เรียนรู้วิทยาศาสตร์ข้อมูลด้วยโปรแกรม R และ RStudio: พื้นฐานการโปรแกรมภาษา R การสร้างภาพข้อมูล และการทำความสะอาดข้อมูล

Data Science using R and RStudio: Basic R Programming, Data Visualization, and Data Wrangling

สุทธิพงษ์ มีใย

สาขาวิชาวิศวกรรมขนส่ง สำนักวิชาวิศวกรรมศาสตร์
มหาวิทยาลัยเทคโนโลยีสุรนารี

คำนำ

ตำราเรียนรู้วิทยาศาสตร์ข้อมูลด้วยโปรแกรม R และ RStudio: พื้นฐานการโปรแกรมภาษา R การสร้างภาพข้อมูล และการทำความสะอาดข้อมูล (Data Science using R and RStudio: Basic R Programming, Data Visualization and Data Wrangling) เล่มนี้ ผู้เขียนจัดทำขึ้นเพื่อใช้เป็นส่วนหนึ่งในเอกสารประกอบการเรียนวิชา ENG22 3015 การเรียนรู้ของเครื่องและการวิเคราะห์ข้อมูล (Machine Learning and Data Analysis) สำหรับนักศึกษาสาขาวิชาวิศวกรรมขนส่ง สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ตำราเล่มนี้รวบรวมจากประสบการณ์สอนในห้องปฏิบัติการคอมพิวเตอร์ ข้อมูลจากหลายแหล่ง รวมทั้งข้อมูลความต้องการใช้งานของสถานประกอบการ เพื่อให้ตรงตามวัตถุประสงค์รายวิชา และนำไปใช้ในการวิเคราะห์ข้อมูลสำหรับงานขนส่งและโลจิสติกส์ และงานด้านวิทยาศาสตร์ข้อมูล (Data science) ที่กำลังเป็นที่นิยมในปัจจุบัน

ในขณะที่เขียนตำราเล่มนี้มีเอกสาร/หนังสือภาษาไทยที่เน้นเรื่องการสร้างภาพข้อมูลอยู่น้อยมาก โดยเฉพาะการทำความสะอาดข้อมูลไม่มีหนังสือที่กล่าวถึงเรื่องนี้โดยเฉพาะ ดังนั้นตำราเล่มนี้นอกจากจะปูพื้นฐานการใช้โปรแกรมภาษา **R** (Basic R programming) แล้วยังเน้นให้นักศึกษา/ผู้อ่านมีความรู้ความเข้าใจการสร้างภาพข้อมูล (Data visualization) โดยเฉพาะการทำความสะอาดข้อมูล (Data wrangling) ซึ่งขั้นตอนดังกล่าวใช้เวลาถึงร้อยละ 80 ของเวลาการทำงานทั้งหมด (Andrews, 2021) ซึ่งจะช่วยให้การทำงานมีประสิทธิภาพมากขึ้น ช่วยลดเวลาการทำงานในขั้นตอนดังกล่าวได้มาก เนื้อหาในเอกสารฉบับนี้ไม่ได้เน้นการใช้ **R** กับงานวิเคราะห์ทางด้านสถิติ (Statistics) หรือแบบจำลอง Machine Learning ผู้ที่สนใจสามารถศึกษาได้ในเอกสารอื่นที่เกี่ยวข้องได้

เนื้อหาแบ่งออกเป็น 3 ส่วน ได้แก่

ส่วนที่ 1 พื้นฐานการโปรแกรมภาษา **R** (Basic R Programming) ประกอบด้วย (1) บทนำ (Introduction) (2) เวกเตอร์ (Vectors) เมทริกซ์ (Matrixes) และอาร์เรย์ (Arrays) (3) ลิสต์ (Lists) เดต้าเฟรม (data.frame) และเดต้าเทเบิ้ล (data.table) (4) ข้อมูลที่ไม่ใช่ตัวเลข (Non-Numeric data types) ค่าเฉพาะ (Special values) คลาส (Class) และการแปลงชนิดข้อมูล (Coercion) (5) การพล็อตเบื้องต้น (Basic plotting) ไฟล์ (Files) และชุดข้อมูลสำหรับโปรแกรม **R** (6) การโปรแกรมเบื้องต้น (7) R Markdown และ R Notebook

ส่วนที่ 2 การสร้างภาพข้อมูล (Data Visualization) ประกอบด้วย (8) `ggplot2` เบื้องต้น (9) การพล็อตกราฟ 1 ตัวแปร (X): ตัวแปรต่อเนื่อง หรือตัวแปรแบ่งกลุ่ม (10) การพล็อตกราฟ 2 ตัวแปร (X and Y): ตัวแปรต่อเนื่อง หรือตัวแปรแบ่งกลุ่ม (11) การพล็อตกราฟ 2 ตัวแปร (X and Y): X ตัวแปรแบ่งกลุ่ม

และ Y ตัวแปรต่อเนื่อง (12) พารามิเตอร์ต่าง ๆ ในกราฟ (Graphical Parameters) (13) ส่วนขยายของ **ggplot2** (Extensions of **ggplot2**)

ส่วนที่ 3 การทำความสะอาดข้อมูล (Data Wrangling) ประกอบด้วย (14) โครงสร้างข้อมูลแบบ **tibble** และเดต้าเฟรม (data.frame) (15) การจัดการข้อมูลด้วยแพ็กเกจ **tidyr** (16) การจัดการข้อมูลด้วยแพ็กเกจ **dplyr** (17) การนำเข้าข้อมูลด้วยแพ็กเกจ **readr** และ **readxl** (18) การจัดการข้อความ (Strings) ด้วยแพ็กเกจ **stringr** (19) การจัดการแฟกเตอร์ (Factors) ด้วยแพ็กเกจ **forcats** (20) การจัดการวันและเวลาด้วยแพ็กเกจ **lubridate** (21) การวนลูปด้วยแพ็กเกจ **purrr**

ผู้เรียบเรียงหวังเป็นอย่างยิ่งว่าเอกสารฉบับนี้จะเป็นประโยชน์ต่อนักศึกษาและผู้ที่สนใจทั่วไปที่ประสงค์จะเรียนรู้หลักการใช้โปรแกรมภาษา **R** หากมีข้อแนะนำที่เป็นประโยชน์ต่อการปรับปรุงเอกสารฉบับนี้ กรุณาแจ้งผู้เรียบเรียงจักเป็นพระคุณยิ่ง (email: sutthi@sut.ac.th)

ผศ. ร.อ. ดร.สุทธิพงษ์ มีใย
สาขาวิชาวิศวกรรมขนส่ง
สำนักวิชาวิศวกรรมศาสตร์
มหาวิทยาลัยเทคโนโลยีสุรนารี

ธันวาคม 2567

สารบัญ

สารบัญตาราง

สารบัญรูป

ส่วนที่ 1
พื้นฐานการโปรแกรมภาษา R (Basic R Programming)

บทที่ 1
บทนำ (Introduction)

1.1 รู้จักโปรแกรมภาษา R

R เป็นภาษาโปรแกรมที่ได้รับอิทธิพลมาจากภาษาโปรแกรม **S** (**S** มาจากอักษรตัวแรกของ Statistics) ซึ่งถูกพัฒนาขึ้นมาโดยนักวิจัยใน Bell Laboratories ตั้งแต่ปี 1976 (Davies, 2016) (การตั้งชื่อใช้อักษรตัวเดียวเช่นเดียวกับภาษา **C**) เพื่อใช้ในงานวิเคราะห์ข้อมูลและสถิติ ต่อมาได้ถูกพัฒนาเป็นโปรแกรมสถิติทางการค้าโดยเพิ่มเติมความสามารถทางด้านกราฟิก (Graphical user interface, GUI) ที่มีชื่อว่าโปรแกรม S-Plus (Chambers, 1998) ต่อมาในปี 1992 Ross Ihaka และ Robert Gentleman จากมหาวิทยาลัย Auckland ประเทศนิวซีแลนด์ได้ร่วมกันพัฒนาโปรแกรมภาษา **R** ขึ้นมา (**R** มาจากอักษรตัวแรกของ Ross และ Robert) **R** ได้รับความนิยมและแพร่หลายอย่างรวดเร็ว เนื่องจากฟรีและเป็น Open-source (Matloff, 2011) ทำให้มีนักพัฒนาทั่วโลกช่วยกันปรับปรุงประสิทธิภาพของ **R** อย่างต่อเนื่อง (Hothorn & Everitt, 2009) โปรแกรม **R** อยู่ภายใต้การดูแลขององค์กรที่ไม่แสวงหากำไร **R** เป็นส่วนหนึ่งในโครงการ GNU (ดูรายละเอียดได้จาก https://www.r-project.org/)

R เป็นภาษาโปรแกรมที่นิยมใช้วิเคราะห์ทางด้านสถิติ (Crawley, 2012) การแสดงผลทางกราฟิก (Graphics) และการสร้างภาพจากข้อมูล (Data visualization) (Chang, 2012; Hadley Wickham, 2016) การทำเหมืองข้อมูล (Data mining) (Zhao, 2012) และงานด้านวิทยาสตร์ข้อมูล (Data Science) (Koushik & Ravindran, 2016) ที่กำลังเป็นที่นิยมในปัจจุบัน[1] รวมทั้งยังใช้ในการวิเคราะห์และประมวลผลข้อมูลทั่วไปได้อย่างมีประสิทธิภาพ โปรแกรม **R** มีข้อดีหลายประการ (Matloff, 2011) ดังต่อไปนี้

- **R** เป็น Freeware ไม่มีค่าใช้จ่ายในการจัดซื้อ/จัดหา ผู้ใช้ไม่ต้องกังวลเรื่องการละเมิดลิขสิทธิ์เหมือนกับโปรแกรมสำเร็จรูปทางสถิติอื่น ๆ
- **R** มีฟังก์ชันใช้สำหรับการวิเคราะห์งานหลากหลายประเภท เช่น สถิติ คณิตศาสตร์ กราฟิก เหมืองข้อมูล ข้อมูลเดือนธันวาคม ปี 2024 จากเว็บไซต์ https://cran.r-project.org ระบุว่ามี **R** แพ็กเกจมากกว่า 21,738 แพ็กเกจ และมีผู้พัฒนาแพ็กเกจเพิ่มขึ้นตลอดเวลา
- **R** มีชนิดข้อมูลที่สะดวกต่อการใช้งาน ไม่ว่าจะเป็นเวกเตอร์ เมทริกซ์ หรือเดต้าเฟรม ทำให้สามารถเขียนโปรแกรมได้กระชับ สร้างชุดคำสั่งได้รวดเร็วกว่าภาษาโปรแกรมโดยทั่วไป เช่น C/C++, Java เป็นต้น

[1] ข้อมูลเดือนธันวาคม ปี พ.ศ. 2567

- **R** มีความสามารถด้านกราฟิก (Graphics) สูง ผู้ใช้สามารถสร้างภาพกราฟิกได้ตามต้องการ
- **R** มี RStudio ซึ่งเป็น editor ที่ใช้ง่าย ประสิทธิภาพสูง มี environment แสดงค่าตัวแปรต่าง ๆ ทำให้การตรวจข้อผิดพลาดทำได้ง่าย
- **R** สามารถเรียกโปรแกรมทางสถิติอื่น ๆ เข้ามาใช้งานได้ มีการพัฒนาให้เชื่อมโยงกับโปรแกรมทางสถิติ/โปรแกรมบางตัวได้ เช่น Excel, SPSS, SAS, Stata เป็นต้น
- **R** สามารถเปิด/เรียกใช้งานไฟล์โปรแกรมอื่น ๆ ได้หลากหลาย เช่น ไฟล์นามสกุล txt, csv, dbf, xlsx, HTML เป็นต้น
- **R** มีเอกสารและตำราเป็นภาษาอังกฤษจำนวนมาก เอกสารหลาย ๆ เล่มดาวน์โหลดได้โดยไม่เสียค่าใช้จ่าย
- **R** มีชุมชน (Community) ผู้ใช้อยู่มากมายทั่วโลก สามารถค้นหาวิธีการแก้ปัญหาได้ง่ายทาง Internet
- **R** รองรับผู้ใช้แทบทุกระบบปฏิบัติการ (Operating systems) ทั้ง Windows, MacOS, หรือ Linux/Unix
- **R** รองรับการคำนวณแบบคู่ขนาน (Parallel computing) ทำให้สามารถประมวลผลการคำนวณที่ซับซ้อน โดยใช้การกระจายไปคำนวณที่เครื่องคอมพิวเตอร์หลายตัวหรือหลาย Core ได้พร้อม ๆ กัน
- **R** มีแพ็กเกจที่สามารถเปิดไฟล์ขนาดใหญ่ ๆ ได้ ไฟล์ขนาดใหญ่มาก ๆ ที่ประมวลผลโดยใช้ Excel มักจะมีปัญหาเรื่องหน่วยความจำของเครื่องไม่เพียงพอ ทำให้ทำงานช้าและไม่มีประสิทธิภาพ **R** สามารถทำงานกับไฟล์ที่มีขนาดใหญ่ สามารถจัดการกับหน่วยความจำได้อย่างมีประสิทธิภาพ

R มีข้อด้อยคือ ดูเหมือนไม่มี User interface[2] ให้ใช้งานได้ง่าย ๆ เหมือน SPSS Minitab หรือ Stata ต้องเขียนคำสั่งและรันผ่าน Console แต่ถ้าผู้ใช้คุ้นเคยกับ **R** บ้างแล้ว จะพบว่า **R** มีความยืดหยุ่นสูง ใช้งานได้อย่างมีประสิทธิภาพ สามารถสั่งการทำงานของโปรแกรมโดยนำชุดคำสั่ง (Code files) กลับมาใช้ใหม่ได้ง่าย (Reproducibility) เมื่อข้อมูลมีการเปลี่ยนแปลงไป ลดระยะเวลาในการทำงานซ้ำ ๆ ทวนสอบ/ตรวจสอบงานที่ทำไว้ได้ง่าย และสามารถสื่อสารกับผู้ใช้อื่นหรือชุมชน (**R** Community) ได้ง่ายเนื่องจากชุดคำสั่งเป็นเท็กไฟล์ (Text files) ธรรดาที่เปิดด้วย editor อะไรก็ได้ ข้อด้อยอีกประการหนึ่งคือ ผู้ใช้มักจะมองว่า **R** ทำงานช้าเมื่อเทียบกับโปรแกรมภาษาอื่น ๆ เช่น C/C++, Java เป็นต้น แต่ผู้เขียนมีความเห็นว่า เมื่อพิจารณาเวลาใน

[2] R มีโปรแกรมเสริมที่มี User interface บางตัว เช่น R Commander ทำให้ใช้งานได้คล้ายโปรแกรมสำเร็จรูปอื่น ๆ แต่ในเอกสารฉบับนี้จะไม่กล่าวถึง เนื่องจากจะเน้นที่โปรแกรม R และโปรแกรม RStudio

การเขียนโปรแกรมรวมกับเวลาประมวลผลโปรแกรมด้วยแล้ว พบว่า งานด้านวิทยาศาสตร์ข้อมูลรวมทั้งงานวิเคราะห์ข้อมูลส่วนใหญ่ การใช้โปรแกรม **R** มีความสะดวกรวดเร็วกว่ามาก การตรวจสอบข้อผิดพลาดทำได้ง่ายกว่าเนื่องจากเป็นโปรแกรมประเภท Interpreted language และมี Environment แสดงค่าตัวแปรต่าง ๆ นอกจากนี้จากประสบการณ์ของผู้ที่นำภาษา **R** ไปใช้งานจริงหลายคน พบว่า การใช้ **R** ช่วยเพิ่มประสิทธิภาพการทำงานมากกว่าเมื่อเทียบกับการใช้ Excel แต่เพียงอย่างเดียว

ผู้อ่านอาจจะสงสัยว่านอกจาก **R** แล้วมีภาษาอื่นที่ใช้แทน **R** ได้หรือไม่? **R** ถูกสร้างมาเพื่อทำหน้าที่วิเคราะห์ข้อมูลทางสถิติ แต่ก็มีประสิทธิภาพในการทำงานด้านวิทยาศาสตร์ข้อมูล (Data science) ได้เป็นอย่างดี และมีฟังก์ชันด้าน Data visualization ที่มีความสามารถสูงมาก ส่วนภาษาที่เป็นที่นิยมอย่าง Python ออกแบบมาเพื่องานโปรแกรมทั่วไป (General programming language) ซึ่งมีแพ็กเกจทำงานด้านวิทยาศาสตร์ข้อมูลได้เป็นอย่างดี และเหมาะกับการประยุกต์ใช้งานด้านปัญญาประดิษฐ์ (Artificial intelligence) มากกว่า **R** นอกจากนี้ยังมีภาษาค่อนข้างใหม่อย่างเช่น Julia ซึ่งเหมาะกับการคำนวณด้านวิศวกรรม และการทำงานด้านวิทยาศาสตร์ข้อมูลได้ดีอีกภาษาหนึ่ง ผู้เขียนเลือกที่นำภาษา **R** มาใช้ทั้งการปฏิบัติงานจริง[3] งานวิจัย[4] และใช้ในการเรียนการสอนเนื่องจากเหตุผลดังนี้

1. ภาษา **R** เหมาะกับการวิเคราะห์ข้อมูลทางสถิติ ซึ่งในบรรดาภาษาทั้งหมดที่กล่าวถึงยังไม่มีภาษาไหนเหมาะกับงานด้านสถิติเท่ากับ **R**
2. ภาษา **R** ทำงานด้านวิทยาศาสตร์ข้อมูลได้เป็นอย่างดีไม่ด้อยไม่กว่าภาษาอื่น มีแพ็กเกจ `tidyverse` ซึ่งทำให้การเขียน code เพื่อการวิเคราะห์ข้อมูลที่ซับซ้อนทำได้ง่าย โดยทั่วไปการทำงานในขั้นตอนการทำความสะอาดข้อมูลใช้เวลาถึงร้อยละ 80 ของเวลาทั้งหมด (Andrews, 2021) การใช้แพ็กเกจ `tidyverse` จะช่วยให้การทำงานมีประสิทธิภาพมากขึ้น ช่วยลดเวลาการทำงานในขั้นตอนดังกล่าวได้มาก
3. ภาษา **R** ใช้งานได้ง่าย มี RStudio ซึ่งเป็น IDE ที่มี Environment แสดงค่าตัวแปรต่าง ๆ (คล้ายกับ Environment ของ MatLab) และดัชนี (Index) ของเวกเตอร์ (เทียบเท่ากับ array ใน

[3] ตัวอย่างการประยุกต์ใช้ R ในการปฏิบัติงาน ได้แก่ (1) การวิเคราะห์ข้อมูลจากแบบสำรวจจุดต้นทางและจุดปลายทางการเดินทางเพื่อพัฒนาแบบจำลองการคาดการณ์ความต้องการเดินทางด้วยระบบรางในเขตกรุงเทพมหานครและปริมณฑล (พื้นที่ต่อเนื่อง) ระยะที่ 2 (2) การวิเคราะห์ข้อมูลจากแบบสำรวจการสัมภาษณ์ครัวเรือน เพื่อพัฒนาแบบจำลองการคาดการณ์ความต้องการเดินทางด้วยระบบรถไฟฟ้ารางเบาเมืองพัทยา และ (3) การสร้างแบบจำลองโครงข่ายระบบขนส่งสาธารณะในเขตกรุงเทพมหานครและปริมณฑล

[4] ตัวอย่างการประยุกต์ใช้ R ในงานวิจัย ได้แก่ (1) การจัดเตรียมข้อมูลอุบัติเหตุและการวิเคราะห์ข้อมูล เพื่อสร้างแบบจำลองอุบัติเหตุสำหรับรถจักรยานยนต์บนทางหลวงขนาด 2 ช่องจราจร (2) การจัดเตรียมข้อมูลและการประมาณค่าพารามิเตอร์ด้วย Apollo Choice Modelling ในงานวิจัย เช่น การวิเคราะห์ความยืดหยุ่นการเดินทางระหว่างเมือง เป็นต้น

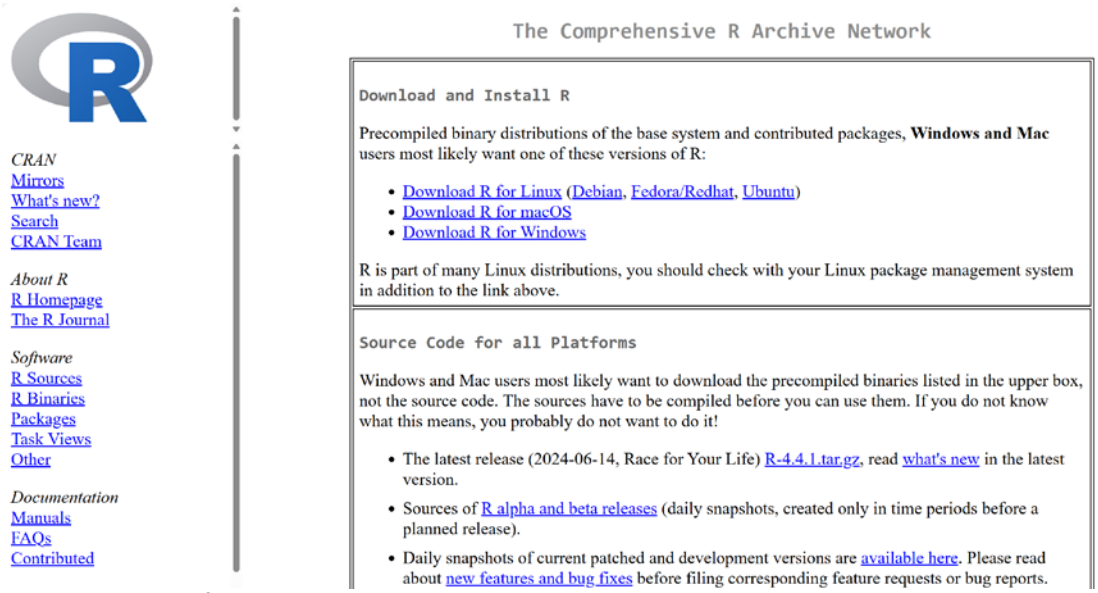
ภาษาคอมพิวเตอร์อื่น) เริ่มจากเลข 1 ทำให้เหมาะกับการคำนวณทางคณิตศาสตร์ สถิติ และวิศวกรรมโดยทั่วไป

4. ภาษา **R** มี R Markdown และ R Notebook ซึ่งสามารถเขียนคำอธิบาย ใส่สัญลักษณ์ สูตร สมการ เครื่องหมาย รูปภาพ หรือ Comments ต่าง ๆ ได้เหมือนการเขียน HTML ร่วมกับการเขียนโค้ดอยู่บน Platform เดียวกัน (คล้ายกับ Jupyter notebook[5] ในภาษา Python)

1.1.1 การดาวน์โหลดโปรแกรม R

ผู้ใช้สามารถดาวน์โหลดโปรแกรม **R** ได้โดยการไปที่หน้าเว็บไซต์ CRAN (Comprehensive **R** Archive Network) ตาม URL ต่อไปนี้ https://cran.r-project.org/ แสดงหน้าเว็บไซต์ในรูปที่ 1-1

ตัวอย่างในเอกสารนี้จะใช้ระบบปฏิบัติการวินโดว์ โดยคลิกที่ Download **R** for Windows หลังจากนั้น คลิกเลือก Base และคลิกที่ Download **R**-(Version) for Windows ในวงเล็บเป็นตัวเลข Version ที่เผยแพร่ เช่น Download **R**-4.4.1 for Windows หลังจากนั้นคอมพิวเตอร์จะทำการดาวน์โหลดโปรแกรมสำหรับติดตั้งโปรแกรม **R** และจะได้ไฟล์ชื่อ **R**-(Version)-win.exe เช่น **R**-4.4.1-win.exe เป็นต้น

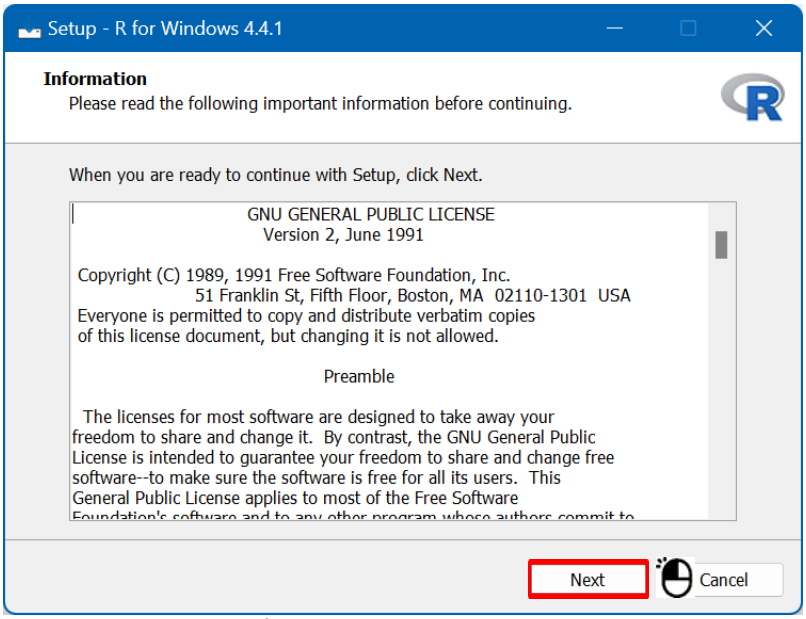


รูปที่ 1-1 เว็บไซต์ CRAN (Comprehensive R Archive Network)

[5] ผู้สนใจสามารถศึกษาเพิ่มเติมได้จาก https://jupyter.org/

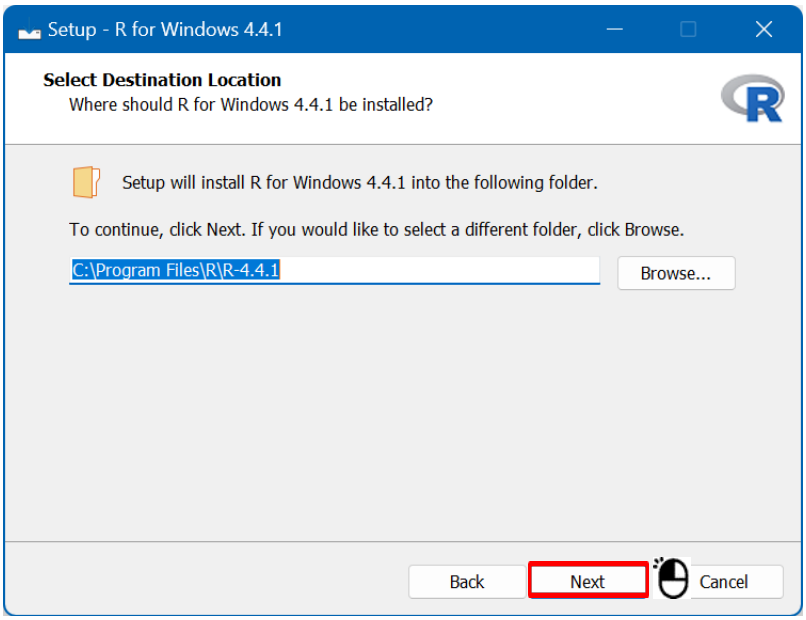
1.1.2 การติดตั้งโปรแกรม R

เลือกไฟล์สำหรับติดตั้งที่ดาวน์โหลดมา ดับเบิ้ลคลิก (Double click) ที่ไฟล์ดังกล่าว คอมพิวเตอร์จะถามว่าต้องการติดตั้งโปรแกรม **R** หรือไม่ ให้ผู้ใช้ตอบตกลง (Yes) หลังจากนั้นทำการเลือกภาษาที่ต้องการ ในที่นี้จะเลือกภาษาอังกฤษและตอบตกลง (Yes) หลังจากนั้นจะปรากฏ License agreement ให้ผู้ใช้อ่านรายละเอียดเกี่ยวกับลิขสิทธิ์ เสร็จแล้วคลิก Next ดังรูปที่ 1-2



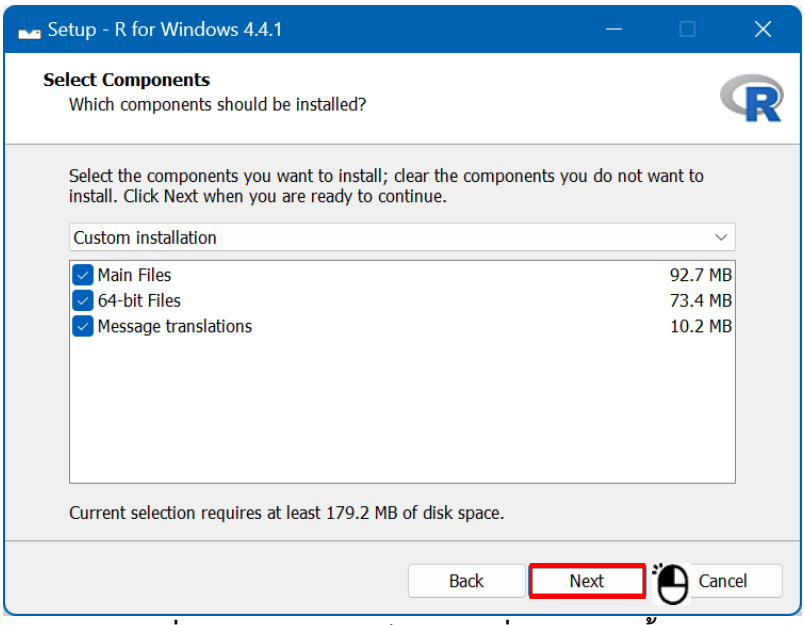
รูปที่ 1-2 License agreement

โปรแกรมจะให้เลือกโฟลเดอร์ (Folder) ที่จะติดตั้งโปรแกรม **R** ให้เลือกโฟลเดอร์ที่ต้องการติดตั้ง ตัวโปรแกรมจะเลือกค่าโดยปริยาย (Default value) ที่ C:\Program Files**R****R**-(Version) ดังรูปที่ 1-3



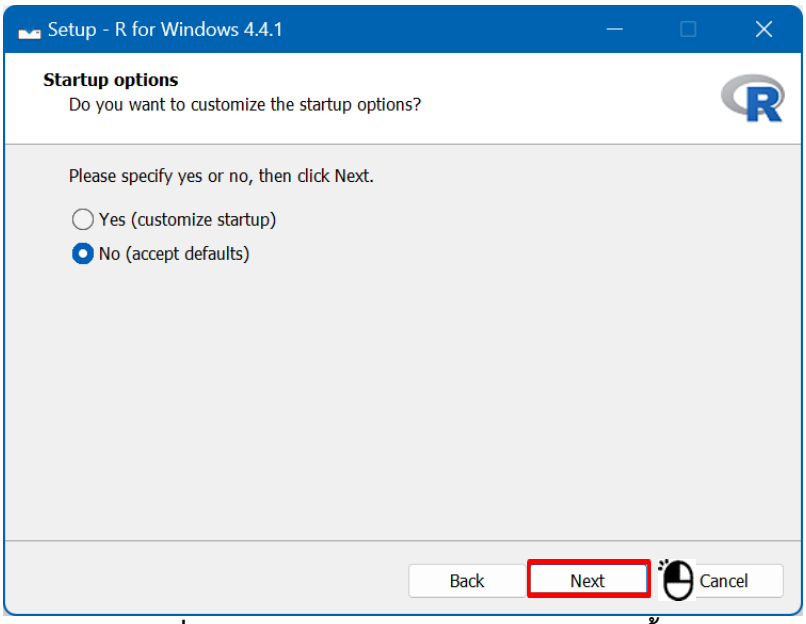
รูปที่ 1-3 การเลือกโฟลเดอร์ที่ต้องการติดตั้งโปรแกรม R

เลือกองค์ประกอบของโปรแกรมที่ต้องการติดตั้ง โดยคลิกเลือก ☑ ที่องค์ประกอบที่ต้องการ อาทิ เลือก Main Files, Message translations, 64-bit Files แล้วแต่ระบบปฏิบัติการที่ใช้อยู่ แล้วคลิก Next ดังรูปที่ 1-4



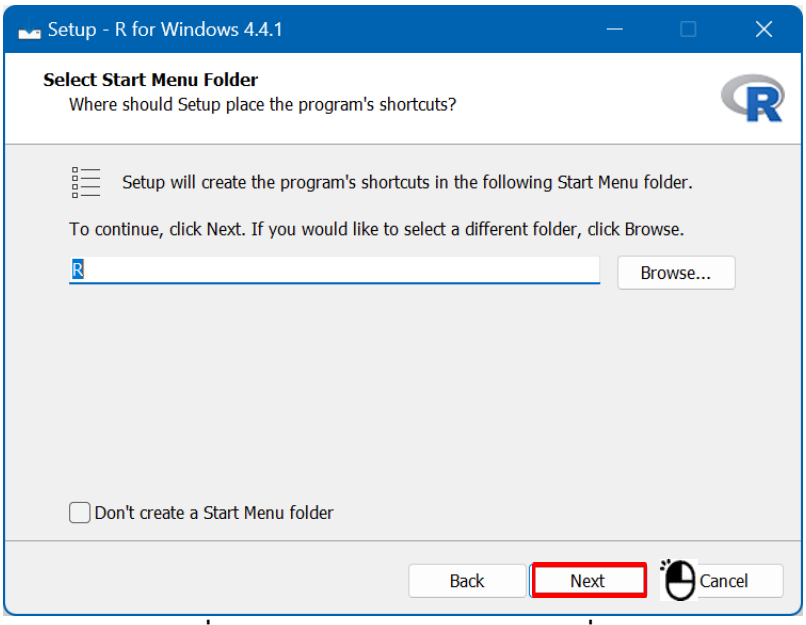
รูปที่ 1-4 การเลือกองค์ประกอบที่ต้องการติดตั้ง

หลังการนั้นโปรแกรมจะให้ผู้ใช้ระบุว่าต้องการตั้งค่าตอนเริ่มโปรแกรมหรือไม่ ในที่นี้จะไม่ปรับเปลี่ยนอะไร ใช้ค่าปริยายโดยเป็นการเลือก No (accept defaults) และคลิก Next ดังรูปที่ 1-5



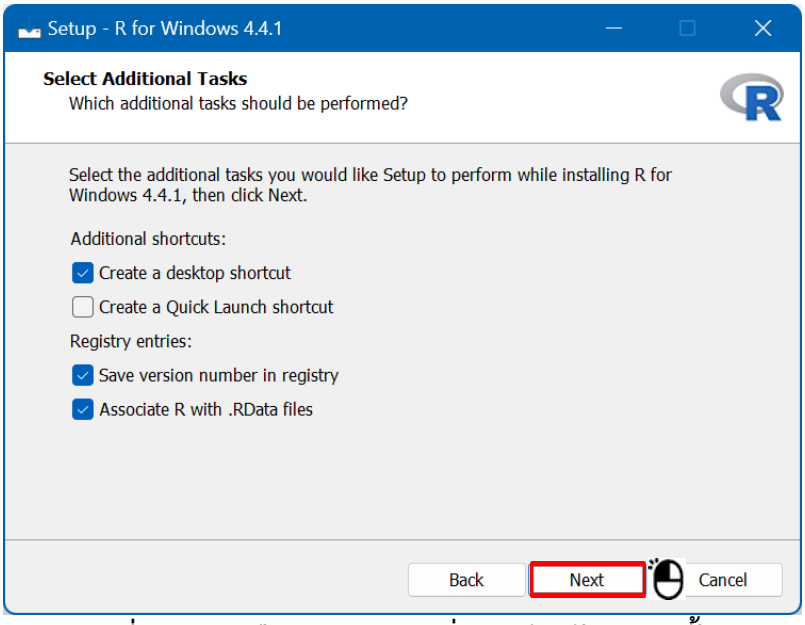
รูปที่ 1-5 การเลือก Options สำหรับการติดตั้ง

หลังจากนั้นโปรแกรมจะให้ผู้ใช้ระบุเกี่ยวกับการเลือก Menu Folder เริ่มต้น ในที่นี้จะไม่ปรับเปลี่ยนอะไร ใช้ค่าปริยาย และคลิก Next ดังรูปที่ 1-6



รูปที่ 1-6 การเลือก Menu Folder เริ่มต้น

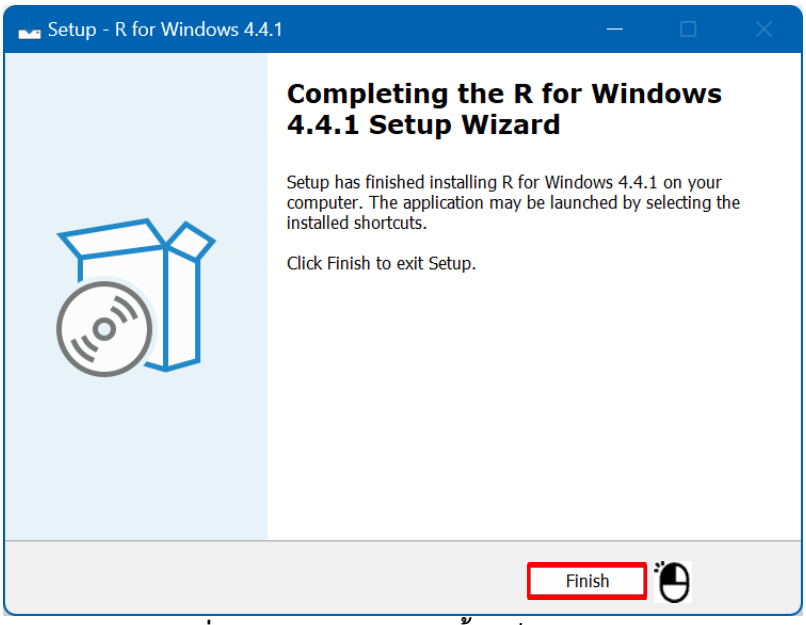
หลังการนั้นโปรแกรมจะให้ผู้ใช้ระบุว่าต้องการตั้งค่าเพิ่มเติมหรือไม่ ในที่นี้จะไม่ปรับเปลี่ยนอะไร ใช้ค่าปริยายโดยเป็นการเลือก Create a desktop shortcut, Save version number in registry และ Associate **R** with .RData files และคลิก Next ดังรูปที่ 1-7



รูปที่ 1-7 การเลือก Options เพิ่มเติมสำหรับการติดตั้ง

R จะทำการติดตั้งองค์ประกอบต่าง ๆ โดยใช้เวลาสักครู่ เมื่อติดตั้งเรียบร้อยจะแสดงผลดังรูปที่ 1-8 คลิก Finish เป็นอันเสร็จสิ้นการติดตั้งโปรแกรม **R**

หมายเหตุ: รายละเอียดปลีกย่อยในการติดตั้งโปรแกรม **R** อาจจะแตกต่างกันออกไป สำหรับโปรแกรม **R** แต่ละรุ่น (Version)



รูปที่ 1-8 โปรแกรม R ติดตั้งเสร็จเรียบร้อย

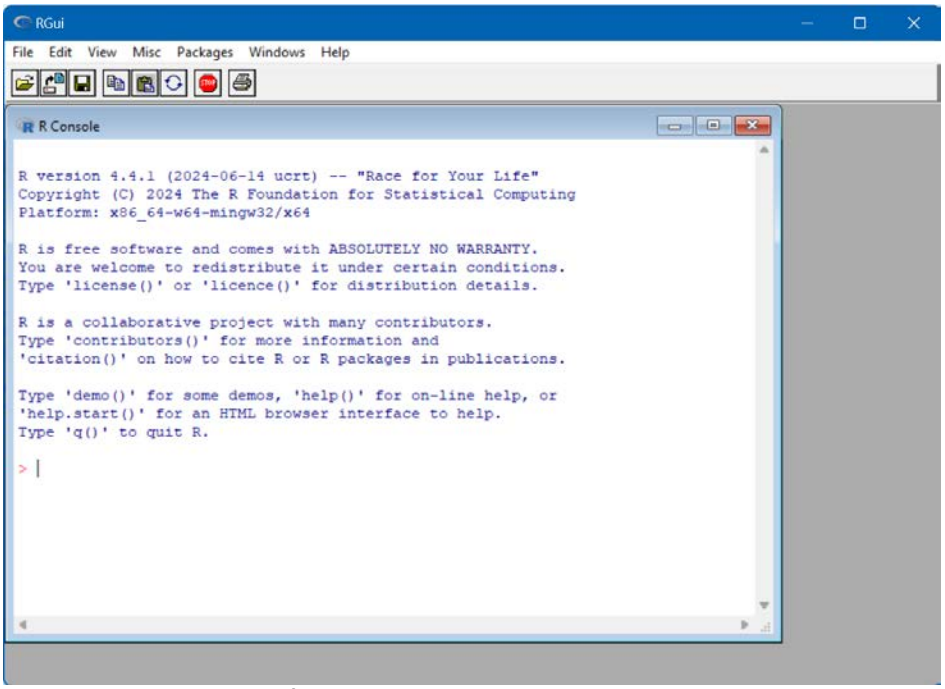
เมื่อต้องการใช้งานโปรแกรม **R** เรียกใช้ได้หลายวิธี เช่น เรียกจาก Startup menu หรือ คลิกที่ shortcut โปรแกรม **R** หรือ คลิกที่ Quick Launch Icon

1.1.3 แนะนำโปรแกรม R

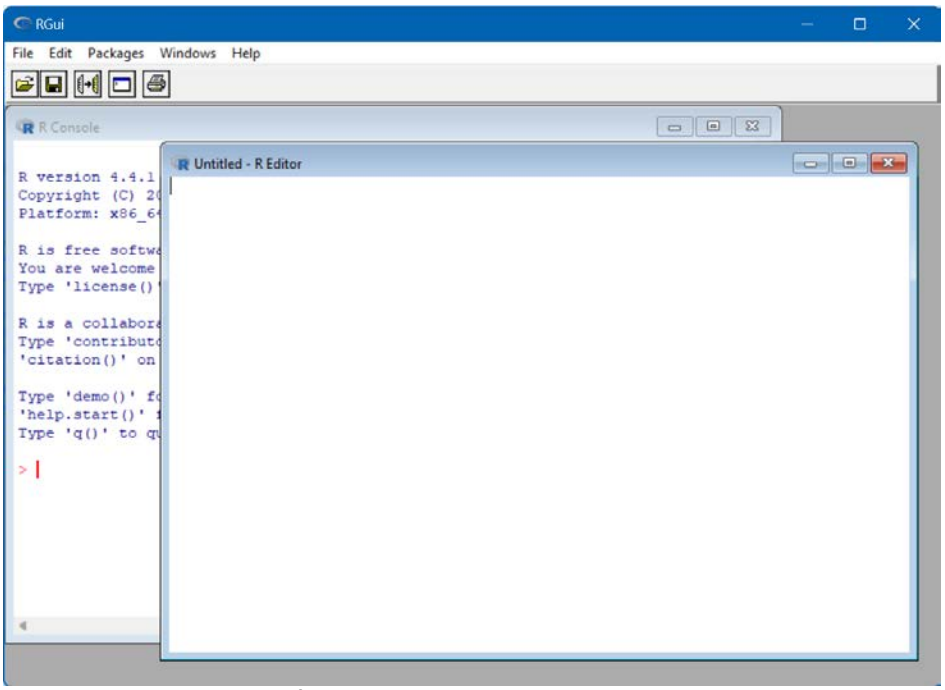
ตัวโปรแกรม **R** เป็นโปรแกรมแบบ Interpreter โดยจะประมวลผลทีละบรรทัด และแสดงผลการรันและข้อความเตือน (Warning message) ออกมาทางหน้าจอ (**R** Concole) ภาษาโปรแกรม **R** มีลักษณะ Case-sentitive คือ อักษรตัวใหญ่และอักษรตัวเล็ก **R** จะมองว่าแตกต่างกันเหมือนกับภาษา C/C++, Java, Python เป็นต้น

เมื่อเปิดโปรแกรม **R** ขึ้นมาจะได้ **R** Console ดังรูปที่ 1-9 สามารถเปิด **R** Editor เพื่อพิมพ์ชุดคำสั่ง โดยเลือก Menu > File > New script[6] หรือกด Ctrl+N ดังรูปที่ 1-10

[6] เอกสารฉบับนี้จะใช้เครื่องหมายมากกว่า > ในการเลือกเมนูต่าง ๆ ของโปรแกรม เช่น Menu > File หมายถึงเลือกคำสั่ง Menu ต่อด้วย File

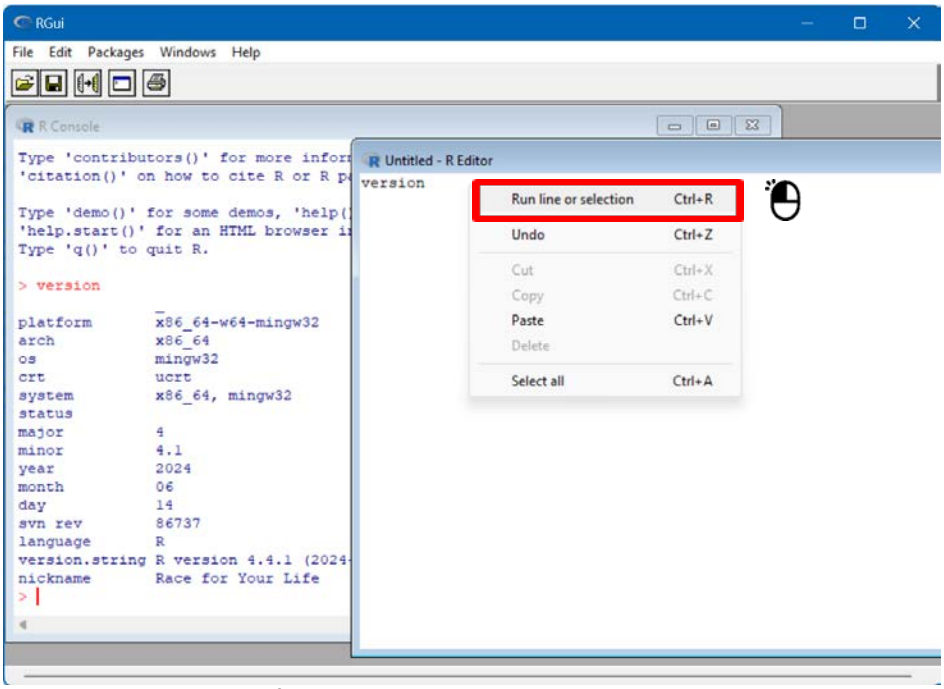


รูปที่ 1-9 โปรแกรม R และ R Console



รูปที่ 1-10 โปรแกรม R และ R Editor

พิมพ์ตัวอย่างคำสั่ง **version** ใน **R** Editor และสั่งรันชุดคำสั่งโดยการคลิกขวาที่ **R** Editor เลือก Run line or selection หรือ Ctrl+R ในตำแหน่งแถวที่ต้องการรัน หรือเลือกชุดคำสั่งที่ต้องการรัน ดังรูปที่ 1-11 และสามารถบันทึกชุดคำสั่งเป็นไฟล์เก็บไว้ใช้งานต่อไป โดยการเลือก Menu > File > Save หรือ Ctrl+S เลือกโฟลเดอร์และตั้งชื่อไฟล์ที่ต้องการ



รูปที่ 1-11 R Editor และการรันโปรแกรม

ถ้าต้องการเปิดไฟล์คำสั่งที่เคยสร้างไว้แล้ว เลือก Menu > File > Open script... ให้เลือกโฟลเดอร์และไฟล์ที่ต้องการ

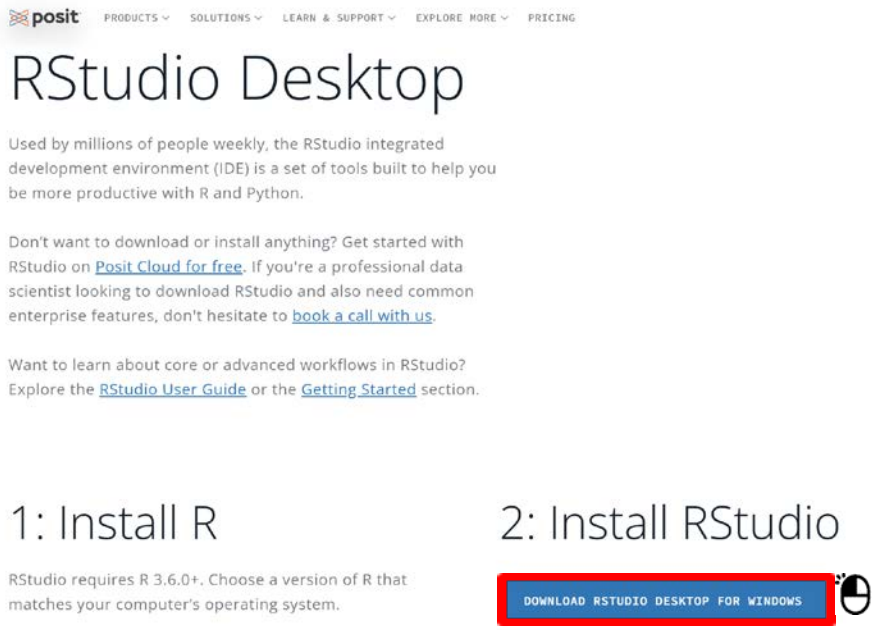
1.2 รู้จักโปรแกรม RStudio

RStudio เป็นโปรแกรม Integrated Development Environments (IDEs) ที่ออกแบบสำหรับการใช้งานโปรแกรม **R** โดยเฉพาะ เหมือนกับโปรแกรม **R** คือ ไม่มีค่าใช้จ่าย รองรับผู้ใช้แทบทุกระบบปฏิบัติการ (Operating systems) เช่น Windows, MacOS, หรือ Linux/Unix แต่ก่อนที่จะติดตั้งโปรแกรม RStudio จะต้องติดตั้งโปรแกรม **R** พื้นฐานก่อน (ดูหัวข้อก่อนหน้านี้) RStudio ทำให้การใช้งาน **R** ง่าย สะดวก และคล่องตัวมากขึ้น ในเอกสารฉบับนี้จะใช้โปรแกรม RStudio เป็น IDE หลักในการใช้งานต่อ ๆ ไป

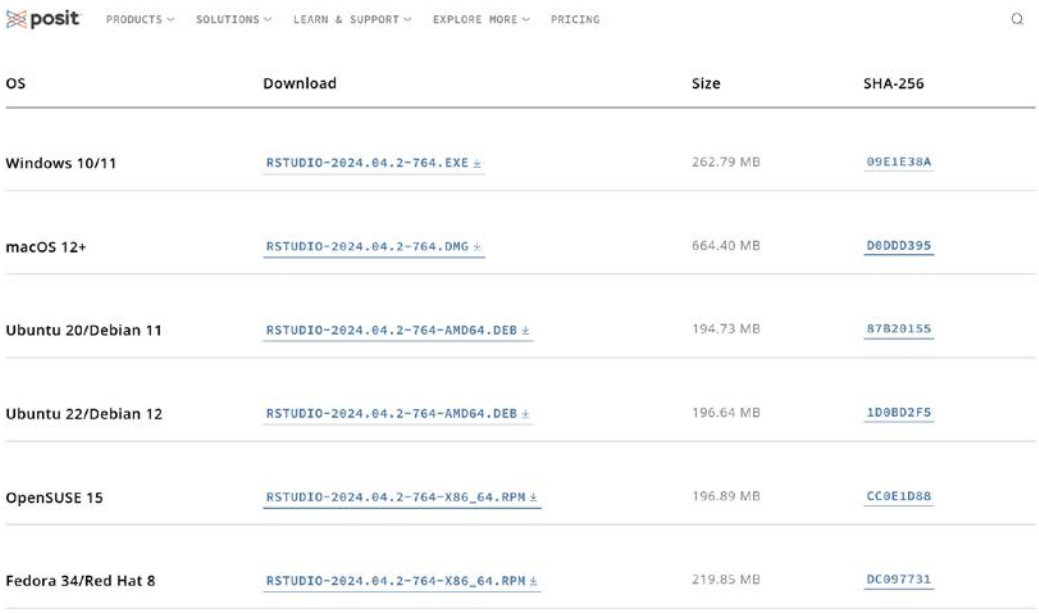
1.2.1 การดาวน์โหลดโปรแกรม RStudio

ผู้ใช้สามารถดาวน์โหลดโปรแกรม RStudio ได้โดยไปที่เว็บไซต์ Posit ตาม URL ต่อไปนี้ https://posit.co/download/rstudio-desktop/ แสดงหน้าเว็บไซต์ในรูปที่ 1-12 โดยตัวอย่างในเอกสารนี้จะใช้ระบบปฏิบัติการวินโดว์ ให้คลิกเลือก Download RStudio Desktop for Windows ซึ่งจะได้ RStudio Version ล่าสุดสำหรับ Windows หลังจากนั้นคอมพิวเตอร์จะทำการดาวน์โหลดโปรแกรมสำหรับติดตั้งโปรแกรม RStudio และจะได้ไฟล์ชื่อ RStudio-(Version).exe เช่น RStudio-2024.04.2-764.exe เป็นต้น

หากผู้ใช้ต้องการดาวน์โหลดโปรแกรมสำหรับติดตั้งในระบบปฏิบัติการอื่น ๆ ให้เลื่อนเว็บไซต์ลงมาจะเจอหน้าเว็บไซต์สำหรับดาวน์โหลดโปรแกรมในแต่ละระบบปฏิบัติการแสดงดังรูปที่ 1-13 จากนั้นคลิกเลือกโปรแกรมสำหรับติดตั้งในคอลัมน์ Download ที่ตรงกับระบบปฏิบัติการของเครื่องคอมพิวเตอร์



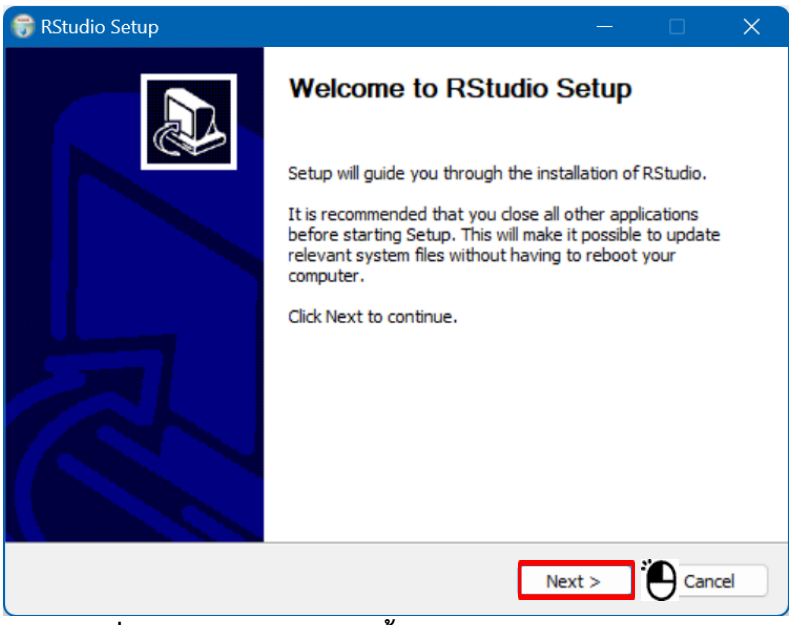
รูปที่ 1-12 เว็บไซต์ Posit สำหรับดาวน์โหลดโปรแกรม RStuduio



รูปที่ 1-13 เว็บไซต์ RStudio และลิงค์ดาวน์โหลดโปรแกรม

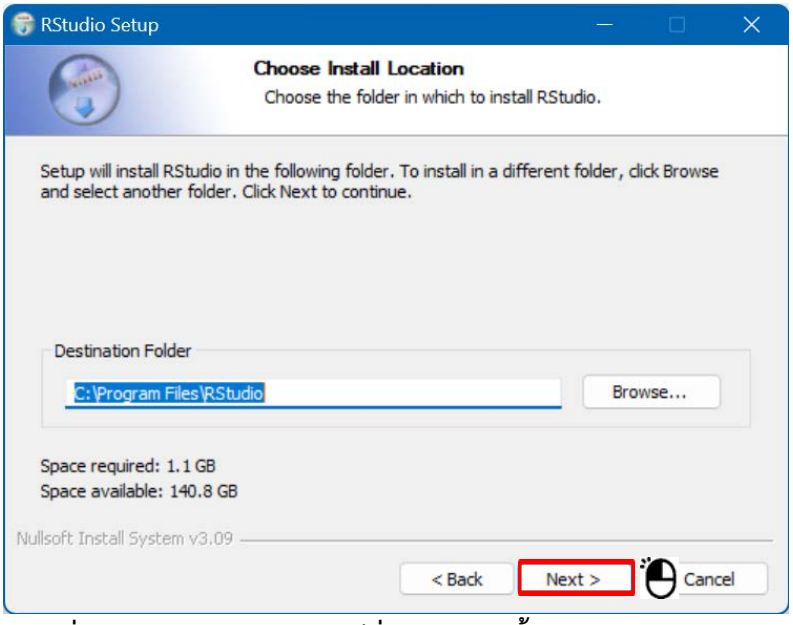
1.2.2 การติดตั้งโปรแกรม RStudio

เลือกไฟล์สำหรับติดตั้งที่ดาวน์โหลดมา ดับเบิ้ลคลิก (Double clicks) ที่ไฟล์ดังกล่าว คอมพิวเตอร์จะถามว่าต้องการติดตั้งโปรแกรม RStudio หรือไม่ ให้ผู้ใช้ตอบตกลง (Yes) หลังจากนั้นจะปรากฏหน้าต่าง RStudio Setup คลิก Next ดังรูปที่ 1-14



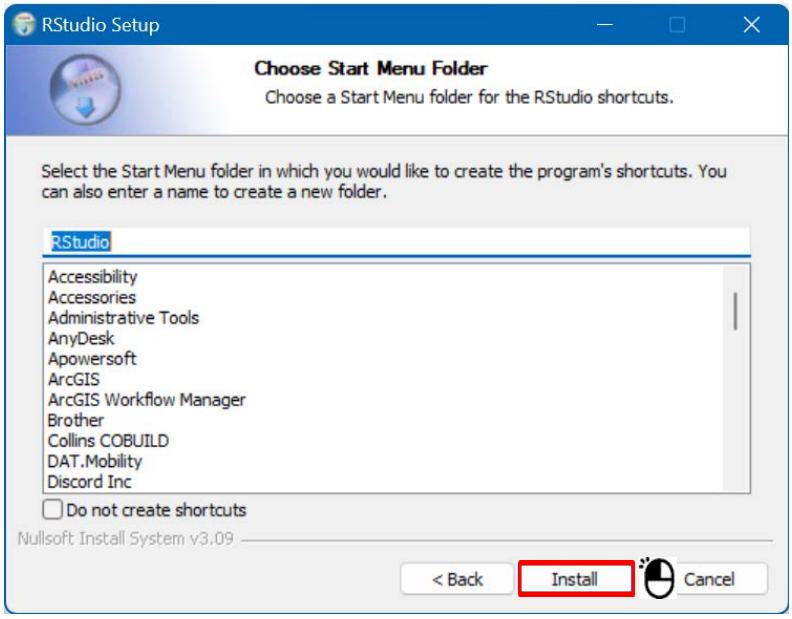
รูปที่ 1-14 หน้าต่างการติดตั้งโปรแกรม RStudio Setup

โปรแกรมจะให้เลือกโฟลเดอร์ (Folder) ที่จะติดตั้งโปรแกรม RStudio และโปรแกรมจะเลือกค่าโดยปริยายที่ C:\Program Files\RStudio ให้เลือกโฟลเดอร์ที่ต้องการติดตั้ง จากนั้นคลิก Next ดังรูปที่ 1-15



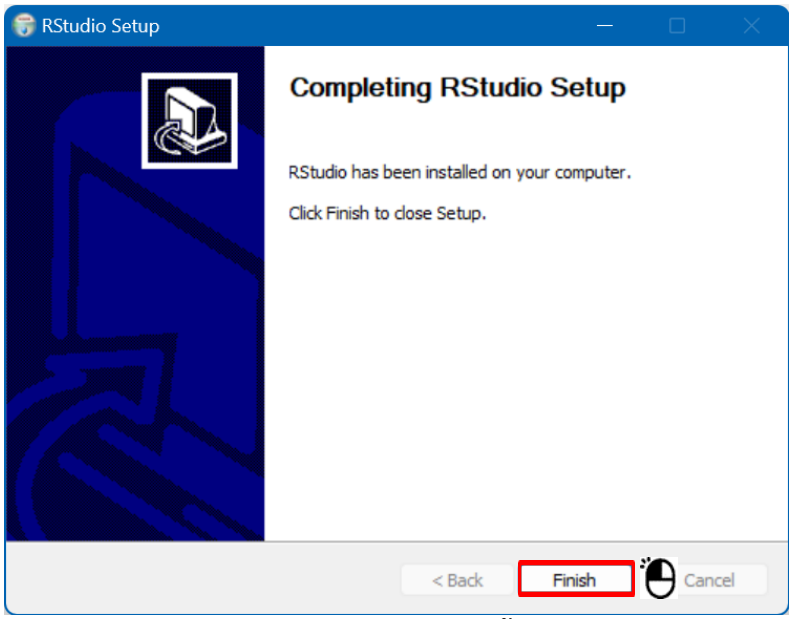
รูปที่ 1-15 การเลือกโฟลเดอร์ที่ต้องการติดตั้งโปรแกรม RStudio

โปรแกรมจะให้เลือก Menu Folder เริ่มต้น ในที่นี้จะไม่ปรับเปลี่ยนอะไร ใช้ค่าปริยาย จากนั้นคลิก Install ดังรูปที่ 1-16



รูปที่ 1-16 การเลือก Menu Folder เริ่มต้นของ RStudio

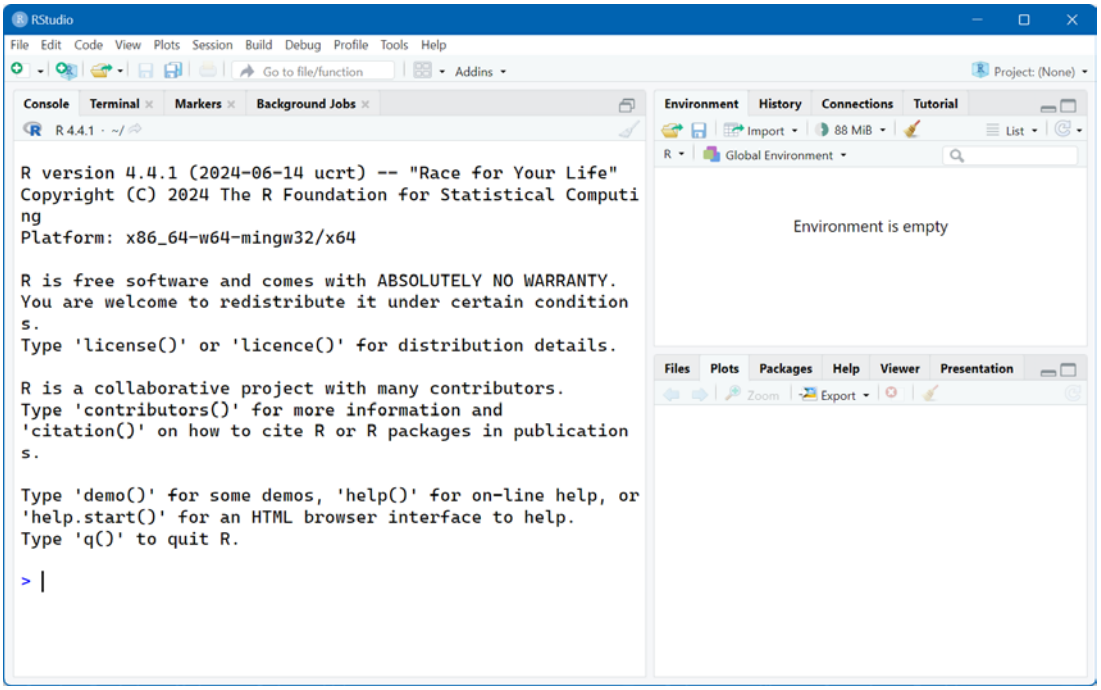
RStudio จะทำการติดตั้งองค์ประกอบต่าง ๆ เมื่อติดตั้งเรียบร้อยจะแสดงผลดังรูปที่ 1-17 คลิก Finish เป็นอันเสร็จสิ้นการติดตั้งโปรแกรม RStudio



รูปที่ 1-17 โปรแกรม RStudio ติดตั้งเสร็จเรียบร้อย

1.2.3 แนะนำโปรแกรม RStudio

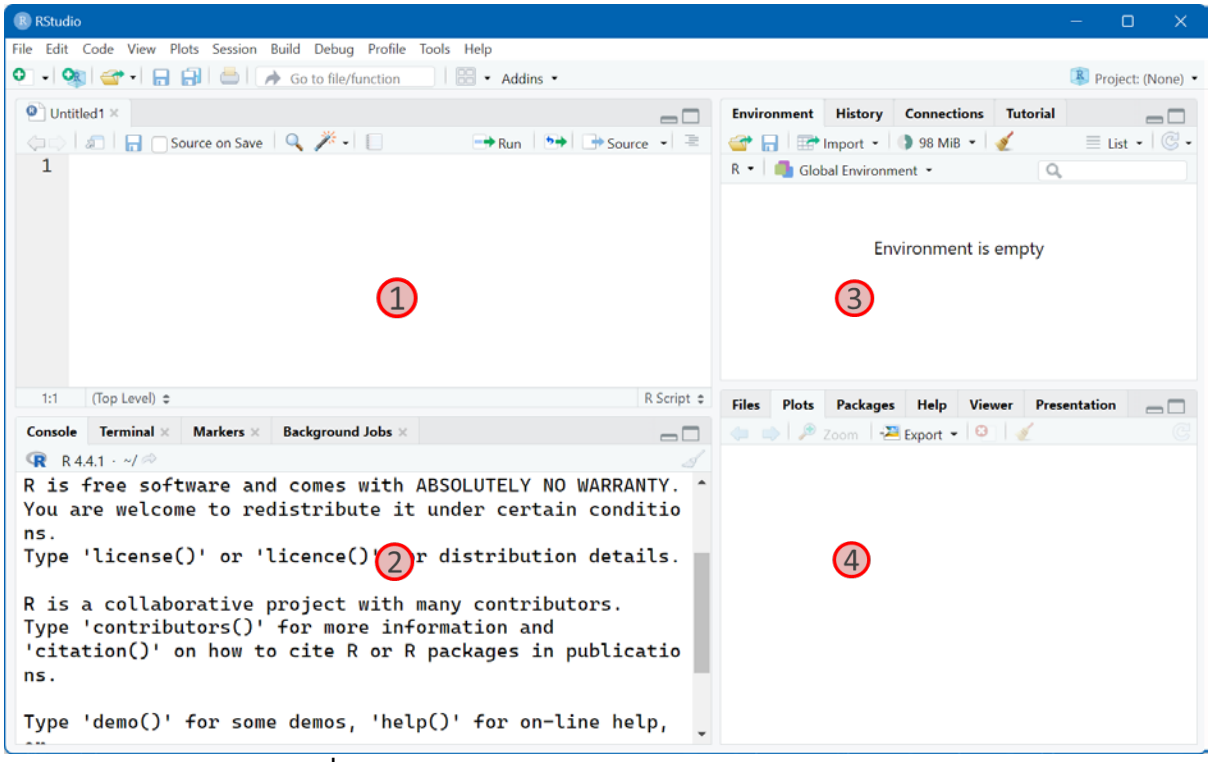
เมื่อเปิดโปรแกรม RStudio ครั้งแรกจะได้หน้าตาของโปรแกรมดังรูปที่ 1-18 ประกอบด้วย Console ทางด้านซ้ายมือ ผู้ใช้สามารถพิมพ์คำสั่งตอบโต้ได้เหมือนโปรแกรม **R** โดยทั่วไป



รูปที่ 1-18 โปรแกรม RStudio

ผู้ใช้สามารถสร้างไฟล์ชุดคำสั่งได้โดยเลือก Menu > File > New File > **R** Script หรือ Ctrl+Shift+N จะได้หน้าต่างสำหรับการพิมพ์ชุดคำสั่งด้านบนซ้ายมือ ดังรูปที่ 1-19 ผู้ใช้สามารถพิมพ์คำสั่งลงในหน้าต่าง **R** Script และสั่งรันโดยการกด Ctrl+Enter หรือ Ctrl+R

RStudio ประกอบด้วยหน้าต่างทั้งหมด 4 ส่วน คือ (1) **R** Script (2) Console, Terminal, Markers, Background Jobs (3) Environment, History, Connections, Tutorial และ (4) Files, Plots, Packages, Help, Viewer, Presentation ดังรูปที่ 1-19



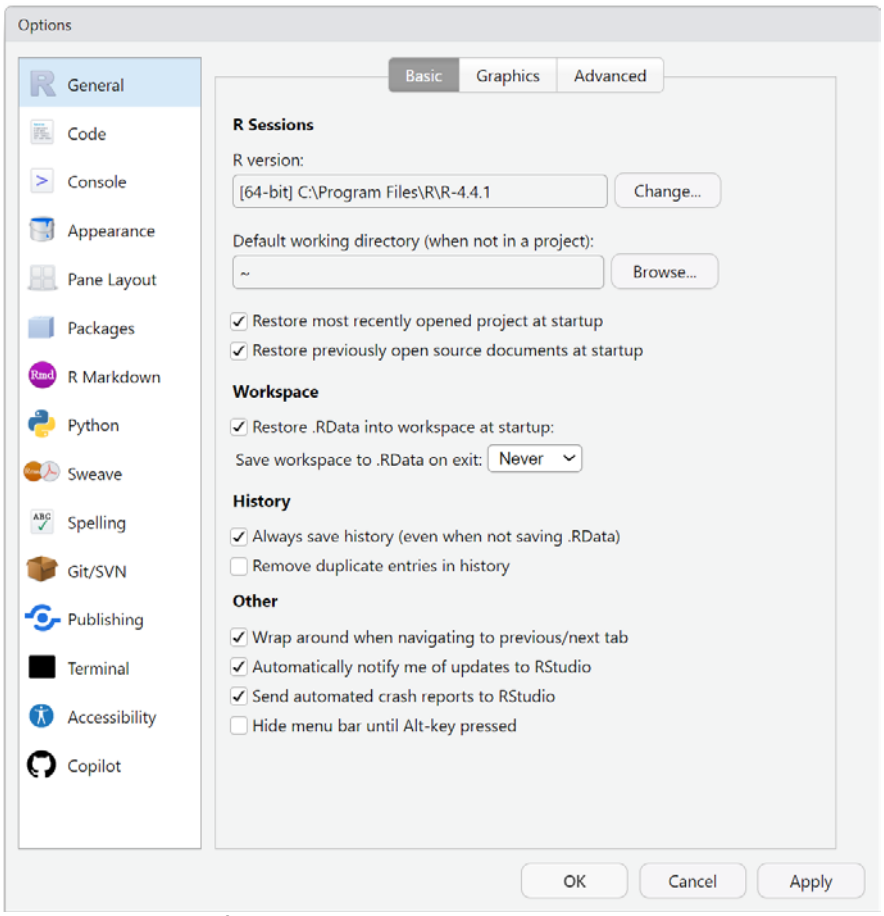
รูปที่ 1-19 โปรแกรม RStudio พร้อมหน้าต่าง R Script

จุดเด่นอย่างหนึ่งของ RStudio คือ ตัว R Script Editor จะแสดงสีของคำสั่งต่าง ๆ ตามธีมที่กำหนดไว้ (Color-themed code highlighting) แสดงการจับคู่วงเล็บ (Bracket matching) ให้สังเกตได้ง่าย เขียนคำสั่งได้สะดวก ลดข้อผิดพลาดลงได้มาก รวมทั้งสามารถเติมคำสั่งที่เหลือ (Autocomplete) ให้ผู้ใช้พิมพ์ไม่กี่อักษร ตัวโปรแกรมก็จะมี Pop-up คำสั่งที่เหลือมาให้เลือกใช้ได้

ผู้ใช้สามารถกำหนดและปรับแต่งการใช้งานได้ตามที่ต้องการ โดยเลือก Menu > Tools > Global Options... จะได้หน้าต่าง Options ดังรูปที่ 1-20 Options ประกอบด้วยส่วนต่าง ๆ ที่สำคัญ ดังนี้

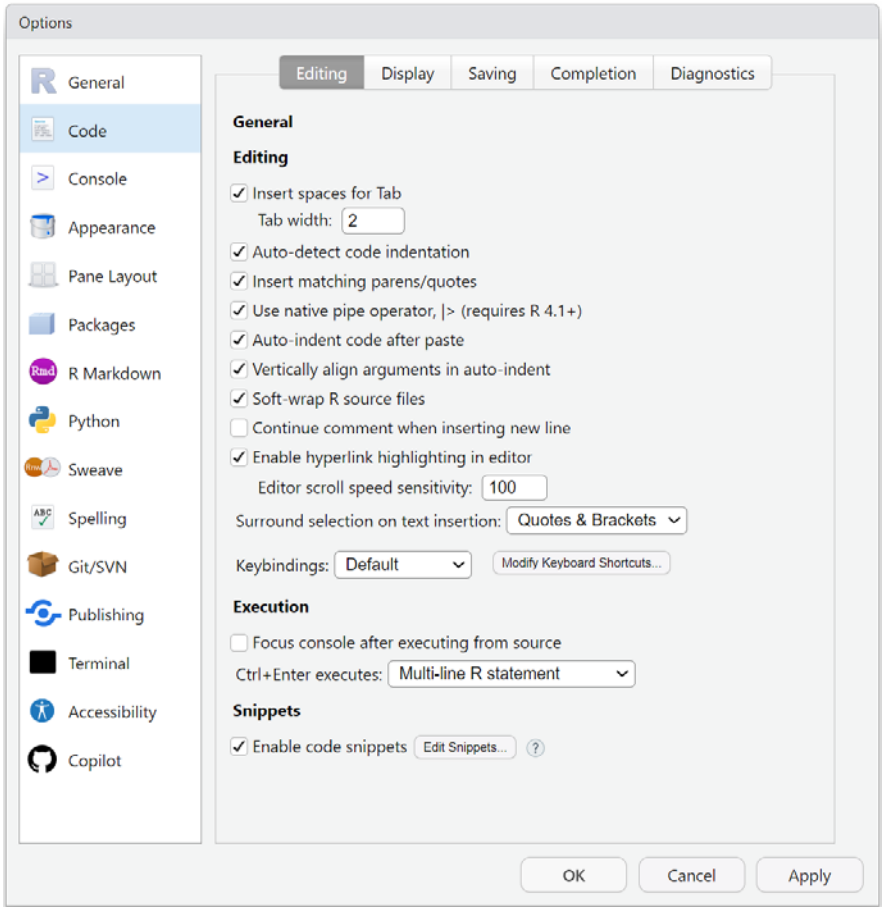
- General: สำหรับกำหนดค่าทั่วไป เช่น ตัวโปรแกรม R ที่ใช้สำหรับโปรแกรม RStudio กำหนดตำแหน่งของโฟลเดอร์เริ่มต้น เป็นต้น
- Code: สำหรับกำหนดค่าใน Editor การแสดงผล Editor และการกำหนด Code completion เป็นต้น
- Console: สำหรับปรับแต่งการแสดงผลใน Console เป็นต้น
- Appearance: สำหรับกำหนดธีมของ Editor ลักษณะตัวอักษร (Font) ขนาดของตัวอักษร (Size) เป็นต้น
- Pane Layout: สำหรับตำแหน่งของหน้าต่างทั้ง 4 ส่วน ได้แก่ คือ (1) Source (2) Console (3) Environment, History, Connections, Tutorial และ (4) Files, Plots, Packages, Help, Viewer, Presentation เป็นต้น
- Packages: สำหรับคุณสมบัติต่าง ๆ ของแพ็กเกจ

- อื่น ๆ ได้แก่ R Markdown, Python, Sweave, Spelling, Git/SVN, Publishing, Terminal, Accessibility และ Copilot



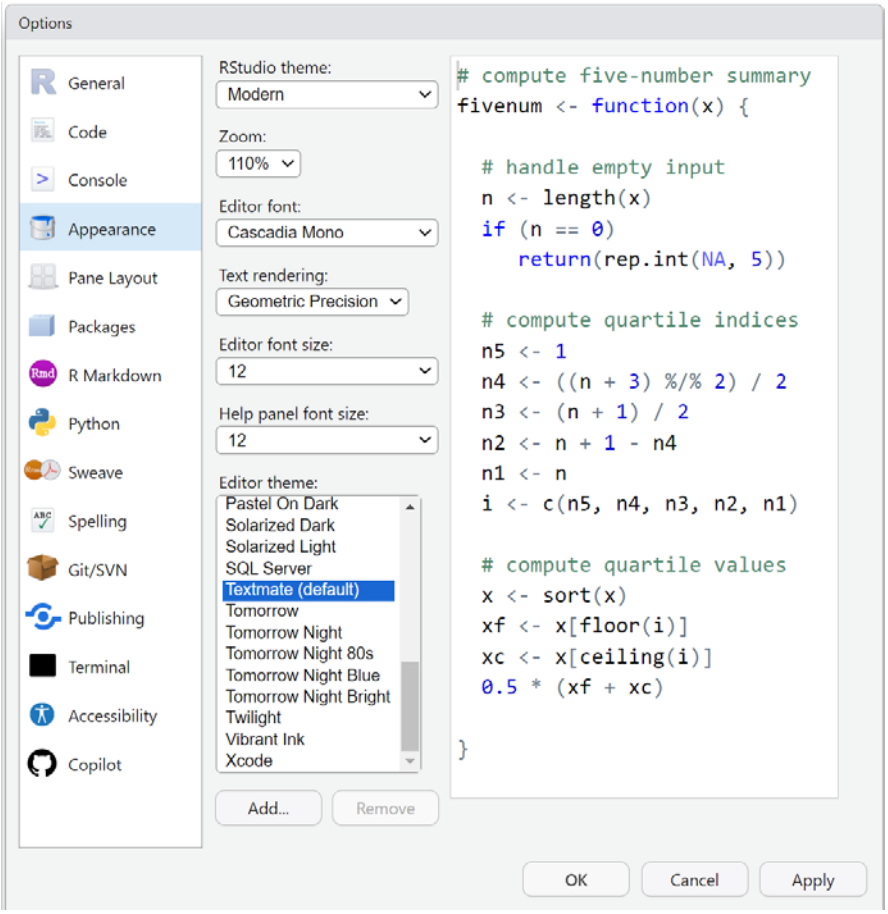
รูปที่ 1-20 Options ของโปรแกรม RStudio

ผู้ใช้สามารถปรับแต่ง Editor โดยเลือก Menu > Tools > Global Options... > Code แล้วเลือก Tab ต่าง ๆ ได้แก่ Editing, Display, Saving, Completion, Diagnostics แล้วเลือกปรับรายละเอียดที่ต้องการดังรูปที่ 1-21



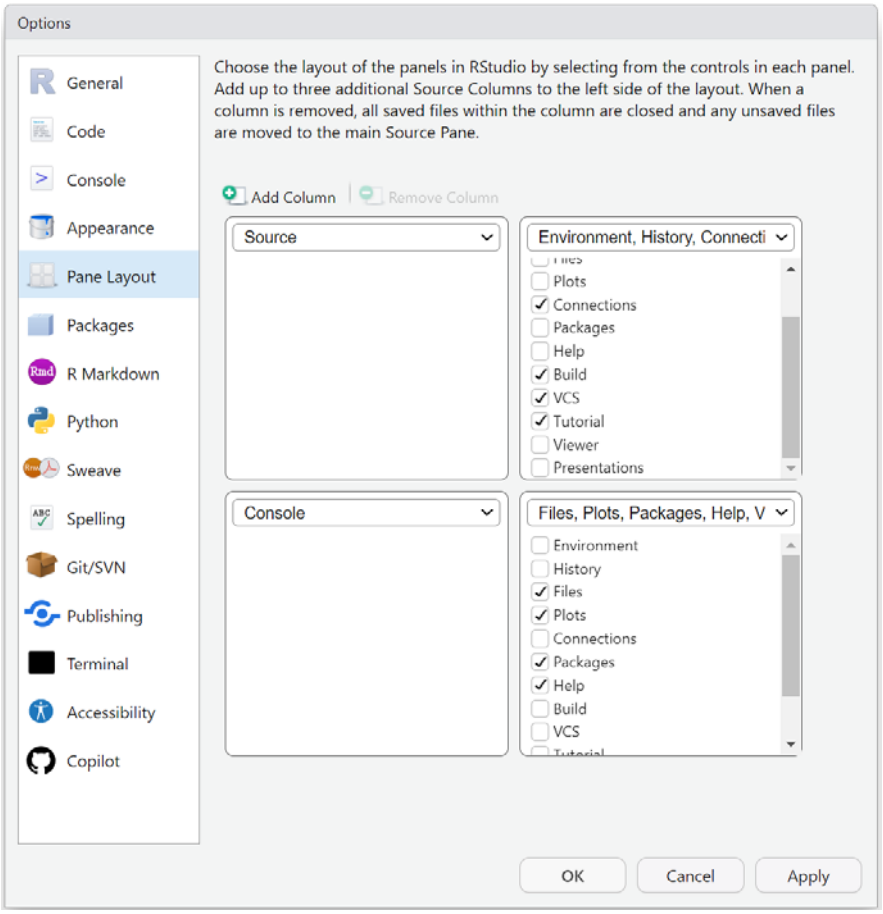
รูปที่ 1-21 การปรับแต่ง Editor

ผู้ใช้สามารถปรับแต่งหรือเปลี่ยนธีมของ Editor โดยเลือก Menu > Tools > Global Options... > Appearance แล้วเลือกธีมรวมทั้งรายละเอียดที่ต้องการดังรูปที่ 1-22



รูปที่ 1-22 การปรับแต่งธีม Editor

ผู้ใช้สามารถดัดแปลงหน้าต่างทั้ง 4 ส่วนของ RStudio โดยการเลือก Menu > Tools > Global Options... > Pane Layout แล้วเลือกแสดงส่วนประกอบต่าง ๆ ในแต่ละส่วนด้วยการเลือก(1) Source (2) Console (3) Environment, History, Connections หรือ (4) Files, Plots, Packages, Help, Viewer ดังรูปที่ 1-23



รูปที่ 1-23 การปรับแต่ง Pane Layout

1.3 การใช้งานโปรแกรม RStudio และ R

1.3.1 การรันโปรแกรม

การรันโปรแกรม **R** มี 2 แบบ ได้แก่ (1) แบบตอบโต้ผู้ใช้ (Interactive mode) และ (2) แบบชุดคำสั่ง (Batch mode) การรันโปรแกรมแบบตอบโต้ผู้ใช้ (Interactive mode) ผู้ใช้สามารถพิมพ์คำสั่งที่ Console โดยตรงหรือที่ Editor Pane และสั่งรันโดยการกด Ctrl+Enter หรือ Ctrl+R ได้ เช่น การหาเลขสุ่มที่มีการกระจายแบบปกติ (Normal distribution) ค่าเฉลี่ยเท่ากับ 0 ส่วนเบี่ยงเบนมาตรฐานเท่ากับ 1 $N(0, 1)$ จำนวน 10 ค่า เขียนเป็นคำสั่งคือ `rnorm(10)` ได้ดังนี้

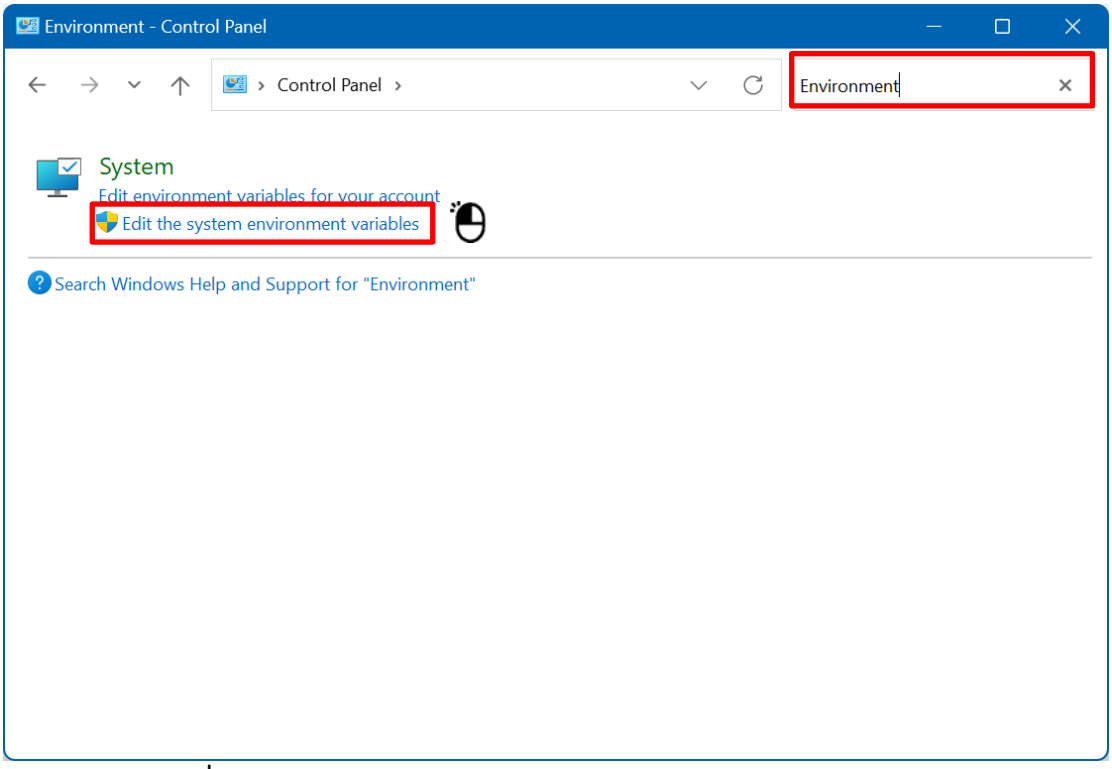
```
> rnorm(10)
 [1]  0.2847394 -1.8303788 -0.8353794  0.8737840  0.5174660 -1.6316359
 [7] -0.3067997  1.2016404  0.3576609 -0.3848432
```

ค่าโดยปริยายของ **R** จะแสดงเครื่องหมาย > ที่ Console ผู้ใช้สามารถเปลี่ยนการแสดงเครื่องหมายดังกล่าวได้ โดยใช้คำสั่ง `options()` พร้อมกับระบุพารามิเตอร์[7] `prompt` (ค่าที่ต้องการแสดงผล) ดังนี้

```
> options(prompt = "R> ")
R>
```

ผู้ใช้สามารถกดลูกศรขึ้น (↑) หรือลง (↓) บนแป้นพิมพ์เพื่อเลื่อนให้ Console แสดงผลคำสั่งก่อนหรือหลัง ตามลำดับ

การรันโปรแกรมแบบชุดคำสั่ง (Batch mode) ก่อนอื่นต้องไปกำหนด path ให้โปรแกรม **R** สามารถรันที่โฟลเดอร์ไหนก็ได้ โดยเปิด Control Panel โดยใช้คำสำคัญ (Keywords) "environment" ที่ช่องค้นหา (search) แล้วเลือก Edit the system environment variables ได้ดังรูปที่ 1-24

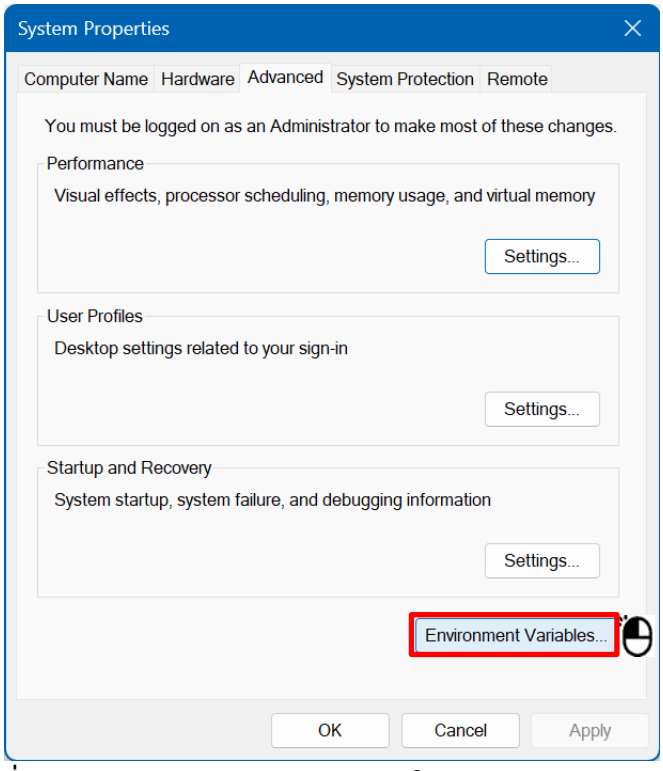


รูปที่ 1-24 การค้นหา Edit the system environment variables

คลิกเลือก Environment Variables... ในแท็บ Advanced ดังรูปที่ 1-25

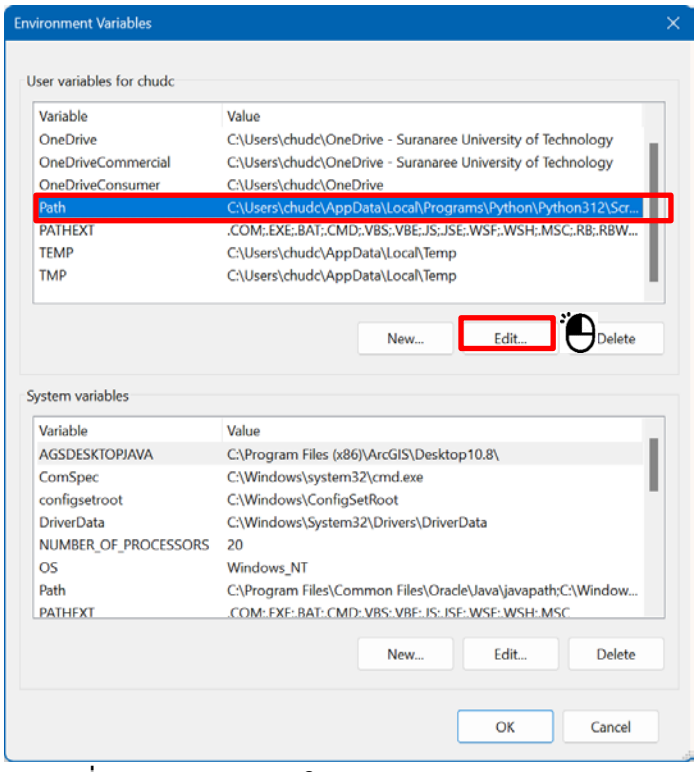
[7] เอกสารฉบับนี้จะใช้คำว่า พารามิเตอร์ (Parameters) และ อาร์กิวเมนต์ (Arguments) แทนกันได้ (Interchangeable)

นิยามโดยทั่วไป Parameters หรือ Formal parameters หมายถึง ตัวแปรที่กำหนดไว้หรืออยู่ในวงเล็บในการนิยามฟังก์ชัน ส่วน Arguments หรือ Actual paremeters หมายถึง ค่าที่ส่งเข้าไปในฟังก์ชันตอนที่เรียกฟังก์ชันมาใช้งาน



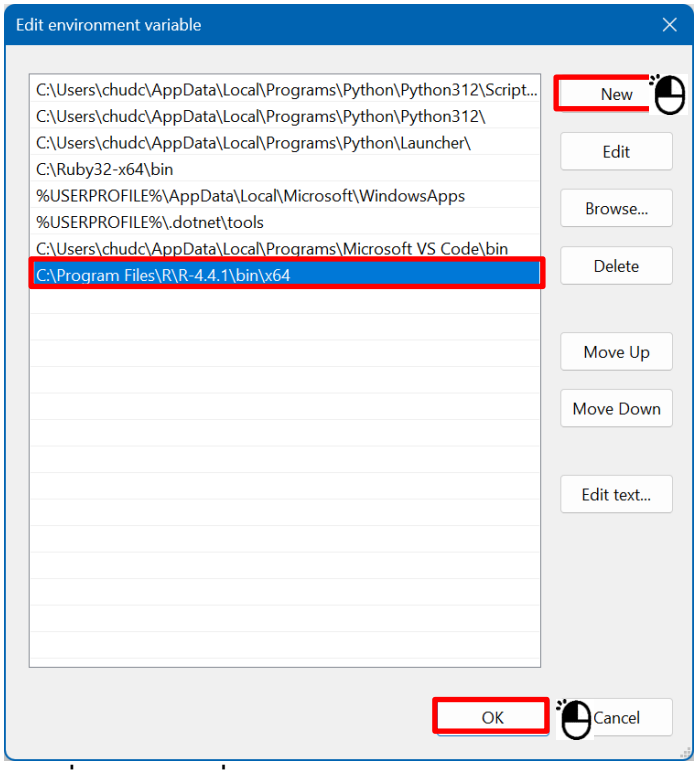
รูปที่ 1-25 Environment Variables ใน System Properties

เลือก Path ที่หน้าต่างด้านบน (หน้าต่าง User variables for ...) แล้วคลิก Edit... ดังรูปที่ 1-26



รูปที่ 1-26 Edit Path ใน Environment Variables

คลิก New แล้วเพิ่มคำสั่ง C:\Program Files\R\R-(version)\bin\x64 เช่น C:\Program Files\R\R-4.1.1\bin\x64 แล้วคลิก OK ไปจนกว่าจะออกจากการ Setting ดังรูปที่ 1-27 เป็นอันสิ้นสุดการกำหนด Path ให้กับโปรแกรม **R** ซึ่งหมายความว่า ผู้ใช้สามารถรันโปรแกรม **R** ที่โฟลเดอร์/ตำแหน่งไหนก็ได้ในเครื่องคอมพิวเตอร์



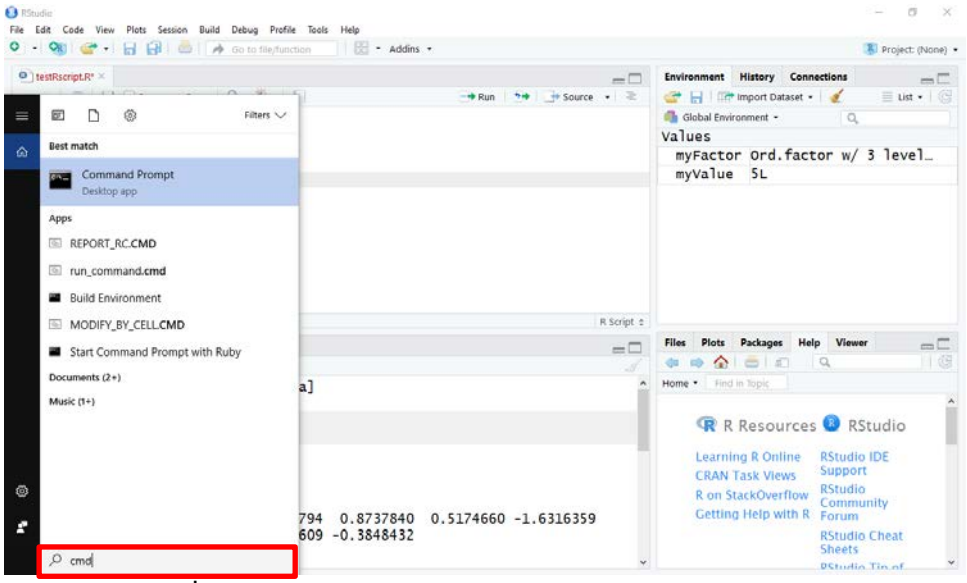
รูปที่ 1-27 การเพิ่ม Path ใน Environment Variables

การรันโปรแกรมแบบชุดคำสั่ง (Batch mode) จะต้องสร้างชุดคำสั่งเก็บเป็นไฟล์ไว้ก่อน ตัวอย่างเช่น สร้างชุดคำสั่งเก็บไว้ในไฟล์ชื่อว่า `testBatchMode.R` ดังนี้

```
pdf("output.pdf")
hist(rnorm(100))
dev.off()
```

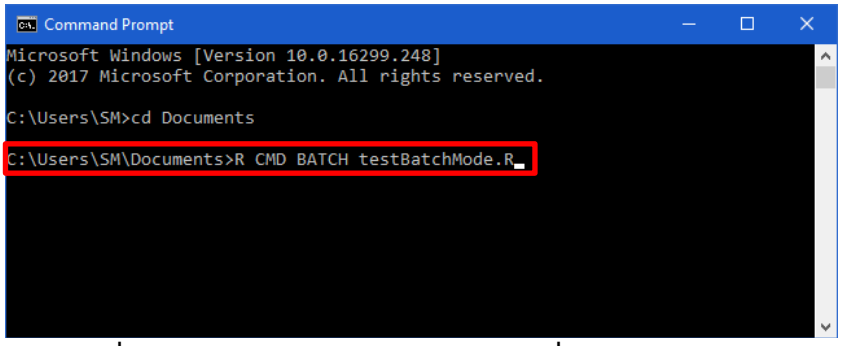
คำสั่งแรกเป็นการกำหนดให้ไฟล์ที่จะสร้างเป็นไฟล์ pdf ชื่อว่า output.pdf คำสั่งที่สองเป็นการสร้างฮิสโตแกรมโดยมีข้อมูลที่มีการกระจายแบบปกติมาตรฐาน (Normal distribution) ค่าเฉลี่ยเท่ากับ 0 ส่วนเบี่ยงเบนมาตรฐานเท่ากับ 1 $N(0, 1)$ จำนวน 100 ข้อมูล ส่วนคำสั่งสุดท้ายเป็นการเขียนฮิสโตแกรมลงบนไฟล์ในเครื่อง และทำการปิดไฟล์ดังกล่าว

การรันโปรแกรมแบบชุดคำสั่ง (Batch mode) ให้ไปที่ Command Prompt ใน Windows หรือค้นหาโดยใช้คำสำคัญ (Keywords) "cmd" ที่ช่อง Search Windows บน Task bar ดังรูปที่ 1-28



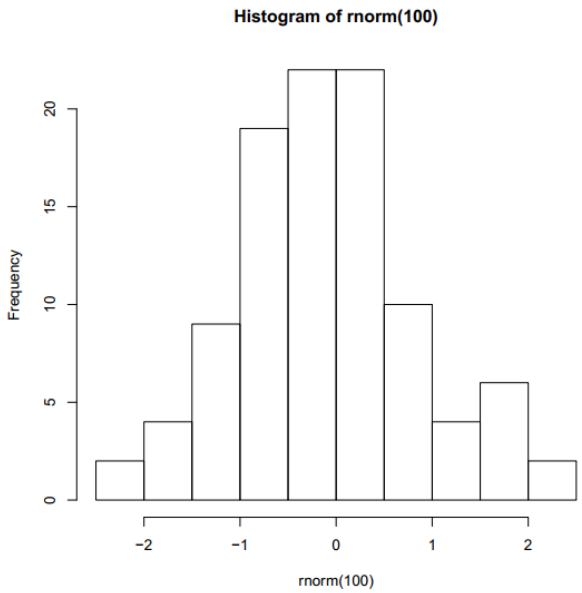
รูปที่ 1-28 การค้นหา Command Prompt ใน Windows

คลิกเลือก Command Prompt จะได้หน้าต่าง Command Prompt แล้วทำการเปลี่ยนตำแหน่งที่ทำงานไปที่โฟลเดอร์ที่เก็บไฟล์ `testBatchMode.R` ที่ได้บันทึกไว้แล้ว เช่น ในตัวอย่างนี้เปลี่ยนไปที่โฟลเดอร์ C:\Users\SM\Documents โดยใช้คำสั่ง "cd Documents" และสั่งรันโปรแกรม **R** โดยใช้คำสั่ง `"R CMD BATCH testBatchMode.R"` ดังรูปที่ 1-29 มีข้อสังเกตว่า **CMD** และ **BATCH** จะต้องเป็นอักษรตัวใหญ่เท่านั้น



รูปที่ 1-29 การรันโปรแกรม R แบบชุดคำสั่ง (Batch mode)

ผลการรันจะได้ไฟล์ pdf ชื่อว่า output.pdf เมื่อเปิดไฟล์ pdf ดังกล่าวจะได้ฮิสโตแกรม คล้ายกับรูปที่ 1-30 ดังนี้



รูปที่ 1-30 ผลการรันโปรแกรม R แบบชุดคำสั่ง (Batch mode)

R สามารถรันชุดคำสั่งดังกล่าวใน **R** หรือ RStudio ได้โดยการใช้คำสั่ง `source()` และระบุไฟล์คำสั่ง หรือใช้คำสั่ง Ctrl+Shift+S เช่น

```
R> source("C:/Users/SM/Documents/2_R/testRscript.R")  # option
R> source("~/2_R/testRscript.R")  # ~ represent home directory
```

ออกจากโปรแกรม **R** โดยใช้ Menu > File > Exit ใน **R** หรือ Menu > File > Quit Session... ใน RStudio หรือใช้คำสั่ง `q()` ดังนี้

```
R> q()
Save workspace image to ~/2_R/.RData? [y/n]: n
```

1.3.2 การคอมเมนต์ (Comments)

การเขียนโปรแกรมที่ดีควรจะใส่คอมเมนต์ เพื่ออธิบายการทำงานของโปรแกรม ทำให้ผู้อื่นทำความเข้าใจชุดคำสั่งได้ง่าย รวมทั้งเป็นประโยชน์ต่อผู้เขียนเอง เมื่อต้องกลับมาทำความเข้าใจคำสั่งที่ตัวเองเคยเขียนไว้ **R** สามารถใส่คอมเมนต์โดยใช้เครื่องหมาย (#) นำหน้าข้อความเพื่อไม่ให้โปรแกรม **R** ทำการประมวลผลคำสั่งที่ตามมาในบรรทัดนั้นได้ ดังอย่างต่อไปนี้

```
R> # Comments in R…
R> 2+2  # This is the command to work out two plus two
```

1.3.3 โฟลเดอร์การทำงาน (Working directory)

R ตรวจสอบโฟลเดอร์ที่กำลังทำงานอยู่ (Working directory) ด้วยคำสั่ง `getwd()` ดังอย่างต่อไปนี้

```
R> > getwd()
[1] "C:/Users/R_lover/Documents"
```

สำหรับระบบปฏิบัติการวินโดว์ (Windows) ค่าโดยปริยายกำหนดให้ทำงานภายใต้โฟลเดอร์ Home ของโปรแกรม **R** ตามปกติจะอยู่ที่โฟลเดอร์ C:\Users\UserX\Documents (UserX คือชื่อผู้ใช้) มีข้อสังเกตว่า ระบบปฏิบัติการวินโดว์จะใช้เครื่องหมาย Back slash (****) ในการคั่นระหว่างชื่อโฟลเดอร์ ในขณะที่ **R** จะใช้เครื่องหมาย Forward slash (**/**) ในการคั่นระหว่างชื่อโฟลเดอร์แทน ดังนั้น ผู้ใช้ **R** ในระบบปฏิบัติการวินโดว์ต้องระวังข้อผิดพลาดจากจุดดังกล่าว

ถ้าไม่ระบุ Path ให้กับชื่อไฟล์ โปรแกรม **R** จะทำงานที่โฟลเดอร์ Home ผู้ใช้สามารถกำหนดโฟลเดอร์สำหรับการทำงาน (Working directory) ด้วยคำสั่ง **setwd()** ตามด้วยโฟลเดอร์ที่ต้องการทำงาน ดังนี้

```
R> setwd("C:/Users/R_lover/Documents/R")
```

1.3.4 แพ็กเกจ (Packages)

R มีคำสั่งสำหรับการทำงาน การคำนวณ การวิเคราะห์ทางสถิติ การสร้างกราฟ และอื่น ๆ ติดตั้งมาพร้อมกับติดตั้งโปรแกรม เรียกว่า คำสั่งภายใน (Build-in command) ซึ่งสามารถเรียกใช้งานได้ทันที แต่ก็มีบางงานที่ต้องการรายละเอียดเฉพาะทางนอกเหนือจากคำสั่งภายใน เพื่อเพิ่มความสามารถดังกล่าว **R** สามารถติดตั้งแพ็กเกจ (Packages) หรือบางทีเรียกว่า ไลบรารี (Libraries) เพิ่มเติมได้อีก ขั้นตอนการติดตั้งและใช้งานแพ็กเกจประกอบด้วย 2 ขั้นตอน ได้แก่ (1) การติดตั้ง (Install) แพ็กเกจ และ (2) การโหลด (Load) แพ็กเกจเข้ามาเก็บไว้ในหน่วยความจำเพื่อทำงานต่อไป ดังนี้

ขั้นตอนแรกเป็นการติดตั้งแพ็กเกจลงในโปรแกรม **R** โดยใช้คำสั่ง **install.packages()** ตามด้วยชื่อแพ็กเกจที่อยู่ภายในเครื่องหมาย double quote **" "** ดังนี้

```
R> install.packages("ggplot2")
```

ขั้นตอนที่สองเป็นการโหลดแพ็กเกจเข้ามาที่หน่วยความจำ (Memory) ของเครื่อง โดยใช้คำสั่ง **library()** หรือ **require()** ตามด้วยชื่อแพ็กเกจ ดังนี้

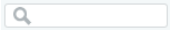
```
R> library("ggplot2")  # option library(ggplot2)
R> require("ggplot2")  # option require(ggplot2)
```

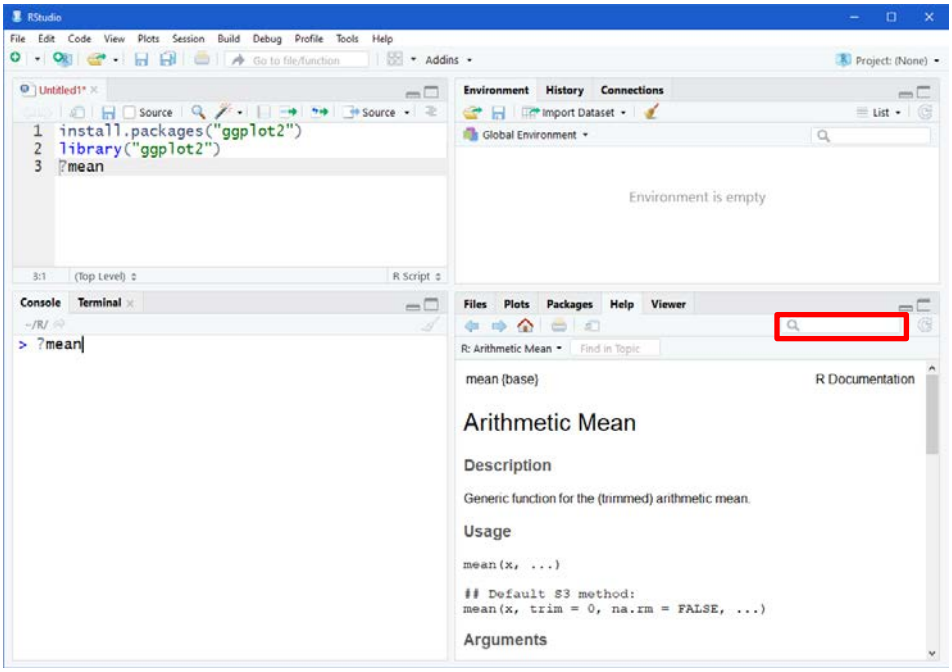
พารามิเตอร์ที่ผ่านเข้ามาในคำสั่ง **library()** หรือ **require()** จะมีเครื่องหมาย double quote **" "** หรือไม่มีก็ได้

1.3.5 การขอความช่วยเหลือ (Help)

การขอความช่วยเหลือใน **R** ให้พิมพ์เครื่องหมายคำถาม **?** หรือคำสั่ง **help()** แล้วตามด้วยคำสั่ง/ฟังก์ชันที่ต้องการขอความช่วยเหลือ ดังอย่างต่อไปนี้

```
R> ?mean
R> help("mean")
```

คำสั่งข้างบนเป็นการขอความช่วยเหลือ (Help) ของคำสั่ง **mean()** ผลที่ได้จะแสดงในหน้าต่าง Help ด้านล่างซ้ายมือ ดังรูปที่ 1-31 หรือพิมพ์คำสั่งที่ต้องการขอความช่วยเหลือในช่องว่างต่อจากเครื่องหมายแว่นขยาย  ในหน้าต่าง Help ได้โดยตรง



รูปที่ 1-31 การขอความช่วยเหลือ (Help)

คำอธิบายใน Help ประกอบด้วยข้อมูลที่สำคัญ ได้แก่

- Description: เป็นสรุปคำอธิบายคำสั่ง/ฟังก์ชัน
- Usage: เป็นคำอธิบายรูปแบบการใช้งานคำสั่ง/ฟังก์ชัน แสดงลำดับของพารามิเตอร์และค่าโดยปริยายของแต่ละพารามิเตอร์แต่ละตัว
- Parameters: แสดงคำอธิบายรายละเอียดของพารามิเตอร์แต่ละตัว
- Value: แสดงค่าที่ส่งกลับมาของฟังก์ชัน (ถ้ามี)
- References: แสดงที่มา/แหล่งอ้างอิงของคำสั่ง/ฟังก์ชัน
- See Also: แสดงคำสั่ง/ฟังก์ชันที่เกี่ยวข้อง
- Examples: แสดงตัวอย่างการใช้งาน

ถ้าไม่แน่ใจว่าพิมพ์ชื่อคำสั่งถูกต้องหรือไม่ ใช้เครื่องหมาย `??` หรือคำสั่ง `help.search()` ตามด้วยชื่อที่ต้องการหาในเครื่องหมาย double quote **R** จะค้นหาสตริง[8] ทุกตัวใน Help files และรายงานผลการค้นให้ทราบ ตัวอย่างเช่น

```
R> ??"mean"
R> help.search("mean")
```

R ยังสามารถแสดงตัวอย่างการใช้งานคำสั่งทาง Console ได้โดยใช้คำสั่ง `example()` ตามด้วยชื่อฟังก์ชันที่ต้องการ เช่น ต้องการตัวอย่างคำสั่ง `mean()` เขียนได้ดังนี้

```
R> example(mean)

mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim=0.10))
[1] 8.75 5.50
```

1.3.6 พื้นที่ทำงาน (Workspaces) และไฟล์คำสั่ง (Script Files)

เมื่อทำงานด้วยโปรแกรม RStudio ผู้ใช้สามารถบันทึกพื้นที่ทำงาน (Workspaces) ด้วยการเลือก Menu > Session > Save Workspace As... เลือกโฟลเดอร์และตั้งชื่อไฟล์ที่ต้องการ หรือใช้คำสั่ง `save.image()` ตามด้วยชื่อไฟล์ Workspaces จะมีนามสกุล ***.RData*** ซึ่งจะเก็บข้อมูลวัตถุ (Objects) เช่น ตัวแปรต่าง ๆ ที่อยู่ใน R environment ขณะที่ทำงานอยู่ ผู้ใช้สามารถนำกลับมาใช้ได้ในภายหลังด้วย Menu > Session > Load Workspace... หรือใช้คำสั่ง `load()` ตามด้วยชื่อไฟล์ ตัวอย่างเช่น

```
R> save.image("my_work_space.RData")
R> load("my_work_space.RData")
```

ถ้าไม่ระบุ Workspaces โปรแกรม **R** จะบันทึกในไฟล์ที่ไม่มีชื่อ นามสกุล *.RData* โดยอัตโนมัติ และจะทำการโหลด Workspaces ดังกล่าวเข้ามาโดยอัตโนมัติ เมื่อเปิดโปรแกรม **R** ขึ้นมา

นอกจาก Workspace โปรแกรม **R** ยังมีไฟล์คำสั่ง (Script Files) ที่สามารถบันทึกเก็บไว้ในไฟล์นามสกุล ***.R*** โดยเลือก Menu > File > Save หรือ Ctrl+S แต่ถ้าต้องการบันทึกเป็นไฟล์ใหม่ให้เลือก Menu > File > Save As... และเรียกกลับมาใช้โดยเลือก Menu > File > Open File... หรือ Ctrl+O

[8] สตริง (Strings) หรือตัวอักษร บางครั้งเรียกว่าสายอักขระ เป็นการนำอักขระ (Characters) หลายตัวมาประกอบกัน

1.4 สัญลักษณ์และข้อตกลง (Conventions)

เอกสารฉบับนี้จะใช้สัญลักษณ์และข้อตกลง (Conventions) เพื่อให้คำสั่งหรือข้อความมีความแตกต่างกันอย่างชัดเจน ดังต่อไปนี้

คำสั่ง/ฟังก์ชันจะแสดงด้วย `Font Consolas` เช่น `mean()` โดยทั่วไปคำสั่งจะอยู่ภายในเส้นกรอบบน/ล่าง และขึ้นต้นด้วยเครื่องหมาย `R>` ดังนี้

```
R> 22/7
[1] 3.14285
```

ชุดคำสั่งจะแสดงด้วย `Font Consolas` ภายในเส้นกรอบบน/ล่าง ดังนี้

```
for(my_value in 1:5) {
  cat("Hello World, ")
  cat("my_value is ", my_value, "\n")
}
```

คำสั่งที่ยาวมากกว่า 1 บรรทัด จะใส่ indent ให้อ่านชุดคำสั่งได้ง่าย ดังนี้

```
R> my_factor <- factor(x = c("Small", "Medium", "Medium", "Large", "Large"),
                       Levels = c("Small", "Medium", "Large"),
                       Ordered = TRUE)
```

Tidyverse style guide ได้แนะนำสไตล์การเขียนชุดคำสั่งใน **R** ไว้ ดูรายละเอียดเพิ่มเติมได้ตาม URL ต่อไปนี้ https://style.tidyverse.org/files.html สรุปสาระสำคัญ ได้ดังนี้

1.4.1 การใช้สัญลักษณ์และข้อตกลงสำหรับไฟล์และตัวแปร

1. ชื่อไฟล์ (File names) ควรจะสื่อความหมาย และลงท้ายด้วย ***.R*** ใช้อักษรตัวเล็กทั้งหมด ควรหลีกเลี่ยงการใช้ตัวอักษรพิเศษ แต่ละคำคั้นด้วยเครื่องหมายขีดเส้นใต้ _ (Underscores) หรือเครื่องหมายลบ - (Hyphens)

```
# Good
fit_models.R
utility_functions.R

# Bad
fit models.R
foo.R
stuff.R
```

2. ชื่อไฟล์ (File names) ที่แต่ละไฟล์มีลำดับการทำงาน ควรตั้งชื่อที่มีหมายเลขกำกับ และถ้าจำนวนไฟล์มีมากกว่า 10 ไฟล์ควรจะใช้เลข 0 นำหน้าชื่อ ดังนี้

```
00_download.R
01_explore.R
...
09_model.R
10_visualize.R
```

3. ใช้ commented lines ด้วย - หรือ = แยกส่วนต่าง ๆ ออกจากกันเพื่อให้โค้ดอ่านง่าย

```
# Load data --------------------------

# Plot data --------------------------
```

4. ชื่อตัวแปร ควรใช้อักษระตัวเล็กทั้งหมด ตัวเลข และเครื่องหมายขีดเส้นใต้ _ (Underscores) การตั้งชื่อตัวแปรควรใช้คำนาม ชื่อควรจะกระชับ และสื่อความหมาย

```
# Good
day_one
day_1

# Bad
DayOne
dayone
first_day_of_the_month
djm1
```

5. ชื่อฟังก์ชัน ควรใช้อักษระตัวเล็กทั้งหมด และเครื่องหมายขีดเส้นใต้ _ (Underscores) การตั้งชื่อฟังก์ชันควรใช้คำกริยา ชื่อควรจะกระชับ และสื่อความหมาย

```
# Good
add_row()
permute()

# Bad
row_adder()
permutation()
```

6. ควรหลีกเลี่ยงชื่อตัวแปรหรือชื่อฟังก์ชันที่มีอยู่ใน R แล้ว

```
# Bad
T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

1.4.2 การใช้สัญลักษณ์และข้อตกลงสำหรับไวยากรณ์

7. ใส่ช่องว่าง 1 ตัว (A space) หลัง commas และไม่มีช่องว่างหน้า commas

```
# Good
x[, 1]

# Bad
x[,1]
x[ ,1]
x[ , 1]
```

8. การเรียกใช้ฟังก์ชันไม่มีช่องว่างทั้งก่อนและหลังเครื่องหมายวงเล็บ

```
# Good
mean(x, na.rm = TRUE)
```

```
# Bad
mean (x, na.rm = TRUE)
mean( x, na.rm = TRUE )
```

9. เมื่อใช้กับคำสั่ง `if, for,` หรือ `while` ควรใส่ช่องว่าง 1 ตัวก่อนและหลังเครื่องหมายวงเล็บ `()`

```
# Good
if (debug) {
  show(x)
}

# Bad
if(debug){
  show(x)
}
```

10. ใส่ช่องว่าง 1 ตัวหลังเครื่องหมายวงเล็บ `()` เมื่อเขียนฟังก์ชัน

```
# Good
function(x) {}

# Bad
function (x) {}
function(x){}
```

11. เมื่อใช้เครื่องหมาย embracing operator `{{ }}` ควรมีช่องว่างเพื่อเน้นตัว operator

```
# Good
max_by <- function(data, var, by) {
  data |>
    group_by({{ by }}) |>
    summarise(maximum = max({{ var }}, na.rm = TRUE))
}

# Bad
max_by <- function(data, var, by) {
  data |>
    group_by({{by}}) |>
    summarise(maximum = max({{var}}, na.rm = TRUE))
}
```

12. ควรมีช่องว่าง 1 ตัวก่อนและหลัง infix operators เช่น `= + - * <-`

```
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

13. ไม่ควรมีช่องว่างก่อนและหลัง สำหรับ operators ที่มีลำดับความสำคัญเป็นลำดับต้น ๆ ในการคำนวณ ได้แก่ `::, :::, $, @, [, [[, ^, unary -, unary +,` และ `:`

```
# Good
sqrt(x^2 + y^2)
df$z
x <- 1:10

# Bad
sqrt(x ^ 2 + y ^ 2)
df $ z
x <- 1 : 10
```

14. ไม่ควรมีช่องว่างก่อน Single-sided formulas เมื่อทางขวามือของเครื่องหมายมีเพียงค่าเดียว

```
# Good
~foo
tribble(
  ~col1, ~col2,
  "a",   "b"
)

# Bad
~ foo
tribble(
  ~ col1, ~ col2,
  "a", "b"
)
```

15. มีข้อยกเว้น ควรมีช่องว่างสำหรับ Single-sided formulas เมื่อทางขวามือของเครื่องหมายมีหลายค่า

```
# Good
~ .x + .y

# Bad
~.x + .y
```

16. ไม่ควรมีช่องว่างก่อนและหลังเครื่องหมาย tidy evaluation `!!` (bang-bang) และ `!!!` (bang-bang-bang)

```
# Good
call(!!xyz)

# Bad
call(!! xyz)
call( !! xyz)
call(! !xyz)
```

17. การใช้คำสั่งขอความช่วยเหลือ (Help) ไม่ควรมีช่องว่างก่อนและหลังเครื่องหมาย ?

```
# Good
package?stats
?mean

# Bad
package ? stats
? mean
```

18. สามารถเพิ่มช่องว่างเพื่อจัด alignment สำหรับเครื่องหมาย = หรือ <- ได้

```
# Good
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)

# Also fine
list(
  total = a + b + c,
  mean = (a + b + c) / n
)
```

19. ชื่อพารามิเตอร์/อากิวเมนต์ (Named arguments) ที่มีการใช้บ่อย ๆ เช่น ข้อมูลที่ป้อนเข้ามาใน ฟังก์ชัน มักจะละไว้ไม่เขียนชื่อพารามิเตอร์ตัวนั้น

```
# Good
mean(1:10, na.rm = TRUE)

# Bad
mean(x = 1:10, , FALSE)
mean(, TRUE, x = c(1:10, NA))
```

20. ในการเขียน code blocks ด้วยเครื่องหมายปีกกา **{}** ควรใช้เครื่องหมาย **{** เป็นตัวสุดท้ายของ บรรทัด และ **}** เป็นตัวแรกของบรรทัด

```
# Good
if (y < 0 && debug) {
  message("y is negative")
}

if (y == 0) {
  if (x > 0) {
    log(x)
  } else {
    message("x is negative or zero")
  }
} else {
  y^x
}

# Bad
if (y < 0 && debug) {
message("Y is negative")
}

if (y == 0)
{
    if (x > 0) {
      log(x)
    } else {
  message("x is negative or zero")
```

```
    }
} else { y ^ x }
```

21. แต่ละแถวความยาวไม่ควรเกิน 80 ตัวอักษร ควรแยกแต่ละบรรทัดสำหรับแต่ละ option หรือแต่ละพารามิเตอร์

```
# Good
do_something_very_complicated(
  something = "that",
  requires = many,
  arguments = "some of which may be long"
)

# Bad
do_something_very_complicated("that", requires, many, arguments,
                              "some of which may be long"
                              )
```

22. ควรใส่ option หรือพารามิเตอร์ที่เกี่ยวข้องกันอยู่ในบรรทัดเดียวกัน

```
# Good
paste0(
  "Requirement: ", requires, "\n",
  "Result: ", result, "\n"
)

# Bad
paste0(
  "Requirement: ", requires,
  "\n", "Result: ",
  result, "\n")
```

23. การกำหนดค่าให้ตัวแปรใช้เครื่องหมาย `<-` แทนการใช้ `=`

```
# Good
x <- 5

# Bad
x = 5
```

24. ข้อความ (Text) ให้อยู่ภายในเครื่องหมาย double quote ยกเว้นเมื่อต้องการ double quote อยู่ในข้อความให้ใช้ single quote แทน

```
# Good
"Text"
'Text with "quotes"'
'<a href="http://style.tidyverse.org">A link</a>'

# Bad
'Text'
'Text with "double" and \'single\' quotes'
```

25. ใช้ **TRUE** และ **FALSE** แทน **T** และ **F**

26. ใช้คำสั่ง return เมื่อต้องการให้ฟังก์ชันคืนค่าก่อนที่จะทำงานถึงบรรทัดสุดท้าย บรรทัดสุดท้ายไม่ต้องใช้คำสั่ง return เนื่องจาก **R** จะคืนค่าที่บรรทัดสุดท้ายอยู่แล้ว

```
# Good
find_abs <- function(x) {
  if (x > 0) {
    return(x)
  }
  x * -1
}
add_two <- function(x, y) {
  x + y
}

# Bad
add_two <- function(x, y) {
  return(x + y)
}
```

1.4.3 การใช้สัญลักษณ์และข้อตกลงสำหรับการใส่หมายเหตุ (Comment)

27. การใส่หมายเหตุ (Comment) ใช้เครื่องหมาย **#** นำหน้าส่วนที่ต้องการ comment ในการทำการวิเคราะห์ข้อมูล ถ้ามีการ comment มากกว่าการโค้ด แนะนำให้ใช้ **R** Markdown แทน **R** Script และ comment ในฟังก์ชันควรจะอธิบายว่าทำไม (Why) ไม่ใช่บอกว่าทำอะไร (What) หรือทำอย่างไร (How)
28. การเขียนหมายเหตุ (Comment) ควรเขียนเป็นรูปประโยค ใช้ตัวใหญ่หรือตัวเล็กตาม Sentence case และใส่เครื่องหมายจุด เมื่อจบประโยคถ้ามีตั้งแต่ 2 ประโยคขึ้นไป

```
# Good
# Objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Do not use `is.list()`. Objects like data frames must be treated
# as leaves.
x <- map_if(x, is_bare_list, recurse)


# Bad
# objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Objects like data frames are treated as leaves.
x <- map_if(x, is_bare_list, recurse)
```

1.4.4 การใช้สัญลักษณ์และข้อตกลงสำหรับการส่งคำสั่งต่อเนื่อง (Command Piping)

29. การใช้เครื่องหมาย **|>** หรือ **%>%** (Pipe)[9] สำหรับการส่งคำสั่งต่อเนื่อง (Command piping) ควรมีช่องว่าง 1 ตัวก่อน **|>** และตามด้วยการขึ้นบรรทัดใหม่ และมีย่อหน้า (Indent) ในบรรทัดหลังจากเครื่องหมาย **|>**

```
# Good
iris |>
  group_by(Species) |>
  summarize_if(is.numeric, mean) |>
  ungroup() |>
  gather(measure, value, -Species) |>
  arrange(value)

# Bad
iris |> group_by(Species) |> summarize_all(mean) |>
ungroup() |> gather(measure, value, -Species) |>
arrange(value)
```

30. แถวที่ยาวมากให้ขึ้นบรรทัดใหม่ และแต่ละบรรทัดมีพารามิเตอร์แยกออกจากกัน บรรทัดละ 1 พารามิเตอร์

```
iris |>
  group_by(Species) |>
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
  )
```

31. การใช้เครื่องหมาย **|>** สำหรับการส่งคำสั่งต่อเนื่อง 1 ขั้นตอน (Short pipe) อาจจะอยู่ในบรรทัดเดียวก็ได้ แต่ถ้าต้องการ **|>** เผื่อในอนาคตก็ทำเป็น 2 บรรทัดตามปกติ

```
iris |> arrange(Species)

iris |>
  arrange(Species)
```

32. บางครั้งอาจจะใช้การ Short pipe สำหรับการสร้างพารามิเตอร์ในการ Pipe ที่มีหลายขั้นตอนได้

```
# Good
x |>
  select(a, b, w) |>
```

[9] เครื่องหมาย **|>** หรือใช้คำสั่ง Ctrl+Shift+m เป็นเครื่องหมาย Pipe ของ **R** พื้นฐาน (base **R**) และต้องการ **R version 4.1** ขึ้นไป ส่วนเครื่องหมาย **%>%** เป็นเครื่องหมาย Pipe ของ แพ็กเกจ magrittr ซึ่งก่อนใช้งานต้องโหลด (Load) แพ็กเกจเข้ามาที่หน่วยความจำของเครื่องด้วยคำสั่ง `library(magrittr)`

```
  left_join(y |> select(a, b, v), by = c("a", "b"))

# Better
x_join <- x |> select(a, b, w)
y_join <- y |> select(a, b, v)
left_join(x_join, y_join, by = c("a", "b"))
```

33. การกำหนดค่าให้ตัวแปรต่อจากการส่งคำสั่งต่อเนื่อง (Command Piping) ทำได้ 3 แบบ ได้แก่

```
# 1.Variable name and assignment on separate lines
iris_long <-
  iris |>
  gather(measure, value, -Species) |>
  arrange(-value)

# 2.Variable name and assignment on the same line
iris_long <- iris |>
  gather(measure, value, -Species) |>
  arrange(-value)

# 3. Assignment at the end of the pipe with ->
iris |>
  gather(measure, value, -Species) |>
  arrange(-value) ->
  iris_long
```

1.4.5 การใช้สัญลักษณ์และข้อตกลงสำหรับ ggplot2

34. การใช้เครื่องหมาย + สำหรับเพิ่ม layer ให้กับคำสั่ง **ggplot** มีวิธีการใช้คล้ายกับ |> กล่าวคือ มีช่องว่างก่อนหน้าเครื่องหมาย + 1 ตัว และตามด้วยบรรทัดใหม่

```
# Good
iris |>
  filter(Species == "setosa") |>
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()

# Bad
iris |>
  filter(Species == "setosa") |>
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
    geom_point()

# Bad
iris |>
  filter(Species == "setosa") |>
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) + geom_point()
```

35. ถ้าคำสั่งยาวมาก ให้ขึ้นย่อหน้าใหม่และแยกบรรทัดสำหรับแต่ละพารามิเตอร์

```
# Good
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
```

```
  labs(
    x = "Sepal width, in cm",
    y = "Sepal length, in cm",
    title = "Sepal length vs. width of irises"
  )

# Bad
ggplot(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  geom_point() +
  labs(x = "Sepal width, in cm", y = "Sepal length, in cm", title = "Sepal …)
```

36. ให้ทำการเตรียมข้อมูล (Data manipulation) ก่อนที่จะทำการเรียกคำสั่ง ggplot

```
# Good
iris |>
  filter(Species == "setosa") |>
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()

# Bad
ggplot(filter(iris, Species == "setosa"),
  aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
```

1.5 สรุปท้ายบท

บทนี้เป็นการแนะนำให้รู้จักโปรแกรมภาษา **R** และโปรแกรม RStudio ความเป็นมาของโปรแกรม **R** ความสำคัญของโปรแกรม **R** ข้อดี ข้อด้อย การดาวน์โหลดและการติดตั้งโปรแกรมภาษา **R** และโปรแกรม RStudio แนะนำการใช้งานเบื้องต้น การรันโปรแกรม (Run) การใส่คอมเมนต์ (Comments) การจัดการกับโฟลเดอร์การทำงาน (Working directory) การจัดการกับแพ็คเกจ (Packages) การขอความช่วยเหลือ (Help) การจัดการพื้นที่ทำงาน (Workspaces) ไฟล์คำสั่ง (Script files) รวมถึงการใช้สัญลักษณ์และข้อตกลง (Conventions) ในเอกสารฉบับนี้

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	**คำอธิบาย**
`options()`	กำหนดออปชันต่าง ๆ
`cd`	เปลี่ยนโฟลเดอร์ บน Windows
`cmd`	รัน Command prompt บน Windows
`R CMD BATCH`	รันโปรแกรม **R** แบบชุดคำสั่ง (Batch mode)
`source()`	รันชุดคำสั่งใน **R** หรือ RStudio
`q()`	ออกจากโปรแกรม **R** หรือ RStudio
`#`	การคอมเมนต์ (Comments)
`|>` หรือ `%>%`	Pipe สำหรับการส่งคำสั่งต่อเนื่อง (Command Piping)

คำสั่ง	คำอธิบาย
`<-`	การกำหนดค่า
`getwd()`	ตรวจสอบโฟลเดอร์ที่กำลังทำงานอยู่ (Working directory)
`setwd()`	กำหนดโฟลเดอร์สำหรับการทำงาน (Working directory)
`install.packages()`	การติดตั้ง (Install) แพ็กเกจ (Packages)
`library()` หรือ `require()`	การโหลด (Load) แพ็กเกจเข้ามาที่หน่วยความจำของเครื่อง
`?` หรือ `help()`	การขอความช่วยเหลือ (Help)
`??` หรือ `help.search()`	ค้นหาสตริงทุกตัวใน Help files
`example()`	แสดงตัวอย่างการใช้งานคำสั่ง
`save.image()`	บันทึกพื้นที่ทำงาน (Workspaces)
`load()`	โหลดพื้นที่ทำงาน (Workspaces) ที่เก็บเป็นไฟล์ไว้กลับมาใช้

1.6 แบบฝึกหัดท้ายบท

1. โปรแกรม **R** คืออะไร?
2. บอกข้อดี ข้อด้อยของโปรแกรม **R**?
3. โปรแกรม RStudio คืออะไร แตกต่างกับโปรแกรม **R** อย่างไร?
4. สร้างคอมเมนต์ในโปรแกรม **R** ได้อย่างไร?
5. กำหนดโฟลเดอร์การทำงาน (Working directory) ได้อย่างไร?
6. ทำไมต้องมีแพ็กเกจ?
7. เพิ่มแพ็กเกจได้อย่างไร?
8. มีการขอความช่วยเหลือใน **R** ด้วยวิธีการอะไรบ้าง?
9. การกำหนดพื้นที่ทำงาน (Workspaces) มีประโยชน์อย่างไร?
10. tidyverse style guide คืออะไร?
11. tidyverse style guide มีประโยชน์อย่างไร?
12. ตามที่ tidyverse style guide แนะนำ ชุดคำสั่งต่อไปนี้ มีข้อเสียอะไรบ้าง (อาจมีมากกว่า 1 จุด)?

12.1

```
my_factor <- factor(x = c("Small","Medium","Large"),levels = month)
```

12.2

```
myFactor<-factor(x=c("Small","Medium","Large"), levels = month)
```

12.3

```
my.factor<- factor(x = c("Small","Medium","Large"), levels=month)
```

12.4

```
My_mean <- mean(x[ ,2]) # Here is a comment
```

12.5

```
myMean <- mean(x[,2])  #  Here is a comment
```

12.6

```
for( i in 1:10 )
```

12.7

```
#Here is a comment
x[1,]
x[,2]
```

12.8

```
   #here is a comment
if (is.null(xlim)) {xlim <- c(0, 1)}
```

12.9

```
if (is.null(ylim)) ylim = c(0, 0.06)
```

12.10

```
if (x > 10) {
  y = 5
}
else {
  y = 10
}
```

12.11

```
newRecord <- data.frame(name = "Ben",age =
             27,gender = factor("M",levels = levels(gender)))
```

12.12

```
# Computes the sample covariance between two vectors.
#
# Args:
#   x: One of two vectors.
#   y: The other vector.
#
# Returns:
#   The sample covariance between x and y.
CalculateSampleCovariance <- function(x, y, verbose = TRUE) {
  if (n <= 1 || n != length(y)) {
    stop("Parameters x and y have different lengths: ",
         length(x), " and ", length(y), ".")
  }
```

12.13

```
myFunction <- function(query, property, days,
                       show.plot=T, month)
```

12.14

```
table.new <- table(df[days.from.opt<0, "id"])
```

บทที่ 2

เวกเตอร์ (Vectors) เมทริกซ์ (Matrices) และอาร์เรย์ (Arrays)

2.1 การคำนวณทางคณิตศาสตร์เบื้องต้น

R สามารถใช้ในการคำนวณเสมือนเครื่องคิดเลขได้ โดยใช้เครื่องหมายทางคณิตศาสตร์และวงเล็บ ลำดับการทำงานโดยปกติจะเริ่มจากซ้ายไปขวา แต่จะให้ลำดับความสำคัญกับเครื่องหมายที่แตกต่างกันตามลำดับ ดังตารางต่อไปนี้

ตารางที่ 2-1 ลำดับความสำคัญของตัวดำเนินการ (Operator precedence)

ตัวดำเนินการ	ความหมาย	ลำดับความสำคัญ
( )	วงเล็บ	1
^	ยกกำลัง	2
* /	คูณ หาร	3
+ -	บวก ลบ	4

ตัวอย่างการคำนวณด้วย **R**

```
R> 3+4
[1] 7

R> 15/7
[1] 2.142857

R> 15/7+5
[1] 7.142857

R> 15/(7+5)
[1] 1.25

R> 3^2
[1] 9

R> 2^3
[1] 8
```

R สามารถทำการคำนวณที่ซับซ้อนได้ ตัวอย่างเช่น

$$4^2 + \frac{3 \times 16}{5} - 8$$

```
R> 4^2+3*16/5-8
[1] 17.6
```

$$\frac{5^2 \times (7-3)}{40-8+3}$$

```
R> 5^2*(7-3)/(40-8+3)
[1] 2.857143
```

$$2^{3+2} - 5 + 13^{2^{1.1-\frac{1}{3}}}$$

```
R> 2^(3+2)-5+13^((2)^(1.1-1/3))
[1] 105.5579
```

$$\left(\frac{1.23 \times (4-0.56)}{12}\right)^{\frac{1}{2}}$$

```
R> (1.23*(4-0.56)/12)^(1/2)
[1] 0.5938013
```

รากที่สอง (Square root) หาได้โดยใช้ฟังก์ชัน `sqrt()` เช่น

```
R> sqrt(9)  # or sqrt(x = 9)
[1] 3

R> sqrt(5.5)  # or sqrt(x = 5.5)
[1] 2.345208
```

logarithm หาได้โดยใช้ฟังก์ชัน `log()` เช่น logarithm ของ 81 ฐาน 3 ($\log_3 81$) เท่ากับ 4 เนื่องจาก $3^4 = 81$

```
R> log(81, base = 3)
[1] 4
```

สำหรับ exponential หาได้โดยใช้ฟังก์ชัน `exp()` ส่วน natural logarithm หาได้โดยใช้ฟังก์ชัน `log()` โดยไม่ต้องกำหนดพารามิเตอร์ `base` เช่น

```
R> exp(4)
[1] 54.59815

R> log(54.59815)
[1] 4
```

เมื่อตัวเลขมีค่าสูงมากหรือต่ำมาก โดยทั่วไปมักใช้สัญลักษณ์ e (e-notation) เช่น $x \times 10^y$ แทนด้วยสัญลักษณ์ xey เช่น

- $1.23456789 \times 10^{12}$ แทนด้วย $1.23456789e12$
- $1.23456789 \times 10^{10}$ แทนด้วย $1.23456789e10$

R จะแสดงตัวเลขดังกล่าวไม่เกิน 7 หลัก และมีเครื่องหมาย `+/-` ต่อจากสัญลักษณ์ e เช่น

```
R> 1234567890000
[1] 1.234568e+12

R> 0.0000123456789
```

```
[1] 1.234568e-05
```

2.2 การกำหนดค่าให้กับตัวแปร (Assignment)

R เก็บตัวแปรเพื่อใช้ในการคำนวณต่อไป โดยใช้เครื่องหมาย **<-** หรือ **=** แต่โดยทั่วไปนิยมใช้เครื่องหมาย **<-** tidyverse style guide ก็แนะนำให้ใช้เครื่องหมาย **<-** เช่น

```
R> x <- 3
R> x
[1] 3

R> x = x + 1  # this overwrites the previous value of x
R> x
[1] 4

R> my_number = 12.3
R> y <- my_number * x
R> y
[1] 49.2
```

2.3 เวกเตอร์ (Vectors)

ในภาษาโปรแกรมโดยทั่วไป เช่น C/C++, Java มองว่าเวกเตอร์ (Vectors) แตกต่างจากตัวแปรเดี่ยว (Scalars) ตัวอย่างเช่น ให้ตัวแปร x เป็นตัวแปร Scalar ชนิดจำนวนเต็ม (Integer) ส่วนตัวแปร y เป็นตัวแปรเวกเตอร์ชนิดจำนวนเต็ม เขียนเป็นชุดคำสั่งในภาษา C/C++ ได้ดังต่อไปนี้

```
int x;
int y[5];
```

คำสั่งข้างต้นเป็นการสั่งให้คอมไพเลอร์จองหน่วยความจำสำหรับตัวแปร x เป็นตัวแปรชนิดจำนวนเต็ม 1 ตัว และตัวแปร y เป็นตัวแปรแบบอาร์เรย์ (Arrays) ชนิดจำนวนเต็ม 5 ตัว (อาร์เรย์ใน C/C++ คือข้อมูลแบบเวกเตอร์ในภาษา **R**)

ตัวแปรในภาษา **R** จะเก็บข้อมูลหลายค่าในเวกเตอร์ตัวเดียว แม้แต่ค่าเพียงค่าเดียวก็ถือว่าเป็นเวกเตอร์ที่มีสมาชิกเพียง 1 ตัว (One-element vector) เช่น ค่าตัวเลข 25.5 เก็บไว้ในตัวแปร my_value ตัวแปรแบบเวกเตอร์มีประโยชน์มากในการคำนวณหรือการเปรียบเทียบ การสร้างเวกเตอร์จะใช้ฟังก์ชัน **c()** ในการรวมตัวแปร/ตัวเลข/ค่าหลาย ๆ ตัวเข้าด้วยกันเป็นเวกเตอร์ เช่น my_vector เก็บค่า 1, 3, 5, และ 42

```
R> my_value <- 25.5
R> my_value
[1] 25.5

R> my_vector <- c(1, 3, 5, 42)
R> my_vector
[1]  1  3  5 42
```

ค่าที่ผ่านเข้ามาในเวกเตอร์สามารถคำนวณได้ หรือรับค่าจากตัวแปรก่อนหน้านี้ได้ เช่น my_vector2 รับค่าตัวเลข 2, -2, 3, ..., 5+(3-2.2)/14 และตัวแปร my_value ที่สร้างไว้ก่อนหน้านี้

```
R> my_vector2 <- c(2, -2, 3, 3.5, 2e+03, 81^0.5, 5+(3-2.2)/14, my_value)
[1]    2.000000   -2.000000    3.000000    3.500000 2000.000000    9.000000
[7]    5.057143   25.500000
```

ตัวอย่างต่อไปแสดงการรวมเวกเตอร์ 2 ตัวเข้าด้วยกัน

```
R> my_vector3 <- c(my_vector, my_vector2)
R> my_vector3
[1]    1.000000    3.000000    5.000000   42.000000    2.000000   -2.000000
[7]    3.000000    3.500000 2000.000000    9.000000    5.057143   25.500000
```

ค่าที่ผ่านเข้ามาในเวกเตอร์ my_vector3 ได้แก่ เวกเตอร์ my_vector และ my_vector2 จะเรียงตามลำดับก่อน/หลัง ตามค่าที่ส่งเข้ามาในฟังก์ชัน `c()`

R สร้างลำดับตัวเลขได้ เช่น

```
R> 5:24
[1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

R สร้างลำดับตัวเลข และเก็บไว้ในตัวแปรได้ เช่น

```
R> begin <- 2.3
R> my_vector <- begin:(-30+1.5)
R> my_vector
 [1]   2.3   1.3   0.3  -0.7  -1.7  -2.7  -3.7  -4.7  -5.7  -6.7  -7.7
[12]  -8.7  -9.7 -10.7 -11.7 -12.7 -13.7 -14.7 -15.7 -16.7 -17.7 -18.7
[23] -19.7 -20.7 -21.7 -22.7 -23.7 -24.7 -25.7 -26.7 -27.7
```

R หาความยาวของเวกเตอร์ด้วยฟังก์ชัน `length()` เช่น

```
R> length(c(1, 3, 8, 9))
[1] 4

R> length(4:15)
[1] 12
```

2.3.1 สร้างลำดับตัวเลขด้วย seq()

R สร้างลำดับตัวเลขด้วยฟังก์ชัน `seq()` โดยเริ่มต้นจากกำหนดค่าให้ `from` ไปยัง `to` และเพิ่มขึ้น/ลดลงครั้งละ `by` เช่น

```
R> seq(from = 3, to = 18, by = 3)
[1]  3  6  9 12 15 18
```

หรือกำหนดค่าเริ่มต้นให้ `from` ไปยัง `to` และแบ่งข้อมูลเป็นส่วน ๆ เท่ากับจำนวนที่กำหนด `length.out` เช่น

```
R> seq(from = 3, to = 18, length.out = 10)
[1]  3.000000  4.666667  6.333333  8.000000  9.666667 11.333333
```

```
[7] 13.000000 14.666667 16.333333 18.000000
```

กำหนดตัวแปรเก็บไว้ และใช้ในการสร้างลำดับตัวเลขได้ เช่น

```
R> begin <- 4.2
R> seq(from = begin, to = -10, by = -1.5)
[1]  4.2  2.7  1.2 -0.3 -1.8 -3.3 -4.8 -6.3 -7.8 -9.3
```

2.3.2 ทำซ้ำด้วย rep()

R สร้างลำดับตัวเลขซ้ำ ๆ ด้วยฟังก์ชัน `rep()` โดยกำหนดค่าหรือเวกเตอร์ที่ต้องการทำซ้ำ และจำนวนครั้งในการทำซ้ำสำหรับค่าหรือเวกเตอร์ `times` และจำนวนครั้งในการทำซ้ำสำหรับสมาชิกแต่ละตัวในเวกเตอร์ `each` เช่น

```
R> rep(2, times = 5)
[1] 2 2 2 2 2

R> rep(c(1, 2, 3), times = 3)
[1] 1 2 3 1 2 3 1 2 3

R> rep(c(1, 2, 3), each = 3)
[1] 1 1 1 2 2 2 3 3 3

R> rep(c(1, 2, 3), times = 2, each = 3)
[1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
```

2.3.3 เรียงลำดับตัวเลขด้วย sort()

การเรียงลำดับตัวเลขใช้ฟังก์ชัน `sort()` ค่าโดยปริยายจะเรียงลำดับตัวเลขจากน้อยไปมาก `decreasing = FALSE` ถ้าต้องการเรียงจากมากไปน้อยใช้พารามิเตอร์ `decreasing = TRUE` เช่น

```
R> sort(c(2.5, -1, -9, 3.6))
[1] -9.0 -1.0  2.5  3.6

R > sort(c(2.5, -1, -9, 3.6), decreasing = TRUE)
[1]  3.6  2.5 -1.0 -9.0
```

2.3.4 การดึงสมาชิกจากเวกเตอร์

ค่าที่แสดงผลทางหน้าจอ (Console) เริ่มด้วยวงเล็บสี่เหลี่ยม [1] หมายถึง แสดงผลเวกเตอร์ลำดับที่ 1 ต่อด้วยเวกเตอร์ลำดับถัดไป จนขึ้นบรรทัดใหม่ แล้วแสดงตำแหน่งของเวกเตอร์ลำดับถัดไป ลำดับของเวกเตอร์สามารถใช้อ้างอิงเพื่อดึงสมาชิกออกมาจากเวกเตอร์ได้ เช่น

```
R> my_vector <- c(2.5, -1.8, -9, 3, 3, 3, 4.6, 10, 9, 50)
R> my_vector[1]
[1]  2.5

R> my_value <- my_vector[2]
[1]  -1.8
```

```
R> my_vector[length(my_vector)]
[1]  50
```

R ลบสมาชิกจากเวกเตอร์ได้ โดยใช้เครื่องหมายลบ (Negative) เช่น -1 หมายถึง ไม่รวมสมาชิกในเวกเตอร์ลำดับที่ 1

```
R> my_vector[-1]
[1] -1.8 -9.0  3.0  3.0  3.0  4.6 10.0  9.0 50.0

R> my_vector[-2]
[1]  2.5 -9.0  3.0  3.0  3.0  4.6 10.0  9.0 50.0
```

การดึงหรือลบสมาชิกจากเวกเตอร์ ไม่ทำให้ค่าเวกเตอร์เริ่มต้นเปลี่ยนแปลง ยกเว้นจะกำหนดค่าใหม่ให้กับเวกเตอร์เดิม เช่น

```
R> new_vector <- my_vector[-1]
R> my_vector
[1]  2.5 -1.8 -9.0  3.0  3.0  3.0  4.6 10.0  9.0 50.0

R> my_vector <- my_vector[-1]
R> my_vector
[1] -1.8 -9.0  3.0  3.0  3.0  4.6 10.0  9.0 50.0
```

R ดึงสมาชิกในเวกเตอร์ออกมาได้หลายตัว โดยใช้เวกเตอร์แสดงลำดับที่ของสมาชิกแต่ละตัว (Vectors of indexes) เช่น

```
R> my_vector[c(1, 3, 5)]
[1]  2.5 -9.0  3.0

R> my_vector[1:4]
[1]  2.5 -1.8 -9.0  3.0

R> my_vector[4:2]
[1]  3.0 -9.0 -1.8
```

R ลบสมาชิกในเวกเตอร์ออกหลายตัวได้ โดยใช้เวกเตอร์แสดงลำดับ (Vectors of indexes) และเครื่องหมายลบ (Negative) เช่น

```
R> my_vector[-c(1, 3, 5)]
[1] -1.8  3.0  3.0  4.6 10.0  9.0 50.0
```

R แทนที่สมาชิกในเวกเตอร์ (Overwrite) ได้ โดยการกำหนดค่าใหม่ให้กับสมาชิกแต่ละตัว เช่น ถ้าต้องการแทนที่สมาชิกตัวที่ 1 ด้วย 7 จะต้องกำหนดค่า ดังนี้

```
R> my_vector[1] <- 7
R> my_vector
[1]  7.0 -1.8 -9.0  3.0  3.0  3.0  4.6 10.0  9.0 50.0
```

ถ้าต้องการแทนที่สมาชิกตัวที่ 1, 3, 5 ด้วย 7, 8, 9 จะต้องกำหนดค่า ดังนี้

```
R> my_vector[c(1, 3, 5)] <- c(7, 8, 9)
R> my_vector
```

```
[1]  7.0 -1.8  8.0  3.0  9.0  3.0  4.6 10.0  9.0 50.0
```

ถ้าต้องการแทนที่สมาชิกตัวที่ 7 ถึง 10 ด้วย 2.2 จะต้องกำหนดค่า ดังนี้

```
R> my_vector[7:10] <- 2.2
R> my_vector
[1]  7.0 -1.8  8.0  3.0  9.0  3.0  2.2  2.2  2.2  2.2
```

2.3.5 การคำนวณโดยใช้เวกเตอร์ (Vectorization)

การคำนวณโดยใช้เวกเตอร์ (Vector-oriented, vectorization, หรือ element-wise) ใน **R** จะทำการคำนวณภายในเวกเตอร์ แทนการวนรอบ (Iteration) ของภาษาโปรแกรมคอมพิวเตอร์โดยทั่วไป ซึ่งการคำนวณโดยใช้เวกเตอร์มีประโยชน์อย่างมาก ทำให้การพัฒนาโปรแกรมมีประสิทธิภาพสูงขึ้น เขียนคำสั่งได้กระชับ อ่านได้ง่าย ใช้เวลาในการเขียนคำสั่งน้อยลง รวมทั้งประมวลผลได้รวดเร็วกว่าเมื่อเทียบกับการวนรอบ (Iteration) โดยใช้โปรแกรมคอมพิวเตอร์ทั่วไป เช่น

```
R> my_vector1 <- 1.5:6.5
R> my_vector1
[1]  1.5  2.5  3.5  4.5  5.5  6.5

R> my_vector1 - c(1, 3, 5, 7, 9, 11)
[1]  0.5 -0.5 -1.5 -2.5 -3.5 -4.5
```

R ทำการคำนวณโดยการจับคู่สมาชิกแต่ละตัวตามลำดับ เช่น สมาชิกตัวแรกได้จาก my_vector1 คือ 1.5 และจาก c(1, 3, 5, 7, 9, 11) คือ 1 ได้ผลลัพธ์เท่ากับ 0.5 (1.5–1 = 0.5) สมาชิกตัวที่สองได้จาก 2.5–3 = –0.5

เมื่อความยาวของเวกเตอร์ที่ใช้คำนวณไม่เท่ากัน **R** จะเติมสมาชิกในเวกเตอร์ที่มีความยาวน้อยกว่าโดยการเพิ่มสมาชิกเข้าไปในลักษณะเป็นวงรอบ (Recycle) เช่น

```
R> my_vector3 <- c(1, -1, 2, -2)
R> my_vector1 * my_vector3
[1]  1.5 -2.5  7.0 -9.0  5.5 -6.5
Warning message:
In my_vector1 * my_vector3 :
  longer object length is not a multiple of shorter object length
```

R สามารถคำนวณโดยใช้ค่าเดี่ยว ๆ (Scalar) กับเวกเตอร์ได้ เช่น

```
R> my_value <- 2
R> my_vector1 + my_value
[1]  3.5  4.5  5.5  6.5  7.5  8.5
```

R สามารถคำนวณโดยใช้ฟังก์ชันร่วมกับเวกเตอร์ได้ เช่น

```
R> my_vector1
[1]  1.5  2.5  3.5  4.5  5.5  6.5

R> sum(my_vector1)  # Summation
[1] 24
```

```
R> prod(my_vector1)  # Multiplier
[1] 2111.484
```

R แทนที่สมาชิกหลายตัวในเวกเตอร์ (Overwrite) ได้ โดยการกำหนดค่าใหม่ให้กับสมาชิกแต่ละตัว เช่น

```
R> my_vector1[c(1, 2, 5, 6)] <- c(2.2, -2.2)
R> my_vector1
[1]  2.2 -2.2  3.5  4.5  2.2 -2.2
```

2.4 การสร้างเมทริกซ์

R สร้างเมทริกซ์ด้วยฟังก์ชัน `matrix()` โดยการกำหนดข้อมูลนำเข้าเป็นเวกเตอร์ กำหนดจำนวนแถว `nrow` และจำนวนคอลัมน์ `ncol` จำนวนสมาชิกในเวกเตอร์ต้องเท่ากับจำนวนแถวคูณจำนวนคอลัมน์ เช่น ถ้าต้องการสร้างเมทริกซ์ A ขนาด 2 แถว 3 คอลัมน์โดยเรียงข้อมูลตามคอลัมน์ (Column-by-column) เขียนคำสั่งดังนี้

```
R> A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
R> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

ถ้าต้องการสร้างเมทริกซ์ A ขนาด 2 แถว 3 คอลัมน์โดยเรียงข้อมูลตามแถว (Row-by-row) จะต้องกำหนด `byrow = TRUE` ดังนี้

```
R> A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
R> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

ถ้าไม่กำหนดจำนวนแถวและจำนวนคอลัมน์ R จะแปลให้เป็นเมทริกซ์คอลัมน์เดียว (Single-column matrix) ดังนี้

```
R> matrix(c(1, 2, 3, 4, 5, 6))
R> #  similar to -- matrix(c(1, 2, 3, 4, 5, 6), nrow = 6, ncol = 1)
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

2.4.1 การรวมเวกเตอร์ตามแนวแถวและคอลัมน์ (Row and Column Bindings)

R สามารถรวมเวกเตอร์ที่มีความยาวเท่ากันเป็นเมทริกซ์ได้ โดยใช้ฟังก์ชัน **rbind()** ในการรวมตามแถว (Row bindings) และ **cbind()** ในการรวมตามคอลัมน์ (Column bindings) เช่น

```
R> rbind(1:3, 4:6)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> cbind(c(1, 4), c(2, 5), c(3, 6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

2.4.2 ขนาดของเมทริกซ์ (Matrix Dimensions)

R ใช้ฟังก์ชัน **dim()** แสดงขนาดของเมทริกซ์ ใช้ฟังก์ชัน **nrow()** แสดงจำนวนแถว และฟังก์ชัน **ncol()** แสดงจำนวนคอลัมน์ เช่น

```
R> B <- rbind(c(1, 2, 3), 4:6, c(7, 8, 9), 10:12)
R> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

R> dim(B)
[1] 4 3
R> nrow(B)
[1] 4
R> ncol(B)
[1] 3
```

2.5 การดึงสมาชิกจากเมทริกซ์

การดึงสมาชิกของเมทริกซ์เหมือนกับการดึงสมาชิกของเวกเตอร์ แตกต่างกันที่การดึงสมาชิกจากเมทริกซ์ต้องกำหนดขนาด (Dimension) เพิ่มขึ้น โดยใส่มิติของเมทริกซ์ในเครื่องหมาย **[]** เช่น

```
R> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)
R> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

R> A[2, 3]
[1] 6
```

2.5.1 การดึงแถว คอลัมน์ และแนวทแยง (Row, Column and Diagonal Extractions)

R ดึงข้อมูลแถวหรือคอลัมน์ได้ โดยการระบุแถวหรือคอลัมน์ที่ต้องการและให้ค่าในเครื่องหมาย **[]** อีกหนึ่งตัวว่างไว้ แต่ยังคงมีเครื่องหมายจุลภาค (Comma) คั่นระหว่างแถวหรือคอลัมน์ เช่น

```
R> A[1, ]
[1] 1 2 3

R> A[, 2]
[1] 2 5 8
```

R ดึงข้อมูลแถวหรือคอลัมน์ที่ซับซ้อนได้ โดยใช้เวกเตอร์ช่วย ตัวอย่างเช่น คำสั่งแรกเป็นการดึงแถวที่ 1 และ 2 คำสั่งที่สองเป็นการดึงคอลัมน์ที่ 3 และ 1 ส่วนคำสั่งที่สามเป็นการดึงแถวที่ 3 และ 1 รวมทั้งคอลัมน์ที่ 2 และ 3

```
R> A[1:2, ]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> A[, c(3, 1)]
     [,1] [,2]
[1,]    3    1
[2,]    6    4
[3,]    9    7

R> A[c(3, 1), 2:3]
     [,1] [,2]
[1,]    8    9
[2,]    2    3
```

R ดึงข้อมูลตามแนวทแยง (Diagonal) ได้โดยใช้ฟังก์ชัน **diag()** ดังนี้

```
R> diag(A)
[1] 1 5 9
```

2.5.2 การละ (Omitting) ข้อมูลบางส่วนในเมทริกซ์

R สามารถละ (Omitting) ข้อมูลแถวหรือคอลัมน์ได้ โดยการระบุเครื่องหมายลบ (Negative) ตัวอย่างเช่น คำสั่งแรกไม่รวมแถวที่ 1 คำสั่งที่สองไม่รวมแถวที่ 1 แต่แสดงคอลัมน์ที่ 3 และ 2 ส่วนคำสั่งที่สามไม่รวมแถวที่ 1 และคอลัมน์ที่ 2 ส่วนคำสั่งสุดท้ายไม่รวมแถวที่ 1 คอลัมน์ที่ 2 และ 3

```
R> A[-1, ]
     [,1] [,2] [,3]
[1,]    4    5    6
[2,]    7    8    9

R> A[-1, 3:2]
     [,1] [,2]
[1,]    6    5
```

```
[2,]    9    8

R> A[-1, -2]
     [,1] [,2]
[1,]    4    6
[2,]    7    9

R> A[-1, -c(2, 3)]
[1] 4 7
```

2.5.3 การแทนที่ (Overwriting) ข้อมูลในเมทริกซ์

R สามารถแทนที่ (Overwriting) ข้อมูลแถวหรือคอลัมน์ได้ โดยการระบุค่าที่จะแทนที่ ด้วยการกำหนดค่า (Assignment) ให้กับเมทริกซ์ทางซ้ายมือ ดังนี้

```
R> B <- A
R> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

แทนที่ข้อมูลแถวที่ 2 ด้วย 10, 11, 12 ดังนี้

```
R> B[2, ] <- 10:12
R> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   10   11   12
[3,]    7    8    9
```

แทนที่ข้อมูลแถวที่ 1 และ 3 และคอลัมน์ที่ 2 ด้วย 20 ดังนี้

```
R> B[c(1, 3), 2] <- 20
R> B
     [,1] [,2] [,3]
[1,]    1   20    3
[2,]   10   11   12
[3,]    7   20    9
```

แทนที่ข้อมูลแถวที่ 3 ด้วยข้อมูลคอลัมน์ที่ 3 ดังนี้

```
R> B[3, ] <- B[, 3]
R> B
     [,1] [,2] [,3]
[1,]    1   20    3
[2,]   10   11   12
[3,]    3   12    9
```

แทนที่ข้อมูลแถวที่ 1 และ 3 คอลัมน์ที่ 2 และ 1 ด้วยข้อมูล 33, -33, 44, -44 ดังนี้

```
R> B[c(1, 3), 2:1] <- c(33, -33, 44, -44)
R> B
     [,1] [,2] [,3]
```

```
[1,]   44   33    3
[2,]   10   11   12
[3,]  -44  -33    9
```

แทนที่ข้อมูลตามแนวทแยง (Diagonal) ด้วย 0 ดังนี้

```
R> diag(B) <- 0
R> B
     [,1] [,2] [,3]
[1,]    0   33    3
[2,]   10    0   12
[3,]  -44  -33    0
```

2.6 ตัวดำเนินการเมทริกซ์ (Matrix Operations)

2.6.1 Matrix Transpose

R สามารถสลับข้อมูลแถวกับคอลัมน์ได้ด้วยฟังก์ชัน `t()` ดังนี้

```
R> A <- rbind(c(1, 2, 3), c(4, 5, 6))
R> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> t(A)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

2.6.2 Identity Matrix

R สร้างเมทริกซ์เอกลักษณ์ (Identity matrix) โดยใช้ฟังก์ชัน **`diag()`** และระบุขนาดของเมทริกซ์ ดังนี้

```
R> A <- diag(3)
R> A
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

2.6.3 การคูณเมทริกซ์ด้วยสเกลาร์ (Scalar Multiple of a Matrix)

การคูณเมทริกซ์ด้วยสเกลาร์ **R** สามารถใช้ตัวเลขหรือตัวแปรที่เก็บตัวเลข คูณกับเมทริกซ์ได้โดยตรง ดังนี้

```
R> A <- rbind(c(1, 2, 3), c(4, 5, 6))
R> A
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> A*2
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    8   10   12

R> 2*A
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    8   10   12
```

2.6.4 การบวกและลบเมทริกซ์ (Matrix Addition and Subtraction)

การบวกและลบเมทริกซ์ทำได้เมื่อเมทริกซ์มีขนาดเท่ากัน โดยจะทำการบวกหรือลบสมาชิกในตำแหน่งเดียวกัน (Element-wise) ดังนี้

```
R> A <- cbind(c(1, 2, 3), c(4, 5, 6))
R> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

R> B <- cbind(c(7, 8, 9), c(10, 11, 12))
R> B
     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12

R> A - B
     [,1] [,2]
[1,]   -6   -6
[2,]   -6   -6
[3,]   -6   -6
```

2.6.5 การคูณเมทริกซ์ (Matrix Multiplication)

การคูณเมทริกซ์จะใช้เครื่องหมาย **%*%** ดังนี้

```
R> A <- rbind(c(1, 2, 3), c(4, 5, 6))
R> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> B <- cbind(c(-1, -2, -3), c(-4, -5, -6))
R> B
     [,1] [,2]
```

```
[1,]   -1   -4
[2,]   -2   -5
[3,]   -3   -6

R> A %*% B
     [,1] [,2]
[1,]  -14  -32
[2,]  -32  -77

R> B %*% A
     [,1] [,2] [,3]
[1,]  -17  -22  -27
[2,]  -22  -29  -36
[3,]  -27  -36  -45
```

ถ้าใช้เครื่องหมาย * เพียงอย่างเดียว (ไม่มีเครื่องหมาย %) จะหมายถึง การคูณสมาชิกในตำแหน่งเดียวกัน (Element-wise) โดยจะคูณกันได้ก็ต่อเมื่อเมทริกซ์ทั้งสองตัวมีขนาดเท่ากัน

2.6.6 Matrix Inversion

วิธีการหนึ่งในการทำ Matrix inversion คือ ใช้ฟังก์ชัน `solve()` ดังนี้

```
R> A <- matrix(c(3, 4, 1, 2), nrow = 2, ncol = 2)
R> A
     [,1] [,2]
[1,]    3    1
[2,]    4    2

R> B <- solve(A)
R> B
     [,1] [,2]
[1,]    1 -0.5
[2,]   -2  1.5

R> A %*% B
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

2.7 อาร์เรย์หลายมิติ (Multidimensional Arrays)

เวกเตอร์และเมทริกซ์ใน R เป็นเสมือนอาร์เรย์แบบหนึ่งในภาษาโปรแกรมทั่ว ๆ ไป ซึ่งเก็บข้อมูลหลาย ๆ ตัวไว้ในตัวแปรเดียวกัน เวกเตอร์เก็บข้อมูล 1 คอลัมน์หรือ 1 แถว เมทริกซ์เป็นชุดของเวกเตอร์หลายตัวที่มีขนาดเท่ากัน ในขณะที่อาร์เรย์หลายมิติเป็นชุดของเมทริกซ์หลายตัวที่มีขนาดเท่ากันทั้งหมด

2.7.1 การสร้างอาร์เรย์

R สร้างอาร์เรย์ได้ด้วยฟังก์ชัน `array()` โดยกำหนดข้อมูลสมาชิกในอาร์เรย์ และมิติของอาร์เรย์ด้วยพารามิเตอร์ `dim` ตัวอย่างเช่น อาร์เรย์ 3 มิติขนาด 3 แถว 4 คอลัมน์และ 2 เลเยอร์ (3 rows, 4 columns, 2 layers) ดังนี้

```
R> A <- array(1:24, dim = c(3, 4, 2))
R> A
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

อาร์เรย์มากกว่า 3 มิติสามารถสร้างได้ด้วยวิธีเดียวกัน ตัวอย่างเช่น อาร์เรย์ 4 มิติขนาด ขนาด 3 แถว 4 คอลัมน์ 2 เลเยอร์ และ 3 บล็อก (3 rows, 4 columns, 2 layers, 3 blocks) ดังนี้

```
R> B <- array(rep(1:24, times = 3), dim = c(3, 4, 2, 3))
R> B
, , 1, 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2, 1

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

```
, , 1, 2

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2, 2
```

```
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

, , 1, 3

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2, 3

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

2.7.2 การดึงข้อมูลจากอาร์เรย์

เหมือนการดำเนินการในเมทริกซ์ R ดึงข้อมูลแถว คอลัมน์ หรือมิติอื่น ๆ จากอาร์เรย์ได้ โดยการระบุมิติที่ต้องการและให้ค่าอื่นในเครื่องหมาย [ ] ว่างไว้ แต่ยังคงมีเครื่องหมายจุลภาค (Comma) คั่นระหว่างมิติ

ถ้าต้องการแสดงข้อมูลในแถวที่ 2 ของเลเยอร์ที่ 2 เขียนคำสั่งได้ดังนี้

```
R> A[2, , 2]
[1] 14 17 20 23
```

ถ้าต้องการแสดงข้อมูลคอลัมน์ที่ 3 และ 1 ของแถวที่ 2 ของเลเยอร์ที่ 2 เขียนคำสั่งได้ดังนี้

```
R> A[2, c(3, 1), 2]
[1] 20 14
```

การดึงข้อมูลแล้วได้เวกเตอร์หลาย ๆ ตัว R จะแสดงผลโดยใช้คอลัมน์ในเมทริกซ์ ตัวอย่างเช่น แสดงแถวแรกของทุกเลเยอร์ โดยแสดงผลเลเยอร์ที่ 1 ด้วยคอลัมน์แรกของเมทริกซ์ และแสดงผลเลเยอร์ที่ 2 ด้วยคอลัมน์ที่ 2 ของเมทริกซ์ ดังนี้

```
R > A[1, , ]
     [,1] [,2]
[1,]    1   13
[2,]    4   16
[3,]    7   19
[4,]   10   22
```

คำสั่งต่อไปนี้แสดงแถวที่ 2 คอลัมน์ที่ 1 เลเยอร์ที่ 1 และบล็อกที่ 2

```
R> B[2, 1, 1, 2]
[1] 2
```

แสดงแถวที่ 1 และบล็อกที่ 1 ของทุกคอลัมน์ ทุกเลเยอร์ ดังนี้

```
R> B[1, , , 1]
     [,1] [,2]
[1,]    1   13
[2,]    4   16
[3,]    7   19
[4,]   10   22
```

R สร้างสับเซตข้อมูลโดยใช้เวกเตอร์ระบุตำแหน่งดัชนี ตัวอย่างเช่น แสดงแถวที่ 3 และ 2 ของคอลัมน์ที่ 1 ทุกเลเยอร์ ทุกบล็อก ดังนี้

```
R > B[3:2, 1, , ]
, , 1

     [,1] [,2]
[1,]    3   15
[2,]    2   14

, , 2

     [,1] [,2]
[1,]    3   15
[2,]    2   14

, , 3

     [,1] [,2]
[1,]    3   15
[2,]    2   14
```

R สามารถละ (Omitting) ข้อมูลบางส่วนได้ โดยการระบุเครื่องหมายลบ (Negative) รวมทั้งการแทนที่ (Overwriting) ข้อมูลสามารถทำได้เช่นเดียวกับการดำเนินการในเมทริกซ์

2.8 สรุปท้ายบท

บทนี้เป็นการใช้ R ในการคำนวณทางคณิตศาสตร์เบื้องต้น การกำหนดค่าให้กับตัวแปร (Assignment) อธิบายข้อมูลชนิดที่เป็นเวกเตอร์ (Vectors) การรวมสมาชิกแต่ละตัวเข้าเป็นเวกเตอร์ด้วยคำสั่ง `c()` การสร้างลำดับตัวเลขด้วยคำสั่ง `:` และคำสั่ง `seq()` การทำซ้ำด้วยคำสั่ง `rep()` การเรียงลำดับตัวเลขด้วยคำสั่ง `sort()` การดึงสมาชิกจากเวกเตอร์ (Subsetting) และการคำนวณโดยใช้เวกเตอร์ (Vectorization)

การสร้างเมทริกซ์ (Matrix) การรวมเวกเตอร์ตามแนวแถวและคอลัมน์ (Row and column bindings) การแสดงขนาดของเมทริกซ์ (Matrix dimensions) การสร้างสับเซตและการดึงสมาชิกของเมทริกซ์ การดึงแถว/คอลัมน์และแนวทแยง (Row, column and diagonal extractions) การละ (Omitting) ข้อมูลบางส่วนในเมทริกซ์ การแทนที่ (Overwriting) ข้อมูลในเมทริกซ์ อธิบายตัวดำเนินการเมทริกซ์ (Matrix operations) ได้แก่ การสลับข้อมูลแถวกับคอลัมน์ (Matrix transpose) การสร้างเมทริกซ์เอกลักษณ์ (Identity matrix) การคูณเมทริกซ์ด้วยสเกลาร์ (Scalar multiple of a matrix) การบวกและลบเมทริกซ์

(Matrix addition and subtraction) การคูณเมทริกซ์ (Matrix multiplication) การทำ Matrix inversion การสร้างอาร์เรย์ (Arrays) สับเซต (Subset) และการดึงข้อมูล (Extraction) จากอาร์เรย์ การละ (Omitting) และการแทนที่ (Overwriting) ข้อมูลบางส่วน

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
(), ^, *, /, +, -	วงเล็บ ยกกำลัง คูณ หาร บวก ลบ ตามลำดับ
sqrt()	หารากที่สอง (Square root)
log()	หาค่า Logarithm
exp()	หาค่า Exponential
<- หรือ =	กำหนดค่าให้กับตัวแปร (Assignment)
c()	รวมสมาชิกแต่ละตัวเข้าเป็นเวกเตอร์
x:y	สร้างลำดับตัวเลขจาก x ไปยัง y โดยการเพิ่มหรือลดทีละ 1
length()	หาความยาวของเวกเตอร์
seq()	สร้างลำดับตัวเลข
rep()	สร้างลำดับตัวเลขซ้ำ ๆ
sort()	เรียงลำดับตัวเลข
x[y]	เวกเตอร์ x สมาชิกลำดับที่ y
x[-y]	ลบสมาชิกลำดับที่ y ของเวกเตอร์ x
sum()	หาผลรวม
prod()	หาผลคูณ
matrix()	สร้างเมทริกซ์ (Matrix)
rbind()	รวมเวกเตอร์ตามแถว (Row bindings)
cbind()	รวมเวกเตอร์ตามคอลัมน์ (Column bindings)
dim()	แสดงขนาดของเมทริกซ์
nrow()	แสดงจำนวนแถวของเมทริกซ์
ncol()	แสดงจำนวนคอลัมน์ของเมทริกซ์
A[x,y]	เมทริกซ์ A แถว x คอลัมน์ y
A[-x,-y]	เมทริกซ์ A ไม่รวมแถว x ไม่รวมคอลัมน์ y
diag()	ดึงข้อมูลตามแนวทแยง (Diagonal) และสร้างเมทริกซ์เอกลักษณ์ (Identity matrix)
t()	สลับข้อมูลแถวและคอลัมน์ (Transpose)

คำสั่ง	คำอธิบาย
`sort()`	เรียงลำดับตัวเลข
`x[y]`	เวกเตอร์ x สมาชิกลำดับที่ y
`x[-y]`	ลบสมาชิกลำดับที่ y ของเวกเตอร์ x
`+, -, *`	บวก ลบ และคูณเมทริกซ์ สำหรับสมาชิกในตำแหน่งเดียวกัน (Element-wise)
`%*%`	คูณเมทริกซ์ (Matrix algebra multiplication)
`solve()`	หา Matrix inversion
`array()`	สร้างอาร์เรย์ (Array)

2.9 แบบฝึกหัดท้ายบท

1. ข้อใดคือ ยกกำลัง 3 ของลบ 5 แล้วลบด้วย 3? (อาจมีมากกว่า 1 คำตอบ)

   1.1 $(-5)\text{^}3 - 3$
   1.2 $-5\text{^}3 - 3$
   1.3 $(-5)\text{^}(3 - 3)$
   1.4 $-5\text{^}(3 - 3)$

2. ข้อใดให้ผลลัพธ์เท่ากันบ้าง? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

   2.1 $7 \times 3 + 25$
   2.2 $7 \times (3 + 25)$
   2.3 $(7 \times (3 + 25))$
   2.4 $(7 \times 3) + 25$

3. ข้อใดให้ผลลัพธ์เท่ากันบ้าง? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

   3.1 $5^2 \times 4 - 1.5^{1/2}$
   3.2 $(5^2 \times 4) - (1.5)^{1/2}$
   3.3 $5^2 \times (4 - 1.5)^{1/2}$
   3.4 $5^2 \times (4 - 1.5^{1/2})$
   3.5 $(5^2) \times (4 - 1.5)^{(1/2)}$
   3.6 $(5^2 \times 4 - 1.5)^{1/2}$

4. คำนวณหาค่าต่อไปนี้

   4.1 $log_e 0.5$
   4.2 $log_{10} 0.5$

5. เขียนตัวเลขต่อไปนี้ โดยใช้สัญลักษณ์ e (e-notation)? (เขียนด้วยตัวเองก่อน แล้วตรวจสอบด้วย **R**)

   5.1 12345678910111213
   5.2 -0.00000123456789

6. สร้างตัวแปรในการเก็บค่า $5^2 \times 4 - 1.5^{1/2}$
7. สร้างตัวแปรในการเก็บค่า 1, 2, 3, ..., 100
8. สร้างตัวแปรในการเก็บค่าระหว่าง 10 ถึง -10 โดยลดลงครั้งละ 0.6
9. สร้างตัวแปรในการเก็บค่าระหว่าง 10 ถึง -10 โดยมีตัวเลขทั้งหมด 20 ค่า
10. หาความยาวของตัวแปรที่สร้างไว้ในข้อ 7, 8 และ 9
11. สร้างตัวแปรในการเก็บค่า 10 จำนวน 100 ค่า
12. สร้างตัวแปรในการเก็บค่า 10, 20, และ 30 วนไปทั้งหมด 10 รอบ

13. สร้างตัวแปรในการเก็บค่า 10 ทั้งหมด 10 ตัว เก็บค่า 20 ทั้งหมด 10 ตัว และเก็บค่า 30 ทั้งหมด 10 ตัวต่อกันไป
14. เรียงลำดับตัวเลขในข้อ 8. และข้อ 9. จากน้อยไปมาก
15. เรียงลำดับตัวเลขในข้อ 12. และข้อ 13. จากมากไปน้อย
16. ให้แสดงค่าลำดับที่ 2 ถึง 4 ในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ออกทางหน้าจอ (Console)
17. ให้ลบค่าลำดับที่ 2 และ 4 ในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ออกทางหน้าจอ (Console) โดยไม่ทำให้ค่าใน your_vector เปลี่ยนแปลง
18. ให้ลบค่าลำดับที่ 2 และ 4 ในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ออกจากตัวแปร your_vector อย่างถาวร
19. ให้แทนที่ค่าลำดับที่ 1 4 และ 6 ในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ด้วย 20, 30 และ 40 ตามลำดับ
20. ให้เพิ่มค่าในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ด้วย 10 กับสมาชิกทุกตัวในเวกเตอร์
21. ให้เพิ่มค่าในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13 ด้วย 1, 2, 3, ... กับสมาชิกทุกตัวในเวกเตอร์ตามลำดับ
22. หาผลรวมของสมาชิกทุกตัวในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13
23. หาผลคูณของสมาชิกทุกตัวในเวกเตอร์ your_vector = 1, -3, 5, -7, 9, -11, 13
24. สร้างตัวแปรในการเก็บเมทริกซ์ 2x4 ตามแถว โดยสมาชิกแต่ละตัวประกอบด้วย 10, 20, 30, 40, 50, 60, 70, 80 ตามลำดับ
25. สร้างตัวแปรในการเก็บเมทริกซ์ 2x4 ตามคอลัมน์ โดยสมาชิกแต่ละตัวประกอบด้วย 10, 20, 30, 40, 50, 60, 70, 80 ตามลำดับ
26. แสดงขนาดของเมทริกซ์ข้อ 1 และ 2
27. เพิ่มเมทริกซ์ข้อ 1 คอลัมน์ที่ 5 ด้วยเวกเตอร์ (90, 100)
28. เพิ่มเมทริกซ์ข้อ 1 แถวที่ 3 ด้วยเวกเตอร์ (90, 100, 110, 120)
29. แสดงสมาชิกแถวที่ 1 คอลัมน์ที่ 2 และ 3 ของเมทริกซ์ข้อ 1
30. แสดงสมาชิกทุกแถว ไม่รวมคอลัมน์ที่ 2 ของเมทริกซ์ข้อ 1
31. แทนที่สมาชิกคอลัมน์ที่ 3 ของเมทริกซ์ข้อ 1 ด้วยเวกเตอร์ (130, 140)
32. สลับข้อมูลแถวและคอลัมน์ (Transpose) ของเมทริกซ์ข้อ 1
33. คูณเมทริกซ์ข้อ 1 ด้วย 3
34. บวกเมทริกซ์ข้อ 1 กับเมทริกซ์ข้อ 2
35. คูณเมทริกซ์ข้อ 1 ด้วยเมทริกซ์ข้อ 2 ที่สลับข้อมูลแถวและคอลัมน์ (Transpose) แล้ว

36. สร้างอาร์เรย์ 3 มิติขนาด 2 แถว 3 คอลัมน์และ 2 เลเยอร์ ประกอบด้วยตัวเลข 10, 20, 30, ... ตามลำดับ
37. แสดงข้อมูลแถวที่ 1 ของเลเยอร์ที่ 2
38. แสดงข้อมูลคอลัมน์ที่ 1 และ 3 ของเลเยอร์ที่ 2
39. เวกเตอร์ เมทริกซ์ และอาร์เรย์ใน **R** สัมพันธ์กันอย่างไร?
40. ลบข้อมูลบางแถวหรือคอลัมน์ในเมทริกซ์ได้อย่างไร?
41. หาผลรวมของสมาชิกทุกตัวในเมทริกซ์ได้อย่างไร?

บทที่ 3
ลิสต์ (Lists) เดต้าเฟรม (data.frame) และเดต้าเทเบิ้ล (data.table)

เวกเตอร์และอาร์เรย์มีข้อจำกัดที่สำคัญคือ ข้อมูลในเวกเตอร์และอาร์เรย์จะต้องเป็นข้อมูลชนิดเดียวกัน ในบทนี้จะกล่าวถึงโครงสร้างข้อมูลที่สามารถเก็บข้อมูลต่างประเภทไว้ในโครงสร้างข้อมูลเดียวกัน ได้แก่ ลิสต์ (Lists) เดต้าเฟรม (data.frame) และเดต้าเทเบิ้ล (data.table)

3.1 ลิสต์ (Lists)

ลิสต์เป็นโครงสร้างข้อมูลอีกประเภทหนึ่งที่สามารถเก็บข้อมูลต่างประเภท ได้แก่ เวกเตอร์ แฟกเตอร์ เมทริกซ์ อาร์เรย์ ตัวเลข สตริง หรือตัวลิสต์เองซ้อนอยู่ภายในลิสต์เดียวกันได้

3.1.1 การสร้างลิสต์

การสร้างลิสต์ใช้ฟังก์ชัน `list()` พร้อมกับระบุสมาชิกที่จะเก็บในลิสต์ และคั่นด้วยเครื่องหมายจุลภาค (Comma) ตัวอย่างเช่น ต้องการเก็บสตริง ตัวเลข เมทริกซ์ และเวกเตอร์ เขียนคำสั่งดังนี้

```
R> my_list <- list("Hello", 12.5, matrix(1:4, nrow = 2, ncol = 2), c(T, T, F, T))
R> my_list
[[1]]
[1] "Hello"

[[2]]
[1] 12.5

[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[4]]
[1]  TRUE  TRUE FALSE  TRUE
```

หาความยาวของลิสต์ด้วยฟังก์ชัน `length()` ดังนี้

```
R> length(my_list)
[1] 4
```

R ดึงข้อมูลในลิสต์โดยระบุดัชนีภายในเครื่องหมาย `[[` และ `]]` แต่ถ้าใช้เครื่องหมาย `[]` จะหมายถึงแสดงลำดับของลิสต์ ดังนี้

```
R> my_list[[1]]
[1] "Hello"

R> my_list[1]
[[1]]
```

```
[1] "Hello"
```

การดึงข้อมูลในลิสต์โดยใช้เครื่องหมาย [[ ]] จะเหมือนกับการอ้างถึงสมาชิกภายในลิสต์นั้น R สามารถนำข้อมูลที่ดึงออกมาใช้คำนวณต่อไปได้ เช่น

```
R> cat(my_list[[1]], "World!")
Hello World!

R> my_list[[2]] + 5
[1] 17.5

R> my_list[[3]] + 1.1
     [,1] [,2]
[1,]  2.1  4.1
[2,]  3.1  5.1

R> my_list[[3]][1, 2]
[1] 3

R> my_list[[3]][1, ]
[1] 1 3
```

R แทนที่ข้อมูลในลิสต์โดยใช้การกำหนดค่ากลับไปในลิสต์ เช่น

```
R> my_list[[1]]
[1] "Hello"

R> my_list[[1]] <- paste(my_list[[1]], "World!")
R> my_list
[[1]]
[1] "Hello World!"

[[2]]
[1] 12.5

[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[4]]
[1]  TRUE  TRUE FALSE  TRUE
```

ถ้าต้องการสมาชิกตัวที่ 4 และตัวที่ 2 ของลิสต์ จะต้องเขียนคำสั่งภายในเครื่องหมาย [] แต่ถ้าใช้ [[ ]] จะหมายถึงการดึงสมาชิกภายในลิสต์ ดังนี้

```
R> my_list[c(4, 2)]
[[1]]
[1]  TRUE  TRUE FALSE  TRUE

[[2]]
[1] 12.5

R> my_list[[c(4, 2)]]
```

```
[1] TRUE
```

3.1.2 การตั้งชื่อสมาชิกในลิสต์

R สามารถตั้งชื่อสมาชิกแต่ละตัวในลิสต์ เพื่อให้ง่ายต่อการใช้งานได้ โดยกำหนดเวกเตอร์ชื่อโดยใช้คำสั่ง names() ดังนี้

```
R> names(my_list) <- c("my_string", "my_value", "my_matrix", "my_logical")
R> my_list
$my_string
[1] "Hello World!"

$my_value
[1] 12.5

$my_matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$my_logical
[1]  TRUE  TRUE FALSE  TRUE
```

R สามารถอ้างถึงค่าภายในลิสต์โดยใช้ชื่อลิสต์ตามด้วย $ และชื่อสมาชิกในลิสต์ แทนการใช้เครื่องหมาย [[ ]] เช่น

```
R> my_list$my_string
[1] "Hello World!"

R> my_list$my_matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

การสร้างสับเซตยังคงเหมือนการใช้คำสั่ง R โดยทั่วไป เช่น

```
R> all(my_list$my_matrix[, 2] == my_list[[3]][, 2])
[1] TRUE
```

คำสั่งข้างบนเป็นการตรวจสอบว่าสมาชิกในสับเซตจากลิสต์ชื่อ my_list เหมือนกันทุกตัว

R สร้างลิสต์พร้อมกับการตั้งชื่อสมาชิกในลิสต์ได้ ดังนี้

```
R> my_list2 <- list(value1 = c(my_list[[4]], T, F), value2 = "new element",
                    value3 = my_list$my_matrix+1)
R> my_list2
$value1
[1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE

$value2
[1] "new element"

$value3
```

```
     [,1] [,2]
[1,]    2    4
[2,]    3    5
```

คำสั่งข้างบนเป็นการสร้างลิสต์พร้อมกับการตั้งชื่อสมาชิกในลิสต์ได้แก่ value1, value2 และ value2 ตามลำดับ

R แสดงชื่อสมาชิกในลิสต์โดยใช้ฟังก์ชัน `names()` เช่น

```
R> names(my_list)
[1] "my_string"   "my_value"    "my_matrix"   "my_logical"

R> names(my_list2)
[1] "value1" "value2" "value3"
```

3.1.3 ลิสต์ที่อยู่ในลิสต์

ลิสต์สามารถเก็บข้อมูลลิสต์อื่นอยู่ภายในลิสต์นั้นได้ โดยการกำหนดค่าให้ลิสต์ที่ต้องการเก็บค่าดังกล่าว ตัวอย่างเช่น ต้องการเก็บ my_list อยู่ภายใต้ my_list2 โดยใช้ชื่อว่า value4

```
R> my_list2$value4 <- my_list
R> my_list2
$value1
[1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE

$value2
[1] "new element"

$value3
     [,1] [,2]
[1,]    2    4
[2,]    3    5

$value4
$value4$my_string
[1] "Hello World!"

$value4$my_value
[1] 12.5
```

```
$value4$my_matrix
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$value4$my_logical
[1]  TRUE  TRUE FALSE  TRUE
```

R อ้างถึงสมาชิกในลิสต์โดยใช้ชื่อหรือดัชนีได้ เช่น

```
R> my_list2$value4$my_string
[1] "Hello World!"

R> my_list2$value4$my_logicals[1:2]
[1] TRUE TRUE

R> my_list2[[4]]$my_value
[1] 12.5
```

3.2 เดต้าเฟรม (data.frame)

เดต้าเฟรม (data.frame) เป็นโครงสร้างข้อมูลอีกประเภทหนึ่งคล้ายกับลิสต์ (List) แต่มีข้อแตกต่างที่สำคัญคือ สมาชิกในเดต้าเฟรมต้องเป็นเวกเตอร์ที่มีความยาวเท่ากัน เดต้าเฟรมมีประโยชน์อย่างมากในการเก็บข้อมูลเพื่อการวิเคราะห์ทางด้านสถิติ เนื่องจากข้อมูลที่เก็บส่วนมากมีความยาวแถว (Record) เท่ากัน

3.2.1 การสร้างเดต้าเฟรม

R สร้างเดต้าเฟรมด้วยฟังก์ชัน `data.frame()` โดยกำหนดให้ข้อมูลที่จะเก็บในเดต้าเฟรมเป็นเวกเตอร์ที่มีความยาวเท่ากัน ตัวอย่างเช่น ต้องการเก็บข้อมูลลูกค้าประกอบด้วยชื่อ อายุ และเพศ ดังนี้

```
R> my_data <- data.frame(name = c("Big", "Dan", "June", "James", "Kate"),
                         age = c(23, 33, 25, 48, 55),
                         gender = factor(c("M", "M", "F", "M", "F")))
R> my_data
   name age gender
1   Big  23      M
2   Dan  33      M
3  June  25      F
4 James  48      M
5  Kate  55      F
```

คำสั่งข้างบนเป็นการสร้างเดต้าเฟรมเก็บชื่อ อายุ และเพศของลูกค้า จะเห็นว่าข้อมูลนำเข้าจะต้องเป็นเวกเตอร์ที่มีความยาวเท่ากัน ถ้าใส่ข้อมูลที่มีความยาวเวกเตอร์ไม่เท่ากัน R จะทำการเติมข้อมูลให้เวกเตอร์ที่มีความยาวน้อยกว่า ซึ่งทำให้เกิดข้อผิดพลาดได้ ข้อมูลแต่ละแถวในเดต้าเฟรมเรียกว่า ระเบียน (Record) แต่ละคอลัมน์เรียกว่า ตัวแปร (Variable) หรือ feature

R อ้างถึงข้อมูลแต่ละตัวในเดต้าเฟรมด้วยดัชนีแถวและคอลัมน์ คล้ายกับการอ้างถึงสมาชิกในเมทริกซ์ เช่น

```
R> my_data[3, 2]
[1] 25
```

คำสั่งข้างบนเป็นการอ้างถึงสมาชิกแถวที่ 3 และคอลัมน์ที่ 2 ซึ่งแสดงอายุเท่ากับ 25 ถ้าต้องการอ้างถึงสมาชิกในแถวที่ 2 ถึง 5 ของคอลัมน์ที่ 3 เขียนคำสั่งได้ดังนี้

```
R> my_data[2:5, 3]
[1] M F M F
Levels: F M
```

คำสั่งข้างบนเป็นการแสดงเพศของลูกค้าในแถวที่ 2 ถึง 5 ถ้าต้องการแสดงเพศและชื่อตามลำดับ เขียนคำสั่งได้ดังนี้

```
R> my_data[, c(3, 1)]
  gender  name
1      M   Big
2      M   Dan
3      F  June
4      M James
5      F  Kate
```

R อ้างถึงสมาชิกในเดต้าเฟรมโดยใช้ชื่อได้ เช่นเดียวกับการอ้างถึงสมาชิกในลิสต์ เช่น

```
R> my_data$age
[1] 23 33 25 48 55
```

R อ้างถึงสมาชิกในเดต้าเฟรมโดยการสร้างสับเซตได้ เช่น

```
R> my_data$age[1:3]
[1] 23 33 25
```

R แสดงขนาดของเดต้าเฟรมได้โดยใช้ฟังก์ชัน `nrow()` `ncol()` และ `dim()` เช่น

```
R> nrow(my_data)
[1] 5

R> ncol(my_data)
[1] 3

R> dim(my_data)
[1] 5 3
```

ในการสร้างเดต้าเฟรม R จะแปลงสตริงให้เป็นแฟกเตอร์โดยอัตโนมัติ เช่น

```
R> my_data$name
[1] Big   Dan   June  James Kate
Levels: Big Dan James June Kate
```

การหลีกเลี่ยงการแปลงสตริงเป็นแฟกเตอร์โดยอัตโนมัติ ในการสร้างเดต้าเฟรมต้องใส่พารามิเตอร์ `stringsAsFactors = FALSE` เช่น

```
R> my_data <- data.frame(name = c("Big", "Dan", "June", "James", "Kate"),
                         age = c(23, 33, 25, 48, 55),
                         gender = factor(c("M", "M", "F", "M", "F")),
                         stringsAsFactors = FALSE)
R> my_data$name
[1] "Big"   "Dan"   "June"  "James" "Kate"
```

3.2.2 การเพิ่มตัวแปร/ข้อมูลและการรวมเดต้าเฟรม

การเพิ่มข้อมูลแถว (Record) เข้าไปในเดต้าเฟรม R ใช้ฟังก์ชัน `rbind()` และส่วนการเพิ่มตัวแปร (Variable) และข้อมูลตามแนวคอลัมน์ใช้ฟังก์ชัน `cbind()` เช่นเดียวกับเมทริกซ์ โดยในขั้นแรกจะทำการสร้างเดต้าเฟรมใหม่ ดังนี้

```
R> new_record <- data.frame(name = "Ben", age=27,
                            gender = factor("M",
                                            levels = levels(my_data$gender)))
R> new_record
  name age gender
1  Ben  27      M
```

ในการเพิ่มข้อมูลแถว (Record) เข้าไปในเดต้าเฟรมเดิม ต้องระวังให้ตัวแปรมีชนิดและชื่อเหมือนกับตัวแปรในเดต้าเฟรมเดิม R สามารถใช้ฟังก์ชัน `rbind()` ในการเพิ่มข้อมูลแถว (Record) ได้ดังนี้

```
R> my_data <- rbind(my_data, new_record)
R> my_data
   name age gender
1   Big  23      M
2   Dan  33      M
3  June  25      F
4 James  48      M
5  Kate  55      F
6   Ben  27      M
```

การเพิ่มตัวแปร (Variable) และข้อมูลในคอลัมน์เข้าไปในเดต้าเฟรมเดิม จะต้องสร้างเวกเตอร์ตัวแปรใหม่ให้มีความยาวเท่ากับเดต้าเฟรมเดิม และใช้ฟังก์ชัน `cbind()` ตัวอย่างเช่น ต้องการเพิ่มความพึงพอใจในการใช้บริการของลูกค้า (Satisfaction) ต่ำ/กลาง/สูง ขั้นแรกทำการสร้างเวกเตอร์ขึ้นมาก่อน โดยสมมติว่าสร้างเวกเตอร์ชื่อ sat ดังนี้

```
R> sat <- c("High", "Medium", "High", "Low", "Medium", "High")
R> sat <- factor(sat, levels = c("Low", "Medium", "High"))
R> sat
[1] High   Medium High   Low    Medium High
Levels: Low Medium High
```

ขั้นต่อไปทำการรวมเวกเตอร์ sat กับเดต้าเฟรมเดิมโดยใช้ฟังก์ชัน `cbind()` ดังนี้

```
R> my_data <- cbind(my_data, sat)
R> my_data
   name age gender    sat
1   Big  23      M   High
2   Dan  33      M Medium
3  June  25      F   High
4 James  48      M    Low
5  Kate  55      F Medium
6   Ben  27      M   High
```

การเพิ่มตัวแปร/ข้อมูลเข้าไปในเดต้าเฟรมเดิม นอกจากฟังก์ชัน **rbind()** และ **cbind()** แล้วยังสามารถเพิ่มตัวแปร/ข้อมูลเข้าไปได้โดยกำหนดค่าตัวแปรใหม่ให้กับเดต้าเฟรมได้ เช่น ถ้าต้องการสร้างข้อมูลอายุเป็นหน่วยเดือน โดยกำหนดให้ตัวแปรใหม่ชื่อ age_month เขียนคำสั่งได้ดังนี้

```
R> my_data$age_month <- my_data$age * 12
R> my_data
   name age gender    sat age_month
1   Big  23      M   High       276
2   Dan  33      M Medium       396
3  June  25      F   High       300
4 James  48      M    Low       576
5  Kate  55      F Medium       660
6   Ben  27      M   High       324
```

3.2.3 การดึงข้อมูลโดยใช้เวกเตอร์ตรรกะ

R สามารถใช้เวกเตอร์ตรรกะในการดึงข้อมูลจากเดต้าเฟรมได้ เช่นเดียวกับข้อมูลชนิดอื่น ๆ เช่น ต้องการแสดงข้อมูลเฉพาะลูกค้าชาย

```
R> my_data$gender == "M"
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE

R> my_data[my_data$gender == "M", ]
   name age gender    sat age_month
1   Big  23      M   High       276
2   Dan  33      M Medium       396
4 James  48      M    Low       576
6   Ben  27      M   High       324
```

R ละบางคอลัมน์ที่ไม่ต้องการแสดงผลได้ เช่น ต้องการข้อมูลเฉพาะลูกค้าชาย ไม่ต้องการแสดง gender ที่อยู่ในคอลัมน์ที่ 3

```
R> my_data[my_data$gender == "M", -3]
   name age    sat age_month
1   Big  23   High       276
2   Dan  33 Medium       396
4 James  48    Low       576
6   Ben  27   High       324
```

R แสดงผลโดยระบุชื่อตัวแปรได้ เช่น ต้องการข้อมูลเฉพาะลูกค้าชาย แสดงเฉพาะชื่อ อายุ และความพึงพอใจ

```
R> my_data[my_data$gender == "M", c("name", "age", "sat")]
   name age    sat
1   Big  23   High
2   Dan  33 Medium
4 James  48    Low
6   Ben  27   High
```

R สร้างสับเซตโดยกำหนดเงื่อนไขที่ซับซ้อนได้ เช่น ต้องการข้อมูลเฉพาะลูกค้าชายที่อายุมากกว่า 25 ปี

```
R> my_data[my_data$gender == "M" & my_data$age > 25, ]
   name age gender    sat age_month
2   Dan  33      M Medium       396
4 James  48      M    Low       576
6   Ben  27      M   High       324
```

ถ้าไม่มีข้อมูลตามเงื่อนไขที่กำหนด R จะแสดงผลศูนย์แถว ดังนี้

```
R> my_data[my_data$age < 20, ]
[1] name      age       gender    sat       age_month
<0 rows> (or 0-length row.names)
```

3.3 เดต้าเทเบิ้ล (data.table)

ไฟล์ขนาดใหญ่มาก ๆ มักจะมีปัญหาเรื่องใช้เวลาประมวลผลนานเกินไป ทำให้ทำงานช้า ขาดประสิทธิภาพ นอกจากนี้อาจจะมีปัญหาเรื่องหน่วยความจำของเครื่องไม่เพียงพอ R มีแพ็กเกจชื่อว่า data.table ใช้จัดการกับปัญหาดังกล่าวได้อย่างมีประสิทธิภาพ มีข้อดีหลายประการ (Dowle, 2018) สรุปได้ดังนี้

- จัดการกับข้อมูลขนาดใหญ่มาก ๆ ได้ (ตัวอย่างเช่น ข้อมูลที่มีขนาดถึง 100 GB)
- ประมวลผลคำสั่งต่าง ๆ ได้รวดเร็ว
- มีคำสั่งที่ยืดหยุ่น ใช้งานง่าย ทำให้เขียนชุดคำสั่งได้สะดวก รวดเร็วขึ้น

เดต้าเทเบิ้ล (data.table) เป็นส่วนขยายของเดต้าเฟรม (data.frame) ทำให้คำสั่งต่าง ๆ ในเดต้าเทเบิ้ลใกล้เคียงกับเดต้าเฟรม นอกจากนี้ยังสามารถแปลงข้อมูลเดต้าเฟรมเป็นเดต้าเทเบิ้ลได้อีกด้วย

3.3.1 รูปแบบการใช้งานเดต้าเทเบิ้ล (data.table)

รูปแบบการใช้งานเดต้าเทเบิ้ล (data.table) ประกอบด้วย 3 ส่วนหลัก คือ

```
DT[i, j, k]

# R: i = rows          ==> SQL: where
# R: j = columns       ==> SQL: select
# R: k = groups        ==> SQL: grouped by

# Take DT, subset rows using "i", then compute "j" grouped by "k"
```

- ส่วนแรก i เป็นการกำหนดแถวที่ต้องการ (On which row?)
- ส่วนที่สอง j เป็นการบอกว่าให้ประมวลผลอะไร (What to do?)
- ส่วนสุดท้าย k เป็นการจัดกลุ่มข้อมูล (Grouped by what?)

พารามิเตอร์ **i, j, k** ในเดต้าเทเบิ้ล (data.table) เทียบเท่าคำสั่ง SQL[10] ได้แก่ where, select, and grouped by ตามลำดับ

การติดตั้งแพ็กเกจ **data.table** ในเครื่องคอมพิวเตอร์ แล้วโหลด **data.table** มาไว้ในหน่วยความจำ ทำได้ดังนี้

```
R> install.packages("data.table")
R> library(data.table)
```

เพื่อความสะดวก จะใช้ข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** คือ mtcars เพื่อแสดงตัวอย่างดังนี้

```
R> str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

R> head()
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

แปลงข้อมูลในรูปเดต้าเฟรม (data.frame) ให้เป็นเดต้าเทเบิ้ล (data.table) โดยใช้คำสั่ง **as.data.table()** ดังนี้

```
R> my_DT <- as.data.table(mtcars)

R> class(my_DT)
[1] "data.table" "data.frame"
```

[10] SQL ย่อมาจาก Structured Query Language เป็นภาษามาตรฐานที่ใช้จัดการฐานข้อมูล

คำสั่ง **class()** แสดงให้เห็นว่า my_DT เป็นทั้งเดต้าเทเบิ้ล (data.table) และเดต้าเฟรม (data.frame)

สามารถเข้าถึงตัวแปร/คอลัมน์ (Column) ด้วยคำสั่งเช่นเดียวกับเดต้าเฟรม (data.frame) เช่น ต้องการเข้าถึงตัวแปรชื่อว่า mpg ใช้คำสั่ง ดังนี้

```
R> my_DT$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
[15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
[29] 15.8 19.7 15.0 21.4
```

R สามารถสร้างเดต้าเทเบิ้ล (data.table) โดยใช้คำสั่ง **data.table()** ดังนี้

```
R> DT <- data.table(x = rep(c("b", "a", "c"), each = 2), y = c(1, 3), v = 1:6)
R> DT
   x y v
1: b 1 1
2: b 3 2
3: a 1 3
4: a 3 4
5: c 1 5
6: c 3 6
```

เดต้าเทเบิ้ล (data.table) เป็นส่วนขยายของเดต้าเฟรม (data.frame) คุณสมบัติต่าง ๆ ของเดต้าเทเบิ้ล (data.table) ได้รับการถ่ายทอดมาจากเดต้าเฟรม (data.frame) ดังนี้

```
R> DF <- data.frame(x = rep(c("b", "a", "c"), each = 2), y = c(1, 3), v = 1:6)
R> identical(dim(DT), dim(DF))
[1] TRUE

R> identical(DF$a, DT$a)
[1] TRUE

R> is.list(DF)
[1] TRUE

R> is.list(DT)
[1] TRUE

R> is.data.frame(DT)
[1] TRUE
```

3.3.2 การสับเซตแถว i (Subset rows "i")

การสับเซตแถว **i** (Subset rows "i") ทำได้เช่นเดียวกับเดต้าเฟรม (Dataframes) เช่น

```
R> DT[2,]  # 2nd row
   x y v
1: b 3 2

R> DT[2]  # comma "," can be ignored in data.table
   x y v
```

```
1: b 3 2

R> DT[3:2]  # 3rd and 2nd row
   x y v
1: a 1 3
2: b 3 2
```

แสดงข้อมูลทั้งหมดยกเว้นแถวที่ 2 ถึงแถวที่ 4

```
R> DT[!2:4]  # all rows other than 2:4, option: DT[-(2:4)]
   x y v
1: b 1 1
2: c 1 5
3: c 3 6
```

สามารถใส่เงื่อนไขที่ต้องการได้ เช่น ต้องการตัวแปร v มากกว่า 3

```
R> DT[v>3]  # all rows where DT$v > 3
   x y v
1: a 3 4
2: c 1 5
3: c 3 6
```

ใส่เงื่อนไขที่ซับซ้อนด้วยนิพจน์ (Expression) ที่ประกอบด้วยเครื่องหมายและ (**&**) หรือ (**|**) ได้ เช่น ต้องการตัวแปร x ไม่เท่ากับ "a" และตัวแปร y มากกว่า 2

```
R> DT[x!="a" & y>2]  # compound logical expressions
   x y v
1: b 3 2
2: c 3 6
```

เรียงข้อมูลจากน้อยไปมาก หรือกลับกันได้ เช่น ต้องการเรียงตามตัวแปร x จากน้อยไปมาก

```
R> DT[order(x)]  # no need for order(DT$x)
   x y v
1: a 1 3
2: a 3 4
3: b 1 1
4: b 3 2
5: c 1 5
6: c 3 6
```

ต้องการเรียงข้อมูลตามตัวแปร x จากมากไปน้อย

```
R > DT[order(-x)]
   x y v
1: c 1 5
2: c 3 6
3: b 1 1
4: b 3 2
5: a 1 3
6: a 3 4
```

ต้องการเรียงตามตัวแปร x จากน้อยไปมาก แล้วเรียงตามตัวแปร y จากมากไปน้อย

```
R> DT[order(x, -y)]
   x y v
1: a 3 4
2: a 1 3
3: b 3 2
4: b 1 1
5: c 3 6
6: c 1 5
```

3.3.3 การเลือก/คำนวณคอลัมน์หรือตัวแปร j (Select/compute on column "j")

การเลือกคอลัมน์หรือตัวแปร **j** (Select column "j") ทำได้โดยการกำหนดเวกเตอร์หรือลิสต์เข้าไป เช่น ต้องการคอลัมน์ v โดยผ่านค่า v แบบเวกเตอร์และส่งค่ากลับแบบเวกเตอร์

```
R> DT[, v]  # v column (as vector)
[1] 1 2 3 4 5 6
```

ต้องการคอลัมน์ v โดยผ่านค่า v แบบลิสต์และส่งค่ากลับแบบเดต้าเทเบิ้ล (data.table)

```
R> DT[, list(v)]  # v column (as list)
   v
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6

R> DT[, .(v)]  # same as above, .() is a shorthand alias to list()
   v
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
```

ในเดต้าเทเบิ้ล (data.table) ผู้ใช้สามารถใช้คำสั่ง **.()** แทน **list()** ได้

การคำนวณคอลัมน์ **j** (Compute on column "j") ทำได้โดยกำหนดฟังก์ชันที่ต้องการ เช่น ต้องการหาผลรวมของคอลัมน์ v โดยให้คืนค่ากลับมาเป็นเวกเตอร์ ดังนี้

```
R> DT[, sum(v)]  # sum of column v, returned as vector
[1] 21
```

ถ้าต้องการให้คืนค่ากลับมาเป็นเดต้าเทเบิ้ล ต้องผ่านลิสต์เข้าไป ดังนี้

```
R> DT[, .(sum(v))]  # same, but return data.table (column autonamed V1)
   V1
1: 21
```

คำสั่งข้างต้นจะตั้งชื่อตัวแปร/คอลัมน์ให้อัตโนมัติชื่อว่า v1 ผู้ใช้สามารถตั้งชื่อเองได้ เช่น ต้องการตั้งชื่อตัวแปร/คอลัมน์ว่า my_sum เขียนคำสั่งได้ดังนี้

```
R> DT[, .(my_sum = sum(v))]  # same, but column named "my_sum"
   my_sum
1:     21
```

สามารถคืนค่ากลับเป็นเดต้าเทเบิ้ลได้หลายคอลัมน์

```
R> DT[, .(v, v_power = v*2, v_mean = mean(v))]
   v v_power v_mean
1: 1       2    3.5
2: 2       4    3.5
3: 3       6    3.5
4: 4       8    3.5
5: 5      10    3.5
6: 6      12    3.5
```

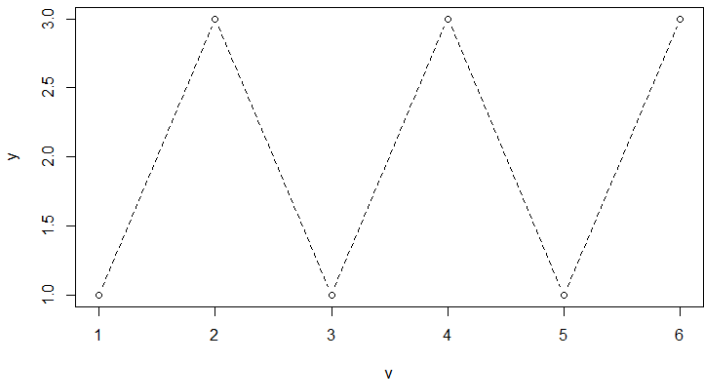
สามารถสับเซตแถว **i** พร้อมกับเลือก/คำนวณคอลัมน์ **j** ได้

```
R> DT[2:5, sum(v)]  # sum(v) over rows 2 and 5, return vector
[1] 14

R> DT[2:5, .(sum = sum(v))]  # same, but return data.table with column "Sum"
   sum
1:  14
```

ผู้ใช้สามารถใช้คำสั่งอะไรก็ได้ในคอลัมน์ **j** เช่น ต้องการพล็อตกราฟระหว่าง v และ y ทำได้ ดังนี้

```
R> DT[, plot(v, y, type = "b", lty = 2)]
```



สามารถใส่ชุดคำสั่งในคอลัมน์ **j** ภายในเครื่องหมาย **{ }** ได้เช่น ต้องการพิมพ์ค่า x และพล็อตกราฟระหว่าง v และ y ดังนี้

```
DT[, { print(x)
       plot(v, y, type = "b", lty = 2)
     } ]
[1] "b" "b" "a" "a" "c" "c"
NULL
```

3.3.4 การจัดกลุ่มโดย k (Grouped by "k")

หลังจากสับเซตแถว **i** และเลือกตัวแปร/คอลัมน์ **j** หรือคำนวณตัวแปร/คอลัมน์ **j** แล้ว ผู้ใช้สามารถจัดกลุ่มผลลัพธ์ที่ได้ตาม **k** (Grouped by "k") ได้ดังนี้

```
R> DT[, sum(v), by = x]  # ad hoc by, order of groups preserved in result
   x V1
1: b  3
2: a  7
3: c 11

R> DT[, sum(v), keyby = x]  # same, but order the result on by column "x"
   x V1
1: a  7
2: b  3
3: c 11

R> DT[, sum(v), by = x][order(x)]  # same but by chaining expressions
   x V1
1: a  7
2: b  3
3: c 11
```

ผู้ใช้สามารถจัดกลุ่มผลลัพธ์ที่ได้ตาม **k** (Grouped by "k") โดยที่ **k** เป็นลิสต์ประกอบด้วยตัวแปร/คอลัมน์หลายตัวแปร/คอลัมน์ นอกจากนี้ยังเรียงตามตัวแปร/คอลัมน์ได้หลายลักษณะ ดังนี้

```
R> DT <- data.table(x = rep(c("b", "a", "c"), each = 4), y = c(1, 2), v = 1:12)
R> DT[, .(sum = sum(v)), by = .(x,y)][order(x,-y)]  # sum by x,y order by x,-y
   x y sum
1: a 2  14
2: a 1  12
3: b 2   6
4: b 1   4
5: c 2  22
6: c 1  20
```

สามารถสร้างสับเซตได้อย่างรวดเร็วโดยใช้ Binary search ได้ดังนี้

```
# fast ad hoc row subsets (subsets as joins)
R> DT["a", on = "x"]  # same as x == "a" but uses binary search (fast)
   x y v
1: a 1 5
2: a 2 6
3: a 1 7
4: a 2 8
```

คำสั่งต่อไปเหมือนคำสั่งข้างบน แต่ใช้ **.()** ไม่ต้องใช้เครื่องหมายจุลภาค ("") กับทุกตัวแปร

```
R> DT["a", on = .(x)]  # same, for convenience, no need to quote every column
   x y v
1: a 1 5
2: a 2 6
```

```
3: a 1 7
4: a 2 8

R> DT[.("a"), on = "x"]  # same as above
R> DT[x == "a"]  # same, single "==" internally optimised (binary search)
R> DT[.("b", 2), on = c("x", "y")]  # join on columns x, y (binary search)
   x y v
1: b 2 2
2: b 2 4

R> DT[.("b", 2), on = .(x, y)]  # same, but using on = .()
R> DT[.("b", 1:3), on = c("x", "y")]  # no match returns NA
   x y  v
1: b 1  1
2: b 1  3
3: b 2  2
4: b 2  4
5: b 3 NA
```

นอกจากนี้เดต้าเทเบิ้ล (data.table) ยังมีความสามารถอื่น เช่น รวม (Join) เดต้าเทเบิ้ลอื่น เพิ่ม (Add) ปรับปรุง (Update) ลบ (Delete) ข้อมูลในเดต้าเทเบิ้ลได้อย่างรวดเร็ว ผู้ที่สนใจสามารถศึกษาเพิ่มเติมได้จากแหล่งข้อมูลอื่น ๆ เช่น https://github.com/Rdatatable/data.table/wiki หรือพิมพ์ **?data.table** ที่ console

3.4 สรุปท้ายบท

ในบทนี้กล่าวถึงโครงสร้างข้อมูลที่สามารถเก็บข้อมูลต่างประเภทได้ ประกอบด้วย ลิสต์ (Lists) เดต้าเฟรม (data.frame) และเดต้าเทเบิ้ล (data.table) ลิสต์เป็นโครงสร้างข้อมูลประเภทหนึ่งซึ่งสามารถเก็บข้อมูลต่างประเภท ข้อมูลที่สามารถเก็บในลิสต์ ได้แก่ เวกเตอร์ แฟกเตอร์ เมทริกซ์ อาร์เรย์ ตัวเลข สตริง หรือตัวลิสต์เองซ้อนอยู่ภายในลิสต์เดียวกันได้ กล่าวถึงการสร้างลิสต์ การตั้งชื่อสมาชิกในลิสต์ ลิสต์ที่อยู่ในลิสต์

เดต้าเฟรม (data.frame) เป็นโครงสร้างข้อมูลอีกประเภทหนึ่งคล้ายกับลิสต์ แต่มีข้อแตกต่างที่สำคัญ คือ สมาชิกในเดต้าเฟรมต้องเป็นเวกเตอร์ที่มีความยาวเท่ากัน เดต้าเฟรมมีประโยชน์อย่างมากในเก็บข้อมูลเพื่อการวิเคราะห์ทางด้านสถิติ กล่าวถึงการสร้างเดต้าเฟรม การเพิ่มตัวแปร/ข้อมูล การรวมเดต้าเฟรม การดึงข้อมูลโดยใช้เวกเตอร์ตรรกะ

เดต้าเทเบิ้ล (data.table) เป็นส่วนขยายของเดต้าเฟรม ใช้กับไฟล์ขนาดใหญ่มาก ๆ ได้อย่างมีประสิทธิภาพ กล่าวถึงรูปแบบการใช้งานเดต้าเทเบิ้ล (**i, j, k**) การสับเซตแถว **i** การเลือก/คำนวณคอลัมน์หรือตัวแปร **j** การจัดกลุ่มโดย **k**

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`list()`	สร้างลิสต์ (List)
`length()`	หาความยาวของลิสต์
`[[ ]]`	อ้างถึงสมาชิกภายในลิสต์
`[ ]`	แสดงลำดับของลิสต์
`names()`	กำหนดชื่อตัวแปรในลิสต์/วัตถุ หรือแสดงชื่อตัวแปรในลิสต์/วัตถุ
`$`	ใช้อ้างถึงสมาชิกภายในลิสต์/เดต้าเฟรม โดยใช้ชื่อลิสต์/เดต้าเฟรมตามด้วยเครื่องหมาย **$**
`data.frame()`	สร้างเดต้าเฟรม
`[ , ]`	ดึงข้อมูลจากเดต้าเฟรม
`nrow()`	แสดงจำนวนแถวของเดต้าเฟรม
`ncol()`	แสดงจำนวนคอลัมน์/ตัวแปรของเดต้าเฟรม
`dim()`	แสดงขนาด/มิติของเดต้าเฟรม
`rbind()`	การเพิ่มข้อมูลเข้าไปในเดต้าเฟรม
`cbind()`	การเพิ่มตัวแปรและข้อมูลในคอลัมน์เข้าไปในเดต้าเฟรม
`str()`	เรียกดูโครงสร้างข้อมูล
`head()`	แสดงข้อมูล 6 ลำดับแรก
`install.packages()`	การติดตั้ง (Install) แพ็กเกจ (Packages)
`library()` หรือ `require()`	การโหลด (Load) แพ็กเกจเข้ามาที่หน่วยความจำของเครื่อง
`data.table()`	สร้างเดต้าเทเบิ้ล (data.table)
`as.data.table()`	แปลงข้อมูลในรูปเดต้าเฟรม (Dataframes) ให้เป็นเดต้าเทเบิ้ล (data.table)
`class()`	หาว่าสมาชิกในวัตถุเป็นคลาสอะไร
`identical()`	ตรวจสอบว่าวัตถุเหมือนกันทุกประการ
`is.list()`	ตรวจสอบว่าเป็นสิสต์หรือไม่
`is.data.frame()`	ตรวจสอบว่าเป็นเดต้าเฟรม (Dataframes) หรือไม่
`order()`	เรียงข้อมูลภายในเดต้าเทเบิ้ล (data.table)
`.()`	สร้างลิสต์ (List) ในเดต้าเทเบิ้ล (data.table) เช่นเดียวกับคำสั่ง `list()`
`by = k`	จัดกลุ่มผลลัพธ์ที่ได้ตาม **k** (Grouped by k) ในเดต้าเทเบิ้ล (data.table)
`keyby = k`	จัดกลุ่มผลลัพธ์ที่ได้ตาม **k** (Grouped by k) และเรียงตาม **k** ในเดต้าเทเบิ้ล (data.table)
`on =`	สร้างสับเซตโดยใช้ Binary search ในเดต้าเทเบิ้ล (data.table)

3.5 แบบฝึกหัดท้ายบท

1. ให้สร้างลิสต์ที่ประกอบด้วย (1) เลขจำนวนเต็ม -10, -8, -6, ..., 10 (2) เมทริกซ์ขนาด 3x4 ประกอบด้วยเลขจำนวนเต็มตั้งแต่ 1 ถึง 9 (3) เวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) ประกอบด้วยข้อมูล 3 ระดับ ได้แก่ ต่ำ (Low) ปานกลาง (Medium) และสูง (High) โดยมีค่าระดับต่ำ < ปานกลาง < สูง ประกอบด้วยข้อมูล ดังนี้ c("Low", "Medium", "Low", "Medium", "High", "High", "Low") และ (4) ข้อมูลเวกเตอร์ตรรกะประกอบด้วย c(T, F, T, F, T, T, F, F)

   1.1 ให้ดึงสมาชิกตัวที่ 4 ของเวกเตอร์เลขจำนวนเต็มชุดแรกที่เป็นสมาชิกในลิสต์
   1.2 ให้ดึงสมาชิกตัวที่ 4 และ 6 ของเวกเตอร์เลขจำนวนเต็มชุดแรกที่เป็นสมาชิกในลิสต์
   1.3 ให้ดึงสมาชิกทุกตัวที่มากกว่า 0 ของเวกเตอร์เลขจำนวนเต็มชุดแรกที่เป็นสมาชิกในลิสต์
   1.4 ให้ดึงสมาชิกแถวที่ 3 คอลัมน์ที่ 2 และ 4 ของเมทริกซ์ที่เป็นสมาชิกในลิสต์
   1.5 ให้เปรียบเทียบสมาชิกตัวที่ 1 ว่ามีค่าน้อยกว่าสมาชิกตัวที่ 2 ของเวกเตอร์ที่เป็นแฟกเตอร์หรือไม่
   1.6 ให้หาว่าสมาชิกในเวกเตอร์ที่เป็นแฟกเตอร์ ลำดับที่เท่าไรบ้างที่มีค่าเท่ากับ High

2. ให้สร้างเดต้าเฟรม (Dataframe) ประกอบด้วยชื่อ (Name) เพศ (Gender) และความพึงพอใจในการทำงาน (Job satisfaction) โดยที่เพศเป็นแฟกเตอร์ และความพึงพอใจในการทำงานเป็นแฟกเตอร์ที่มีระดับชั้น (Levels) ดังนี้ Low < Medium < High ข้อมูลประกอบด้วย

Name	Gender	Job satisfaction
Mark	Male	Low
Dan	Male	Medium
Anne	Female	High
June	Female	Medium
James	Male	High
Mary	Female	Low

   2.1 มีพนักงานเข้ามาใหม่ชื่อ Sophia เพศหญิง ความพึงพอใจในการทำงานสูง ให้เพิ่มข้อมูลดังกล่าวเข้าไปในเดต้าเฟรม
   2.2 พนักงานทั้งหมด ได้แก่ Mark, Dan, Anne, June, James, Mary และ Sophia มีอายุเท่ากับ 35, 48, 28, 33, 52, 43 และ 25 ตามลำดับ ให้เพิ่มข้อมูลตัวแปรดังกล่าวเข้าไปในเดต้าเฟรม
   2.3 ให้แสดงข้อมูลพนักงานหญิงที่มีความพึงพอใจในการทำงานระดับปานกลางขึ้นไป
   2.4 ให้แสดงข้อมูลพนักงานที่มีชื่อขึ้นต้นด้วยอักษร M (Hint: ใช้ฟังก์ชัน `substr()` ช่วย)

3. ให้สร้างเดต้าเทเบิ้ล (data.table) จากเดต้าเฟรม (Dataframe) ในข้อ 2.

3.1 ให้แสดงข้อมูลแถวที่ 2

3.2 ให้แสดงข้อมูลแถวที่ 4 ถึงแถวที่ 2 ตามลำดับ

3.3 ให้แสดงข้อมูลทั้งหมดที่ไม่ใช่แถวที่ 2 ถึงแถวที่ 5

3.4 ให้แสดงข้อมูลพนักงานที่มีอายุมากกว่า 40 ปี และมีความพึงพอใจในการทำงานระดับปานกลางขึ้นไป

3.5 ให้เรียงลำดับข้อมูลตามความพึงพอใจในการทำงานจากมากไปน้อย และในแต่ละระดับความพึงพอใจในการทำงานให้เรียงจากอายุน้อยไปมาก

3.6 ให้ดึงข้อมูลอายุออกมาเป็นเวกเตอร์

3.7 ให้ดึงข้อมูลอายุออกมาเป็นเดต้าเทเบิ้ล (data.table)

3.8 ให้แสดงข้อมูลชื่อ อายุ และอายุเฉลี่ย

3.9 ให้แสดงข้อมูลชื่อ ความพึงพอใจในการทำงาน อายุ และอายุเฉลี่ย สำหรับพนักงานที่มีความพึงพอใจในการทำงานสูง และมีอายุน้อยกว่า 30 ปี

3.10 ให้แสดงข้อมูลอายุเฉลี่ย แบ่งกลุ่มตามเพศ

3.11 ให้แสดงข้อมูลอายุเฉลี่ย แบ่งกลุ่มตามความพึงพอใจในการทำงาน โดยเรียงจากอายุมากไปน้อย

3.12 ให้แสดงข้อมูลอายุเฉลี่ย แบ่งกลุ่มตามความพึงพอใจในการทำงาน โดยเรียงจากอายุน้อยไปมาก

บทที่ 4
ชนิดข้อมูลที่ไม่ใช่ตัวเลข (Non-Numeric Data Types)
คลาส (Class) และการแปลงชนิดข้อมูล (Coercion)

ในบทนี้จะกล่าวถึงชนิดข้อมูลที่ไม่ใช้ตัวเลข ได้แก่ ตรรกะ (Logical) อักขระ (Character) และแฟกเตอร์ (Factors) ค่าเฉพาะ (Special Values) ได้แก่ ค่าอนันต์ (Infinity) ค่าที่ไม่ใช่ตัวเลข (**NaN**) ค่า **NA** และค่า **NULL** ชนิดข้อมูล แอททริบิวต์ (Attributes) วัตถุ (Objects) และคลาส (Classes) ฟังก์ชัน Is-Dot ในการตรวจสอบวัตถุ และฟังก์ชัน As-Dot ในการแปลงชนิดข้อมูล

4.1 ค่าตรรกะ (Logical Values)

ค่าตรรกะ (Logical Values) เป็นค่าจริงหรือเท็จ อาจจะตีความค่าดังกล่าวเป็น ใช่/ไม่ใช่ (Yes/No) หนึ่ง/ศูนย์ (One/Zero) พอใจ/ไม่พอใจ (Satisfied/Not satisfied) ใน **R** แทนด้วย **TRUE** หรือ **FALSE** (ตัวใหญ่) อาจจะย่อเป็น **T** หรือ **F** (ตัวใหญ่) ได้ดังนี้

```
R> my_logical1 <- TRUE
R> my_logical1
[1] TRUE

R> my_logical2 <- F
R> my_logical2
[1] FALSE

R> my_logical3 <- c(T, F, T, T, T, F, T, F, T, F, F, F)
R> my_logical3
 [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[11] FALSE FALSE

R> my_logical4 <- matrix(my_logical3, nrow = 3, ncol = 4, byrow = my_logical1)
R> my_logical4
     [,1]  [,2]  [,3]  [,4]
[1,] TRUE FALSE  TRUE  TRUE
[2,] TRUE FALSE  TRUE FALSE
[3,] TRUE FALSE FALSE FALSE
```

4.1.1 ตัวดำเนินการเปรียบเทียบ (Comparison Operator) และตัวดำเนินการตรรกะ (Logical Operator)

R เปรียบเทียบค่าต่าง ๆ โดยใช้ตัวดำเนินการเปรียบเทียบ ดังนี้

ตารางที่ 4-1 ตัวดำเนินการเปรียบเทียบ (Comparison Operators)

ตัวดำเนินการ	คำอธิบาย
==	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าเท่ากับ
<=	น้อยกว่าเท่ากับ

โดยทั่วไป ตัวดำเนินการเปรียบเทียบใช้กับข้อมูลตัวเลข เช่น

```
R> 2 == 3
[1] FALSE

R> 2 > 3
[1] FALSE

R> (2-3) <= 1
[1] TRUE

R> 2 != (2+3)
[1] TRUE
```

ตัวดำเนินการเปรียบเทียบประยุกต์ใช้กับเวกเตอร์ได้ โดย **R** จะทำการเปรียบเทียบสมาชิกแต่ละคู่หรือเปรียบเทียบสมาชิกแต่ละตัวกับสเกลาร์ได้ เช่น

```
R> my_vector1 <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
R> my_vector2 <- c(-4, -3, -2, -1, 0, 1, 2, 3, 4, 5)
R> length(my_vector1) == length(my_vector2)
[1] TRUE

R> my_vector1 == my_vector2
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE

R> my_vector1 < my_vector2
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE

R> my_vector1 <= (my_vector2 + 3)
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

R> my_vector2 > 3
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

R ประยุกต์ใช้กับเมทริกซ์และอาร์เรย์ได้ โดย **R** จะทำการเปรียบเทียบสมาชิกแต่ละคู่ หรือเปรียบเทียบสมาชิกแต่ละตัวกับสเกลาร์ได้เช่นเดียวกันเวกเตอร์

R มีฟังก์ชัน any() จะคืนค่า TRUE เมื่อมีสมาชิกตัวใดตัวหนึ่งเป็นจริง และในทางตรงกันข้ามจะคืนค่า FALSE และฟังก์ชัน all() จะคืนค่า TRUE เมื่อมีสมาชิกทุกตัวเป็นจริง และในทางตรงกันข้ามจะคืนค่า FALSE เช่น

```
R> my_vector1 < my_vector2
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
R> my_vector3 <-  my_vector1 < my_vector2

R> any(my_vector3)
[1] TRUE

R> all(my_vector3)
[1] FALSE
```

การเปรียบเทียบเงื่อนไขที่ซับซ้อน (มากกว่า 1 เงื่อนไข) จะใช้ตัวดำเนินการตรรกะช่วย ดังนี้

ตารางที่ 4-2 ตัวดำเนินการตรรกะ (Logical operators)

ตัวดำเนินการ	คำอธิบาย	ผลลัพธ์
&	และ (AND) เปรียบเทียบสมาชิกแต่ละตัว (Element-wise)	TRUE & TRUE ได้ TRUE TRUE & FALSE ได้ FALSE FALSE & TRUE ได้ FALSE FALSE & FALSE ได้ FALSE
&&	และ (AND) เปรียบเทียบสมาชิกตัวแรก (Single comparison)	ผลลัพธ์เหมือน & ข้างบน
|	หรือ (OR) เปรียบเทียบสมาชิกแต่ละตัว (Element-wise)	TRUE | TRUE ได้ TRUE TRUE | FALSE ได้ TRUE FALSE | TRUE ได้ TRUE FALSE | FALSE ได้ FALSE
||	หรือ (OR) เปรียบเทียบสมาชิกตัวแรก (Single comparison)	ผลลัพธ์เหมือน | ข้างบน
!	ไม่ (NOT)	!TRUE ได้ FALSE !FALSE ได้ TRUE

เครื่องหมาย && มีลำดับความสำคัญก่อน || ในการคำนวณจะต้องใส่วงเล็บ เพื่อให้ลำดับการทำงานก่อนหลังถูกต้อง ตัวอย่างเช่น

```
R> (FALSE || TRUE) && FALSE
[1] FALSE

R> !TRUE && FALSE
[1] FALSE

R> !FALSE && TRUE
```

```
[1] TRUE

R> (2 > 1) || (4 != 5)
[1] TRUE
```

เครื่องหมาย **&** และ **|** จะทำการเปรียบเทียบสมาชิกแต่ละตัว แต่เครื่องหมาย **&&** และ **||** จะทำการเปรียบเทียบสมาชิกตัวแรกเท่านั้น ตัวอย่างเช่น

```
R> my_vector1 <- c(T, T, T, T, T, F, F, F, F, F)
R> my_vector2 <- c(F, T, F, T, F, T, F, T, F, T)

R> my_vector1 & my_vector2
 [1] FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

R> my_vector1 | my_vector2
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE

R> my_vector1 && my_vector2
[1] FALSE

R> my_vector1 || my_vector2
[1] TRUE
```

ค่า **TRUE** หรือ **FALSE** แทนด้วย 1 และ 0 ตามลำดับ สามารถคำนวณโดยใช้ตัวดำเนินการทางคณิตศาสตร์ได้ ตัวอย่างเช่น

```
R> TRUE + TRUE
[1]  2

R> FALSE - TRUE
[1] -1

R> T + F + T + F + T + F
[1]  3
```

สามารถคำนวณตัวเลขทางคณิตศาสตร์ โดยใช้ตัวดำเนินการตรรกะได้ ตัวอย่างเช่น

```
R> 1 && 1
[1] TRUE

R> 1 || 0
[1] TRUE

R> 1 && 0
[1] FALSE
```

4.1.2 การดึงข้อมูลโดยค่าตรรกะ (Logical Subsetting and Extraction)

ในการดึงข้อมูล แทนที่จะระบุดัชนีของเวกเตอร์โดยตรง **R** สามารถทำการดึงข้อมูลโดยใช้ค่าตรรกะได้ โดยการสร้างเวกเตอร์ตรรกะ (Logical flag vector) **R** จะสร้างสับเซตหรือดึงข้อมูลเฉพาะค่าที่เป็น **TRUE** เท่านั้น ตัวอย่างเช่น

```
R> my_vector <- c(-4, -3, -2, -1, 0, 1, 2, 3, 4, 5)

R> my_vector[c(T, T, F, F, T, T, F, F, T, T)]
[1] -4 -3  0  1  4  5

R> my_vector > 0
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

R> my_vector[my_vector > 0]
[1] 1 2 3 4 5

R> my_vector[(my_vector > 0) & (my_vector < 4)]
[1] 1 2 3
```

R สามารถแทนที่ข้อมูลโดยการสร้างเวกเตอร์ตรรกะ เช่นเดียวกับการแทนที่ข้อมูลในเวกเตอร์ทั่วไป เช่น

```
R> my_vector[(my_vector < 0)] <- 999
R> my_vector
[1] 999 999 999 999   0   1   2   3   4   5
```

R มีฟังก์ชัน `which()` แสดงตำแหน่งของสมาชิกที่มีค่าเป็นจริงในเวกเตอร์ เช่น

```
R> which(c(T, T, F, F, T))
[1] 1 2 5

R> which(my_vector < 999)
[1]  5  6  7  8  9 10
```

สามารถใช้ฟังก์ชัน `which()` ในการลบสมาชิกที่ไม่ต้องการออกจากเวกเตอร์ได้ เช่น

```
R> my_vector <- my_vector[-which(my_vector >= 999)]
R> my_vector
[1] 0 1 2 3 4 5
```

4.2 อักขระและสตริง (Characters and Strings)

4.2.1 การสร้างสตริง

อักขระเป็นตัวอักษร/หรือสัญลักษณ์เพียงตัวเดียว ส่วนสตริง (Strings) เป็นการนำอักขระ (Characters) แต่ละตัวมาประกอบกัน อักขระและสตริงจะอยู่ภายในเครื่องหมายอัญประกาศ (Double quotation mark, ") เช่น

```
R> my_string <- "Hello World!"
R> my_string
[1] "Hello World!"

R> length(my_string)
[1] 1
```

R จะมองว่าสตริงเป็นค่า 1 ค่า หรือเป็นเวกเตอร์ที่มีความยาวเท่ากับ 1 ในการนับจำนวนตัวอักขระใช้ฟังก์ชัน nchar() เช่น

```
R> nchar(my_string)
[1] 12
```

ตัวเลขที่อยู่ภายในเครื่องหมาย double quote R จะมองว่าเป็นสตริง ไม่สามารถใช้คำนวณทางคณิตศาสตร์ได้ เช่น

```
R> your_string <- "12.34"
R> your_string
[1] "12.34"

R> your_string * 3
Error in your_string * 3 : non-numeric parameter to binary operator
```

สตริงใช้กับตัวดำเนินการเปรียบเทียบและตัวดำเนินการตรรกะได้ เช่น

```
R> "apply" == "apply"
[1] TRUE

R> "apply" != "banana"
[1] TRUE

R> c("apply", "banana", "orange") == "apply"
[1]  TRUE FALSE FALSE
```

R จะเปรียบเทียบตัวอักขระแรกก่อนแล้วค่อยดูตัวอักขระถัดไป R เปรียบเทียบตัวอักขระโดยใช้ลำดับตัวอักขระก่อนหลัง ตัวอักขระที่มาก่อนจะมีค่าน้อยกว่าตัวอักขระที่อยู่ถัดไป เช่น

```
R> "apply" < "banana"
[1] TRUE

R> "apply" < "apricot"
[1] TRUE

R> "orange" > "Apply"
[1] TRUE
```

ตัวอักษรตัวใหญ่มีค่ามากกว่าตัวอักษรตัวเล็ก เช่น

```
R> "Apply" > "apply"
[1] TRUE

R> "banana" > "bAnana"
[1] FALSE
```

อักขระส่วนมากสามารถใช้ได้ เช่น

```
R> my_string <- "&2 _ 4 **%.? $ymbolic ,; "
R> my_string
[1] "&2 _ 4 **%.? $ymbolic ,; "
```

ยกเว้นเครื่องหมาย Backslash, \ โดยทั่วไปเรียกว่าอักขระหลีก (Escape character) ใช้สำหรับควบคุมคำสั่งแสดงผลพิเศษ

4.2.2 การเชื่อมสตริง

R มีฟังก์ชันในการเชื่อมสตริง 2 ตัวคือ `cat()` และ `paste()` โดย `cat()` จะแสดงผลทางจอภาพ (Console) และไม่มีการส่งค่ากลับ (Return) ส่วน `paste()` จะมีการส่งค่ากลับ (Return) เช่น

```
R> my_string <- c("powerful", "R", "is")
R> length(my_string)
[1] 3

R> my_string
[1] "powerful" "R"          "is"

R> cat(my_string[2], my_string[3], "totally", my_string[1], "!")
R is totally powerful !

R> paste(my_string[2], my_string[3], "totally", my_string[1], "!")
[1] "R is totally powerful !"
```

ทั้ง `cat()` และ `paste()` มีพารามิเตอร์ตัวเลือก (Optional parameter) ชื่อ `sep` สำหรับกำหนดตัวคั่น (separation) ระหว่างสตริงได้ เช่น

```
R> paste(my_string[2], my_string[3], "totally", my_string[1], "!", sep = "-")
[1] "R-is-totally-powerful-!"

R> paste(my_string[2], my_string[3], "totally", my_string[1], "!", sep = "")
[1] "Ristotallypowerful!"
```

R สามารถผ่านค่าตัวแปรเข้าไปคำนวณและแสดงผลใน `cat()` และ `paste()` ได้ เช่น

```
R> a <- 5
R> b <- 6.6
R> paste("The value in 'a' is ", a, ".", sep = "")
[1] "The value in 'a' is 5."

R> paste("The value in 'b' is ", b, ".", sep = "")
[1] "The value in 'b' is 6.6."

R> paste("The result of a+b is ", a+b, ".", sep = "")
[1] "The result of a+b is 11.6."
```

4.2.3 อักขระหลีก (Escape Characters)

R มีอักขระหลีกสำหรับควบคุมตำแหน่งการแสดงผล เช่นเดียวกับภาษาโปรแกรมทั่วไป อาทิ ภาษา C Java ดังนี้

ตารางที่ 4-3 อักขระหลีก (Escape characters)

อักขระหลีก	คำอธิบาย
\n	ขึ้นบรรทัดใหม่ (New line)
\t	ไปคอลัมน์ถัดไป (Horizontal tab)
\b	ถอยหลัง 1 ช่องว่าง (Backspace)
\\	ใช้แทนอักขระ \ (Backslash)
\"	ใช้แทนอักขระ " (Double quote)

R สามารถผ่านค่าตัวแปรเข้าไปคำนวณและแสดงผลใน **cat()** และ **paste()** ได้ เช่น

```
R> cat("This is a string\nIt is split\t\tto new\b\n\n\tlines")
This is a string
It is split     to new

   Lines

R> cat("This is a backslash: \\ \n\"This is a quotation.\"")
This is a backslash: \
"This is a quotation."
```

R กำหนดโฟร์เดอร์ (Folder) ที่ใช้งานประจำ ด้วยฟังก์ชัน **setwd()** โดยใช้ Forward slash (**/**) แทนตัวคั่นระหว่างโฟร์เดอร์ เช่น

```
R> setwd("c:/Users/Students/Documents/R/")
```

4.2.4 สับสตริงและแมชชิ่ง (Substrings and Matching)

R มีฟังก์ชัน **substr()** สำหรับดึงอักขระบางส่วนของสตริง โดยกำหนดตำแหน่งเริ่มต้น **start** และตำแหน่งสุดท้าย **stop** เช่น ต้องการดึงตำแหน่งที่ 7 ถึง 12 ออกมาจากสตริง เขียนได้ดังนี้

```
R> my_string <- "Hello world!"
R> substr(my_string, start = 7, stop = 12)
[1] "world!"
```

R กำหนดค่าเข้าไปในฟังก์ชัน **substr()** เพื่อแทนที่สตริงได้ ถ้าสตริงที่ต้องการแทนที่สั้นหรือยาวกว่าตำแหน่งที่ระบุ **R** จะแทนที่เท่าที่สตริงให้มา เช่น ต้องการแทนที่ตำแหน่งที่ 7 ถึง 12 ด้วย "friend" เขียนคำสั่งได้ดังนี้

```
R> substr(my_string, start = 7, stop = 12) <- "friend"
R> my_string
[1] "Hello friend"
```

R ใช้ฟังก์ชัน **sub()** และ **gsub()** เพื่อแทนที่สตริงได้โดยการค้นหาคำที่เหมือนกัน แล้วแทนที่คำดังกล่าวด้วยสตริงที่กำหนด โดย **sub()** จะแทนเฉพาะคำแรกที่พบ ส่วน **gsub()** จะแทนทุกคำที่พบ ตัวอย่างเช่น

```
R> my_string <- "How many world could you live world-wide"
R> sub(my_string, pattern = "world", replacement = "country")
[1] "How many country could you live world-wide"

R> my_string <- "How many world could you live world-wide"
R> gsub(my_string, pattern = "world", replacement = "country")
[1] "How many country could you live country-wide"
```

R มีฟังก์ชัน **grep()** ที่สามารถค้นหาและแทนที่สตริงได้หลากหลายรูปแบบ สามารถค้นคว้าเพิ่มเติมได้จาก help โดยพิมพ์ **?grep** ที่ **R** console

4.3 แฟกเตอร์ (Factors)

แฟกเตอร์มีประโยชน์อย่างมากในการจัดการกับตัวแปรที่ไม่ต่อเนื่องและสามารถแบ่งเป็นกลุ่มได้ (Categorical variables) ตัวอย่างข้อมูลพนักงานที่จะใช้อธิบายต่อไป แสดงได้ดังนี้

Name	Gender	Month of birth
Big	Male (1)	June
Dan	Male (1)	April
Jan	Female (0)	September
June	Female (0)	June
James	Male (1)	July
A	Male (1)	November
B	Female (0)	December
Bird	Male (1)	January

กำหนดค่าเริ่มต้นให้กับ **R** ได้ดังนี้

```
R> first_name <- c("Big", "Dan", "Jan", "June", "James", "A", "B", "Bird")
R> gender_num <- c(1, 1, 0, 0, 1, 1, 0, 1)
R> gender_char <- c("Male", "Male", "Female", "Female", "Male", "Male",
                    "Female", "Male")
```

4.3.1 การสร้างแฟกเตอร์

การสร้างตัวแปรแบบแบ่งกลุ่ม (Categorical variables) ทำได้โดยใช้ฟังก์ชัน **factor()** ดังนี้

```
R> gender_num_fac <- factor(gender_num)
R> gender_num_fac
[1] 1 1 0 0 1 1 0 1
Levels: 0 1

R> gender_char_fac <- factor(gender_char)
R> gender_char_fac
```

```
[1] Male   Male   Female Female Male   Male   Female Male
Levels: Female Male
```

แฟกเตอร์ใน R เก็บข้อมูลแบบแบ่งกลุ่มเป็นระดับชั้น (Levels) สามารถตรวจสอบระดับชั้นโดยใช้ฟังก์ชัน `levels()` เช่น

```
R> levels(gender_num_fac)
[1] "0" "1"

R> levels(gender_char_fac)
[1] "Female" "Male"
```

R สามารถเปลี่ยนแปลงระดับชั้นได้ โดยการกำหนดค่าเวกเตอร์ระดับชั้นใหม่ (Relabel) ให้กับฟังก์ชัน `levels()` เช่น

```
R> levels(gender_num_fac) <- c("1", "2")
R> gender_num_fac
[1] 2 2 1 1 2 2 1 2
Levels: 1 2
```

เวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) สามารถสร้างสับเซตได้เช่นเดียวกับเวกเตอร์ทั่วไป เช่น

```
R> gender_char_fac[2:5]
[1] Male   Female Female Male
Levels: Female Male

R> gender_char_fac[c(1:2, 5, 7)]
[1] Male   Male   Male   Female
Levels: Female Male
```

เวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) สามารถเปรียบเทียบเงื่อนไขได้เช่นเดียวกับเวกเตอร์ทั่วไป เช่น

```
R> gender_num_fac == "2"
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE

R> gender_char_fac == "Male"
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE
```

R ใช้เวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) ช่วยในการดึงข้อมูลตามเงื่อนไขที่ต้องการได้ เช่น

```
R> first_name[gender_char_fac == "Male"]
[1] "Big"   "Dan"   "James" "A"     "Bird"
```

4.3.2 การเรียงลำดับชั้น (Ordering Levels)

ถ้ามีการกำหนดค่าเริ่มต้นเดือนเกิดให้กับ R ดังนี้

```
R> month_of_birth <- c("Jun", "Apr", "Sep", "Jun", "Jul", "Nov", "Dec", "Jan")
```

การเก็บข้อมูลเดือนเกิดในลักษณะนี้มีปัญหาตามมาคือ เดือนเกิดไม่ครอบคลุมทั้ง 12 เดือน และไม่สามารถเปรียบเทียบเดือนเกิดก่อนหลังได้ การกำหนดค่าเริ่มต้นเดือนเกิดจะกำหนดเป็นแฟกเตอร์ให้กับ **R** ได้ดังนี้

```
R> month <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
              "Oct", "Nov", "Dec")
R> month
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

R> month_of_birth_fac <- factor(month_of_birth, levels = month, ordered = TRUE)
R> month_of_birth_fac
[1] Jun Apr Sep Jun Jul Nov Dec Jan
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < ... < Dec
```

หลังจากกำหนดให้เป็นแฟกเตอร์แล้ว **R** สามารถเปรียบเทียบเดือนเกิดได้ดังนี้

```
R> month_of_birth_fac[2]
[1] Apr
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < ... < Dec

R> month_of_birth_fac[3]
[1] Sep
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < ... < Dec

R> month_of_birth_fac[2] < month_of_birth_fac[3]
[1] TRUE
```

4.3.3 การเชื่อมต่อและตัดแฟกเตอร์ (Combining and Cutting)

โดยปกติเวกเตอร์สามารถต่อกันได้ เช่น

```
R> vector1 <- c(1, 2, 3, 4, 5)
R> vector2 <- c(6, 7, 8, 9)
R> c(vector1, vector2)
[1] 1 2 3 4 5 6 7 8 9
```

แต่เวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) ไม่สามารถต่อกันได้โดยตรง ต้องแยกแฟกเตอร์ออกเป็นสมาชิกแต่ละตัวก่อนแล้วค่อยเชื่อมต่อกัน และกำหนดให้เป็นแฟกเตอร์อีกครั้ง สมมติว่ามีการเพิ่มเดือนเกิด ดังนี้

```
R> month_new <- factor(c("Apr", "Jul", "Jul"), levels = month, ordered = TRUE)
R> month_new
[1] Apr Jul Jul
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < ... < Dec
```

ถ้ามีการเชื่อมต่อแฟกเตอร์เดิมและแฟกเตอร์ใหม่ จะได้ผลเป็นเลขจำนวนเต็ม ดังนี้

```
R> c(month_of_birth_fac, month_new)
[1]  6  4  9  6  7 11 12  1  4  7  7
```

R มีฟังก์ชัน `levels()` ในการแสดงระดับชั้นข้อมูล ตัวอย่างเช่น

```
R> levels(month_of_birth_fac)
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

ถ้าใช้เลขจำนวนเต็มแสดงค่าดัชนีประกอบกับฟังก์ชัน `levels()` เพื่อแสดงระดับชั้นข้อมูล จะได้เวกเตอร์ที่ต้องการเชื่อมต่อ ดังนี้

```
R> levels(month_of_birth_fac)[c(month_of_birth_fac, month_new)]
 [1] "Jun" "Apr" "Sep" "Jun" "Jul" "Nov" "Dec" "Jan" "Apr" "Jul" "Jul"
```

หลังจากนั้นกำหนดค่าเวกเตอร์ดังกล่าวให้กับตัวแปรใหม่ แล้วค่อยกำหนดให้เป็นแฟกเตอร์อีกครั้ง ดังนี้

```
R> month_final <- levels(month_of_birth_fac)[c(month_of_birth_fac, month_new)]
R> month_final_fac <- factor(month_final, levels = month, ordered = TRUE)
R> month_final_fac
 [1] Jun Apr Sep Jun Jul Nov Dec Jan Apr Jul Jul
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < ... < Dec
```

R สามารถแบ่งข้อมูลเลขต่อเนื่องออกเป็นช่วง ๆ เช่น เล็ก/กลาง/ใหญ่ หรือสูง/ต่ำ โดยใช้ฟังก์ชัน `cut()` โดยสมมติข้อมูลเริ่มต้น ดังนี้

```
R> my_data <- c(0.5, 1.4, 5.3, 2.2, 5.5, 4.1, 0.7, 3.4, 4.2, 1.1)
```

ถ้าต้องการแบ่งข้อมูลออกเป็นเล็ก/กลาง/ใหญ่ โดยกลุ่มเล็กอยู่ในช่วง `[0, 2)` หรือ (0 <= x < 2) กลุ่มกลางอยู่ในช่วง `[2, 4)` หรือ (2 <= x < 4) กลุ่มใหญ่อยู่ในช่วง `[4, 6]` หรือ (4 <= x <= 6) โดยเครื่องหมาย `[` หรือ `]` จะรวมค่าในตำแหน่งนั้นด้วย ส่วนเครื่องหมาย `(` หรือ `)` ไม่รวมค่าในตำแหน่งนั้น คำสั่งต่อไปนี้ จะได้ค่าชั้นเล็ก `(0, 2]` ซึ่งยังไม่ตรงกับความต้องการ

```
R> break_value <- c(0, 2, 4, 6)
R> cut(my_data, breaks = break_value)
 [1] (0,2] (0,2] (4,6] (2,4] (4,6] (4,6] (0,2] (2,4] (4,6] (0,2]
Levels: (0,2] (2,4] (4,6]
```

แก้ไขโดยการกำหนดพารามิเตอร์ `right = FALSE` เป็นการสั่งให้ช่วงข้อมูลที่กำหนดทางขวาเป็นช่วงเปิด กล่าวอีกอย่างหนึ่งคือ ไม่รวมตำแหน่งทางขวา ดังนี้

```
R> cut(my_data, breaks = break_value, right = FALSE)
 [1] [0,2) [0,2) [4,6) [2,4) [4,6) [4,6) [0,2) [2,4) [4,6) [0,2)
Levels: [0,2) [2,4) [4,6)
```

แต่ยังมีปัญหาคือ ชั้นข้อมูลสุดท้ายยังไม่รวมตำแหน่งขวา แก้ไขโดยการกำหนดพารามิเตอร์ `include.lowest = TRUE` ดังนี้

```
R> cut(my_data, breaks = break_value, right = FALSE, include.lowest = TRUE)
 [1] [0,2) [0,2) [4,6] [2,4) [4,6] [4,6] [0,2) [2,4) [4,6] [0,2)
Levels: [0,2) [2,4) [4,6]
```

R สามารถกำหนดชื่อระดับชั้น (Labels) โดยการกำหนดเวกเตอร์เพิ่มเข้าไปในพารามิเตอร์ **labels** ดังนี้

```
R> label_value <- c("Small", "Medium", "Large")
R> cut(my_data, breaks = break_value, right = FALSE, include.lowest = TRUE,
       Labels = label_value)
 [1] Small  Small  Large  Medium Large  Large  Small  Medium Large Small
Levels: Small Medium Large
```

4.4 ค่าเฉพาะ (Special Values)

R มีการเก็บค่าบางอย่างเป็นค่าเฉพาะ เช่น เมื่อมีข้อมูลในเดต้าเฟรมหายไป (Missing Value) ค่าที่ไม่ใช้ตัวเลข หรือค่าอนันต์ (Infinity) ดังต่อไปนี้

4.4.1 ค่าอนันต์ (Infinity)

เมื่อค่าตัวเลขมีค่าสูงมาก ๆ R จะเก็บค่าดังกล่าวไว้ในวัตถุ (Object) พิเศษชื่อ **Inf** ซึ่งใช้ได้กับเวกเตอร์ที่เป็นตัวเลขเท่านั้น R สร้างตัวแปรสำหรับเก็บค่า Inf ได้ดังนี้

```
R> my_inf <- Inf
R> my_inf
[1] Inf

R> my_vector <- c(123, Inf, 4.5, -6, 7.89, Inf)
R> my_vector
[1] 123.00    Inf   4.50  -6.00   7.89    Inf

R> my_value <- 999999^100
R> my_value
[1] Inf
```

คำสั่งข้างต้นเป็นการกำหนดวัตถุ **Inf** ชื่อ my_inf กำหนดเวกเตอร์ชื่อ my_vector ซึ่งประกอบด้วยตัวเลขและค่า **Inf** คำสั่งสุดท้ายตัวเลขค่าสูงมาก ๆ (999999 ยกกำลัง 100) ถูกเก็บไว้ในวัตถุ **Inf** ชื่อ my_value

R เก็บค่าลบอนันต์ (Negative Infinity) ไว้ในวัตถุ (Object) พิเศษชื่อ **-Inf** เช่น

```
R> my_vector2 <- c(-34, 56, -Inf, Inf, -Inf, -789)
R> my_vector2
[1]  -34   56 -Inf  Inf -Inf -789
```

Inf และ **-Inf** ทำงานกับตัวดำเนินการทางคณิตศาสตร์ได้ เช่น เมื่อ **Inf** คูณค่าติดลบจะได้ **-Inf** การบวก/ลบ/หรือคูณกับ **Inf** จะได้ **Inf** ดังนี้

```
R> Inf * -3
[1] -Inf

R> Inf + 3
[1] Inf
```

```
R> 5 * -Inf
[1] -Inf

R> -77.7 * -Inf
[1] Inf

R> Inf - 888.8
[1] Inf

R> Inf + Inf
[1] Inf

R> Inf / 9
[1] Inf
```

เมื่อหารตัวเลขใด ๆ ด้วย `Inf` จะได้ค่าศูนย์ ดังนี้

```
R> 88 / Inf
[1] 0

R> -77 / Inf
[1] 0
```

ค่าตัวเลขที่ไม่ใช่ศูนย์เมื่อหารด้วยศูนย์ จะได้ `Inf` หรือ `-Inf` แล้วแต่เครื่องหมายบวก/ลบ เช่น

```
R> 88 / 0
[1] Inf

R> -77 / 0
[1] -Inf

R> Inf / 0
[1] Inf
```

ถ้าต้องการตรวจสอบว่าเป็นตัวเลขอนันต์ หรือไม่เป็นตัวเลขอนันต์ ผู้ใช้สามารถใช้ฟังก์ชัน `is.infinite()` และ `is.finite()` ตามลำดับ เช่น

```
R> my_vector2
[1]  -34   56 -Inf  Inf -Inf -789

R> is.infinite(my_vector2)
[1] FALSE FALSE  TRUE  TRUE  TRUE FALSE

R> is.finite(my_vector2)
[1]  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

`Inf` และ `-Inf` สามารถเปรียบเทียบกันด้วยตัวดำเนินการเปรียบเทียบได้ เช่น

```
R> -Inf < Inf
[1] TRUE

R> Inf < Inf
[1] FALSE
```

```
R> my_vector2 == Inf
[1] FALSE FALSE FALSE  TRUE FALSE FALSE

R> my_vector2 == -Inf
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE
```

4.4.2 ค่าที่ไม่ใช่ตัวเลข (NaN)

บางครั้งผลการคำนวณไม่สามารถระบุเป็นตัวเลข **Inf** หรือ **-Inf** ได้ **R** มีวิธีการกำหนดค่าดังกล่าวโดยใช้ค่าที่ชื่อว่า **NaN** ย่อมาจากค่าที่ไม่ใช่ตัวเลข (Not a Number) เช่น

```
R> my_NaN <- NaN
R> my_NaN
[1] NaN

R> my_vector <- c(NaN, 23.4, -5, NaN, 6666.66, Inf, -Inf)
R> my_vector
[1]     NaN   23.40   -5.00     NaN 6666.66     Inf    -Inf
```

NaN ใช้แสดงผลการคำนวณที่ไม่สามารถหาค่าที่เป็นตัวเลขได้ เช่น

```
R> Inf / Inf
[1] NaN

R> -Inf + Inf
[1] NaN

R> 0 / 0
[1] NaN
```

ตัวดำเนินการทางคณิตศาสตร์ที่เกี่ยวข้องกับ **NaN** จะได้ผลลัพธ์เป็น **NaN** เช่น

```
R> NaN + 5
[1] NaN

R> 3+5*(2-2)/0
[1] NaN

R> 2.5^(-Inf/Inf)
[1] NaN
```

คำสั่งแรกเป็นการบวก 5 กับค่าที่ไม่ใช่ตัวเลข (Not a Number) ผลยังคงเป็น **NaN** คำสั่งที่สอง (2-2)/0 เป็น 0/0 ได้ผลเป็น **NaN** และคำสั่งสุดท้าย **-Inf/inf** ได้ผลเป็น **NaN**

ในการตรวจสอบว่าเป็นค่าที่ไม่ใช่ตัวเลข (Not a Number) หรือไม่ **R** ใช้ฟังก์ชัน **is.nan()** เช่น

```
R> my_vector
[1]     NaN   23.40   -5.00     NaN 6666.66     Inf    -Inf

R> is.nan(my_vector)
[1]  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE

R> !is.nan(my_vector)
```

```
[1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE

R> !is.nan(my_vector) | is.infinite(my_vector)
[1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE

R> my_vector[-which(is.nan(my_vector))]
[1]   23.40   -5.00 6666.66     Inf    -Inf
```

คำสั่งแรกแสดงค่าเวกเตอร์ my_vector คำสั่งที่สองตรวจสอบว่าสมาชิกแต่ละตัวในเวกเตอร์ my_vector ว่ามีค่าที่ไม่ใช่ตัวเลข (Not a Number) หรือไม่ คำสั่งที่สามเป็นนิเสธ (Negative) ของคำสั่งที่สอง คำสั่งที่สี่เป็นการตรวจสอบว่าสมาชิกแต่ละตัวในเวกเตอร์ my_vector ว่ามีค่าที่ไม่ใช่ตัวเลข (Not a Number) หรือเป็นค่าอนันต์ (Infinity) หรือไม่ คำสั่งสุดท้ายเป็นการละ (Omiting) ค่าที่ไม่ใช่ตัวเลข (Not a Number)

4.4.3 ค่า NA

ในการเก็บข้อมูล บางครั้งมีข้อมูลที่หายไป (Missing Values) เนื่องจากหลายสาเหตุ เช่น ผู้ตอบแบบสำรวจไม่ให้ข้อมูล หรือไม่สามารถหาข้อมูลที่ต้องการได้ ใน **R** มีค่าที่เรียกว่า **NA** ย่อมาจาก Not Available หมายความว่า ไม่มีข้อมูลที่ตำแหน่งดังกล่าวในเวกเตอร์นั้น เช่น

```
R> my_vector1 <- c("apple", "banana", NA, "orange", NA)
R> my_vector1
[1] "apple"  "banana" NA       "orange" NA

R> my_vector2 <- factor(c("blue", NA, NA, "blue", NA, "green", NA, "red", "red"))
R> my_vector2
[1] blue  <NA>  <NA>  blue  <NA>  green <NA>  red   red
Levels: blue green red
```

R ตรวจสอบว่าไม่มีข้อมูล (**NA**) และเป็นค่าที่ไม่ใช่ตัวเลข (**NaN**) โดยใช้ฟังก์ชัน `is.na()` มีข้อสังเกตว่า `is.na()` จะแสดงทั้ง **NA** และ **NaN** ทั้งนี้เพราะในทางปฏิบัติค่าทั้งสองแบบไม่สามารถนำมาคำนวณต่อไปได้ เช่น

```
R> my_vector3 <- c(NA, 2, Inf, 4, NA, 6, NaN, 8, NaN, 10)
R> my_vector3
[1]  NA   2 Inf   4  NA   6 NaN   8 NaN  10

R> is.na(my_vector3)
[1]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE

R> which(is.na(my_vector3))
[1] 1 5 7 9

R> which(is.nan(my_vector3))
[1] 7 9
```

ในการดึงค่าที่ไม่มีข้อมูล (**NA**) และเป็นค่าที่ไม่ใช่ตัวเลข (**NaN**) ออกจากเวกเตอร์ที่กำหนด นอกจากการสร้างสับเซตโดยใช้ค่าดัชนีติดลบ (Negative indexes) แล้ว ยังสามารถทำได้โดยใช้ฟังก์ชัน **na.omit()** เช่น

```
R> my_vector4 <- na.omit(my_vector3)
#  equivalent to: my_vector4 <- my_vector3[-which(is.na(my_vector3))]
R> my_vector4
[1]   2 Inf   4   6   8  10

attr(, "na.action")
[1] 1 5 7 9

attr(, "class")
[1] "omit"
```

เช่นเดียวกับ **NaN** ตัวดำเนินการทางคณิตศาสตร์ที่เกี่ยวข้องกับ **NA** และ **NaN** จะได้ผลลัพธ์เป็น **NA** เช่น

```
R> 1 + 2.3*NA - 5
[1] NA

R> 2 * c(1, 2, 3, NA, NaN, NA, 7, NaN)
[1]   2   4   6  NA NaN  NA  14 NaN

R> NA > 10
[1] NA

R> 15 > NaN
[1] NA
```

4.4.4 ค่า NULL

R ใช้ตัวแปรพิเศษชื่อว่า **NULL** ในการนิยามว่าไม่มีข้อมูลใด ๆ หรือว่าง (Empty) ในขณะที่ **NA** หมายถึง ไม่มีข้อมูลที่ตำแหน่งดังกล่าวในเวกเตอร์ แต่ **NA** สามารถแก้ไข/เพิ่มเติมข้อมูลในตำแหน่งนั้นได้ ดังตัวอย่างต่อไปนี้

```
R> my_null <- NULL
R> my_null
NULL

R> my_NA <- NA
R> my_NA
[1] NA

R> c(1, 2, NA, 4)
[1]  1  2 NA  4

R> c(1, 2, NULL, 4)
[1] 1 2 4
```

คำสั่งที่สี่ (R> my_NA) จะเห็นว่าดัชนีที่ [1] แสดงค่า NA ส่วนคำสั่งที่ห้า (R> c(1, 2, NA, 4)) จะเห็นว่าดัชนีที่ [3] จะแสดงค่า NA ในขณะที่ NULL จะไม่ถูกแสดงเลย ในคำสั่งสุดท้ายจะแสดงเฉพาะดัชนีที่ [1] [2] และ [3] มีค่าเท่ากับ 1 2 และ 4 โดยไม่มี NULL อยู่

พิจารณาเวกเตอร์ NA และเวกเตอร์ NULL ดังต่อไปนี้

```
R> c(NA, NA, NA, NA)
[1] NA NA NA NA

R> c(NULL, NULL, NULL, NULL)
NULL
```

คำสั่งแรกหมายถึงมีที่เก็บข้อมูลสี่ช่อง (Slots) แต่ไม่มีข้อมูลอยู่ ส่วนคำสั่งที่สองเป็นการระบุว่าไม่มีข้อมูล (Empty) สี่ครั้ง ผลก็คือไม่มีวัตถุ (Object) ใด ๆ เกินขึ้น ซึ่งแสดงด้วย NULL

ในการตรวจสอบว่าไม่มีข้อมูล (Empty) หรือไม่มีวัตถุใด ๆ ใช้ฟังก์ชัน is.null() เช่น

```
R> parameter1 <- c("var1", "var2", "var3")
R> is.null(parameter1)
[1] FALSE

R> is.na(parameter1)
[1] FALSE FALSE FALSE
```

คำสั่งแรกเป็นการกำหนดเวกเตอร์พารามิเตอร์ชื่อ parameter1 ในการตรวจสอบว่าไม่มีข้อมูล (Empty) ใช้ฟังก์ชัน is.null() ในคำสั่งที่สอง แต่ถ้าใช้ is.na() ในคำสั่งที่สามจะเป็นการตรวจสอบสมาชิกแต่ละตัวแทนการตรวจสอบสมาชิกทุกตัว

ถ้าต้องการตรวจสอบว่าพารามิเตอร์ในเวกเตอร์ว่างหรือไม่ ผู้ใช้ใช้ฟังก์ชัน is.null() ในขณะที่ฟังก์ชัน is.na() เป็นการตรวจสอบสมาชิกแต่ละตัวแทนการตรวจสอบสมาชิกทุกตัว เช่น

```
R> parameter2 <- c(NA, NA, NA)
R> is.na(parameter2)
[1] TRUE TRUE TRUE

R> is.null(parameter2)
[1] FALSE

R> parameter3 <- c(NULL, NULL, NULL)
R> is.null(parameter3)
[1] TRUE
```

การเข้าถึงสมาชิกที่ไม่มีในลิสต์จะได้ NULL เช่น

```
R> my_list <- list(member1 = c(1, 2, 3, 4), member2 = "Hello")
R> my_list
$member1
[1] 1 2 3 4

$member2
[1] "Hello"
```

```
R> my_list$member3
NULL
```

4.5 ทำความเข้าใจชนิดข้อมูล (Types) คลาส (Classes) และการแปลงชนิดข้อมูล (Coercion)

4.5.1 แอททริบิวต์ (Attributes)

วัตถุใน **R** มีคุณสมบัติของวัตถุแต่ละตัวอยู่เรียกว่า แอททริบิวต์ (Attributes) ตัวอย่างจากบทที่ผ่านมา คุณสมบัติของเมทริกซ์คือ ขนาด เรียกดูโดยใช้ฟังก์ชัน `dim()` และใช้ `levels()` ในการเรียกดูคุณสมบัติของแฟกเตอร์ คุณสมบัติบางอย่างไม่ได้แสดงไว้โดยตรง ในการเรียกดู **R** ใช้ฟังก์ชัน `attributes()` เช่น

```
R> my_matrix1 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
R> my_matrix1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

R> attributes(my_matrix1)
$dim
[1] 3 3
```

R แสดงแอททริบิวต์ได้โดยใช้ชื่อแอททริบิวต์หลังเครื่องหมาย `$` หรือใช้ฟังก์ชัน `attr()` พร้อมกับระบุชื่อแอททริบิวต์ให้พารามิเตอร์ `which` หรือเรียกใช้ชื่อฟังก์ชันที่มีชื่ออยู่โดยตรง เช่น

```
R> attributes(my_matrix1)$dim
[1] 3 3

R> attr(my_matrix1, which="dim")
[1] 3 3

R> dim(my_matrix1)
[1] 3 3
```

พารามิเตอร์บางตัวเป็นทางเลือกให้ผู้ใช้ระบุค่าเข้าไป (Optional Parameters) ถ้าไม่ระบุค่าจะหมายถึง `NULL` เช่น ในเมทริกซ์มีพารามิเตอร์ชื่อ dimnames ใช้แสดงชื่อแถวและคอลัมน์ เช่น

```
R> my_matrix2 <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE,
                        dimnames = list(c("Row1", "Row2", "Row3"),
                                        c("A", "B", "C")))
R> my_matrix2
     A B C
Row1 1 2 3
Row2 4 5 6
Row3 7 8 9

R> attributes(my_matrix2)
$dim
```

```
[1] 3 3

$dimnames
$dimnames[[1]]
[1] "Row1" "Row2" "Row3"

$dimnames[[2]]
[1] "A" "B" "C"
```

แอททริบิวต์ dimnames เป็นลิสต์ที่ซ้อนอยู่ภายในเมทริกซ์ **R** สามารถเข้าถึงได้หลายวิธี เช่น อ้างถึงสมาชิกของลิสต์โดยใช้ฟังก์ชัน **attr()** หรือใช้ฟังก์ชัน **dimnames()** เช่น

```
R> dimnames(my_matrix2)
[[1]]
[1] "Row1" "Row2" "Row3"

[[2]]
[1] "A" "B" "C"
```

แอททริบิวต์บางตัวสามารถเปลี่ยนแปลง/แก้ไขได้ โดยการกำหนดค่าเข้าไปใหม่ เช่น

```
R> dimnames(my_matrix2) <- list(c("R1", "R2", "R3"), c("C1", "C2", "C3"))
R> my_matrix2
   C1 C2 C3
R1  1  2  3
R2  4  5  6
R3  7  8  9
```

คำสั่งข้างต้นเป็นการกำหนดชื่อแถวและคอลัมน์ให้กับ my_matrix2 โดยการผ่านชื่อแถวและคอลัมน์เข้ามาในลิสต์แล้วกำหนดค่าดังกล่าวเข้าไปในฟังก์ชัน **dimnames()**

4.5.2 วัตถุ (Objects) และคลาส (Classes)

R เป็นโปรแกรมเชิงวัตถุ (Object-oriented programming language) หมายความว่า ตัวแปร/ข้อมูลจะถูกเก็บไว้ในรูปวัตถุ (Objects) และวัตถุแต่ละตัวก็จะมีหน้าที่หรือฟังก์ชันแตกต่างกันออกไป วัตถุพื้นฐานใน **R** ได้แก่ เวกเตอร์ เมทริกซ์ และอาร์เรย์

4.5.2.1 เวกเตอร์เดี่ยว (Stand-Alone Vectors)

ขั้นแรกเป็นการสร้างเวกเตอร์ขึ้นมาเดี่ยว ๆ ดังนี้

```
R> num_vector1 <-  1:6
R> num_vector1
[1] 1 2 3 4 5 6

R> num_vector2 <- seq(from = 1, to = 5, length = 6)
R> num_vector2
[1] 1.0 1.8 2.6 3.4 4.2 5.0

R> char_vector <- c("Hello", "world")
```

```
R> char_vector
[1] "Hello" "world"

R> logic_vector <- c(T, T, T, F, F, F)
R> logic_vector
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE

R> factor_vector <- factor(c("Blue", "Green", "Green", "Yellow", "Red", "Red"))
R> factor_vector
[1] Blue   Green  Green  Yellow Red    Red
Levels: Blue Green Red Yellow
```

ในการหาว่าสมาชิกในวัตถุเป็นคลาสอะไร ใช้ฟังก์ชัน `class()` เช่น

```
R> class(num_vector1)
[1] "integer"

R> class(num_vector2)
[1] "numeric"

R> class(char_vector)
[1] "character"

R> class(logic_vector)
[1] "logical"

R> class(factor_vector)
[1] "factor"
```

ในกรณีที่เป็นเวกเตอร์เดี่ยว ๆ (Stand alone vectors) **R** จะแสดงคลาสตามชนิดข้อมูลในเวกเตอร์ คำสั่งแรกแสดงว่าสมาชิกในเวกเตอร์ my_vector1 เป็นคลาสจำนวนเต็ม (Integer) คำสั่งที่สองแสดงว่าสมาชิกในเวกเตอร์ my_vector2 เป็นคลาสตัวเลขที่มีจุดทศนิยม (Numeric) บางครั้งภาษาคอมพิวเตอร์เรียกว่า Floating-point numbers ในการตรวจสอบว่าเป็นตัวเลขหรือไม่ **R** ใช้ฟังก์ชัน `is.numeric()` ซึ่งจะเป็นจริงก็ต่อเมื่อเป็นเลขจำนวนเต็มหรือเลขทศนิยม คำสั่งที่สามแสดงว่าสมาชิกในเวกเตอร์ char_vector เป็นคลาสอักขระ (Character) คำสั่งที่สี่แสดงว่าสมาชิกในเวกเตอร์ logic_vector เป็นคลาสตรรกะ (Logical) และคำสั่งสุดท้ายแสดงว่าสมาชิกในเวกเตอร์ factor_vector เป็นคลาสแฟกเตอร์ (Factor)

4.5.2.2 โครงสร้างข้อมูลแบบอื่น (Other Data Structures)

โดยปกติ **R** จะแสดงคลาสตามโครงสร้างข้อมูล (Data Structures) แทนที่จะแสดงตามชนิดของข้อมูล ยกเว้นถ้าเป็นเวกเตอร์เดี่ยว ๆ **R** จะแสดงคลาสตามชนิดของข้อมูล (ดูหัวข้อก่อนหน้านี้) เช่น

```
R> number_matrix1 <- matrix(num_vector1, nrow = 2, ncol = 3, byrow = TRUE)
R> number_matrix1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6

R> class(number_matrix1)
```

```
[1] "matrix"
```

4.5.2.3 วัตถุที่มีหลายคลาส (Multiple Classes)

วัตถุบางประเภทมีหลายคลาสอยู่ในวัตถุเดียวกัน เช่น แฟกเตอร์ที่มีลำดับชั้น (Ordered factor vectors) เป็นวัตถุที่สืบทอดคุณสมบัติมาจากคลาสแฟกเตอร์ (Factor class) และคลาสลำดับ (Ordered class) ดังนั้นจึงมีทั้งคลาสแฟกเตอร์และคลาสลำดับ ดังนี้

```
R> ordered_vector <- factor(c("Low", "Low", "High", "Medium", "High"),
                            levels = c("Low", "Medium", "High"),
                            ordered = TRUE)
R> ordered_vector
[1] Low    Low    High   Medium   High
Levels: Low < Medium < High

R> class(ordered_vector)
[1] "ordered" "factor"
```

4.5.3 ฟังก์ชัน Is-Dot ในการตรวจสอบวัตถุ

ในการตรวจสอบวัตถุว่าเป็นคลาสหรือประเภทข้อมูลที่ต้องการหรือไม่ R ใช้ฟังก์ชันในกลุ่มที่เรียกว่า is-dot เช่น

```
R> num_vector <- 1:6
R> num_vector
[1] 1 2 3 4 5 6

R> is.integer(num_vector)
[1] TRUE

R> is.numeric(num_vector)
[1] TRUE

R> is.matrix(num_vector)
[1] FALSE

R> is.data.frame(num_vector)
[1] FALSE

R> is.list(num_vector)
[1] FALSE

R> is.vector(num_vector)
[1] TRUE

R> is.logical(num_vector)
[1] FALSE
```

ฟังก์ชัน `is.integer()` `is.numeric()` และ `is.logical()` ใช้ตรวจสอบชนิดข้อมูลในวัตถุ ส่วนฟังก์ชัน `is.matrix()` `is.data.frame()` `is.list()` `is.vector()` ใช้ตรวจสอบชนิดของ

โครงสร้างข้อมูล ผลลัพธ์แสดงว่า num_vector เป็นเลขจำนวนเต็ม (Integer) เป็นตัวเลข (Numeric) และเป็นเวกเตอร์ (Vecter) ไม่ใช่เมทริกซ์ (Matrix) เดต้าเฟรม (Dataframe) ลิสต์ (List) หรือตรรกะ (Logical)

4.5.4 ฟังก์ชัน As-Dot ในการแปลงชนิดข้อมูล

R มีการแปลงชนิดข้อมูลโดยอัตโนมัติ กล่าวคือมีการแปลงชนิดข้อมูลภายในคำสั่งหรือนิพจน์ (Implicit coercion) โดยที่ผู้ใช้ไม่ต้องสั่งให้มีการแปลงชนิดข้อมูล ตัวอย่างเช่น แปลงจากข้อมูลตรรกะไปเป็นตัวเลข โดยเลข 1 ใช้แทนค่าจริง (**TRUE**) และเลข 0 ใช้แทนค่าเท็จ (**FALSE**) ดังนี้

```
R> c(1, 2, 3, 4) + c(T, T, F, F)
[1] 2 3 3 4
```

คำสั่งข้างบนเป็นการบวกกันของเวกเตอร์จำนวนเต็มกับเวกเตอร์ตรรกะ **R** จะทำการแปลงข้อมูลตรรกะให้เป็นตัวเลขจำนวนเต็ม 0/1 และทำการบวกกัน

การเชื่อมต่อสตริงโดยใช้ฟังก์ชัน **paste()** และ **cat()** **R** จะทำการแปลงตัวเลขให้เป็นสตริงก่อนที่จะนำมาเชื่อมต่อกัน เช่น

```
R> value1 <- 12
R> value2 <- F
R> cat("Hello world: ", value1, "; Good bye: ", value2, sep = "")
Hello world: 12; Good bye: FALSE
```

บางครั้งผู้ใช้ต้องการสั่งให้มีการแปลงชนิดข้อมูลด้วยตัวเอง (Explicit coercion) **R** มีฟังก์ชัน as-Dot ใช้กับชนิดข้อมูลพื้นฐานแทบทุกประเภท เช่น

```
R> as.numeric(c(T, T, F, F, F))
[1] 1 1 0 0 0

R> 1:5 + as.numeric(c(T, T, F, F, F))
[1] 2 3 3 4 5

R> my_value <- 67
R> my_char <- as.character(my_value)
R> my_char
[1] "67"

R> my_logic <- F
R> my_logic_char <- as.character(my_logic)
R> my_logic_char
[1] "FALSE"
```

R แปลงสตริงเป็นตัวเลขได้ เมื่อสตริงดังกล่าวมีความหมาย หรืออ่านเป็นตัวเลขได้ เช่น

```
R> as.numeric("12.34")
[1] 12.34

R> as.numeric("This string 12.34 cannot be converted")
[1] NA
Warning message:
```

```
NAs introduced by coercion
```

สำหรับแฟกเตอร์ **R** จะแปลงระดับ (Levels) เป็นเลขจำนวนเต็มเริ่มจาก 1, 2, ... ตามลำดับ เช่น

```
R>ordered_vector <- factor(c("Low", "Low", "High", "Medium", "High"),
                           levels = c("Low", "Medium", "High"),
                           ordered = TRUE)
R> ordered_vector
[1] Low    Low    High   Medium High
Levels: Low < Medium < High

R> as.numeric(ordered_vector)
[1] 1 1 3 2 3
```

คำสั่งข้างบนจะแปลงระดับ Low Medium และ High เป็นตัวเลข 1, 2, 3 ตามลำดับ การแปลงระหว่างโครงสร้างวัตถุแต่ละแบบ มีประโยชน์อย่างมาก เช่น เมื่อต้องการแปลงเมทริกซ์ไปเป็นเวกเตอร์

```
R> my_matrix <- matrix(1:6, nrow = 2, ncol = 3)
R> my_matrix
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

R> as.vector(my_matrix)
[1] 1 2 3 4 5 6
```

เมื่อต้องการแปลงลิสต์ไปเป็นเดต้าเฟรม เช่น

```
R> my_list <- list(var1 = 1:4, var2 = c(T, T, F, F),
                   var3 = factor(c(4, 4, 1, 2)))
R> my_list
$var1
[1] 1 2 3 4

$var2
[1]  TRUE  TRUE FALSE FALSE

$var3
[1] 4 4 1 2
Levels: 1 2 4

R> as.data.frame(my_list)
  var1  var2 var3
1    1  TRUE    4
2    2  TRUE    4
3    3 FALSE    1
4    4 FALSE    2
```

4.6 สรุปท้ายบท

บทนี้กล่าวถึงชนิดข้อมูลที่ไม่ใช่ตัวเลข (Non-numeric data types) ได้แก่ (1) ตรรกะ (Logical) (2) อักขระ (Character) และ (3) แฟกเตอร์ (Factors) สำหรับค่าตรรกะ (Logical values) เป็นค่าจริงหรือเท็จ

อาจจะตีความเป็นใช่/ไม่ใช่ (Yes/No) หนึ่ง/ศูนย์ (One/Zero) การเปรียบเทียบใช้ตัวดำเนินการเปรียบเทียบ ได้แก่ `==, !=, >, <, >=, <=` การเปรียบเทียบเงื่อนไขที่ซับซ้อน (มากกว่า 1 เงื่อนไข) ใช้ตัวดำเนินการตรรกะช่วยได้แก่ `&, &&, |, ||, !` การสร้างอักขระและสตริง (Characters and strings) การเชื่อมสตริง อธิบายอักขระหลีก (Escape characters) การสับสตริงและแมชชิ่ง (Substrings and matching) การสร้างแฟกเตอร์ การเรียงลำดับชั้น (Ordering levels) การเชื่อมต่อและแบ่งแฟกเตอร์ (Combining and cutting)

การเก็บค่าเฉพาะบางอย่าง ได้แก่ (1) ค่าอนันต์ (Infinity) (2) ค่าที่ไม่ใช่ตัวเลข (`NaN`) (3) ค่า `NA` และ (4) ค่า `NULL` เมื่อค่าตัวเลขมีค่าสูงหรือต่ำมาก ๆ `R` จะเก็บค่าดังกล่าวไว้ในวัตถุ (Object) พิเศษชื่อ `Inf` และ `-Inf` ตามลำดับ บางครั้งผลการคำนวณไม่สามารถระบุเป็นตัวเลข `Inf` หรือ `-Inf` ได้ `R` มีวิธีการกำหนดค่าดังกล่าวโดยใช้ชื่อว่า `NaN` ย่อมาจากค่าที่ไม่ใช่ตัวเลข (Not a Number) `NaN` ใช้แสดงผลการคำนวณที่ไม่สามารถหาค่าที่เป็นตัวเลขได้

`R` มีค่าที่เรียกว่า `NA` ใช้จัดการกับข้อมูลข้อมูลที่หายไป (Missing values) `NA` ย่อมาจาก Not available หมายความว่าไม่มีข้อมูลที่ตำแหน่งดังกล่าว ในขณะเดียวกัน `R` ใช้ตัวแปรพิเศษชื่อว่า `NULL` ในการนิยามว่าไม่มีข้อมูลใด ๆ หรือว่าง (Empty) ในขณะที่ `NA` หมายถึงไม่มีข้อมูลที่ตำแหน่งดังกล่าวในเวกเตอร์ แต่สามารถแก้ไข/เพิ่มเติมข้อมูลในตำแหน่งนั้นได้

วัตถุ (Objects) ใน `R` แต่ละตัวมีคุณสมบัติเรียกว่าแอททริบิวต์ (Attributes) วัตถุพื้นฐานใน `R` ประกอบด้วย เวกเตอร์ เมทริกซ์ และอาร์เรย์ `R` จะแสดงคลาส (Class) ตามโครงสร้างข้อมูลแทนที่จะแสดงตามชนิดของข้อมูล ยกเว้นถ้าเป็นเวกเตอร์เดี่ยว `R` จะแสดงคลาสตามชนิดของข้อมูล วัตถุบางประเภทจะมีหลายคลาสอยู่ในวัตถุเดียวกัน นอกจากนี้มีฟังก์ชัน Is-Dot ใช้ในการตรวจสอบวัตถุ ส่วนฟังก์ชัน As-Dot ใช้ในการแปลงชนิดข้อมูล

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	**คำอธิบาย**
`TRUE, T`	จริง
`FALSE, F`	เท็จ
`==`	เท่ากับ
`!=`	ไม่เท่ากับ
`>`	มากกว่า
`<`	น้อยกว่า
`>=`	มากกว่าเท่ากับ
`<=`	น้อยกว่าเท่ากับ
`any()`	คืนค่า `TRUE` เมื่อมีสมาชิกตัวใดตัวหนึ่งเป็นจริง และในทางตรงกันข้ามจะคืนค่า `FALSE`

คำสั่ง	คำอธิบาย
`all()`	คืนค่า **TRUE** เมื่อมีสมาชิกทุกตัวเป็นจริง และในทางตรงกันข้ามจะคืนค่า **FALSE**
&	และ (AND) เปรียบเทียบสมาชิกแต่ละตัว (Element-wise)
&&	และ (AND) เปรียบเทียบสมาชิกตัวแรก (Single comparison)
|	หรือ (OR) เปรียบเทียบสมาชิกแต่ละตัว (Element-wise)
||	หรือ (OR) เปรียบเทียบสมาชิกตัวแรก (Single comparison)
!	ไม่ (NOT)
`which()`	แสดงตำแหน่งของสมาชิกที่มีค่าเป็นจริงในเวกเตอร์
`" "`	เครื่องหมายคำพูดหรืออัญประกาศ (Double quotation mark) ใช้กำหนดขอบเขตของอักขระหรือสตริง
`nchar()`	นับจำนวนตัวอักขระในสตริง
`cat()`	เชื่อมสตริงโดยจะแสดงผลทางจอภาพ (Console) และไม่มีการส่งค่ากลับ (Return)
`paste()`	เชื่อมสตริงและมีการส่งค่ากลับ (Return)
`\n`	ขึ้นบรรทัดใหม่ (New line)
`\t`	ไปคอลัมน์ถัดไป (Horizontal tab)
`\b`	ถอยหลัง 1 ช่องว่าง (Backspace)
`\\`	ใช้แทนอักขระ \ (Backslash)
`\"`	ใช้แทนอักขระ " (Double quote)
`setwd()`	กำหนดโฟร์เดอร์ (Folder) ที่ใช้งาน (Working directory)
`substr()`	ดึงบางส่วนออกจากสตริง หรือแทนที่บางส่วนของสตริง
`sub()`	แทนที่สตริงโดยการค้นหาคำที่เหมือนกัน แล้วแทนที่คำดังกล่าวด้วยสตริงที่กำหนด โดยจะแทนเฉพาะคำแรกที่พบ
`gsub()`	แทนที่สตริงโดยการค้นหาคำที่เหมือนกัน แล้วแทนที่คำดังกล่าวด้วยสตริงที่กำหนด โดยจะแทนทุกคำที่พบ
`grep()`	ค้นหาและแทนที่สตริงได้หลายรูปแบบ
`factor()`	สร้างแฟกเตอร์
`levels()`	ตรวจสอบระดับชั้น หรือกำหนดค่าเวกเตอร์ระดับชั้น (Relabel)
`cut()`	แบ่งข้อมูลเลขต่อเนื่องออกเป็นช่วง ๆ
`Inf` และ `-Inf`	ค่าอนันต์ (Infinity) และค่าลบอนันต์
`is.infinite()`	ตรวจสอบว่าเป็นตัวเลขอนันต์หรือไม่
`is.finite()`	ตรวจสอบว่าไม่เป็นตัวเลขอนันต์

คำสั่ง	คำอธิบาย
`NaN`	ค่าที่ไม่ใช่ตัวเลข (Not a Number)
`NA`	ไม่มีข้อมูลที่ตำแหน่งนั้น ย่อมาจาก Not Available
`is.na()`	ตรวจสอบว่าไม่มีข้อมูล (**NA**) และเป็นค่าที่ไม่ใช่ตัวเลข (**NaN**)
`na.omit()`	ดึงค่าที่ไม่มีข้อมูล (**NA**) และเป็นค่าที่ไม่ใช่ตัวเลข (**NaN**) ออกจากเวกเตอร์ที่กำหนด
`NULL`	ไม่มีข้อมูลใด ๆ หรือว่าง (Empty)
`is.null()`	ตรวจสอบว่าไม่มีข้อมูล (Empty) หรือไม่มีวัตถุใด ๆ
`attributes()`	แสดงคุณสมบัติของวัตถุ
`class()`	ตรวจสอบว่าสมาชิกในวัตถุเป็นคลาสอะไร
`is._()`	ฟังก์ชันในการตรวจสอบวัตถุ
`as._()`	ฟังก์ชันในการแปลงวัตถุ

4.7 แบบฝึกหัดท้ายบท

1. สร้างตัวแปรเก็บข้อมูลตัวเลข 1, 3, 5, ..., 25

   1.1 หาว่ามีสมาชิกตัวไหนเท่ากับ 15

   1.2 หาว่ามีสมาชิกตัวไหนที่มากกว่าหรือเท่ากับ 15

   1.3 หาว่ามีสมาชิกตัวไหนไม่เท่ากับ 15

2. สร้างตัวแปรเก็บข้อมูลตัวเลข 4, -7, 15, -2, 16, -3, 12, 8, -9, 10

   2.1 หาว่าสมาชิกแต่ละตัวเป็นค่าลบหรือไม่

   2.2 หาว่าสมาชิกตัวใดตัวหนึ่งมากกว่า 10 หรือไม่

   2.3 หาว่าสมาชิกตัวทุกตัวมากกว่า 10 หรือไม่

3. ข้อใดให้ผลลัพธ์เท่ากันบ้าง? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

   3.1 $(4 < 2) \mid (5 \geq 3)$
   3.2 $4 < 2 \mid 5 \geq 3$
   3.3 $!(4 < 2) \mid (5 \geq 3)$
   3.4 $(4 < 2) \mid !(5 \geq 3)$

4. ข้อใดให้ผลลัพธ์เท่ากันบ้าง? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

   4.1 $(4 < 2) \ \& \ (5 \geq 3)$
   4.2 $4 < 2 \ \& \ 5 \geq 3$
   4.3 $!(4 < 2) \ \& \ (5 \geq 3)$
   4.4 $(4 < 2) \ \& \ (5 \geq 3)$

5. ข้อต่อไปนี้ได้ผลลัพธ์เท่าไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

   5.1 `TRUE + TRUE + TRUE`
   5.2 `TRUE + FALSE + TRUE`
   5.3 `T + T + T`
   5.4 `T + F + T`
   5.5 `1 && 0`
   5.6 `1 && 1`
   5.7 `1 || 0`
   5.8 `0 || 1`

6. ใช้ข้อมูลจากตัวแปรในข้อ 2

   6.1 ดึงข้อมูลลำดับที่ 2, 6, 7 โดยใช้ค่าตรรกะ

   6.2 แทนที่ข้อมูลที่มีค่าเป็นลบด้วย 0

6.3 มีสมาชิกลำดับไหนบ้างที่มีค่ามากกว่า 0

6.4 ลบสมาชิกที่มีค่าเป็นลบ ออกจากตัวแปรดังกล่าว

7. สร้างตัวแปรเก็บสตริง `"Here is an exercise 4"`
8. หาจำนวนอักขระในสตริงข้อ 7
9. ข้อต่อไปนี้ได้ผลลัพธ์อะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

9.1 `"lemon" == "lemon"`

9.2 `"lemon" != "melon"`

9.3 `c("pineapple", "watermelon", "melon") == "melon"`

9.4 `"lemon" >= "melon"`

9.5 `"Lemon" >= "lemon"`

9.6 `"lemon" >= "leMon"`

10. เชื่อมต่อสตริง `"pineapple", "watermelon", "melon"` ด้วยเครื่องหมาย `"-"`
11. เชื่อมต่อสตริงข้อ 10 ด้วยการเว้นวรรค แล้วเก็บไว้ในตัวแปร
12. สร้างคำสั่ง 1 บรรทัดเพื่อแสดงผลทางจอภาพ (Console) ให้ได้ดังนี้

12.1 `Here is`

12.2 `an exercise 4`

12.3 `It shows...  string manipulation`

13. ตัวแปรสตริงประกอบด้วย `"Here is an exercise 4"` ให้ดึงสตริง `"an exercise"` ออกมาแสดงทางจอภาพ
14. ตัวแปรสตริง `"Here is an exercise 4"` ให้แทนที่ `"exercise"` ด้วย `"example"`
15. ตัวแปรสตริง `"Here is an exercise 3 and exercise 4"` ให้แทนที่ `"exercise"` ทุกตัวด้วย `"example"`
16. นักศึกษาในชั้นเรียนวิชา **R** programming language จำนวน 20 คน เลขที่นักศึกษาในชั้นเรียนเรียงตามลำดับตั้งแต่ 1 ถึง 20 ประกอบด้วยนักศึกษาชาย 13 คน นักศึกษาหญิง 7 คน โดยมีนักศึกษาหญิงในลำดับที่ 2, 4, 6, 11 และ 15-17 ที่เหลือเป็นนักศึกษาชาย นักศึกษาทุกคนใช้โทรศัพท์มือถือ โดยที่นักศึกษาลำดับที่ 2-3, 5, 7-8, 13-14 และ 17 ใช้เครือข่าย AIS นักศึกษาลำดับที่ 1, 6, 9, 11 และ 20 ใช้เครือข่าย Dtac นักศึกษาลำดับที่ 4, 10, 12, 15-16 และ 19 ใช้เครือข่าย True Move H และมีนักศึกษาลำดับที่ 18 ใช้เครือข่ายอื่น ๆ (Others)

16.1 ให้สร้างเวกเตอร์ที่ประกอบด้วยเพศชาย `"M"` (Male) และหญิง `"F"` (Female) และเวกเตอร์ที่ประกอบด้วยเครือข่ายโทรศัพท์มือถือ `"AIS" "Dtac" "True" "My"` และ `"Other"`

16.2 ให้สร้างเวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) ที่ประกอบด้วยเพศนักศึกษาทุกคน และเวกเตอร์ที่เป็นแฟกเตอร์ที่ประกอบด้วยเครือข่ายโทรศัพท์มือถือของนักศึกษาทุกคน

16.3 ใช้สับเซตในการคืนค่าเวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) สำหรับแสดงเครือข่ายโทรศัพท์มือถือของนักศึกษาหญิง

16.4 ใช้สับเซตในการคืนค่าเวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) สำหรับแสดงเพศของนักศึกษาที่ใช้เครือข่าย Dtac

16.5 มีนักศึกษาลงทะเบียนเพิ่มอีก 5 คน เรียงลำดับเพศและเครือข่ายโทรศัพท์มือถือ ได้ดังนี้ **"F" "M" "F" "F" "M"** และ **"AIS" "True" "Dtac" "My" "True"** ตามลำดับ ให้รวมข้อมูลดังกล่าวเข้ากับเวกเตอร์เพศและเครือข่ายโทรศัพท์มือถือเดิม

17. นักศึกษาทั้ง 25 คนในข้อ 16. ทำคะแนนสอบได้ดังนี้ 83, 58, 94, 68, 66, 70, 85, 48, 74, 50, 88, 83, 91, 67, 100, 81, 64, 80, 86, 48, 95, 68, 79, 84, และ 76 ตามลำดับ ให้สร้างเวกเตอร์ที่แบ่งนักศึกษาตามคะแนนสอบออกเป็น 4 กลุ่มดังนี้ สอบตก **"Fail"** คะแนน [0, 50), พอใช้ **"Moderate"** คะแนน [50, 70), และน่าพอใจ **"Satisfied"** คะแนน [70, 80), และดีมาก **"Very satisfied"** คะแนน [80, 100]

18. จากข้อมูลข้อ 17. ให้หาว่านักศึกษาที่มีคะแนนสอบอยู่ในกลุ่มน่าพอใจ **"Satisfied"** ใช้เครือข่ายโทรศัพท์มือถือไหนมากที่สุด

19. สร้างเวกเตอร์ที่ประกอบด้วยตัวเลขดังนี้ -8000, -7000, -6000, -5000, 5000, 6000, 7000, 8000

19.1 ค่าไหนบ้างที่ยกกำลัง 81 แล้วไม่เป็นค่าอนันต์หรือลบอนันต์

19.2 ค่าไหนบ้างที่ยกกำลัง 81 แล้วไม่เป็นค่าลบอนันต์

19.3 ค่าไหนบ้างที่ยกกำลัง 81 แล้วหารด้วย **Inf** เป็นค่าที่ไม่ใช่ตัวเลข (Not a number, **NaN**)

19.4 สร้างเวกเตอร์ my_vector ที่ประกอบด้วยข้อมูลดังนี้ 1, 2, 3, **NA**, 5, **NaN**, 7, **NULL**, 9, 10 ให้ตรวจสอบว่าข้อไหนถูกต้อง? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

19.5 เวกเตอร์ my_vector มีความยาวเท่ากับ 10

19.6 คำสั่ง **which(is.na(my_vector))** มีค่าเท่ากับ 4

19.7 คำสั่ง **is.null(my_vector)** มีค่า **TRUE**

19.8 คำสั่ง **is.null(my_vector[8])** มีค่า **TRUE**

19.9 คำสั่ง **is.na(my_vector[4]/Inf)** มีค่า **TRUE**

20. ให้แปลงข้อมูลเมทริกซ์ต่อไปนี้

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

21. ให้อยู่ในรูปแบบดังนี้

```
[1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"  "11"  "12"
```

22. จากข้อมูลเมทริกซ์ต่อไปนี้

```
     [,1] [,2] [,3] [,4]
[1,]    1   50    1    0
[2,]    2   63    1    1
[3,]    3   70    2    0
[4,]    4   49    3    0
[5,]    5   55    1    1
[6,]    6   68    2    0
```

22.1 ให้แปลงข้อมูลดังกล่าวเป็นเดต้าเฟรม

22.2 จากเดต้าเฟรมในข้อ 23.1 ให้แปลงข้อมูลในคอลัมน์ที่ 3 เป็นแฟกเตอร์

22.3 จากเดต้าเฟรมในข้อ 23.1 ให้แปลงข้อมูลในคอลัมน์ที่ 4 เป็นค่าตรรกะ

บทที่ 5
การพล็อตเบื้องต้น (Basic Plotting)
ไฟล์ (Files) และชุดข้อมูลสำหรับโปรแกรม R

ในบทนี้จะแนะนำวิธีการพล็อตเบื้องต้น โดยใช้เครื่องมือที่ติดตั้งมาพร้อมกับโปรแกรม **R** โดยกล่าวถึงวิธีการพล็อตที่นิยม ได้แก่ Stem and leaf plots, box plots, histogram, scatter plots, density plots, pie charts, และ bar charts การเพิ่มจุด/เส้น/ข้อความ และสัญลักษณ์อื่น ๆ ลงในกราฟที่พล็อตไว้แล้ว นอกจากนี้ยังมีการพล็อตลักษณะอื่น ๆ อีก ผู้ที่สนใจค้นคว้าเพิ่มเติมได้จาก Help ของโปรแกรม **R**

เครื่องมือการพล็อตที่ติดตั้งมาพร้อมกับโปรแกรม **R** ใช้กับการพล็อตที่ไม่ซับซ้อนมากนัก สำหรับการพล็อตที่ซับซ้อนแนะนำให้ใช้แพ็กเกจ **ggplot2** ซึ่งเป็นแพ็กเกจที่นิยมใช้ในการพล็อตมากที่สุดตัวหนึ่งที่สามารถพล็อตกราฟที่ซับซ้อนได้ดี

นอกจากนี้ยังกล่าวถึงการนำเข้าและส่งออกข้อมูลทั้งแป้นพิมพ์ (Keyboard) จอภาพ (Monitor) และการทำงานกับไฟล์ (Files) และชุดข้อมูลสำหรับโปรแกรม R

5.1 การพล็อตเบื้องต้น (Basic Plotting)

5.1.1 Stem-and-Leaf Plots

การสร้าง Stem-and-leaf plots ใช้คำสั่ง **stem()** รูปแบบคำสั่งประกอบด้วย

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

พารามิเตอร์ต่าง ๆ ที่ใช้กับคำสั่ง **stem()** สรุปได้ดังนี้

ตารางที่ 5-1 พารามิเตอร์ที่ใช้กับคำสั่ง stem()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ตัวเลข	
scale	ค่าตัวเลขใช้กำหนดความยาวของช่วงที่พล็อต	1
width	ความกว้างของพล็อต	NULL
atom	ค่า tolerance	1e-08

เพื่อความสะดวกจะใช้ข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ ChickWeight เพื่อแสดงตัวอย่าง ChickWeight เป็นข้อมูลน้ำหนักไก่ (กรัม) ตามระยะเวลาที่เลี้ยง (วัน) เมื่อใช้อาหารประเภทต่าง ๆ (Diet) 4 ประเภท ดังนี้

```
R> ChickWeight
```

```
    weight Time Chick Diet
1       42    0     1    1
2       51    2     1    1
3       59    4     1    1
...
249     90    8    23    2
250    103   10    23    2
 [ reached getOption("max.print") -- omitted 328 rows ]
```

กำหนดให้ตัวแปร weight1 แทนน้ำหนักไก่ที่เลี้ยงด้วยอาหารประเภทที่ 1 (Diet=1) แสดง Stem-and-leaf plots ได้ดังนี้

```
R> weight1 <- ChickWeight$weight[ChickWeight$Diet == 1]
R> stem(weight1)

  The decimal point is 1 digit(s) to the right of the |

   2 | 599
   4 | 011111111112222223333457888999999990111111234455666778899 9
   6 | 00112223344555777788880111112223444679 9
   8 | 11234444578899990123336667888 9
  10 | 00112336667802223556 79
  12 | 00234455683456889
  14 | 112468945777
  16 | 0002234481457
  18 | 124577257899
  20 | 255958
  22 | 037
  24 | 809
  26 | 6
  28 | 8
  30 | 5
```

5.1.2 Box Plots

การสร้าง Box plots ใช้คำสั่ง **boxplot()** รูปแบบคำสั่งประกอบด้วย

```
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
        notch = FALSE, outline = TRUE, names, plot = TRUE,
        border = par("fg"), col = NULL, log = "",
        pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5),
        horizontal = FALSE, add = FALSE, at = NULL)
```

พารามิเตอร์ต่าง ๆ ที่ใช้กับคำสั่ง **boxplot()** สรุปได้ดังนี้

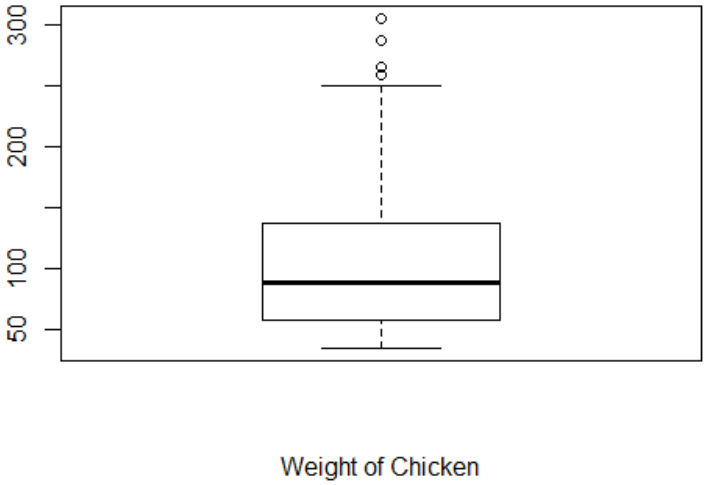
ตารางที่ 5-2 พารามิเตอร์ที่ใช้กับคำสั่ง boxplot()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ตัวเลข หรือลิสต์ที่ประกอบด้วยเวกเตอร์ตัวเลข	
...	ค่าพารามิเตอร์เพิ่มเติม	

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
range	ตัวเลขใช้กำหนดระยะห่างระหว่างหนวด (Whiskers) กับกล่อง (Box)	1.5
width	เวกเตอร์ตัวเลขใช้กำหนดสัดส่วนความกว้าง (Relative widths) ของกล่อง (Box)	NULL
varwidth	ค่าตรรกะ ถ้า varwidth = **TRUE** กล่องจะมีสัดส่วนสัมพันธ์กับรากที่สองของจำนวนข้อมูลในกลุ่ม	FALSE
notch	ค่าตรรกะ ถ้า notch = **TRUE** จะแสดงกล่องแบบบาก (Notch)	FALSE
outline	ค่าตรรกะ ถ้า outline = **FALSE** จะไม่แสดง outliers	TRUE
names	ข้อความที่แสดงแต่ละกล่อง	NULL
plot	ค่าตรรกะ ถ้า plot = **TRUE** จะแสดง Box plot ถ้า plot = **FALSE** จะส่งค่ากลับ	TRUE
border	เวกเตอร์แสดงสีของขอบกล่อง (Box)	par("fg")
col	สีของกล่อง (Box)	NULL
log	ตัวอักษรกำหนดว่าจะให้แกน x-y มีหน่วยเป็น logarithmic scale: "" แสดงว่าไม่มีแกนใดเป็น log, "x" แสดงว่าแกน x เป็น log, "y" แสดงว่าแกน y เป็น log, "xy" แสดงว่าทั้งแกน x และ y เป็น log	""
pars	ค่ากราฟิกพารามิเตอร์	list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5)
horizontal	ค่าตรรกะ ถ้า horizontal = **TRUE** แต่ละกล่องจะอยู่ในแนวนอน ถ้า horizontal = **FALSE** แต่ละกล่องจะอยู่ในแนวตั้ง	FALSE
add	ค่าตรรกะ ถ้า add = **TRUE** จะเพิ่ม boxplot เข้าไปในพล็อตปัจจุบัน	FALSE
at	เวกเตอร์ตัวเลขใช้กำหนดตำแหน่งของกล่อง ถ้า add = **TRUE** ค่า at เท่ากับ 1:n เมื่อ n คือจำนวนกล่อง (Box)	NULL

สร้าง Box plots สำหรับข้อมูล ChickWeight เมื่อเลี้ยงด้วยอาหารประเภทที่ 1 ได้ดังนี้

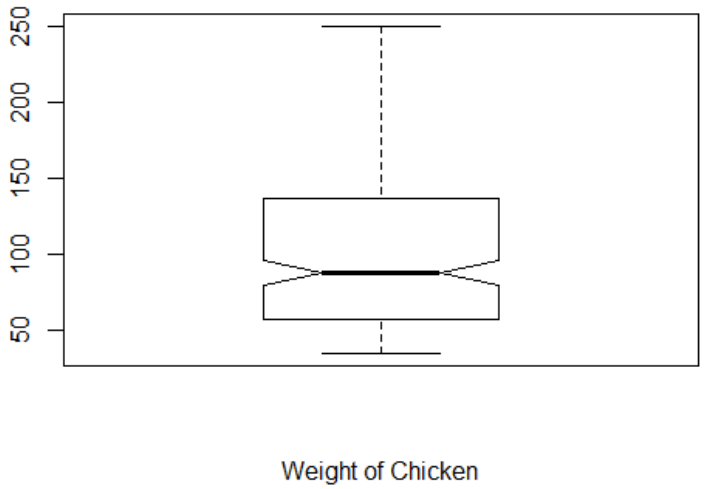
```
R> weight1 <- ChickWeight$weight[ChickWeight$Diet == 1]
R> boxplot(weight1, xlab = "Weight of Chicken")
```



รูปที่ 5-1 Box plot

สร้าง Box plots แบบบาก (Notch) และไม่แสดง Outliers ได้ ดังนี้

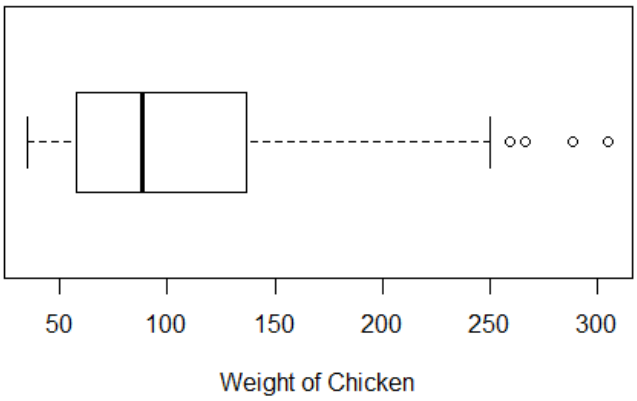
```
R> boxplot(weight1, xlab = "Weight of Chicken", notch = TRUE, outline = FALSE)
```



รูปที่ 5-2 Box plot แบบบาก (Notch) ไม่แสดง Outliers

สร้าง Box plot ในแนวนอนได้ ดังนี้

```
R> boxplot(weight1, xlab = "Weight of Chicken", horizontal = TRUE)
```



รูปที่ 5-3 Box plot แนวนอน

5.1.3 Histogram

การสร้างฮีสโตแกรม (Histogram) ใช้คำสั่ง **hist()** รูปแบบคำสั่งประกอบด้วย

```
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab = yname,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

พารามิเตอร์ต่าง ๆ ที่ใช้กับคำสั่ง **hist()** สรุปได้ดังนี้

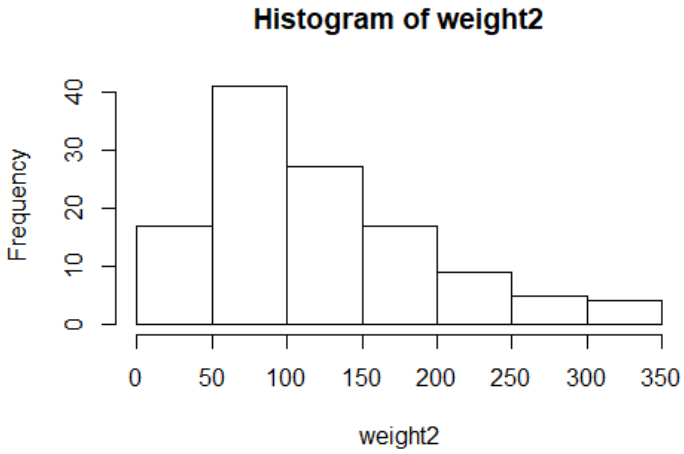
ตารางที่ 5-3 พารามิเตอร์ที่ใช้กับคำสั่ง hist()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ตัวเลข	
breaks	ประกอบด้วยข้อมูลอย่างใดอย่างหนึ่ง ดังนี้ (1) เวกเตอร์ใช้กำหนด breakpoints ระหว่าง cells (2) ฟังก์ชันใช้ในการคำนวณเพื่อกำหนด breakpoints (3) ค่าตัวเลขกำหนดจำนวน cells (4) สตริงใช้ในการคำนวณเพื่อกำหนดจำนวน cells (5) ฟังก์ชันใช้ในการคำนวณเพื่อกำหนดจำนวน cells	"Sturges"
freq	ค่าตรรกะ ถ้า freq = **TRUE** ฮีสโตแกรมจะแสดงความถี่ ผลลัพธ์คือจำนวนนับ ถ้า freq = **FALSE** ฮีสโตแกรมจะแสดงค่า probability densities (ผลรวมเท่ากับ 1)	**NULL**
probability	ค่าความน่าจะเป็น	! freq

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
include.lowest	ค่าตรรกะ ถ้า include.lowest = **TRUE** x[i] จะเท่ากับ breaks และรวมอยู่ในแท่งกราฟด้วย	**TRUE**
right	ค่าตรรกะ ถ้า right = **TRUE** ฮีสโตแกรมจะเป็นลักษณะช่วงเปิดสูงสุด (right-closed intervals or left open intervals)	**TRUE**
density	ค่าตัวเลขแสดงความหนาแน่นของเส้นระบาย (shading lines) หน่วยเป็น เส้น/นิ้ว ถ้า density = **NULL** จะไม่วาดเส้น	**NULL**
angle	ค่าตัวเลขแสดงความชันของเส้นระบาย (shading lines) หน่วยเป็นองศา	45
col	สีของแท่งกราฟ	**NULL**
border	สีของขอบแท่งกราฟ	**NULL**
main	ข้อความหลักของกราฟ (Main title)	paste("Histogram of" , xname)
xlim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน x	range(breaks)
ylim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน y	**NULL**
xlab	ข้อความแสดงบนแกน x	xname
ylab	ข้อความแสดงบนแกน y	**NULL**
axes	ค่าตรรกะเพื่อกำหนดว่าจะแสดงแกน x-y หรือไม่	**TRUE**
plot	ค่าตรรกะ ถ้า plot = **TRUE** จะแสดงฮีสโตแกรม ถ้า plot = **FALSE** จะส่งค่ากลับเป็นลิสต์ของ breaks และ counts	**TRUE**
labels	ค่าตรรกะหรือข้อความ ใช้แสดงข้อความ	**FALSE**
nclass	ค่าจำนวนเต็ม สำหรับ S-Plus compatibility	**NULL**
warn.unused	ค่าตรรกะ ถ้า plot = **FALSE** และ warn.unused = **TRUE** จะแสดงข้อความเตือนเมื่อมีการผ่านกราฟิกพารามิเตอร์เข้ามา	**TRUE**
...	ค่าพารามิเตอร์เพิ่มเติม	

กำหนดให้ตัวแปร weight2 แทนน้ำหนักไก่ที่เลี้ยงด้วยอาหารประเภทที่ 1 แสดงฮีสโตแกรม (Histogram) ได้ ดังนี้

```
R> weight2 <- ChickWeight$weight[ChickWeight$Diet == 2]
R> hist(weight2)
```



รูปที่ 5-4 Histogram

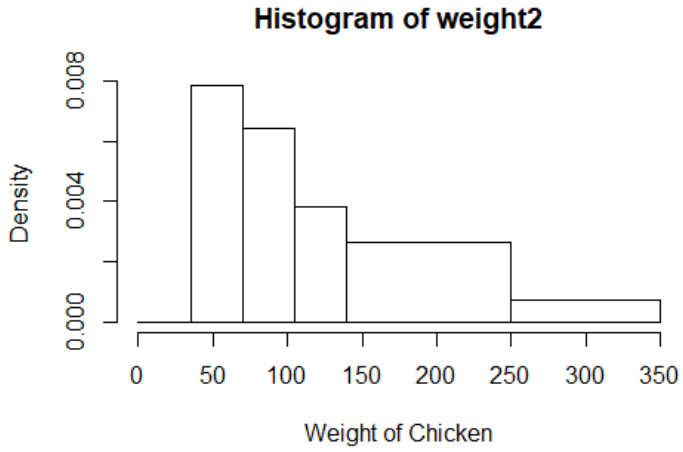
แสดงชื่อแกน x กำหนดจำนวนแท่งของฮีสโตแกรม 12 แท่ง ได้ดังนี้

```
R> hist(weight2, breaks = 12, xlab = "Weight of Chicken")
```

รูปที่ 5-5 Histogram แสดงแท่งกราฟจำนวนหลายแท่ง

แสดงในรูปแบบความน่าจะเป็น `probability = TRUE` และกำหนดความกว้างของแท่งฮีสโตแกรมได้ตามต้องการ โดยกำหนดเวกเตอร์ความกว้างของแท่งฮีสโตแกรม ได้ดังนี้

```
R> bins <- c(seq(from = 0, to = 150, by = 35), 250, 350)
R> hist(weight2, breaks = bins, xlab = "Weight of Chicken", probability = TRUE)
```

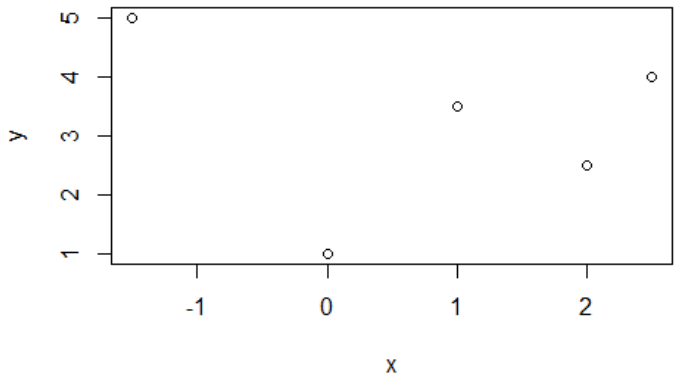


รูปที่ 5-6 Histogram แสดงแท่งกราฟความกว้างไม่เท่ากัน

5.1.4 Scatter Plots

วิธีการพล็อตจุดอย่างง่ายคือ กำหนดจุด (x, y) โดยให้ x เป็นเวกเตอร์ที่มีสมาชิกแต่ละจุดในแนวแกนนอน ส่วน y เป็นเวกเตอร์ที่มีสมาชิกแต่ละจุดในแนวแกนตั้ง เช่น มีจุดที่ต้องการพล็อต ดังนี้ (-1.5, 5.0), (0.0, 1.0), (1.0, 3.5), (2.0, 2.5) และ (2.5, 4.0) สร้างเวกเตอร์ x และ y ใช้คำสั่ง `plot()` กำหนดเวกเตอร์ x และ y เข้าไปในฟังก์ชัน ดังนี้

```
R> x <- c(-1.5, 0.0, 1.0, 2.0, 2.5)
R> y <- c(5.0, 1.0, 3.5, 2.5, 4.0)
R> plot(x, y)
```



รูปที่ 5-7 Scatter plot

ข้อมูลจุด x และ y อาจจะอยู่ในรูปเมทริกซ์หรือลิสต์ก็ได้ เช่น

```
R> xy <- cbind(x, y)
R> xy
          x      y
```

```
[1,]   -1.5    5.0
[2,]    0.0    1.0
[3,]    1.0    3.5
[4,]    2.0    2.5
[5,]    2.5    4.0

R> plot(xy)
```

ฟังก์ชัน **plot()** เป็นฟังก์ชันประเภท generic กล่าวคือ ฟังก์ชัน **plot()** ทำงานกับวัตถุที่ส่งเข้ามาพล็อตได้หลากหลาย รวมทั้งผู้ใช้สามารถกำหนดวิธีการพล็อตในฟังก์ชันดังกล่าวได้ด้วยตัวเอง (User-defined functions)

การพล็อตจุด (Scatter plots) ใช้คำสั่ง **plot()** รูปแบบคำสั่งประกอบด้วย

```
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL, log = "", main = NULL,
     sub = NULL, xlab = NULL, ylab = NULL, ann = par("ann"), axes = TRUE,
     frame.plot = axes, panel.first = NULL, panel.last = NULL, asp = NA, ...)
```

พารามิเตอร์ต่าง ๆ ที่ใช้ในการพล็อตจุด (Scatter plots) สรุปได้ดังนี้

ตารางที่ 5-4 พารามิเตอร์ที่ใช้กับ Scatter plots

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x, y	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ x, y อนุกรมเวลา สูตร ลิสต์ หรือเมทริกซ์	
type	ตัวอักษรกำหนดประเภทการพล็อต: "p" แทนจุด (points), "l" แทนเส้น (lines), "o" แทนเส้นและจุด (overplotted points and lines), "b" แทนจุดและเส้นเชื่อมจุด (points joined by lines), "s" แทนขั้นบันได (stair steps), "h" แทนเส้นแนวตั้ง (histogram-style vertical lines), และ "n" แทนไม่แสดงจุดและเส้น (no points or lines)	"p"
xlim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน x	NULL
ylim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน y	NULL
log	ตัวอักษรกำหนดว่าจะให้แกน x-y มีหน่วยเป็น logarithmic scale: "" แสดงว่าไม่มีแกนใดเป็น log, "x" แสดงว่าแกน x เป็น log, "y" แสดงว่าแกน y เป็น log, "xy" แสดงว่าทั้งแกน x และ y เป็น log	""
main	ข้อความหลักของกราฟ (Main title)	NULL
sub	ข้อความรองของกราฟ (Subtitle)	NULL
xlab	ข้อความแสดงบนแกน x	NULL
ylab	ข้อความแสดงบนแกน y	NULL
ann	ถ้า ann = **TRUE** จะแสดงข้อความหลักและรองของกราฟ รวมทั้งสัญลักษณ์ของแกน x และแกน y ถ้า ann = **FALSE** จะไม่แสดงข้อความหลักและรองของกราฟ รวมทั้งสัญลักษณ์ของแกน x และแกน y	par("ann")

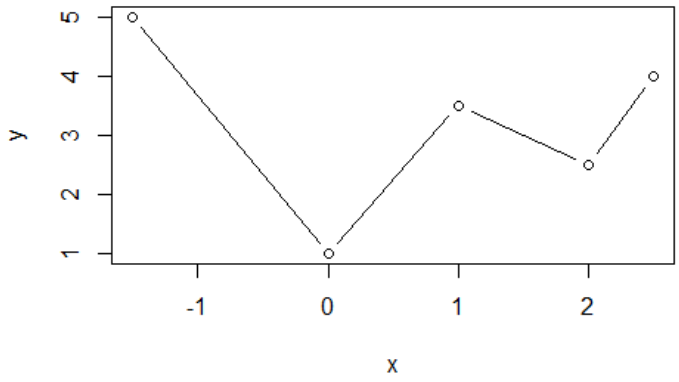
พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
frame.plot	ค่าตรรกะเพื่อกำหนดว่าจะแสดงกรอบ (Frame) หรือไม่	axes
panel.first	นิพจน์ที่จะประมวลผลหลังจากแกนถูกวาดไปแล้ว แต่ก่อนพล็อต	NULL
panel.last	นิพจน์ที่จะประมวลผลหลังจากพล็อตไปแล้ว	NULL
asp	ค่าสัดส่วนเพื่อกำหนด aspect ratio	NA
...	ค่าพารามิเตอร์เพิ่มเติม	
axes	ค่าตรรกะเพื่อกำหนดว่าจะแสดงแกน x-y หรือไม่	TRUE

ค่าพารามิเตอร์เพิ่มเติมที่ใช้งานบ่อย ๆ ประกอบด้วย

- `col` เพื่อกำหนดสีให้กับจุดหรือเส้น
- `pch` ย่อมาจาก point character เพื่อใช้ตัวอักษรแทนจุดที่พล็อต
- `cex` ย่อมาจาก character expansion เพื่อกำหนดขนาดของตัวอักษร (point character) ที่ใช้พล็อตจุด
- `lty` ย่อมาจาก line type เพื่อกำหนดชนิดของเส้นที่เชื่อมจุด
- `lwd` ย่อมาจาก line width เพื่อกำหนดความหนาของเส้นที่เชื่อมจุด

ค่าโดยปริยายกำหนดให้พล็อตเฉพาะจุด `type = "p"` ผู้ใช้สามารถแสดงเส้นเชื่อมระหว่างจุดได้โดยกำหนดพารามิเตอร์ `type = "b"` ดังนี้

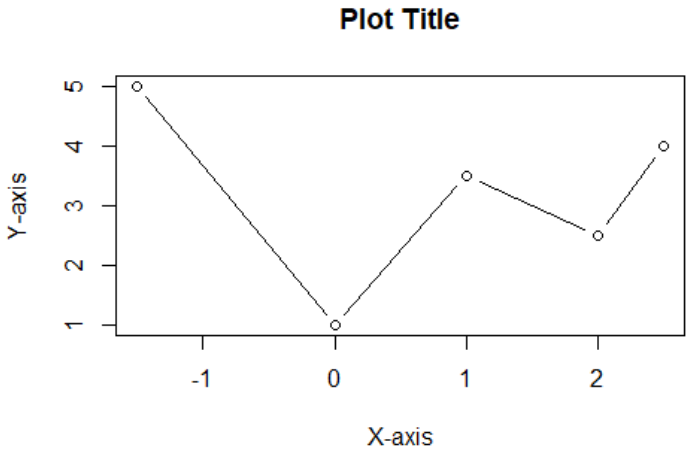
```
R> plot(valueX, valueY, type = "b")
```



รูปที่ 5-8 Scatter plots แสดงเส้นเชื่อมระหว่างจุด

แสดงเส้นข้อความหลัก (Title) และข้อความแสดงบนแกน x และ y ได้ดังนี้

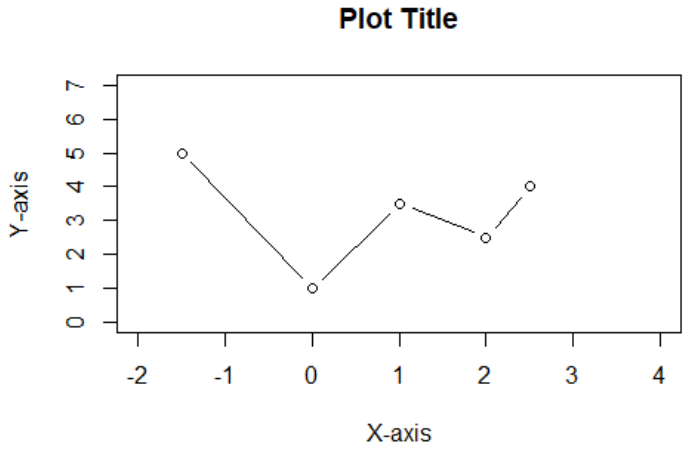
```
R> plot(x, y, type = "b", main = "Plot Title", xlab = "X-axis", ylab = "Y-axis")
```



รูปที่ 5-9 Scatter plots แสดงข้อความหลักและชื่อแกน x-y

ผู้ใช้สามารถกำหนดขอบเขตต่ำสุดและสูงสุดของแกน x-y ด้วยเวกเตอร์ ดังนี้

```
R> plot(x, y, type = "b", main = "Plot Title", xlab = "X-axis", ylab = "Y-axis",
   xlim = c(-2,4), ylim = c(0,7))
```



รูปที่ 5-10 Scatter plots กำหนดขอบเขตแกน x-y

5.1.5 การเพิ่มจุด เส้น ข้อความ และสัญลักษณ์อื่น ๆ (Adding Points, Lines, Text and Others)

ผู้ใช้สามารถเพิ่มจุด เส้น ข้อความ และสัญลักษณ์อื่น ๆ ลงในกราฟที่พล็อตไว้แล้วได้ คำสั่งที่มักนำมาใช้ ประกอบด้วยคำสั่งที่สำคัญ ๆ ดังนี้

- `points()` เพิ่มจุด
- `lines()`, `abline()`, `segments()` เพิ่มเส้น
- `text()` เขียนข้อความเพิ่ม
- `arrows()` เพิ่มลูกศร

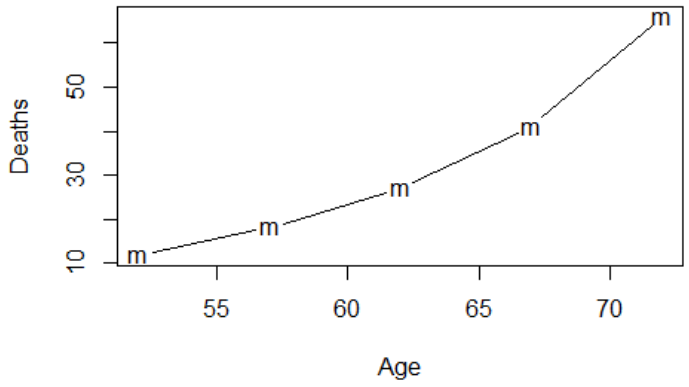
- `legend()` เพิ่มสัญลักษณ์
- `par()` เพิ่มพารามิเตอร์ เช่น ควบคุม Layout ในการพล็อต
- `mar()` กำหนดขอบเขต (Margin) (ใช้คำสั่งนี้ก่อนคำสั่งพล็อตอื่น ๆ)

เพื่อความสะดวก จะใช้ข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ VADeaths เพื่อแสดงตัวอย่าง VADeaths เป็นข้อมูลอัตราการเสียชีวิต (หน่วย x 1000) ในรัฐ Virginia ประเทศสหรัฐอเมริกาในปี ค.ศ. 1940 แยกตามเพศ และเขตเมือง/ชนบท ดังนี้

```
R> VADeaths
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

แสดงกราฟอัตราการเสียชีวิตของเพศชายในเขตชนบท โดยดึงข้อมูลเพศชายในเขตชนบท กำหนดค่ากลาง (Mid points) ของอายุ และสร้างกราฟแสดงจุดด้วยอักษร **"m"** ได้ดังนี้

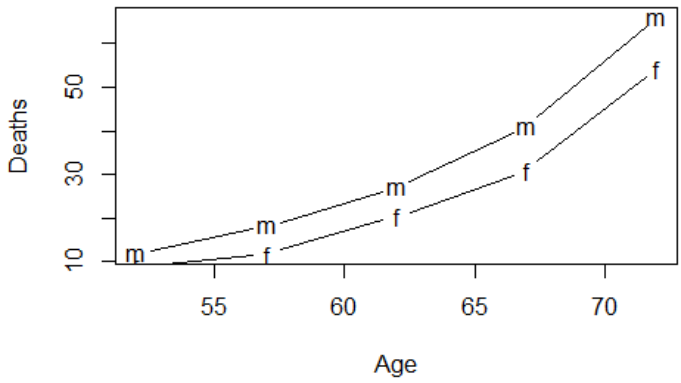
```
R> male_rural_deaths <- VADeaths[, 1]
R> x <- seq(from = 52, to = 72, by = 5)
R> plot(x, male_rural_deaths, type = "b", xlab = "Age", ylab = "Deaths",
       pch = "m")
```



รูปที่ 5-11 กราฟแสดงอัตราการเสียชีวิตของเพศชายในเขตชนบท

ผู้ใช้สามารถเพิ่มเส้นกราฟแสดงกราฟอัตราการเสียชีวิตของเพศหญิงในเขตชนบท ดังนี้

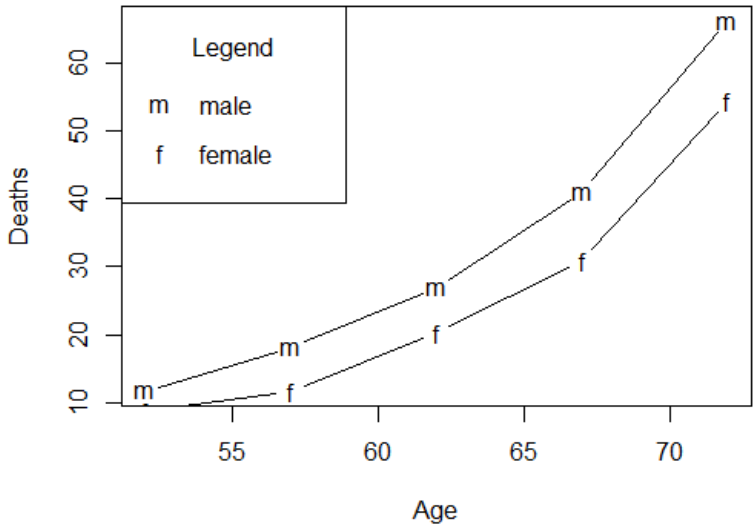
```
R> female_rural_deaths <- VADeaths[, 2]
R> lines(x, female_rural_deaths, type = "b", pch = "f")
```



รูปที่ 5-12 การเพิ่มเส้นโดยใช้คำสั่ง lines()

ผู้ใช้สามารถเพิ่มสัญลักษณ์ (Legend) และขอบเขต (Margin) ได้ดังนี้

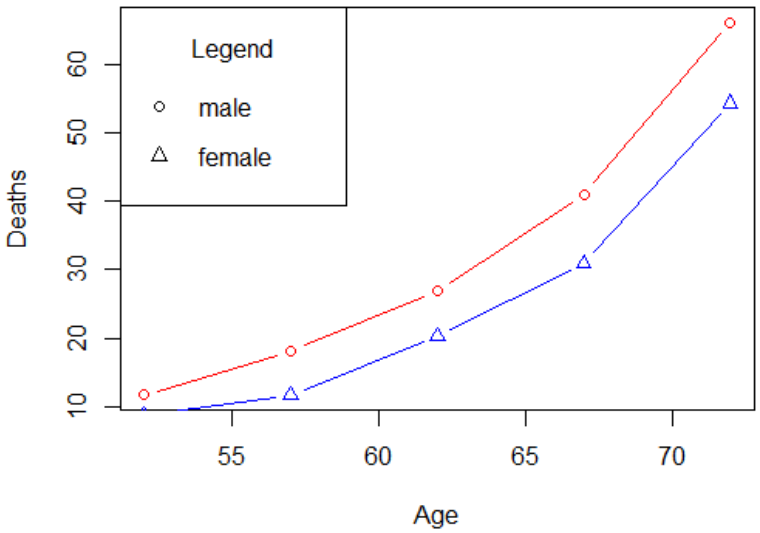
```
R> par(mar = c(bottom = 4.5, left = 4.5, top = 0.2, right = 0.2))
R> plot(x, male_rural_deaths, type = "b", xlab = "Age", ylab = "Deaths",
        pch = "m")
R> lines(x, female_rural_deaths, type = "b", pch = "f")
R> legend("topleft", legend = c("male","female"), title = "Legend",
          pch = c("m","f"))
```



รูปที่ 5-13 การเพิ่มสัญลักษณ์ (Legend) และขอบเขต (Margin)

ผู้ใช้สามารถใช้สี และสัญลักษณ์ต่าง ๆ ได้ดังนี้

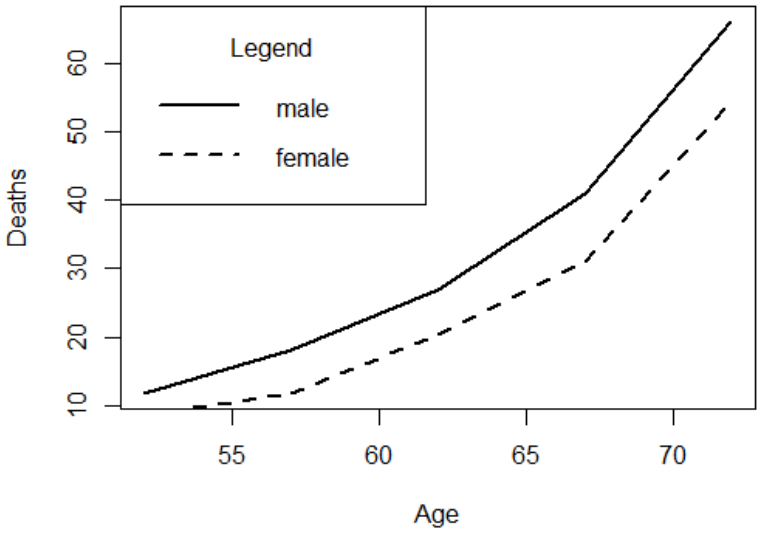
```
R> plot(x, male_rural_deaths, type = "b", xlab = "Age", ylab = "Deaths",
        pch = 1, col = "red")
R> lines(x, female_rural_deaths, type = "b", pch = 2, col = "blue")
R> legend("topleft", legend = c("male","female"), title = "Legend", pch = 1:2)
```



รูปที่ 5-14 การใช้สี (Color) และเครื่องหมาย (Symbol)

ผู้ใช้สามารถแสดงความหนา (Width) และลักษณะของเส้น (Line type) ได้ดังนี้

```
R> plot(x, male_rural_deaths, type = "l", xlab = "Age", ylab = "Deaths",
        lty = 1, lwd = 2)
R> lines(x, female_rural_deaths, type = "l", lty = 2, lwd = 2)
R> legend("topleft", legend = c("male","female"), title = "Legend",
          lty = 1:2, lwd = c(2,2))
```



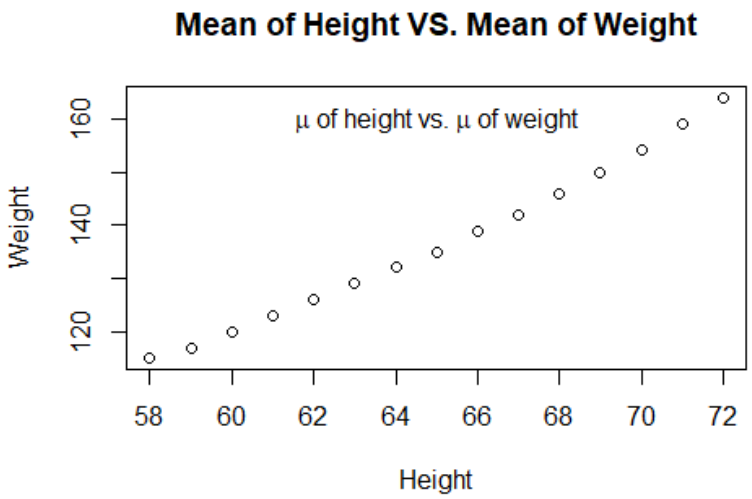
รูปที่ 5-15 การเพิ่มข้อความ (Text) และสัญลักษณ์ (Legend)

ผู้ใช้สามารถเพิ่มข้อความ (Text) หัวกราฟ (Title) และสัญลักษณ์ทางคณิตศาสตร์ ได้ดังนี้

```
R> women
   height weight
1      58    115
```

```
2      59    117
3      60    120
...
14     71    159
15     72    164

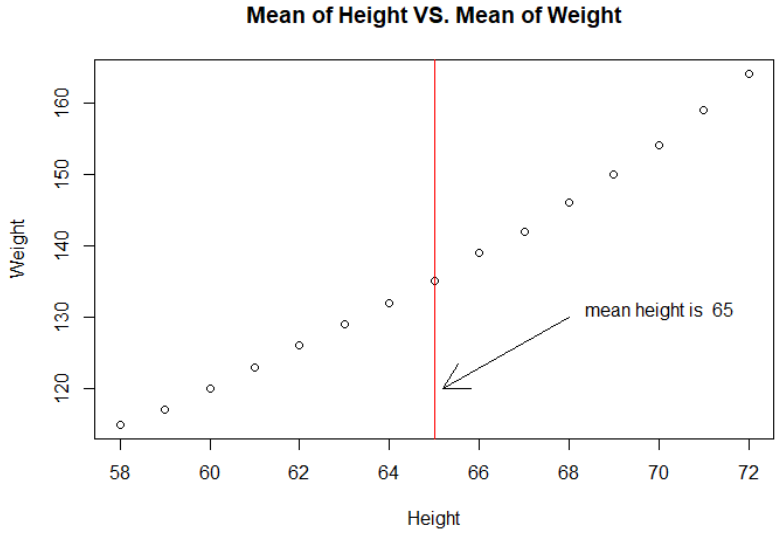
R> height <- women[, 1]
R> weight <- women[, 2]
R> plot(height, weight, xlab = "Height", ylab = "Weight")
R> title(main = "Mean of Height VS. Mean of Weight")
R> text1 <- expression(paste(mu, " of height vs. ", mu, " of weight"))
R> text(65, 160, text1)
```



รูปที่ 5-16 การเพิ่มข้อความ (Text) และข้อความหลัก (Title)

ผู้ใช้สามารถเพิ่มเส้น (Lines) และลูกศร (Arrows) ชี้ไปตำแหน่งที่ต้องการ ได้ดังนี้

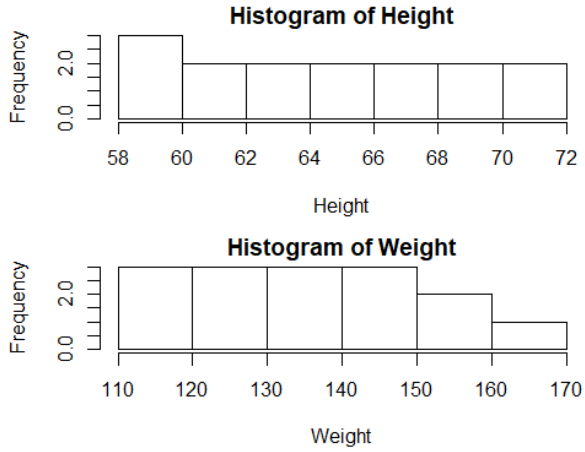
```
R> plot(height, weight, xlab = "Height", ylab = "Weight")
R> title(main = "Mean of Height VS. Mean of Weight")
R> abline(v = mean(height), col = "red")
R> arrows(x0 = 68, y0 = 130, x1 = 65.2, y1 = 120)
R> text(68, 131, paste(" mean height is ", mean(height)), pos = 4)
```



รูปที่ 5-17 การเพิ่มเส้น (Lines) และลูกศร (Arrows)

ผู้ใช้สามารถควบคุม Layout ในการพล็อต โดยการผ่านพารามิเตอร์ `mfrow` เข้าไปในฟังก์ชัน `par()` ในลักษณะ 2 แถว 1 คอลัมน์ ได้ดังนี้

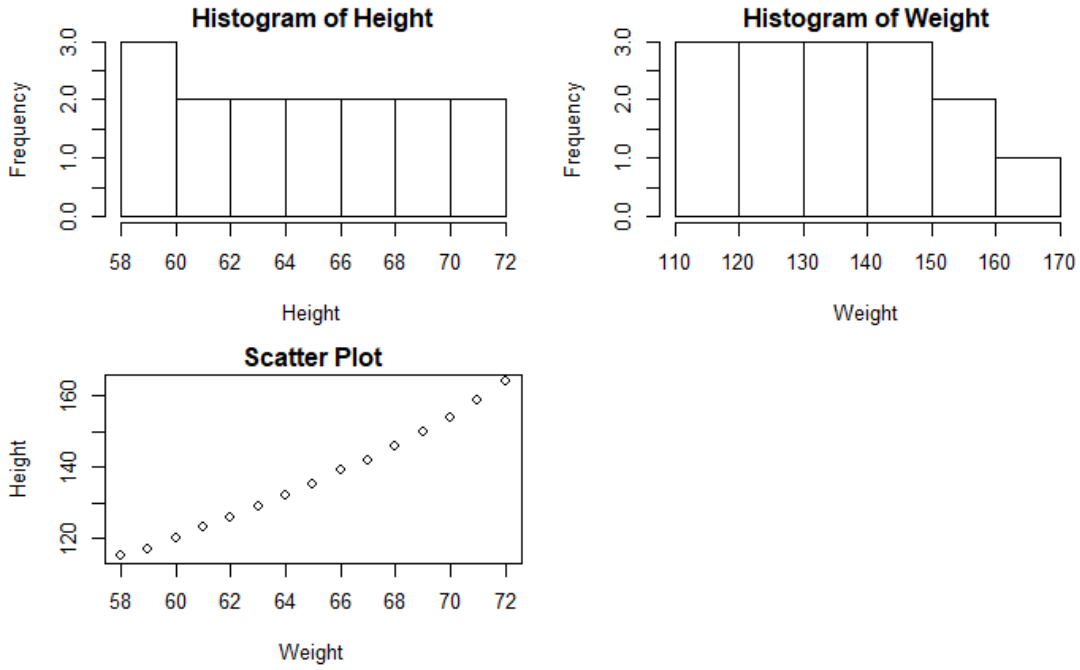
```
R> par(mar = c(bottom = 4.5, left = 4.5, top = 1.5, right = 1))
R> par(mfrow = c(2,1))
R> hist(height, xlab = "Height", main = "Histogram of Height")
R> hist(weight, xlab = "Weight", main = "Histogram of Weight")
```



รูปที่ 5-18 การพล็อตในลักษณะ 2 แถว 1 คอลัมน์

การพล็อตในลักษณะ 2 แถว 2 คอลัมน์

```
R> par(mfrow = c(2,2))
R> hist(height, xlab = "Height", main = "Histogram of Height")
R> hist(weight, xlab = "Weight", main = "Histogram of Weight")
R> plot(height, weight, xlab = "Weight", ylab = "Height", main = "Scatter Plot")
```

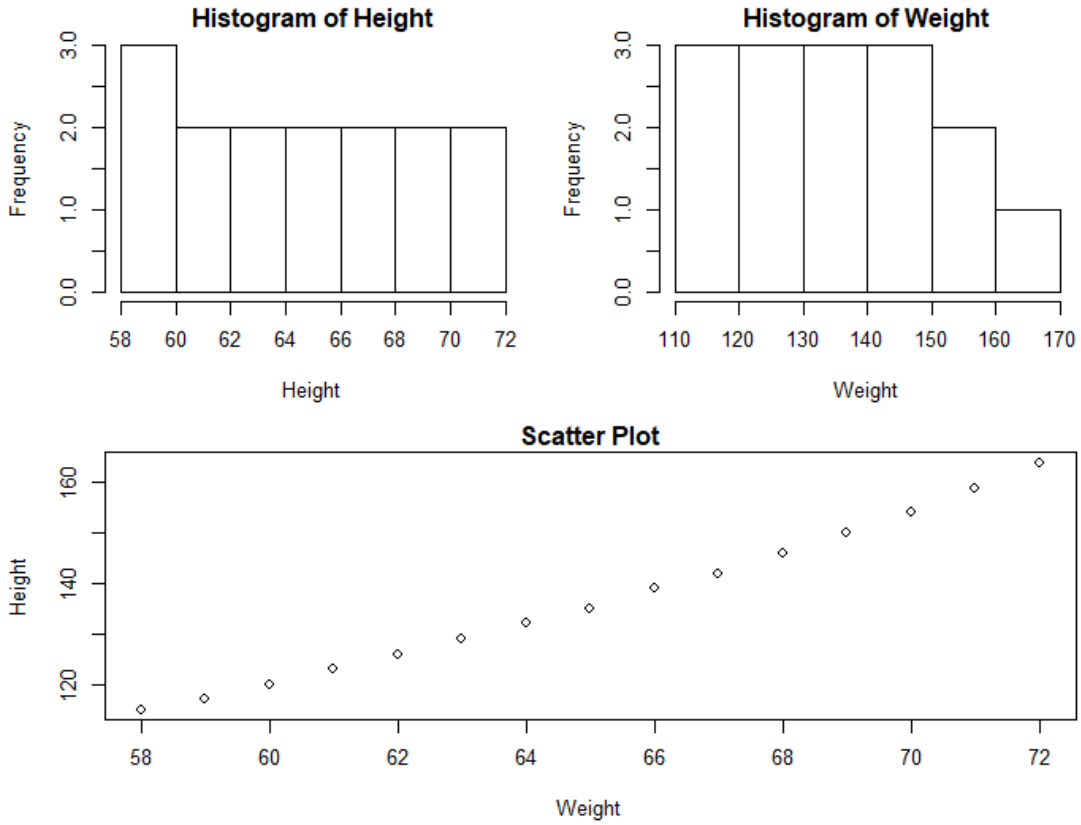


รูปที่ 5-19 การพล็อตในลักษณะ 2 แถว 2 คอลัมน์

ผู้ใช้สามารถควบคุม Layout การพล็อตในลักษณะ 2 แถว 2 คอลัมน์ และทำการรวม (Merge) cells ในแถวที่สอง ได้โดยสร้างเมทริกซ์ขนาด 2X2 แล้วกำหนดค่าควบคุม Layout การพล็อต ดังนี้

```
R> layout_matrix <- matrix(c(1, 2, 3, 3), nrow = 2, ncol = 2, byrow = TRUE)
R> layout_mMatrix
     [,1] [,2]
[1,]    1    2
[2,]    3    3

R> layout(layout_matrix)
R> hist(height, xlab = "Height", main = "Histogram of Height")
R> hist(weight, xlab = "Weight", main = "Histogram of Weight")
R> plot(height, weight, xlab = "Weight", ylab = "Height", main = "Scatter Plot")
```



รูปที่ 5-20 การพล็อตในลักษณะ 2 แถว 2 คอลัมน์ และรวมแถวที่ 2

5.1.6 Density Plots

Kernel density estimation เป็นการวิธีการทางสถิติเพื่อประมาณค่าฟังก์ชันความน่าจะเป็นแบบต่อเนื่อง (Probabiliity density function, PDF) ของตัวแปรสุ่ม (Random variables) ที่กำหนด (Scott, 2015) คล้ายกับการสร้างฮีสโตแกรม (Histogram) ที่มีแกนตั้งเป็นความน่าจะเป็น (Probability) ที่มีผลรวมของพื้นที่ใต้กราฟเท่ากับหนึ่งเสมอ แต่ Kernel density estimation มีวิธีการ (Algorithm) ทำให้เส้นกราฟเรียบขึ้น (Smooth) ในการประมาณ Kernel density ใช้คำสั่ง `density()` ดังต่อไปนี้

```
density(x, ...)
```

พารามิเตอร์ที่ใช้กับคำสั่ง `density()` คือ

ตารางที่ 5-5 พารามิเตอร์ที่ใช้กับคำสั่ง density()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ตัวเลข	-
...	ค่าพารามิเตอร์เพิ่มเติม (ดูได้จาก Help)	-

ดังตัวอย่างต่อไปนี้

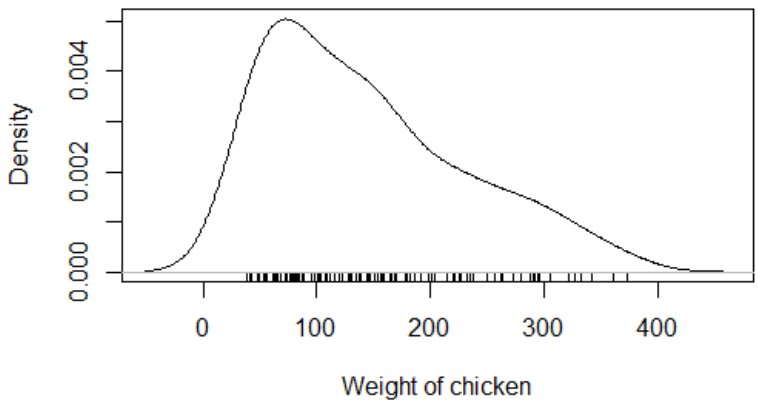
```
R> weight3 <- ChickWeight$weight[ChickWeight$Diet == 3]
R> density(weight3)

Call:
   density.default(x=weight3)

Data: weight3 (120 obs.); Bandwidth 'bw'=29.9

       x                y
 Min.   :-50.69   Min.   :1.639e-06
 1st Qu.: 77.65   1st Qu.:4.080e-04
 Median :206.00   Median :1.609e-03
 Mean   :206.00   Mean   :1.946e-03
 3rd Qu.:334.35   3rd Qu.:3.373e-03
 Max.   :462.69   Max.   :5.044e-03

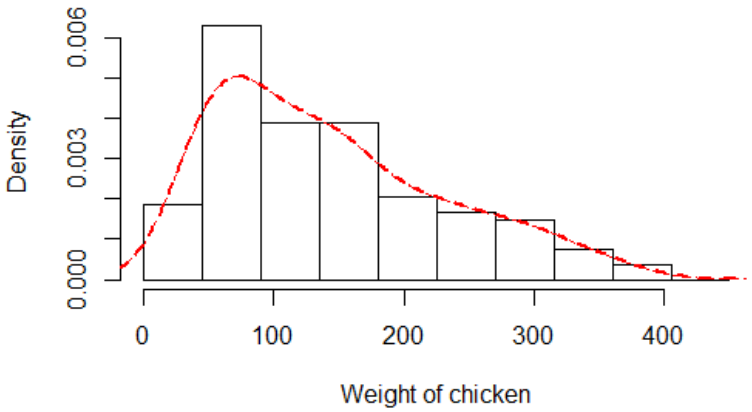
R> plot(density(weight3), xlab = "Weight of chicken", main = "")
R> rug(weight3)
```



รูปที่ 5-21 Kernel density estimation

เช่นเดียวกับคำสั่งอื่น ๆ สามารถพล็อตร่วมกับคำสั่งอื่นได้ ดังตัวอย่างต่อไปนี้

```
R> hist(weight3, xlab = "Weight of chicken", main = "", breaks = seq(0, 450, 45),
        probability = TRUE)
R> lines(density(weight3), col = "red", lty = 2, lwd = 2)
```



รูปที่ 5-22 Kernel density estimation และ histogram

5.1.7 Pie Charts

การสร้าง Pie Charts ใช้คำสั่ง `pie()` รูปแบบคำสั่งประกอบด้วย

```
pie(x, labels = names(x), edges = 200, radius = 0.8,
    clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
    density = NULL, angle = 45, col = NULL, border = NULL,
    lty = NULL, main = NULL, ...)
```

พารามิเตอร์ต่าง ๆ ที่ใช้กับคำสั่ง `pie()` สรุปได้ดังนี้

ตารางที่ 5-6 พารามิเตอร์ที่ใช้กับคำสั่ง pie()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
x	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์ตัวเลข	
labels	เวกเตอร์ตัวอักษร ใช้แสดงข้อความ	names(x)
edges	ค่าตัวเลขที่ใช้แสดงจำนวน Segments ในการวาดนอก Pie chart	200
radius	ค่าตัวเลขที่ใช้แสดงค่ารัศมีของ pie chart (สูงสุดไม่เกิน 1.0)	0.8
clockwise	ค่าตรรกะ ถ้า clockwise = **TRUE** แสดงผลตามเข็มนาฬิกา ถ้า clockwise = **FALSE** แสดงผลทวนเข็มนาฬิกา	**FALSE**
init.angle	ค่ามุมเริ่มต้นของกราฟที่จะพล็อต	If (clockwise) 90 else 0
density	ค่าตัวเลขแสดงความหนาแน่นของเส้นระบาย (Shading lines) หน่วยเป็น เส้น/นิ้ว ถ้า density = **NULL** จะไม่วาดเส้น	**NULL**
angle	ค่าตัวเลขแสดงความชันของเส้นระบาย (Shading lines) หน่วยเป็นองศา	45
col	เวกเตอร์แสดงสีของกราฟ	**NULL**

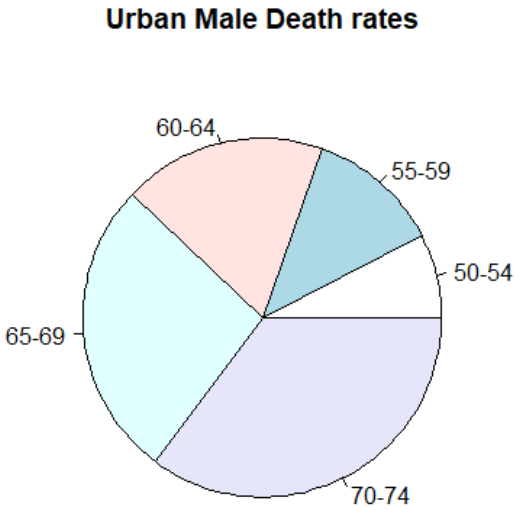
border	พารามิเตอร์ที่ผ่านเข้ามายังฟังก์ชัน polygon()	NULL
lty	ชนิดของเส้น	NULL
main	ข้อความหลักของกราฟ (Main title)	NULL

ต้องการสร้าง Pie chart แสดงสัดส่วนอัตราการเสียชีวิตของเพศชายในเขตเมืองจากข้อมูล VADeaths เริ่มจากการสร้างตัวแปร urbanMale สำหรับเก็บสัดส่วนอัตราการเสียชีวิตของเพศชายในเขตเมือง สร้างตัวแปร ageGroup สำหรับเก็บชื่อแต่ละกลุ่มอายุ และทำการสร้าง Pie chart ดังต่อไปนี้

```
R> urban_male <- VADeaths[, 3]
R> urban_male
50-54 55-59 60-64 65-69 70-74
 15.4  24.3  37.0  54.6  71.1

R> age_group <- rownames(VADeaths)
R> age_group
[1] "50-54" "55-59" "60-64" "65-69" "70-74"

R> pie(urban_male, main = "Urban Male Death rates")
```

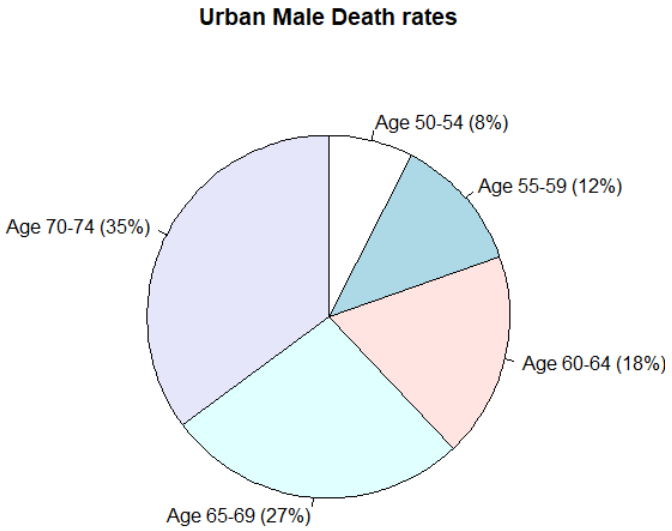


รูปที่ 5-23 Pie chart

ต้องการเพิ่มข้อความแสดงชื่อแต่ละกลุ่มอายุพร้อมกับตัวเลข % โดยการสร้างตัวแปรข้อความชื่อ pieLables และกำหนดชื่อที่ต้องการแสดงผล และให้แสดงผลเริ่มที่ 12 นาฬิกา ในลักษณะตามเข็มนาฬิกา ดังต่อไปนี้

```
R> percent <- round(urban_male/sum(urban_male)*100)
R> pieLabels <- paste("Age ", ageGroup, " (", percent, "%)", sep = "")
R> pieLabels
[1] "Age 50-54 (8%)"  "Age 55-59 (12%)"
[3] "Age 60-64 (18%)" "Age 65-69 (27%)"
[5] "Age 70-74 (35%)"

R> pie(urban_male, main = "Urban Male Death rates", labels = pieLabels,
       clockwise = TRUE, init.angle = 90)
```



รูปที่ 5-24 Pie chart แสดงสัญลักษณ์เริ่มจาก 12 นาฬิกาในทิศทางตามเข็มนาฬิกา

5.1.8 Bar Charts

การสร้าง Bar charts ใช้คำสั่ง `barplot()` รูปแบบคำสั่งประกอบด้วย

```
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
        add = FALSE, args.legend = NULL, ...)
```

พารามิเตอร์ต่าง ๆ ที่ใช้กับคำสั่ง `barplot()` สรุปได้ดังนี้

ตารางที่ 5-7 พารามิเตอร์ที่ใช้กับคำสั่ง barplot()

พารามิเตอร์	รายละเอียด	ค่าโดยปริยาย
height	ข้อมูลที่ต้องการพล็อต ในรูปเวกเตอร์หรือเมทริกซ์ ถ้าค่าที่กำหนดเป็นเมทริกซ์และ beside = **FALSE** แท่งกราฟจะแสดงเป็น stack แต่ถ้า beside = **TRUE** แท่งกราฟจะแสดงต่อกันไปในแนวนอน	
width	เวกเตอร์ตัวเลขแสดงความกว้างของแท่งกราฟ	1
space	ถ้า beside = **FALSE** ค่าที่กำหนดคือช่องว่างระหว่างแท่งกราฟ ถ้า beside = **TRUE** ค่าที่กำหนดคือ เวกเตอร์ที่มีสมาชิก 2 ตัว สมาชิกตัวแรกใช้กำหนดช่องว่างภายในกลุ่ม สมาชิกตัวที่สองใช้กำหนดช่องว่างระหว่างกลุ่ม	If(is.matrix(height) & beside = **TRUE**) c(0,1) else 0.2

names.arg	เวกเตอร์ตัวอักษรแสดงชื่อที่จะพล็อตกราฟแต่ละแท่ง (หรือกลุ่มของแท่งกราฟ)	If(is.matrix(height)) col.names(height) else names(height)
legend.text	เวกเตอร์ตัวอักษรหรือค่าตรรกะ ถ้าเป็นค่าตรรกะสัญลักษณ์ที่ใช้จะเป็นชื่อแถวของ height แต่ถ้าเป็นเวกเตอร์ตัวอักษรจะใช้ค่าจากเวกเตอร์ดังกล่าวแทน	**NULL**
beside	ค่าตรรกะใช้กำหนดว่าจะให้แท่งกราฟแสดงเป็น Stack แสดงต่อกันไปในแนวนอน (สำหรับกราฟแนวตั้ง) ใช้เมื่อ height เป็นเมทริกซ์เท่านั้น	**FALSE**
horiz	ค่าตรรกะใช้กำหนดทิศทางของแท่งกราฟ ถ้า horiz = **FALSE** แท่งกราฟจะแสดงในแนวตั้งจากซ้ายไปขวา horiz = **TRUE** แท่งกราฟจะแสดงในแนวนอนจากล่างขึ้นบน	**FALSE**
density	ค่าตัวเลขแสดงความหนาแน่นของเส้นระบาย (Shading lines) หน่วยเป็น เส้น/นิ้ว ถ้า density = **NULL** จะไม่วาดเส้น	**NULL**
angle	ค่าตัวเลขแสดงความชันของเส้นระบาย (Shading lines) หน่วยเป็น **NULL**	45

พารามิเตอร์	**รายละเอียด**	**ค่าโดยปริยาย**
col	เวกเตอร์แสดงสีของแท่งกราฟ	เทา
border	สีสำหรับขอบของแท่งกราฟ	par("fg")
main	ข้อความหลักของกราฟ (Main title)	**NULL**
sub	ข้อความรองของกราฟ (Subtitle)	**NULL**
xlab	ข้อความแสดงบนแกน x	**NULL**
ylab	ข้อความแสดงบนแกน y	**NULL**
xlim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน x	**NULL**
ylim	เวกเตอร์เพื่อกำหนดขอบเขตต่ำสุดและสูงสุดของแกน y	**NULL**
xpd	ค่าตรรกะใช้กำหนดให้แท่งกราฟสามารถแสดงนอกพื้นที่ได้ (Region)	**TRUE**
log	ตัวอักษรกำหนดว่าจะให้แกน x-y มีหน่วยเป็น logarithmic scale: "" แสดงว่าไม่มีแกนใดเป็น log, "x" แสดงว่าแกน x เป็น log, "y" แสดงว่าแกน y เป็น log, "xy" แสดงว่าทั้งแกน x และ y เป็น log	""
axes	ค่าตรรกะใช้กำหนดให้แกน x-y แสดงหรือไม่	**TRUE**
axisnames	ถ้า axisnames = **TRUE** และ names.arg ไม่เท่ากับ **NULL** จะแสดงแกนที่สองพร้อมข้อความ	**TRUE**

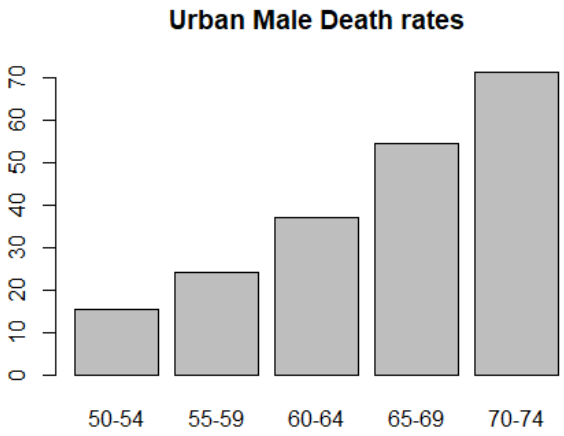
cex.axis	ตัวเลขกำหนดขนาดของตัวเลขที่แสดงในแกน x-y เทียบกับข้อความอื่น (cex.axis = 1 หมายถึงขนาดเท่ากับข้อความอื่น)	par("cex.axis")
cex.names	ตัวเลขกำหนดขนาดของข้อความที่แสดงในแกน x-y เทียบกับข้อความอื่น ๆ (cex.axis = 1 หมายถึงขนาดเท่ากับข้อความอื่น)	par("cex.axis")
inside	ค่าตรรกะใช้กำหนดให้วาดเส้นแยกแท่งกราฟ เมื่อ beside = TRUE	TRUE
plot	ค่าตรรกะใช้แสดงว่าจะพล็อตกราฟหรือไม่	TRUE
axis.lty	ชนิดของเส้นสำหรับแกน x-y	0
offset	เวกเตอร์ตัวเลขแสดงระยะที่ขยับจากแนวแกน x	0
add	ค่าตรรกะใช้กำหนดให้วาดแท่งกราฟเพิ่มเข้ามาในกราฟที่มีอยู่แล้วหรือไม่	FALSE
args.legend	ลิสต์ของพารามิเตอร์ที่ผ่านสัญลักษณ์ (legend) เข้ามา ถ้า legend.text ไม่เท่ากับ NULL	NULL
...	ค่าพารามิเตอร์เพิ่มเติม	

ต้องการสร้าง Bar chart แสดงสัดส่วนอัตราการเสียชีวิตของเพศชายในเขตเมืองจากข้อมูล VADeaths เช่นเดียวกับการสร้าง Pie chart เริ่มจากการสร้างตัวแปร urbanMale สำหรับเก็บสัดส่วนอัตราการเสียชีวิตของเพศชายในเขตเมือง สร้างตัวแปร ageGroup สำหรับเก็บชื่อแต่ละกลุ่มอายุ และทำการสร้าง Bar chart ดังต่อไปนี้

```
R> urban_male <- VADeaths[, 3]
R> urban_male
50-54 55-59 60-64 65-69 70-74
 15.4  24.3  37.0  54.6  71.1

R> age_group <- rownames(VADeaths)
R> age_group
[1] "50-54" "55-59" "60-64" "65-69" "70-74"

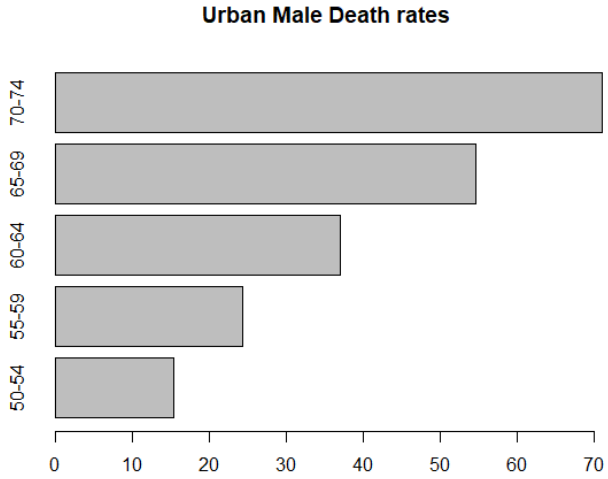
R> barplot(urban_male, main = "Urban Male Death rates")
```



รูปที่ 5-25 Bar chart

ต้องการแสดง Bar Chart ให้แท่งกราฟแสดงในแนวนอน ทำได้ดังนี้

```
R> barplot(urban_male, main = "Urban Male Death rates", horiz = TRUE)
```

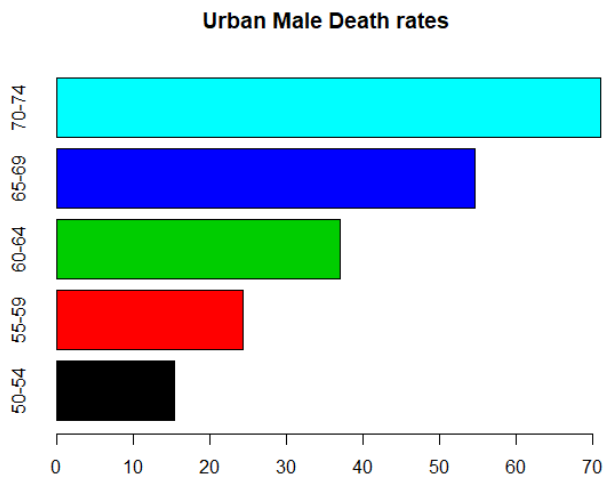


รูปที่ 5-26 Bar chart ในแนวนอน

ต้องการแสดงสีของแท่งกราฟตามกลุ่มอายุ ทำได้โดยสร้างตัวแปรจำนวนกลุ่มอายุ nGroup และผ่านเข้าไปในคำสั่ง `col` ดังต่อไปนี้

```
R> n_group <- nrow(VADeaths)
R> n_group
[1] 5

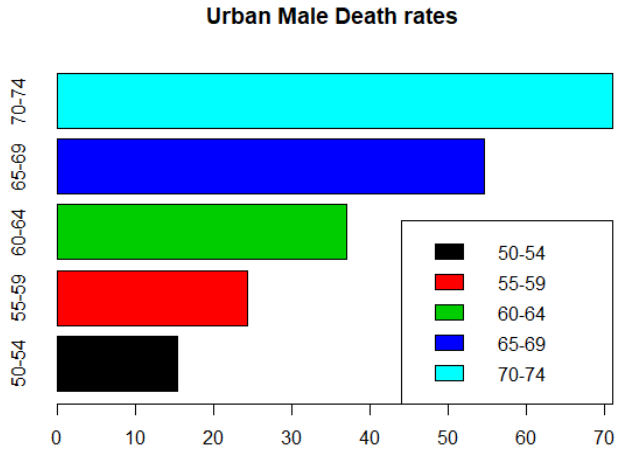
R> barplot(urban_male, main = "Urban Male Death rates", horiz = TRUE,
           Col = 1:n_group)
```



รูปที่ 5-27 Bar chart แสดงสี

ผู้ใช้สามารถเพิ่มสัญลักษณ์ (Legend) คำอธิบายได้ ดังนี้

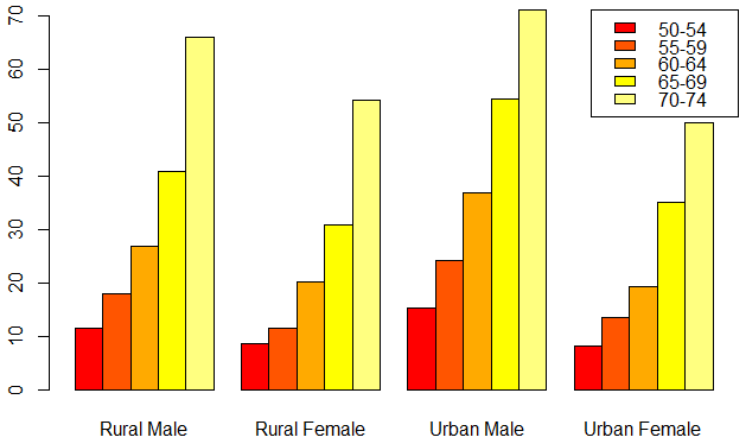
```
R> legend("bottomright", legend = age_group, fill = 1:n_group)
```



รูปที่ 5-28 Bar chart แสดงสีและสัญลักษณ์

สร้าง Bar chart เมื่อมีตัวแปรมากกว่า 1 ตัว โดยการผ่านค่าที่เป็นเมทริกซ์เข้าไป และแสดงสีโดยใช้ฟังก์ชัน `heat.colors()` ดังต่อไปนี้

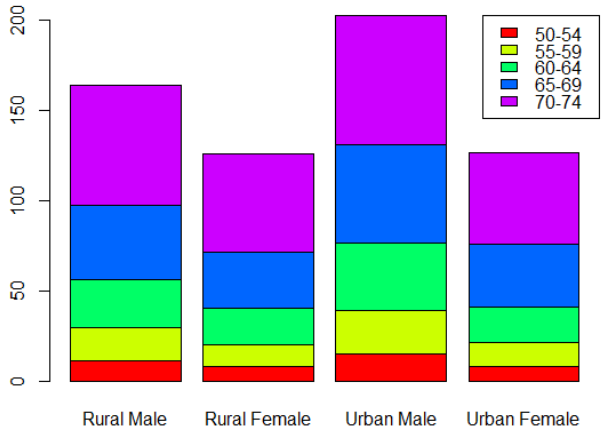
```
R> barplot(VADeaths, beside = TRUE, col = heat.colors(n_group))
R> legend("topright", legend = ageGroup, fill = heat.colors(n_group))
```



รูปที่ 5-29 Bar chart แสดงตัวแปรมากกว่า 1 ตัว

ต้องการแสดง Bar chart เป็นแบบต่อกันในแบบตั้ง (Stack) และเปลี่ยนเป็นสีแบบโทนสายรุ้ง ใช้คำสั่งดังต่อไปนี้

```
R> barplot(VADeaths, beside = FALSE, col = rainbow(n_group))
R> legend("topright", legend = age_group, fill = rainbow(n_group))
```



รูปที่ 5-30 Stacked bar chart

5.2 แป้นพิมพ์ (Keyboard) และจอภาพ (Monitor)

R มีฟังก์ชันในการรับข้อมูลจากแป้นพิมพ์ (Keyboard) และแสดงผลทางจอภาพ (Monitor) หลายแบบ ดังต่อไปนี้

5.2.1 ฟังก์ชัน scan()

ฟังก์ชัน `scan()` ใช้ในการอ่านข้อมูลจากแป้นพิมพ์หรือไฟล์ เข้ามาเป็นเวกเตอร์หรือลิสต์ ตัวอย่างเช่น มีข้อมูลเป็น Text files ชื่อ file1.txt file2.txt file3.txt และ file4.txt ข้อมูล file1.txt ประกอบด้วย

```
1234
5 6
7
```

file2.txt ประกอบด้วยข้อมูล ดังนี้

```
1234
5.5 6
7
```

file3.txt ประกอบด้วยข้อมูล ดังนี้

```
abcd
ef g
h
```

file4.txt ประกอบด้วยข้อมูล ดังนี้

```
abcd
123 4
E
```

เมื่อรันคำสั่ง `scan()` ตามด้วยพารามิเตอร์แต่ละ File จะได้ผลลัพธ์ดังนี้

```
R> scan(file = "file1.txt")  # option scan("file1.txt")
Read 4 items
[1] 1234    5    6    7

R> scan(file = "file2.txt")
Read 4 items
[1] 1234.0    5.5    6.0    7.0

R> scan(file = "file3.txt")
Error in scan("file3.txt") : scan() expected 'a real', got 'abcd'
```

คำสั่งแรกได้เวกเตอร์ที่เป็นจำนวนเต็ม (Integer) 4 ตัว คำสั่งที่สองได้เวกเตอร์ที่เป็นจำนวนจริง (Real) 4 ตัว ส่วนคำสั่งที่สามแสดงข้อผิดพลาด (Error) เนื่องจากโปรแกรมรอรับค่าที่เป็นตัวเลข (Numeric) แต่เจอตัวอักษร (Character)

```
R> scan(file = "file3.txt", what = "")
Read 4 items
[1] "abcd" "ef"   "g"    "h"

R> scan(file = "file4.txt", what = "")
Read 4 items
[1] "abcd" "123"  "4"    "e"
```

แก้ไขโดยการกำหนดพารามิเตอร์ `what = ""` ซึ่งเป็นการกำหนดให้ฟังก์ชัน `scan()` รองรับค่าที่เป็นตัวอักษร (Character) แทนที่จะเป็นตัวเลข (Numeric) ส่วนคำสั่งสุดท้ายรับค่าเป็นตัวอักษรทั้งหมด ถึงแม้ว่าสตริง `"123"` จะดูเหมือนเป็นตัวเลขก็ตาม

ค่าโดยปริยายกำหนดให้สมาชิกแต่ละตัวของเวกเตอร์ที่อ่านค่าเข้ามา แบ่งด้วย Whitespace ซึ่งประกอบด้วยช่องว่าง (Space) แท็บ (Tab) และเครื่องหมายขึ้นบรรทัดใหม่ (Carriage return/ Line feed) ผู้ใช้สามารถใช้พารามิเตอร์ **sep** กำหนดตัวแบ่งได้เอง เช่น ถ้าต้องการแบ่งสมาชิกแต่ละตัวด้วยการขึ้นบรรทัดใหม่ ให้ใช้ **sep = "\n"** ดังนี้

```
R> my_data1 <- scan(file = "file3.txt", what = "")
Read 4 items
R> my_data2 <- scan(file = "file3.txt", what = "", sep = "\n")
Read 3 items

R> my_data1
[1] "abcd" "ef"   "g"    "h"
R> my_data2
[1] "abcd" "ef g" "h"

R> my_data1[2]
[1] "ef"
R> my_data2[2]
[1] "ef g"
```

ถ้าผู้ใช้กำหนดพารามิเตอร์ **file = ""** จะหมายถึง ให้อ่านค่าจากแป้นพิมพ์ (Keyboard) ทาง console ดังนี้

```
R> new_data <- scan(file = "")  # option scan() and input data via console
1: 1.2 2.3
3: 3.4 4.5
5: 5.5
6:
Read 5 items

R> new_data
[1] 1.2 2.3 3.4 4.5 5.5
```

5.2.2 ฟังก์ชัน readline()

ฟังก์ชัน **readline()** ใช้อ่านข้อมูลนำเข้าจากแป้นพิมพ์ (Keyboard) ทาง console เพียง 1 บรรทัด ตัวอย่างเช่น

```
R> my_data <- readline()
abcd ef g

R> my_data
[1] "abcd ef g"
```

ผู้ใช้สามารถใส่พารามิเตอร์ให้แสดงผลเป็นสตริง เพื่อรอรับค่าได้ ดังนี้

```
R > my_data <- readline(prompt = "Please input your name: ")
Please input your name: BNK48

R> my_data
[1] "BNK48"
```

5.2.3 ฟังก์ชัน print()

ฟังก์ชัน `print()` เป็น "Generic function" ใช้ในการแสดงผลข้อมูล ฟังก์ชัน `print()` สามารถใช้กับข้อมูล (วัตถุ) ที่ส่งเข้ามาได้หลายประเภท ตัวอย่างเช่น ถ้าวัตถุที่ส่งมาเป็น table ฟังก์ชันดังกล่าวจะทำการเรียกฟังก์ชัน `print.table()` มาทำงาน ฟังก์ชัน `print()` สามารถพิมพ์ค่าตัวแปร (Variables) หรือนิพจน์ (Expressions) ตัวอย่างการใช้แสดงได้ดังนี้

```
R> my_vector <- 1:5
R> print(my_vector^2)
[1]  1  4  9 16 25
```

5.2.4 ฟังก์ชัน cat()

ฟังก์ชัน `cat()` ใช้ในเชื่อมต่อสตริงและแสดงผลข้อมูลคล้ายกับฟังก์ชัน `print()` แต่ฟังก์ชัน `cat()` จะต้องกำหนดอักขระหลีก (Escape characters) เอง เช่น

```
R> print("Hello world...")
[1] "Hello world..."

R> cat("Hello world...\n")
Hello world...
```

ค่าโดยปริยายของฟังก์ชัน `cat()` กำหนดให้ตัวแบ่งคือ `sep=" "` (ช่องว่าง 1 ตัว) เช่น

```
R> cat(my_vector, "abc", "xyz\n")
1 2 3 4 5 abc xyz
```

สามารถสั่งให้แสดงผลโดยใช้ตัวคั่นแบบอื่น ๆ ได้ เช่น ไม่มีช่องว่าง (`""`) ขึ้นบรรทัดใหม่ (`"\n"`) หรือกำหนดตัวคั่นหลายแบบเป็นเวกเตอร์ ได้ดังนี้

```
R> cat(my_vector, "abc", "xyz\n", sep = "")
12345abcxyz

R> cat(my_vector[1:3], "abc", "xyz\n", sep = "\n")
1
2
3
abc
xyz

R> cat(my_vector, sep = c("-", "-", "-->", "#"))
1-2-3-->4#5
```

5.3 การอ่านไฟล์ (Reading Files)

5.3.1 การอ่านไฟล์เข้ามาเป็นเดต้าเฟรม

ฐานข้อมูลโดยทั่วไปจะอยู่ในรูปแบบตาราง (Table format) บ่อยครั้งมักเป็นข้อมูลเท็กไฟล์ (Text files) ที่ประกอบด้วย 3 ส่วน ได้แก่

- หัวตาราง (Header)
- ตัวคั่น (Delimiter)
- ตัวกำหนดข้อมูลที่หายไป (Missing Value)

โดยปกติข้อมูลเท็กไฟล์จะมีนามสกุล .txt ตัวอย่างข้อมูล file5.txt ประกอบด้วย

```
name gender height
Big M 172
Dan M 180
June F 169
James M *
Kate F 175
```

ข้อมูลข้างต้นมีหัวตาราง (Header) ประกอบด้วย name gender และ height ข้อมูลแต่ละตัวถูกคั่นด้วยช่องว่าง (Space) มีข้อมูลที่หายไป (Missing Value) แทนด้วยเครื่องหมายดอกจัน (*) ข้อมูลจาก file5.txt ไม่สามารถใช้คำสั่ง `scan()` อ่านได้เนื่องจากมีทั้งตัวเลขและสตริง ผู้ใช้สามารถอ่านข้อมูลดังกล่าวเข้ามาเป็นเดต้าเฟรมได้โดยใช้คำสั่ง `read.table()` ดังนี้

```
R> my_data_frame1 <- read.table(file = "file5.txt", header = TRUE,
                                na.strings = "*", stringsAsFactors = FALSE)
R> my_data_frame1
   name gender height
1   Big      M    172
2   Dan      M    180
3  June      F    169
4 James      M     NA
5  Kate      F    175
```

คำสั่งข้างต้น อ่านค่าที่เป็นสตริงแต่ไม่แปลงเป็นแฟกเตอร์ ถ้าผู้ใช้ต้องการแปลงสตริงให้เป็นแฟกเตอร์ตอนอ่านข้อมูลเข้ามาต้องกำหนดพารามิเตอร์ `stringsAsFactors = TRUE` ถ้าข้อมูลที่อ่านเข้ามาไม่มีหัวตารางต้องกำหนดพารามิเตอร์ `header = FALSE`

ข้อมูลอีกประเภทหนึ่งที่นิยมใช้เป็นเท็กไฟล์ที่มีตัวคั่นสมาชิกแต่ละตัวด้วยเครื่องหมายจุลภาค (Comma-separated values) มีนามสกุล .csv ข้อมูลที่อยู่ใน Excel สามารถบันทึกเป็นไฟล์ .csv[11] ได้ ตัวอย่างข้อมูลที่เหมือนกับ file5.txt ข้างต้น แต่ไฟล์ชื่อ file5.csv ประกอบด้วย

```
name,gender,height
Big,M,172
Dan,M,180
June,F,169
James,M,
Kate,F,175
```

ผู้ใช้สามารถอ่านไฟล์นามสกุล .csv เข้ามาเป็นเดต้าเฟรมได้โดยใช้คำสั่ง **read.csv()** ดังนี้

```
R> my_data_frame2 <- read.csv(file = "file5.csv", header = TRUE,
                              stringsAsFactors = FALSE)
R> my_data_frame2
   name gender height
1   Big      M    172
2   Dan      M    180
3  June      F    169
4 James      M     NA
5  Kate      F    175
```

นอกจากนี้ **R** ยังมีแพ็กเกจที่สามารถอ่านข้อมูลไฟล์รูปแบบอื่น ๆ ได้หลายลักษณะ โดยการติดตั้งแพ็กเกจชื่อ foreign

```
R> install.packages("foreign")
R> library(foreign)
```

อ่านไฟล์รูปแบบต่าง ๆ จากโปรแกรมอื่นได้หลายลักษณะ เช่น

- ไฟล์โปรแกรม SPSS ใช้คำสั่ง **read.spss()**
- ไฟล์โปรแกรม SAS ใช้คำสั่ง **read.ssd()** หรือ **read.xport()**
- ไฟล์โปรแกรม Stata ใช้คำสั่ง **read.dta()**
- ไฟล์โปรแกรม Minitab ใช้คำสั่ง **read.mtp()**
- ไฟล์โปรแกรมฐานข้อมูล dBase ใช้คำสั่ง **read.dbf()**

5.3.2 การอ่านไฟล์เข้ามาเป็นเดต้าเทเบิ้ล (data.table)

ไฟล์ขนาดใหญ่มาก ๆ มักจะมีปัญหาเรื่องใช้เวลาประมวลผลนานเกินไป ทำให้ทำงานได้ช้า ขาดประสิทธิภาพ **R** มีแพ็กเกจชื่อว่า data.table ใช้จัดการกับปัญหาดังกล่าวได้อย่างมีประสิทธิภาพ

[11] การอ่านข้อมูลจากโปรแกรม Excel ที่มีไฟล์นามสกุล .xls .xlsx แนะนำให้บันทึกเป็นไฟล์นามสกุล .csv

การอ่านไฟล์เป็นเดต้าเทเบิ้ล (data.table) คล้ายกับการอ่านไฟล์เข้ามาเป็นเดต้าเฟรม แตกต่างที่คำสั่งที่ใช้ การอ่านไฟล์เป็นเดต้าเทเบิ้ล (data.table) ใช้คำสั่ง `fread()` ตัวอย่างต่อไปนี้จะอ่านไฟล์ข้อมูลขนาด 20 MB (จำนวน 253,136 แถว 17 คอลัมน์) จากเว็บไซต์ (https://github.com) และเปรียบเทียบเวลาที่ใช้ระหว่างคำสั่ง `read.csv()` กับ `fread()`

```
R> install.packages("data.table")
R> library(data.table)
R> t1 <- Sys.time()
R> my_data1 <- read.csv("https://github.com/arunsrinivasan/satrdays-
                        workshop/raw/master/flights_2014.csv")
R> t2 <- Sys.time()
R> t2-t1
Time difference of 31.87313 secs

R> t1 <- Sys.time()
R> my_data2 <- fread("https://github.com/arunsrinivasan/satrdays-
                     workshop/raw/master/flights_2014.csv")
R> t2 <- Sys.time()
R> t2-t1
Time difference of 5.459119 secs
```

จะเห็นว่า `read.csv()` ใช้เวลา 31.8 วินาที[12] ในขณะที่ `fread()` ใช้เวลาเพียง 5.4 วินาที คำสั่ง `fread()` เร็วกว่าอย่างเห็นได้ชัด

5.3.3 การอ่านเท็กไฟล์

การอ่านข้อมูลจากเท็กไฟล์ (Text files) บางส่วนหรือทั้งหมด ใช้คำสั่ง `readLines()` ดังนี้

```
R> my_text <- readLines("file5.txt")
R> my_text
[1] "name gender height" "Big M 172"          "Dan M 180"
[4] "June F 169"         "James M 188"        "Kate F 175"
```

คำสั่งข้างต้นอ่านข้อมูลจากเท็กไฟล์ทั้งหมดมาเก็บไว้ในเวกเตอร์ที่มีข้อมูลเป็นสตริงชื่อ my_text สมาชิกแต่ละตัวของเวกเตอร์เก็บข้อมูลแต่ละแถว

[12] ความเร็วในการอ่านข้อมูลอาจจะแตกต่างกันไปขึ้นอยู่กับปัจจัยหลายอย่าง เช่น ความเร็วในการรับส่งข้อมูลทางอินเตอร์เน็ต ความเร็วเครื่องคอมพิวเตอร์ เป็นต้น

5.3.4 การเชื่อมต่อ (Connection)

โปรแกรมภาษา **R** ใช้หลักการเชื่อมต่อ (Connection) เป็นเครื่องมือในการติดต่อกับอุปกรณ์ต่าง ๆ (Input/output operations) เช่น ไฟล์ อินเทอร์เน็ต เป็นต้น รายละเอียดการเชื่อมต่อดูได้จากพิมพ์ **?connection** ที่ Console การเชื่อมต่อโดยทั่วไปใช้คำสั่ง **file()** ดังนี้

```
R> conn <- file("file5.txt", "r")
R> readLines(conn, n = 1)
[1] "name gender height"
R> readLines(conn, n = 1)
[1] "Big M 172"
R> readLines(conn, n = 2)
[1] "Dan M 180"  "June F 169"
R> readLines(conn, n = 2)
[1] "James M 188" "Kate F 175"
R> readLines(conn, n = 2)
character(0)
R> close(conn)
```

คำสั่งแรกเป็นการเชื่อมต่อกับไฟล์ file5.txt ในโหมดการอ่านไฟล์ แล้วกำหนดการเชื่อมต่อดังกล่าวไว้ในตัวแปรชื่อ conn คำสั่งถัดไปเป็นการอ่านไฟล์เข้ามาทีละ 1 บรรทัด (n=1) คำสั่งถัดไปเป็นการอ่านไฟล์เข้ามาทีละ 2 บรรทัด (n=2) จนกระทั่งจบไฟล์ (End of file, EOF) คำสั่งสุดท้ายเป็นการปิดไฟล์

ผู้ใช้สามารถควบคุมการอ่านไฟล์และตรวจสอบการจบไฟล์ (EOF) ได้ ดังนี้

```
conn <- file("file5.txt", "r")
while (TRUE) {
  rec <- readLines(conn, n = 1)
  if (length(rec) == 0) {
    print("reached the end of file")
    break
  } else {
    print(rec)
  }
}
close(conn)
```

ถ้าต้องการให้ File pointer กลับไปที่จุดเริ่มต้นของไฟล์หรือตำแหน่งอื่น ๆ ของไฟล์ ใช้คำสั่ง **seek()** พร้อมกับพารามิเตอร์ **where = position** (ตำแหน่งของ File pointer) ได้ดังนี้

```
R> conn <- file("file5.txt", "r")
R> readLines(conn, n = 1)
[1] "name gender height"
R> readLines(conn, n = 2)
[1] "Big M 172" "Dan M 180"
R> seek(con = conn, where = 0)
[1] 42
R> close(conn)
```

คำสั่ง `seek()` กำหนดให้ `where = 0` หมายถึง ให้ File pointer กลับไปที่จุดเริ่มต้นของไฟล์ พร้อมกับส่งตำแหน่งปัจจุบันของตัวชี้ File pointer เท่ากับ 42 กลับมา

5.3.5 การอ่านไฟล์ทางอินเตอร์เน็ต

การอ่านไฟล์ทางอินเตอร์เน็ตสามารถใช้คำสั่งที่ผ่านมา เช่น `scan()` `read.table()` `read.csv()` ในการอ่านไฟล์ทางอินเตอร์เน็ตได้ โดยการกำหนด URL ของเว็บไซต์ที่ต้องการ ตัวอย่างในการอ่านไฟล์ .csv จากเว็บไซต์ University of California, Los Angeles (UCLA) โดยมี URL คือ https://stats.idre.ucla.edu/stat/data/binary.csv เขียนคำสั่งได้ดังนี้

```
R> my_data <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
R> head(my_data)
  admit gre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
4     1 640 3.19    4
5     0 520 2.93    4
6     1 760 3.00    2
```

5.4 การเขียนไฟล์ (Writing Files)

นอกจากการอ่านไฟล์แล้ว บางครั้งจำเป็นต้องเขียนไฟล์เพื่อนำผลที่ได้ไปดำเนินการต่อ ผู้ใช้ใช้คำสั่ง `write.table()` ในการเขียนไฟล์ไปเป็นเดต้าเฟรม ได้ดังนี้

```
R> my_data <- data.frame(name = c("Big", "Dan", "June", "James", "Kate"),
                         gender = factor(c("M", "M", "F", "M", "F")),
                         height = c(172, 180, 169, 188, 175),
                         stringsAsFactors = FALSE)
R> my_data
   name gender height
1   Big      M    172
2   Dan      M    180
3  June      F    169
4 James      M    188
5  Kate      F    175

R> write.table(my_data, file = "file6.txt")
```

เอาต์พุต (Output) ที่ได้จากการเขียนไฟล์คือ file6.txt ดังนี้

```
"name" "gender" "height"
"1" "Big" "M" 172
"2" "Dan" "M" 180
"3" "June" "F" 169
"4" "James" "M" 188
"5" "Kate" "F" 175
```

คำสั่ง `cat()` ยังสามารถเขียนไฟล์ได้ด้วย ตัวอย่างเช่น

```
cat("Hello world...\n", file = "file7.txt")
cat("Data file line 1\n", file = "file7.txt", append = TRUE)
cat("Data file line 2\n", file = "file7.txt", append = TRUE)
```

เอาต์พุต (Output) ที่ได้จากการเขียนไฟล์คือ file7.txt ดังนี้

```
Hello world...
Data file line 1
Data file line 2
```

ผู้ใช้สามารถใช้คำสั่ง **writeLines()** ในการเขียนไฟล์ได้ โดยกำหนดพารามิเตอร์ต่าง ๆ ดังนี้

```
R> my_data <- c("First element","Second element","Third element")
R> conn <- file("file8.txt", "w")
R> writeLines(my_data, con = conn, sep = "\n")
R> close(conn)
```

คำสั่ง **file()** เป็นการเชื่อมต่อกับไฟล์ file8.txt ในโหมดการเขียนไฟล์ซึ่งกำหนดด้วย **"w"** แล้วกำหนดการเชื่อมต่อดังกล่าวไว้ในตัวแปรชื่อ conn คำสั่ง **writeLines()** เป็นการสั่งให้เขียนไฟล์ file8.txt ด้วยข้อมูลเวกเตอร์ my_data คั่นสมาชิกแต่ละตัวด้วยการขึ้นบรรทัดใหม่ **"\n"** แล้วจึงปิดไฟล์ด้วยคำสั่ง **close()** เอาต์พุต (Output) ที่ได้จากการเขียนไฟล์คือ file8.txt ดังนี้

```
First element
Second element
Third element
```

5.5 ฟังก์ชันที่เกี่ยวข้องกับไฟล์

ฟังก์ชันที่จำเป็นที่เกี่ยวข้องกับไฟล์ ประกอบด้วย

- ฟังก์ชัน **file.info()** แสดงข้อมูล (information) ต่าง ๆ ของไฟล์ที่ต้องการ เช่น ขนาดไฟล์ (size) เป็นโฟลเดอร์หรือไฟล์ (isdir) เวลาในการสร้างไฟล์ เป็นต้น
- ฟังก์ชัน **dir()** แสดงข้อมูลไฟล์ที่อยู่ในโฟลเดอร์
- ฟังก์ชัน **file.exists()** ใช้ตรวจสอบไฟล์ที่ต้องการ
- ฟังก์ชัน **file.remove()** ใช้ย้ายไฟล์ที่ต้องการ
- ฟังก์ชัน **file.rename()** ใช้ในการเปลี่ยนชื่อไฟล์
- ฟังก์ชัน **file.append()** ใช้ในการเชื่อมต่อไฟล์เข้าด้วยกัน
- ฟังก์ชัน **file.copy()** ใช้ในการคัดลอกไฟล์
- ฟังก์ชัน **getwd()** ใช้ตรวจสอบโฟลเดอร์ที่กำลังทำงานอยู่ (Working directory) ว่าอยู่ที่โฟลเดอร์ไหน
- ฟังก์ชัน **setwd()** ใช้กำหนดโฟลเดอร์ที่กำลังทำงานอยู่ (Working directory) ให้ย้ายไปยังโฟลเดอร์ที่กำหนด

ตัวอย่างการใช้งาน แสดงได้ดังนี้

```
R> file.info("file6.txt")
          size isdir mode               mtime               ctime
file6.txt  125 FALSE  666 2018-03-31 12:47:46 2018-03-31 12:47:46
                        atime exe
file6.txt 2018-03-31 12:47:46  no

R> dir()
[1] "file6.txt"
[2] "file7.txt"

R> file.exists("file7.txt")
[1] TRUE

R> getwd()
[1] "C:/Users/SM/Documents"

R> setwd("C:/Users/SM/Documents/2_R")
```

5.6 ชุดข้อมูลสำหรับโปรแกรม R

5.6.1 ชุดข้อมูลที่มีอยู่ในโปรแกรม R (Build-in Data Sets)

โปรแกรม **R** มีชุดข้อมูล (Build-in data sets) ที่ถูกเรียกเข้ามาโดยอัตโนมัติระหว่างการใช้งานโปรแกรม **R** ชุดข้อมูลดังกล่าวอยู่ในแพ็กเกจชื่อ datasets ในการตรวจสอบว่าข้อมูลดังกล่าวประกอบด้วยอะไรบ้าง ใช้คำสั่งดังนี้

```
R> library(help = "datasets")
```

คำสั่ง **data()** จะแสดงชุดข้อมูลที่มีอยู่ในโปรแกรม **R** (Build-in data sets) เช่น

```
R> data()
```

ถ้าต้องการทราบรายละเอียดของชุดข้อมูล **R** ใช้คำสั่ง **?** ตามด้วยชื่อชุดข้อมูลที่ต้องการ เช่น

```
R> ?iris
```

เรียกดูโครงสร้างข้อมูลโดยใช้ฟังก์ชัน **str()** เช่น

```
R> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1
   ...
```

R ใช้คำสั่งต่าง ๆ เหมือนเดต้าเฟรมอื่น ๆ เช่น ในการเข้าถึงข้อมูล iris ใช้ชื่อข้อมูลตามด้วยดัชนีในเครื่องหมาย **[ ]** ดังนี้

```
R> iris[1:10, ]
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
```

5.6.2 ชุดข้อมูลที่เตรียมไว้สำหรับโปรแกรม R (Contributed Data Sets)

โปรแกรม **R** มีชุดข้อมูลที่อยู่ในแพ็กเกจอื่น ๆ ที่เตรียมไว้สำหรับโปรแกรม **R** (Contributed data sets) ผู้ใช้สามารถเรียกใช้ได้โดยการติดตั้ง (Install) แพ็กเกจนั้น แล้วจึงเรียกใช้ชุดข้อมูลที่ต้องการ เช่น ต้องการเรียกใช้ชุดข้อมูล **Forbes2000** ที่อยู่ในแพ็กเกจ **HSAUR2**

```
R> install.packages("HSAUR2")
R> library(HSAUR2)
R> data(Forbes2000)

R> str(Forbes2000)
'data.frame':   2000 obs. of  8 variables:
 $ rank       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ name       : chr  "Citigroup" "General Electric" "American Intl Group"
   "ExxonMobil" ...
 $ country    : Factor w/ 61 levels "Africa","Australia",..: 60 60 60 60 56 60 56
   28 60 60 ...
 $ category   : Factor w/ 27 levels "Aerospace & defense",..: 2 6 16 19 19 2 2 8 9
   20 ...
 $ sales      : num  94.7 134.2 76.7 222.9 232.6 ...
 $ profits    : num  17.85 15.59 6.46 20.96 10.27 ...
 $ assets     : num  1264 627 648 167 178 ...
 $ marketvalue: num  255 329 195 277 174 ...

R> ls()
[1] "Forbes2000"

R> class(Forbes2000)
[1] "data.frame"

R> dim(Forbes2000)
[1] 2000    8

R> names(Forbes2000)
[1] "rank"        "name"        "country"     "category"
[5] "sales"       "profits"     "assets"      "marketvalue"
```

ชุดข้อมูล **Forbes2000** เป็นข้อมูลบริษัทชั้นนำทั่วโลก 2,000 บริษัทในปี 2004 จากนิตยสาร Forbes Magazine (http://www.forbes.com) เป็นข้อมูลที่นำมารวบรวมเก็บไว้ในแพ็กเกจชื่อว่า **HSAUR2** ตัวอย่างข้อมูล 6 แถวแรก แสดงได้ดังนี้

```
R> head(Forbes2000)
  rank                 name        country             category
1    1            Citigroup  United States              Banking
2    2     General Electric  United States        Conglomerates
3    3 American Intl Group  United States            Insurance
4    4            ExxonMobil  United States Oil & gas operations
5    5                   BP United Kingdom Oil & gas operations
6    6      Bank of America  United States              Banking
   sales profits  assets marketvalue
1  94.71   17.85 1264.03      255.30
2 134.19   15.59  626.93      328.54
3  76.66    6.46  647.66      194.87
4 222.88   20.96  166.99      277.02
5 232.57   10.27  177.57      173.54
6  49.01   10.81  736.45      117.55
```

5.7 สรุปท้ายบท

ในบทนี้ได้แนะนำวิธีการพล็อตเบื้องต้นโดยใช้เครื่องมือที่ติดตั้งมาพร้อมกับโปรแกรม **R** ซึ่งเป็นวิธีการพื้นฐานที่นิยมใช้กับการพล็อตที่ไม่ซับซ้อนมากนัก โดยกล่าวถึงการพล็อตที่นิยมใช้ ได้แก่ Stem and leaf plots, box plots, histogram, scatter plots, density plots, pie charts, และ bar charts อธิบายการเพิ่มจุด/เส้น/ข้อความ และสัญลักษณ์อื่น ๆ ลงในกราฟที่พล็อตไว้แล้ว

อินพุต (Input) ทางแป้นพิมพ์ (Keyboard) และจอภาพ (Monitor) **R** มีฟังก์ชันในการรับข้อมูลจากแป้นพิมพ์และแสดงผลเอาต์พุต (Output) ทางจอภาพหลายลักษณะ ได้แก่ **scan() readline() print()** และ **cat()** สามารถอ่านข้อมูลจากเท็กไฟล์ (Text files) บางส่วนหรือทั้งหมดได้ มีฟังก์ชันในการอ่านเท็กไฟล์และไฟล์ที่บันทึกเป็นนามสกุล .csv (นิยมใช้ในโปรแกรมส่วนใหญ่เช่น Excel) เข้ามาเป็นเดต้าเฟรม ได้แก่ **read.table()** และ **read.csv()** ไฟล์ขนาดใหญ่มาก ๆ สามารถอ่านเข้ามาเป็นเดต้าเทเบิ้ล (data.table) ซึ่งใช้จัดการกับไฟล์ขนาดใหญ่ได้อย่างมีประสิทธิภาพ มีคำสั่งในการเขียนไฟล์ได้แก่ **write.table()** และ **writeLines()** รวมทั้งฟังก์ชันที่เกี่ยวข้องกับไฟล์ ได้แก่ **file.info() dir() file.exists() file.remove() file.rename() file.append() file.copy() getwd()** และ **setwd()** เป็นต้น

โปรแกรม **R** มีชุดข้อมูล (Build-in data sets) ที่ถูกเรียกเข้ามาโดยอัตโนมัติระหว่างการใช้งานโปรแกรม นอกจากนี้ยังมีชุดข้อมูลที่อยู่ในแพ็กเกจอื่น ๆ สามารถเรียกใช้ได้โดยการติดตั้ง (Install) แพ็กเกจนั้นแล้วจึงเรียกใช้ชุดข้อมูลที่ต้องการได้

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`stem()`	สร้าง Stem-and-leaf plots
`boxplot()`	สร้าง Box plots
`hist()`	สร้างฮีสโตแกรม (Histogram)
`plot()`	ฟังก์ชัน plot() เป็นฟังก์ชันประเภท Generic ซึ่งทำงานกับวัตถุที่ส่งเข้ามาพล็อตได้หลากหลาย
`col`	กำหนดสีให้กับจุดหรือเส้น
`pch`	ใช้ตัวอักษรแทนจุดที่พล็อต (ย่อมาจาก Point character)
`cex`	กำหนดขนาดของตัวอักษร (Point character) ที่ใช้พล็อตจุด (ย่อมาจาก Character expansion)
`lty`	กำหนดชนิดของเส้นที่เชื่อมจุด (ย่อมาจาก Line type)
`lwd`	กำหนดความหนาของเส้นที่เชื่อมจุด (ย่อมาจาก Line width)
`points()`	เพิ่มจุด
`lines(), abline(), segments()`	เพิ่มเส้น
`text()`	เขียนข้อความเพิ่ม
`arrows()`	เพิ่มลูกศร
`legend()`	เพิ่มสัญลักษณ์
`par()`	เพิ่มพารามิเตอร์
`mar()`	กำหนดขอบเขต (Margin)
`density()`	ประมาณ Kernel density
`pie()`	สร้าง Pie charts
`barplot()`	สร้าง Bar charts
`xlab()`	แสดงข้อความในแกน x
`ylab()`	แสดงข้อความในแกน y
`xlim()`	กำหนดค่าต่ำสุดและค่าสูงสุดที่จะแสดงในแกน x
`ylim()`	กำหนดค่าต่ำสุดและค่าสูงสุดที่จะแสดงในแกน y
`scan()`	อ่านข้อมูลจากแป้นพิมพ์หรือไฟล์

คำสั่ง	คำอธิบาย
`readline()`	อ่านข้อมูลนำเข้าจากแป้นพิมพ์ (Keyboard) ทาง Console เพียง 1 บรรทัด
`print()`	แสดงผลข้อมูล
`cat()`	เชื่อมต่อสตริงและแสดงผลข้อมูลคล้ายกับฟังก์ชัน `print()` แต่ต้องกำหนดอักขระหลีก (Escape characters) เอง
`read.table()`	อ่านข้อมูลเท็กไฟล์ (Text files) เข้ามาเป็นเดต้าเฟรม
`read.csv()`	อ่านไฟล์ csv เข้ามาเป็นเดต้าเฟรม csv เป็นเท็กไฟล์ที่มีตัวคั่นสมาชิกแต่ละตัวด้วยเครื่องหมายจุลภาค (Comma-separated values)
`data.table`	แพ็กเกจชื่อว่า data.table
`fread()`	อ่านไฟล์เป็นเดต้าเทเบิ้ล (data.table)
`readLines()`	อ่านข้อมูลจากเท็กไฟล์ (Text files) บางส่วนหรือทั้งหมด
`seek()`	กำหนดตัวชี้ในไฟล์ (File pointer)
`write.table()`	เขียนเดต้าเฟรมไปเป็นเท็กไฟล์
`writeLines()`	เขียนข้อมูลให้เป็นเท็กไฟล์
`file()`	เชื่อมต่อไฟล์
`close()`	ปิดไฟล์
`file.info()`	แสดงข้อมูล (Information) ต่าง ๆ ของไฟล์
`dir()`	แสดงข้อมูลไฟล์ที่อยู่ในโฟลเดอร์
`file.exists()`	ใช้ตรวจสอบไฟล์ที่ต้องการว่ามีอยู่หรือไม่
`data()`	แสดงชุดข้อมูล หรือโหลดชุดข้อมูลที่กำหนด
`str()`	ดูโครงสร้างข้อมูล
`ls()`	เรียกดูรายชื่อของวัตถุที่อยู่ใน Enviroment

5.8 แบบฝึกหัดท้ายบท

1. ข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ ToothGrowth แสดงความยาวฟันของหนูตะเภา (Guinea pigs) 60 ตัวอย่าง เมื่อได้รับวิตามินซีที่ระดับ 0.5, 1, และ 2 มิลลิกรัม/วัน ด้วยวิธีการ 2 แบบคือ (1) ให้ทางน้ำส้ม (Orange juice, OJ) และ (2) ให้ทาง Ascorbic acid ซึ่งเป็นรูปแบบหนึ่งของวิตามินซี (VC)

   1.1 ให้สร้าง Stem-and-leaf plots แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 2 มิลลิกรัม/วัน

   1.2 ให้สร้าง Box plots แบบบาก (Notch) แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 2 มิลลิกรัม/วัน

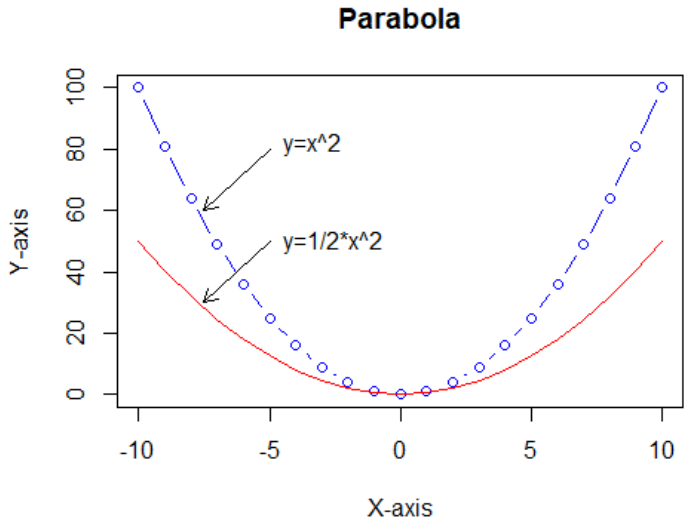
   1.3 ให้สร้าง Box plots แนวนอน แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 2 มิลลิกรัม/วัน

   1.4 ให้สร้าง Histogram แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 0.5 มิลลิกรัม/วัน โดยให้แกน y แสดงความถี่

   1.5 ให้สร้าง Histogram แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 0.5 มิลลิกรัม/วัน โดยให้แกน y แสดงความหนาแน่น

   1.6 ให้สร้าง Histogram แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 0.5 มิลลิกรัม/วัน โดยแท่งกราฟแสดงความกว้างช่วงละ 5 micron (1 micron = 1×10^{-6} meters) ระหว่าง 0 ถึง 15 micron และอีก 10 micron ระหว่าง 15 ถึง 25 micron

2. จากข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ cars ซึ่งแสดงความเร็ว (ไมล์ต่อชั่วโมง) และระยะหยุดของรถยนต์ (ฟุต) ให้สร้างกราฟแสดงความสัมพันธ์ระหว่างความเร็วและระยะหยุดของรถยนต์ โดยให้แสดงความเร็วเป็นกิโลเมตรต่อชั่วโมง และระยะหยุดของรถยนต์เป็นเมตร แสดงข้อความแสดงชื่อกราฟพร้อมกับข้อความกำกับแกน x และแกน y
3. จากข้อมูลในข้อ 1. ให้สร้าง Density plots พร้อมกับ Histogram ในกราฟรูปเดียวกัน แสดงข้อมูลความยาวฟันของหนูตะเภา เมื่อได้รับวิตามินซีที่ระดับ 2 มิลลิกรัม/วัน
4. ให้พล็อตกราฟแสดงกราฟ Parabola 2 เส้น เส้นสีน้ำเงิน $y = x^2$ และเส้นสีแดง $y = \frac{1}{2}x^2$ แสดงชื่อกราฟพร้อมกับข้อความกำกับแกน x และแกน y ลูกศรชี้ไปที่เส้นกราฟทั้งสอง และข้อความแสดงสมการที่ใช้ ดังนี้



5. จากข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ mtcars ให้สร้าง Pie chart แสดงระยะทางรถวิ่งต่อแกลลอนน้ำมัน (ไมล์/แกลลอน) แยกตามจำนวนกระบอกสูบ (Cylinders) 4, 6 และ 8 กระบอกสูบ พร้อมทั้งแสดงข้อความสัดส่วน (ร้อยละ) ของระยะทางรถวิ่งเฉลี่ยต่อแกลลอนน้ำมันแยกตามจำนวนกระบอกสูบ ให้แสดงผลเริ่มที่ 12 นาฬิกา ในลักษณะตามเข็มนาฬิกา เริ่มจาก 4, 6 และ 8 กระบอกสูบ ตามลำดับ
6. จากข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ mtcars ให้สร้าง Bar chart แสดงระยะทางรถวิ่งต่อแกลลอนน้ำมัน (ไมล์/แกลลอน) แยกตามจำนวนกระบอกสูบ (Cylinders) 4, 6 และ 8 กระบอกสูบ ตามลำดับ
7. ให้เขียนชุดคำสั่งรับค่าทางแป้นพิมพ์ ดังนี้

```
Please enter your name: ...
```

8. แล้วแสดงผลออกทาง Console ดังนี้

```
Hello, your name is ...
```

9. ให้อ่านข้อมูลจากเว็บไซต์ University of California, Los Angeles (UCLA) โดยมี URL คือ https://stats.idre.ucla.edu/stat/data/test.txt
10. ให้อ่านข้อมูลจาก https://stats.idre.ucla.edu/stat/data/test_missing.txt
11. ให้อ่านข้อมูลไฟล์ csv ดังนี้ https://stats.idre.ucla.edu/stat/data/test.csv ให้ข้อมูลแถวแรกเป็นตัวแปร
12. ให้เขียนข้อมูลจากข้อ 11. เป็นไฟล์เก็บไว้ใน Home directory ของโปรแกรม **R** แล้วเปิดด้วยโปรแกรม Excel
13. ให้เรียกดูข้อมูลจากชุดข้อมูลที่มีอยู่ในโปรแกรม **R** (Build-in data sets) ชื่อว่า ChickWeight อธิบายว่าเป็นข้อมูลอะไร โครงสร้างข้อมูลประกอบด้วยตัวแปรอะไรบ้าง

14. ให้เรียกดูข้อมูลจากชุดข้อมูลที่เตรียมไว้สำหรับโปรแกรม **R** (Contributed data sets) ชื่อว่า mpg จากแพ็กเกจ **ggplot2** อธิบายว่าเป็นข้อมูลอะไร โครงสร้างข้อมูลประกอบด้วยตัวแปรอะไรบ้าง

บทที่ 6
การโปรแกรมเบื้องต้น

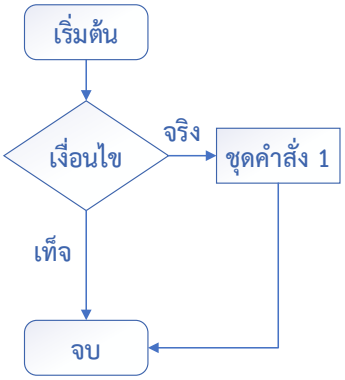
ภาษา R มีคำสั่งตรวจสอบเงื่อนไข การวนรอบ การสร้างฟังก์ชัน การจัดการความผิดพลาด ทำให้ภาษา R สามารถสร้างโปรแกรมที่ซับซ้อนได้เช่นเดียวกับภาษาอื่น ๆ ทำให้จัดการกับข้อมูลที่ซับซ้อนได้เป็นอย่างดี สามารถนำชุดคำสั่ง (Code files) กลับมาใช้ใหม่ได้ง่าย เมื่อข้อมูลมีการเปลี่ยนแปลงไป ลดระยะเวลาในการทำงานซ้ำ ๆ ทวนสอบ/ตรวจสอบงานที่ทำไว้ได้ง่าย ดังรายละเอียดที่จะกล่าวต่อไปนี้

6.1 เงื่อนไข if

6.1.1 เงื่อนไข if อย่างเดียว (Stand-Alone if Statement)

เงื่อนไข `if` อย่างเดียวเขียนได้ดังนี้

```
if (condition) {
  # code1 if condition is TRUE
}
```



เงื่อนไข (Condition) อยู่ภายในเครื่องหมายวงเล็บ `()` หลังคำสั่ง `if` ถ้าเงื่อนไขเป็นจริง (TRUE) จะทำงานคำสั่งภายในเครื่องหมายวงเล็บปีกกา `{ }` แต่ถ้าเงื่อนไขเป็นเท็จ (FALSE) จะข้ามคำสั่งภายในเครื่องหมายวงเล็บปีกกา `{ }` ไป ตัวอย่างเช่น

```
my_number <- 5
your_number <- 10
if (my_number < your_number) {
  my_number <- my_number^2
}
```

ชุดคำสั่งข้างบนมักจะเขียนไว้ใน R Editor หรือ RStudio Editor แล้วจึงรันชุดคำสั่งดังกล่าว โดยหนึ่งในวิธีการต่อไปนี้

- Copy และ paste ชุดคำสั่งจาก Editor ลงใน Console แล้วกด Enter
- ใน RStudio Editor เลือกคำสั่งที่ต้องการรัน แล้วกด Ctrl+Enter หรือกดปุ่ม Run บน Menu > Code > Run Selected Line(s)
- ใน R Editor คลิกขวาเลือก Run line or selection หรือ Ctrl+R ในตำแหน่งแถวที่ต้องการรัน

หลังจากนั้นจึงตรวจสอบค่า my_number ดังนี้

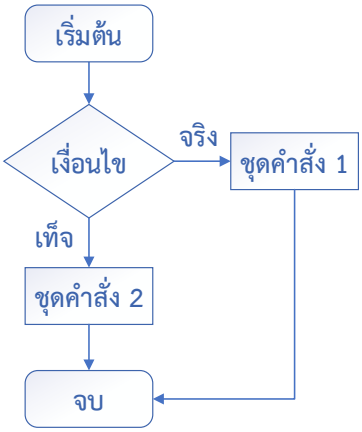
```
R> my_number
[1] 25
```

ชุดคำสั่งข้างต้นเป็นการกำหนดค่าให้ตัวแปร my_number และ your_number เท่ากับ 5 และ 10 ตามลำดับ แล้วตรวจสอบเงื่อนไข **if** ว่า my_number น้อยกว่า your_number หรือไม่ (ตรวจสอบว่า 5 < 10 หรือไม่) ผลลัพธ์ที่ได้เป็นจริง โปรแกรมทำคำสั่งภายในเครื่องหมาย **{ }** โดยกำหนดค่า my_number ให้เท่ากับ my_number ยกกำลังสองได้ผลลัพธ์เท่ากับ 25

6.1.2 เงื่อนไข if-else (if-else Statements)

เงื่อนไข **if-else** เขียนได้ดังนี้

```
if (condition) {
  # code1 if condition is TRUE
} else {
  # code2 if condition is FALSE
}
```



เงื่อนไข (Condition) อยู่ภายในเครื่องหมายวงเล็บ **()** หลังคำสั่ง **if** ถ้าเงื่อนไขเป็นจริง (**TRUE**) จะทำงานคำสั่งภายในเครื่องหมาย **{ }** แรก แต่ถ้าเงื่อนไขเป็นเท็จ (**FALSE**) โปรแกรมจะทำงานคำสั่งภายในเครื่องหมาย **{ }** ถัดไปหลังคำสั่ง **else** ตัวอย่างเช่น

```
my_number <- 20
your_number <- 10
if (my_number < your_number) {
  my_number <- my_number^2
} else {
  my_number <- my_number-1
}
```

ชุดคำสั่งข้างต้นเป็นการกำหนดค่าให้ตัวแปร my_number และ your_number เท่ากับ 20 และ 10 ตามลำดับ แล้วทำการตรวจสอบเงื่อนไข **if** ว่า my_number น้อยกว่า your_number หรือไม่ (ตรวจสอบว่า 20 < 10 หรือไม่) ผลลัพธ์ที่ได้เป็นเท็จ โปรแกรมทำคำสั่งภายในเครื่องหมายวงเล็บปีกกา **{ }** หลังคำสั่ง **else** โดยกำหนดค่า my_number ให้เท่ากับ my_number ลบด้วย 1 ได้ผลลัพธ์เท่ากับ 19

ตรวจสอบค่า my_number ได้ดังนี้

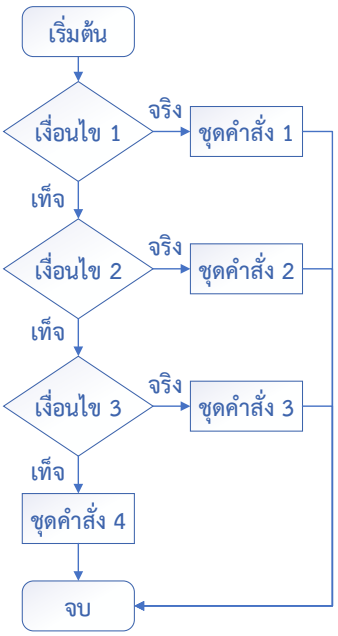
```
R> my_number
[1] 19
```

6.1.3 เงื่อนไข if-else หลายตัว (Multiple if-else Statements)

เงื่อนไข **if-else** หลายตัว เขียนได้ดังนี้

```
if (condition1) {
  # code1 if condition1 is TRUE
```

```
} else if (condition2) {
  # code2 if condition2 is TRUE
} else if (condition3) {
  # code3 if condition3 is TRUE
} else {
  # code4 otherwise
}
```



เงื่อนไข (condition) จะอยู่ภายในเครื่องหมายวงเล็บ **()** หลังคำสั่ง **if** ถ้าเงื่อนไขเป็นจริง (**TRUE**) จะทำงานคำสั่งภายในเครื่องหมาย **{ }** แล้วจบการทำงาน แต่ถ้าเงื่อนไขเป็นเท็จ (**FALSE**) จะทำการตรวจสอบเงื่อนไข **if** ถัดไป ถ้าเงื่อนไขเป็นจริง (**TRUE**) จะทำงานคำสั่งภายในเครื่องหมาย **{ }** ถัดไป ทำอย่างนี้ไปจนกระทั่งถึงคำสั่ง **else** ตัวสุดท้ายจึงจะทำงานคำสั่งภายในเครื่องหมาย **{ }** หลังคำสั่ง **else** ตัวสุดท้าย ตัวอย่างเช่น

```
R> my_salary <- 12000
   if(my_salary < 5000){
      my_tax_rate <- 0.0
   } else if (my_salary < 10000) {
      my_tax_rate <- 0.05
   } else if (my_salary < 15000) {
      my_tax_rate <- 0.10
   } else {
      my_tax_rate <- 0.15
   }
R> my_tax_rate
[1] 0.1
```

ชุดคำสั่งข้างต้นเป็นการกำหนดค่าให้ตัวแปร my_salary เท่ากับ 12,000 แล้วทำการตรวจสอบเงื่อนไข **if** ว่า my_salary น้อยกว่า 5,000 หรือไม่ ผลลัพธ์ที่ได้เป็นเท็จ โปรแกรมทำตรวจสอบเงื่อนไข **if** ต่อไปว่า my_salary น้อยกว่า 10,000 หรือไม่ ผลลัพธ์ที่ได้เป็นเท็จ โปรแกรมทำตรวจสอบเงื่อนไข **if** ต่อไปว่า my_salary น้อยกว่า 15,000 หรือไม่ ผลลัพธ์ที่ได้เป็นจริง กำหนดค่า my_tax_rate ให้เท่ากับ 0.10 แล้วข้ามชุดคำสั่งที่เหลือ และจบการทำงานชุดคำสั่งดังกล่าว

6.1.4 เงื่อนไข ifelse สำหรับตรวจสอบสมาชิกในเวกเตอร์ (ifelse Element-wise Statement Checks)

ถ้าใช้เงื่อนไข **if** หรือ **if-else** ตรวจสอบสมาชิกในเวกเตอร์ จะทำได้เฉพาะสมาชิกตัวแรกเท่านั้น พร้อมกับแสดงข้อความเตือนผู้ใช้ ดังตัวอย่างต่อไปนี้

```
R> if (c(TRUE, FALSE, FALSE, TRUE)) { }
```

```
NULL
Warning message:
In if (c(TRUE, FALSE, FALSE, TRUE)) { :
  the condition has length > 1 and only the first element will be used
```

ในการตรวจสอบสมาชิกในเวกเตอร์ทุกตัว **R** ใช้เงื่อนไข **ifelse** ดังนี้

```
ifelse(test = condition, yes = valueIfTrue, no = valueIfFalse)
```

ตัวอย่างเช่น ต้องการแสดงรากที่สอง (Square root) ของเวกเตอร์ x โดยให้แสดงค่าที่ติดลบด้วย **NA** ดังนี้

```
R> x <- c(3:-2)
R> x
[1]  3  2  1  0 -1 -2

R> sqrt(x)  # show warning message
[1] 1.732051 1.414214 1.000000 0.000000      NaN      NaN
Warning message:
In sqrt(x) : NaNs produced

R> sqrt(ifelse(x >= 0, x, NA))
[1] 1.732051 1.414214 1.000000 0.000000       NA       NA

R> sqrt(ifelse(x >= 0, yes = x, no = NA))  # Option
[1] 1.732051 1.414214 1.000000 0.000000       NA       NA
```

คำสั่ง **ifelse** ข้างต้นทำการตรวจสอบเงื่อนไขว่าเวกเตอร์ x มากกว่าหรือเท่ากับ 0 หรือไม่ ถ้าเป็นจริง (**TRUE**) ให้แสดงค่า x ถ้าเป็นเท็จ (**FALSE**) ให้แสดงค่า **NA**

6.1.5 คำสั่ง switch()

ถ้าผู้ใช้ต้องการกำหนดค่าให้กับตัวแปรร่วมกันเพียง 1 ตัว โดยใช้เงื่อนไข **if-else** ตัวอย่างเช่น ต้องการแสดงราคาบัตรคอนเสิร์ตตามประเภทบัตร ได้แก่ บัตร Class_A, บัตร Class_B, บัตร Class_C และบัตร Class_D ซึ่งมีราคาเท่ากับ 5,000, 3,000, 2,000 และ 1,000 ตามลำดับ ดังตัวอย่างต่อไปนี้

```
if(my_string == "Class_A") {
  my_value <- 5000
} else if(my_string == "Class_B") {
  my_value <- 3000
} else if(my_string == "Class_C") {
  my_value <- 2000
} else if(my_string == "Class_D") {
  my_value <- 1000
} else {
  my_value <- NA
}
```

ผู้ใช้กำหนดค่าเริ่มต้นให้กับตัวแปร my_string เช่น กำหนดให้ my_string มีค่าเท่ากับ Class_C ดังนี้

```
R> my_string <- "Class_C"
```

แล้วรันชุดคำสั่ง **if-else** ด้านบน โปรแกรมจะทำการตรวจสอบว่าค่าที่กำหนดตรงกับเงื่อนไขอะไร ในที่นี้จะได้ my_value เท่ากับ 2,000 เมื่อพิมพ์ค่าดังกล่าวจะได้ผลลัพธ์ ดังนี้

```
R> my_value
[1] 2000
```

ผู้ใช้สามารถใช้คำสั่งที่สั้นและกระชับกว่าคำสั่ง **if-else** โดยการใช้คำสั่ง **switch()** ในรูปแบบดังต่อไปนี้

```
switch(EXPR = value, ...)
```

จากตัวอย่างข้างต้น เขียนด้วยใช้คำสั่ง **switch()** โดยการกำหนดค่า my_string เป็นสตริง ดังต่อไปนี้

```
R> my_string <- "Class_C"
R> my_value <- switch(my_string, Class_A=5000, Class_B=3000,
                      Class_C=2000, Class_D=1000, NA)
R> my_value
[1] 2000
```

หรือเขียนด้วยคำสั่ง **switch()** แบบย่อโดยกำหนดค่า my_order เป็นเลขแสดงลำดับที่ของข้อมูล ได้ดังต่อไปนี้

```
R> my_order <- 3
R> my_value <- switch(my_order, 5000, 3000, 2000, 1000, NA)
R> my_value
[1] 2000
```

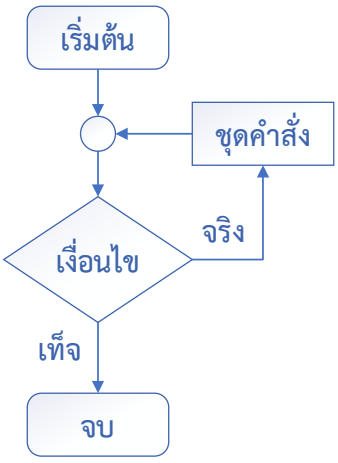
จะเห็นว่าข้อมูล Class_C เป็นข้อมูลในลำดับที่ 3 ซึ่งก็คือ 2,000 จะเห็นได้ว่าในตัวอย่างนี้การเขียนโดยใช้คำสั่ง **switch()** เขียนได้สั้นและกระชับกว่าคำสั่ง **if-else**

6.2 การวนลูป (Loops)

6.2.1 การวนลูปแบบ for

การวนลูปแบบ **for** มีรูปแบบโดยทั่วไปดังนี้

```
for (index in vector) {
  # code here
}
```



index เป็นตัวแปรที่ใช้แทนสมาชิกที่อยู่ใน vector เริ่มจากสมาชิกตัวแรกของ vector ไปจนกระทั่งสมาชิกตัวสุดท้ายของ vector เมื่อเริ่มทำงานโปรแกรมจะทำงานคำสั่งภายในเครื่องหมายวงเล็บปีกกา { } โดยมีตัวแปร index แทนสมาชิกตัวแรกของ vector และวนกลับมาเริ่มทำงานคำสั่ง

ภายในเครื่องหมาย **{ }** อีกครั้งพร้อมกับแทนตัวแปร index ด้วยสมาชิกตัวที่ 2 ของ vector ทำอย่างนี้จนกระทั่งสมาชิกตัวสุดท้ายของ vector เป็นอันสิ้นสุดลูป ตัวอย่างเช่น

```
R> for (i in 3:5) {
     cat("This is item no.", i, "\n")
   }
This is item no. 3
This is item no. 4
This is item no. 5
```

ผู้ใช้สามารถวนลูปโดยใช้ index แทนสมาชิกในเวกเตอร์ หรือใช้ index แทนลำดับในเวกเตอร์ได้ ดังนี้

```
R> my_vector <- c(1, 3, 5, 7)

R> for (i in my_vector) {
     print(i*2)
   }
[1] 2
[1] 6
[1] 10
[1] 14

R> for (i in 1:length(my_vector)) {
     print(my_vector[i] * 2)
   }
[1] 2
[1] 6
[1] 10
[1] 14
```

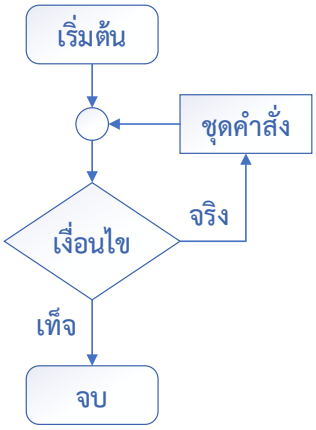
คำสั่งแรกเป็นการกำหนดค่าให้ my_vector มีค่าเป็น 1, 3, 5, 7 ตามลำดับ คำสั่ง **for** แรกเป็นการสร้างลูป โดยมีค่า **i** แทนสมาชิกแต่ละตัวของ my_vector และทำการสั่งพิมพ์ค่า **i*2** ส่วนคำสั่ง **for** สุดท้ายเป็นการสร้างลูป โดยมีค่า **i** แทน index แต่ละตัวของ my_vector เริ่มจาก 1 ไปจนถึงความยาวของ my_vector และทำการสั่งพิมพ์ค่า **my_vector[i] * 2**

6.2.2 การวนลูปแบบ while

การวนลูปแบบ **while** มีรูปแบบโดยทั่วไปดังนี้

```
while (condition) {
  # code1
}
```

ลูป **while** จะทำการตรวจสอบเงื่อนไข (Condition) ก่อน ถ้าเป็นจริง (**TRUE**) จะทำงานคำสั่งภายในเครื่องหมายวงเล็บปีกกา { } จนจบ แล้วตรวจสอบเงื่อนไขอีกจนกระทั่งเงื่อนไขเป็นเท็จจึงจะข้ามคำสั่งภายในเครื่องหมาย { } และสิ้นสุดการทำงาน โดยทั่วไปมักจะกำหนดค่าให้กับตัวแปรที่ใช้ตรวจสอบเงื่อนไขลูปก่อนที่จะเริ่มคำสั่งลูป **while** ดังตัวอย่างต่อไปนี้




```
R> my_value <- 0
R> while (my_value <5) {
     my_value <- my_value + 1  # Add my_value by 1
     cat("my_value is", my_value, "\n")  # print my_value
   }
my_value is 1
my_value is 2
my_value is 3
my_value is 4
my_value is 5
```

คำสั่งแรกเป็นการกำหนดค่าให้กับตัวแปรที่ใช้ตรวจสอบเงื่อนไขลูปชื่อว่า my_value มีค่าเริ่มต้นเท่ากับ 0 คำสั่ง **while** เป็นการสร้างลูป โดยมีเงื่อนไขว่าถ้าค่า my_value น้อยกว่า 5 เป็นจริงจะทำงานคำสั่งภายในเครื่องหมายวงเล็บปีกกา { } เนื่องจากผลลัพธ์ที่ได้เป็นจริง โปรแกรมเข้าไปทำงานคำสั่งถัดไปโดยการเพิ่มค่าให้กับ my_value ด้วยการบวกค่าขึ้นไปอีก 1 และเก็บค่าไว้ในตัวแปร my_value แล้วจึงพิมพ์ค่า my_value ออกทางจอภาพ และทำการตรวจสอบเงื่อนไขอีกจนกระทั่งเงื่อนไขเป็นเท็จจึงจะข้ามคำสั่งภายในเครื่องหมาย { } และสิ้นสุดการทำงาน

6.2.3 การวนลูปด้วยคำสั่ง apply() lapply() sapply() tapply() และ mapply()

โดยทั่วไปการวนลูปแบบ **for** และ **while** เป็นการวนลูปที่ผู้ใช้กำหนดกลไกในการวนลูปเองทั้งหมด มีคำสั่งต่าง ๆ ในการควบคุมการวนลูปชัดเจน (Explicit loops) ใน **R** มีคำสั่งการวนลูปที่เข้าถึงสมาชิกต่าง ๆ ของเวกเตอร์ เมทริกซ์ หรือลิสต์ได้โดยไม่ต้องกำหนดกลไกในการวนลูปเองทั้งหมด (Implicit loops) คำสั่งดังกล่าวทำให้การเขียนชุดคำสั่งสั้น กระชับ และมีประสิทธิภาพมากขึ้น ชุดคำสั่งดังกล่าวได้แก่ **apply()** **lapply()** **sapply()** และ **tapply()**

6.2.3.1 apply()

apply() มีรูปแบบโดยทั่วไปดังนี้

```
apply(X = array, MARGIN = value, FUN = function)
```

apply() จะคืนค่าฟังก์ชันที่ต้องการประมวลผลในแนวแถวหรือคอลัมน์ในรูปเวกเตอร์/ลิสต์/อาร์เรย์จากอาร์เรย์หรือเมทริกซ์ที่กำหนด โดยมีพารามิเตอร์ ดังนี้

- **X** คือ อาร์เรย์หรือเมทริกซ์ที่กำหนดให้
- **MARGIN** เท่ากับ 1 หมายถึงหาค่าที่ต้องการตามแถว ถ้าเท่ากับ 2 หมายถึงหาค่าที่ต้องการตามแนวคอลัมน์
- **FUN** คือ ค่าฟังก์ชันที่ต้องการประมวลผล เช่น sum, mean และ sd เป็นต้น

ตัวอย่างเช่น มีเมทริกซ์ขนาด 3x2 ดังนี้

```
R> my_matrix <- matrix(1:6, nrow = 3, ncol = 2)
R> my_matrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

ถ้าต้องการหาผลรวมในแถว (เก็บไว้ในตัวแปรชื่อว่า rowTotals) ผู้ใช้สามารถใช้คำสั่ง **for** ได้ดังนี้

```
R> row_totals <- rep(NA, times = nrow(my_matrix))  # define a row for total
R> row_totals
[1] NA NA NA

R> for (i in 1:nrow(my_matrix)) {
+    rowTotals[i] <- sum(my_matrix[i,])
+  }
R> row_totals
[1] 5 7 9
```

คำสั่งข้างต้นสามารถใช้คำสั่ง **apply()** ในการคำนวณแทนการใช้คำสั่ง **for** โดยทำการหาค่าผลรวมในแถว และคืนค่ากลับในรูปเวกเตอร์ ได้ดังนี้

```
R> row_totals <- apply(my_matrix, MARGIN = 1, FUN = sum)
R> row_totals
[1] 5 7 9
```

คำสั่ง **apply()** ทำการหาค่าผลรวมในแนวคอลัมน์ และคืนค่ากลับในรูปเวกเตอร์ ได้ดังนี้

```
R> column_totals <- apply(my_matrix, MARGIN = 2, FUN = sum)
R> column_totals
[1]  6 15
```

6.2.3.2 lapply() และ sapply()

lapply() และ **sapply()** ใช้กับเวกเตอร์หรือลิสต์ที่กำหนดให้ **lapply()** จะคืนลิสต์ ที่มีความยาวเท่ากับเวกเตอร์หรือลิสต์ที่กำหนดให้ ส่วน **sapply()** จะคืนค่ากลับในรูปแบบที่อ่านเข้าใจได้ง่ายกว่าในรูปเวกเตอร์หรือเมทริกซ์ (**s** ตัวแรกใน **sapply()** ย่อมาจาก Simplify) **lapply()** และ **sapply()** มีรูปแบบโดยทั่วไปดังนี้

```
lapply(X = vector or list, FUN = function)
sapply(X = vector or list, FUN = function)
```

โดยมีพารามิเตอร์ ดังนี้

- **X** คือ เวกเตอร์หรือลิสต์ที่กำหนดให้
- **FUN** คือ ค่าฟังก์ชันที่ต้องการประมวลผล เช่น sum, mean, min, max, sd เป็นต้น

ตัวอย่างเช่น มีคะแนนวิชา **R** programming ซึ่งมีห้องเรียนทั้งหมด 3 ห้อง (3 sections) แต่ละห้องมีนักศึกษาไม่เท่ากัน ทำให้การกำหนดรูปแบบข้อมูลเป็นเมทริกซ์ (Matrix) หรือเดต้าเฟรม (Dataframe) มีความยุ่งยากเนื่องจากเวกเตอร์แต่ละตัวมีความยาวไม่เท่ากัน ข้อมูลดังกล่าวอยู่ในรูปแบบ ดังนี้

```
R> section1 <- c(75, 82, 64, 91, 68, 77, 81, 58, 67, 70)
R> section2 <- c(86, 59, 78, 69, 82, 74, 87, 65)
R> section3 <- c(56, 63, 59, 72, 75, 69, 66)
R> classes <- list(section1, section2, section3)
```

ผู้ใช้สามารถใช้คำสั่ง `lapply()` และ `sapply()` ในการวิเคราะห์ข้อมูลที่ต้องการได้ เช่น หาค่าเฉลี่ย (Mean) และแสดงสรุป (Summary) ได้ดังนี้

```
R> lapply(classes, mean)
[[1]]
[1] 73.3

[[2]]
[1] 75

[[3]]
[1] 65.71429

R> sapply(classes, mean)
[1] 73.30000 75.00000 65.71429

R> sapply(classes, summary)
         [,1] [,2]     [,3]
Min.    58.00   59 56.00000
1st Qu. 67.25   68 61.00000
Median  72.50   76 66.00000
Mean    73.30   75 65.71429
3rd Qu. 80.00   83 70.50000
Max.    91.00   87 75.00000
```

6.2.3.3 tapply()

`tapply()` ใช้ประมวลผลข้อมูลที่ต้องการคล้ายกับคำสั่ง `apply()` `lapply()` และ `sapply()` แตกต่างกันที่ `tapply()` จะแสดงผลตามเวกเตอร์ที่เป็นแฟกเตอร์ (Factor-valued vectors) ที่กำหนด โดยจะคืนค่าฟังก์ชันที่ต้องการประมวลผลในรูปเวกเตอร์ มีรูปแบบโดยทั่วไปดังนี้

```
tapply(X = vector, INDEX = list, FUN = function)
```

โดยมีพารามิเตอร์ ดังนี้

- **X** คือ วัตถุที่กำหนดให้ โดยทั่วไปเป็นเวกเตอร์
- **INDEX** คือ ลิสต์ของแฟกเตอร์ตั้งแต่ 1 ตัวขึ้นไป
- **FUN** คือ ค่าฟังก์ชันที่ต้องการประมวลผล เช่น sum, mean, sd เป็นต้น

ตัวอย่างเช่น จากข้อมูลที่ติดตั้งมาพร้อมกับ **ggplot2** ชื่อว่า mpg ต้องการหาค่าเฉลี่ยจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) สำหรับรถยนต์ที่วิ่งบนทางหลวง (Highway) แยกตามประเภทรถ สามารถใช้คำสั่ง **tapply()** ได้ดังนี้

```
R> require(ggplot2)
R> my_data <- mpg
R> my_data$class.fac <- factor(my_data$class)
R> tapply(my_data$hwy, INDEX = my_data$class.fac, FUN = mean)
   2seater    compact    midsize    minivan     pickup subcompact        suv
  24.80000   28.29787   27.29268   22.36364   16.87879   28.14286   18.12903
```

คำสั่งแรกเป็นการเรียกแพ็กเกจ **ggplot2** เข้ามาเก็บไว้ในหน่วยความจำ คำสั่งที่สองเป็นการกำหนดให้ข้อมูล mpg เก็บไว้ในตัวแปรชื่อ my_data คำสั่งที่สามเป็นการสร้างตัวแปรแฟกเตอร์ชื่อ class.fac จากตัวแปร class คำสั่งสุดท้ายเป็นการหาค่าเฉลี่ยของจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) สำหรับรถยนต์ที่วิ่งบนทางหลวง (ตัวแปรชื่อ hwy) แยกตามประเภทยานพาหนะ โดยการระบุพารามิเตอร์เวกเตอร์ที่ต้องการหาค่า **my_data$hwy** แฟกเตอร์ที่ต้องการแยกประเภท **INDEX = my_data$class.fac** และฟังก์ชันที่ต้องการประมวลผล **FUN = mean**

6.2.3.4 mapply()

mapply() เป็นการประยุกต์ใช้ฟังก์ชันกับลิสต์หรือเวกเตอร์หลายตัวพร้อม ๆ กัน หรือกล่าวได้ว่าเป็นเหมือนการใช้ **sapply()** ในลักษณะหลายตัวแปร มีรูปแบบโดยทั่วไปดังนี้

```
mapply(FUN, ..., MoreArgs = NULL)
```

โดยมีพารามิเตอร์ ดังนี้

- **FUN** คือ ค่าฟังก์ชันที่ต้องการประมวลผล เช่น sum, mean, sd เป็นต้น
- ... คือ อากิวเมนต์ที่ต้องการ Vectorize
- **MoreArgs** คือ เป็นลิสต์เวกเตอร์อากิวเมนต์

ตัวอย่างเช่น การประยุกต์ใช้คำสั่ง **mapply()** แสดงได้ดังนี้

```
R> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2
```

```
[[3]]
[1] 3 3

[[4]]
[1] 4
```

เป็นการเรียกใช้ฟังก์ชัน **rep()** โดยผ่านเวกเตอร์ที่ต้องการเรียกใช้ฟังก์ชันดังกล่าว (Vectorization) ซึ่งเท่ากับ เวกเตอร์ 1:4 หรือก็คือ c(1, 2, 3, 4) นั้นเอง และกำหนดให้อากิวเมนต์สำหรับฟังก์ชันด้วยเวกเตอร์ 4:1 หรือก็คือ c(4, 3, 2, 1) ผลที่ได้จากสร้างสมาชิกในลิสต์ตัวที่หนึ่งคือ 1 1 1 1 (นำค่า 1 มาทำซ้ำ 4 ครั้ง) สมาชิกในลิสต์ตัวที่สองคือ 2 2 2 (นำค่า 2 มาทำซ้ำ 3 ครั้ง) สมาชิกในลิสต์ตัวที่สามคือ 3 3 (นำค่า 3 มาทำซ้ำ 2 ครั้ง) และสมาชิกในลิสต์ตัวสุดท้ายคือ 4 (นำค่า 4 มาทำซ้ำ 1 ครั้ง)

6.2.3.5 เปรียบเทียบการใช้ฟังก์ชัน lapply() sapply() และ mapply()

เพื่อให้ผู้ใช้เห็นความแตกต่างระหว่าง ฟังก์ชัน **lapply() sapply()** และ **mapply()** ตัวอย่างต่อไปนี้ แสดงการเปรียบเทียบการใช้ฟังก์ชันดังกล่าว

```
R> require(tidyverse)
R> data <- mtcars

# Function to apply
R> mpg_category <- function(mpg) {
        if (mpg > 30) {
             return("High")
        } else if (mpg > 20) {
             return("Medium")
        }
        return("Low")
   }

# Apply to each element
R> lapply(data$mpg, FUN = mpg_category)
[[1]]
[1] "Medium"

[[2]]
[1] "Medium"

[[3]]
[1] "Medium"

[[4]]
[1] "Medium"

[[5]]
[1] "Low"
...
R> sapply(data$mpg, FUN = mpg_category)
[1] "Medium" "Medium" "Medium" "Medium" "Low"    "Low"    "Low"    "Medium"
```

```
 [9] "Medium" "Low"    "Low"    "Low"    "Low"    "Low"    "Low"    "Low"
[17] "Low"    "High"   "High"   "High"   "Medium" "Low"    "Low"    "Low"
[25] "Low"    "Medium" "Medium" "High"   "Low"    "Low"    "Low"    "Medium"
```

คำสั่งแรกเป็นการกำหนดให้ตัวแปร data มีค่าเท่ากับข้อมูล mtcars คำสั่งชุดที่สองเป็นการสร้างฟังก์ชันชื่อ **mpg_category()** ในการคืน (Return) ค่าระยะทางต่อน้ำมันที่ใช้เป็น "High", "Medium", และ "Low" ตามลำดับ คำสั่ง **lapply()** ส่งค่า **data$mpg** ให้กับฟังก์ชัน **mpg_category()** ซึ่งจะคืนค่ากลับมาเป็นลิสต์ตามจำนวนเวกเตอร์ที่ส่งเข้าไป ส่วนคำสั่ง **sapply()** ส่งค่า **data$mpg** ให้กับฟังก์ชัน **mpg_category()** แต่จะคืนค่ากลับมาเป็นเวกเตอร์แทน

สามารถเพิ่มอากิวเมนต์เข้าไปในคำสั่ง **sapply()** ได้ดังนี้

```
# Pass additional arguments after FUN
R> within_range <- function(mpg, low, high) {
     if (mpg >= low & mpg <= high) {
        return(TRUE)
     }
     return(FALSE)
   }

R> index <- sapply(data$mpg, FUN = within_range, low = 15, high = 20)
R> index
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
[12]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[23]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
R> data[index, ]
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

คำสั่งแรกเป็นการสร้างฟังก์ชันชื่อ **within_range()** ในการคืน (Return) ค่าระยะทางต่อน้ำมันที่ใช้เป็นช่วงที่กำหนด โดยคืนค่า **TRUE** เมื่ออยู่ช่วงที่กำหนด และคืนค่า **FALSE** เมื่ออยู่นอกช่วงที่กำหนด คำสั่งถัดมาเป็นการประยุกต์ใช้คำสั่ง **sapply()** ส่งค่า **data$mpg** ให้กับฟังก์ชัน **within_range()** ซึ่งจะคืนค่ากลับมาเป็นเวกเตอร์ **TRUE/FALSE** และคำสั่งสุดท้ายเป็นการแสดงข้อมูลตามเงื่อนไข หรือเฉพาะ index เท่ากับ **TRUE**

สามารถเพิ่มอากิวเมนต์เข้าไปในคำสั่ง **sapply()** ได้ดังนี้

```
# Use mapply to apply a function along multiple vectors
R> mpg_within_standard_range <- function(mpg, cyl) {
        if (cyl == 4) {
           return(within_range(mpg, low = 23, high = 31))
        } else if (cyl == 6) {
           return(within_range(mpg, low = 18, high = 23))
        }
        return(within_range(mpg, low = 13, high = 18))
   }

R> index <- mapply(FUN = mpg_within_standard_range, mpg = data$mpg, cyl =
   data$cyl)
R> index
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
[12]  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE
[23]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
R> data[!index, ]
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

คำสั่งแรกเป็นการสร้างฟังก์ชันชื่อ **mpg_within_standard_range()** ในการคืน (Return) ค่าฟังก์ชัน **within_range()** ดังที่ได้กล่าวไว้ข้างต้น คำสั่งถัดมาเป็นการประยุกต์ใช้คำสั่ง **mapply()** พารามิเตอร์ตัวแรกเป็นการส่งค่าฟังก์ชันที่ต้องการประมวลผลด้วยอากิวเมนต์ **FUN** แล้วตามด้วยเวกเตอร์ที่ต้องการส่งเข้าไปในฟังก์ชันดังกล่าว ได้แก่ **data$mpg** และ **data$cyl** ตามลำดับ คำสั่งสุดท้ายเป็นการแสดงข้อมูลตามเงื่อนไข หรือ ไม่มีใน index

6.3 คำสั่ง break, next และ repeat

6.3.1 break and next

โดยปกติลูป **for** จะหยุดทำงานเมื่อ Index มากกว่าจำนวนสมาชิกในเวกเตอร์ในขณะที่ลูป **while** จะหยุดทำงานเมื่อเงื่อนไข (Condition) เป็นเท็จ (**FALSE**) แต่ผู้ใช้สามารถสั่งให้โปรแกรมออกจากลูปดังกล่าวได้โดยใช้คำสั่ง **break** ดังตัวอย่างเช่น กำหนดให้ my_value และ my_vector มีค่าดังนี้

```
R> my_value <- 20
R> my_vector <- c(3, 2, 1, 0, -1, -2)
```

เมื่อต้องการหาร my_value ด้วย my_vector โดยหารด้วยสมาชิกในเวกเตอร์ทีละตัวด้วยการสร้างลูป **for** ถ้าค่าจากการหารเป็น **Inf** ซึ่งเกิดจากการหารด้วยศูนย์ ตรวจสอบได้โดยใช้ฟังก์ชัน **is.finite()** จะสั่งให้โปรแกรมหยุดทำงานแล้วออกจากลูป **for** ดังนี้

```
R> result <- rep(NA, length(my_vector))
R> result
[1] NA NA NA NA NA NA

R> for (i in 1:length(my_vector)) {
     value <- my_value/my_vector[i]
     if (is.finite(value)) {
        result[i] <- value
     } else {
        break  # break and exit for loop
     }
   }
R> result
[1]  6.666667 10.000000 20.000000        NA        NA        NA
```

จากคำสั่งข้างต้น คำสั่งแรกเป็นการกำหนดค่า **NA** ให้กับเวกเตอร์ result ส่วนลูป **for** จะทำการหาร my_value ด้วยสมาชิก my_vector ทีละตัวจนกระทั่งถึงสมาชิกตัวที่ 4 จะเกิดการหารด้วยศูนย์ได้ค่าจากการหารเท่ากับ **Inf** ซึ่งจะถูกตรวจสอบด้วยเงื่อนไข **if** และฟังก์ชัน **is.finite()** พร้อมกับหยุดการทำงานและออกจากลูป **for** ทำให้ค่าสมาชิกใน result ที่เหลืออีก 3 ค่ายังค่าเท่ากับ **NA** เหมือนเดิม

ส่วนคำสั่ง **next** จะทำการหยุดการทำงานเฉพาะรอบ (Iteration) นั้น และทำงานรอบต่อไปจนกระทั่งครบจำนวนรอบตามที่กำหนดไว้ในลูป **for** หรือตามเงื่อนไขที่กำหนดไว้ในลูป **while** ดังตัวอย่างต่อไปนี้

```
R> result <- rep(NA, length(my_vector))
R> result
[1] NA NA NA NA NA NA

for (i in 1:length(my_vector)) {
  value <- my_value / my_vector[i]
  if (is.finite(value)) {
    result[i] <- value
  } else {
    next  # skip this iteration and continue the for loop
  }
}
R> result
[1]  6.666667  10.000000  20.000000         NA -20.000000 -10.000000
```

6.3.2 repeat

คำสั่ง **repeat** มีรูปแบบโดยทั่วไปดังนี้

```
repeat {
  # code here
}
```

คำสั่ง **repeat{ }** ไม่มีการกำหนดเงื่อนไขหรือตัวควบคุมใด ๆ โปรแกรมจะทำงานคำสั่งภายในเครื่องหมายวงเล็บปีกกา **{ }** ไปไม่สิ้นสุด ดังนั้นผู้ใช้จะต้องสั่งให้โปรแกรมสิ้นสุดการทำงาน ด้วยตัวเองโดยใช้คำสั่ง **break** ตัวอย่างเช่น การสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ซึ่งหาได้จากตัวเลขถัดไปเท่ากับผลบวกของตัวเลขสองตัวก่อนหน้านี้ และกำหนดให้สองตัวแรกคือ 0 และ 1 ตามลำดับ ผู้ใช้ต้องกำหนดเงื่อนไขการหยุดเอง โดยใช้คำสั่ง **break** ดังนี้

```
R> fib1 <- 1
R> fib2 <- 1
R> repeat {
     temp <- fib1 + fib2
     fib1 <- fib2
     fib2 <- temp
     cat(fib2, ", ", sep="")
     if (fib2 > 100) {
        cat("Stop...")
        break  # break and exit repeat loop
     }
   }
2, 3, 5, 8, 13, 21, 34, 55, 89, 144, Stop...
```

สองคำสั่งแรกเป็นการกำหนดค่าให้กับ fib1 และ fib2 เท่ากับ 1 ทั้งคู่ คำสั่ง **repeat** จะสั่งให้คำสั่งภายในเครื่องหมายวงเล็บปีกกา **{ }** ทำงานไปไม่สิ้นสุด คำสั่งภายในเครื่องหมาย **{ }** ประกอบด้วย สร้างตัวแปร temp สำหรับการบวก fib1 และ fib2 แล้วแทนที่ fib1 ด้วย fib2 และแทนที่ fib2 ด้วย temp นั้นคือ fib2 จะเป็นตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) แล้วจึงแสดงผลตัวเลขดังกล่าวออกทางหน้าจอด้วยฟังก์ชัน **cat()** แล้วจึงตรวจสอบว่า fib2 มากกว่าค่าที่กำหนด (ในตัวอย่างนี้คือ 100) หรือไม่ ถ้ามากกว่าก็ทำการหยุดการทำงานด้วยคำสั่ง **break** แล้วพิมพ์ "Stop..." เป็นการสิ้นสุดการทำงาน

6.4 ขอบเขต (Scoping)

การใช้งานโปรแกรม **R** ผู้ใช้จะต้องทำความเข้าใจขอบเขตของตัวแปร/วัตถุต่าง ๆ ที่อยู่ในโปรแกรม **R** ก่อน เพื่อผู้ใช้จะได้ประยุกต์ใช้โปรแกรม **R** ได้อย่างมีประสิทธิภาพ ลดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ ดังรายละเอียดต่อไปนี้

6.4.1 Environments

R จัดการขอบเขตของตัวแปร/วัตถุต่าง ๆ โดยแบ่งเป็น Environments 3 ส่วน ดังนี้

6.4.1.1 Global Environments

ตัวแปร/วัตถุต่าง ๆ ที่ผู้ใช้กำหนดขึ้นจะอยู่ในส่วนของ Global environment ใช้คำสั่ง `ls()` เพื่อแสดงตัวแปร/วัตถุที่อยู่ใน Global environment ปัจจุบัน ได้ดังนี้

```
R> my_value <- 5 + 10
R> my_string <- "Hello world"
R> ls()
[1] "my_string" "my_value"
```

6.4.1.2 Package Environments

ตัวแปร/วัตถุต่าง ๆ ของแพ็กเกจจะอยู่ในส่วนของ Package environment ใช้คำสั่ง `ls()` ตามด้วยชื่อแพ็กเกจ เพื่อแสดงตัวแปร/วัตถุที่อยู่ใน Package environment นั้น ตัวอย่างเช่น แสดงตัวแปร/วัตถุในแพ็กเกจชื่อ datasets ได้ดังนี้

```
R> ls("package:datasets")
  [1] "ability.cov"           "airmiles"              "AirPassengers"
  [4] "airquality"            "anscombe"              "attenu"
  [7] "attitude"              "austres"               "beaver1"
 [10] "beaver2"               "BJsales"               "BJsales.lead"
 #   ------------------------------------------------------------------------
 # Several rows are hidden
 # ------------------------------------------------------------------------
[97] "USPersonalExpenditure" "uspop"                "VADeaths"
[100] "volcano"               "warpbreaks"            "women"
[103] "WorldPhones"           "WWWusage""
```

6.4.1.3 Local Environments

ในการเรียกฟังก์ชัน **R** จะสร้างตัวแปร/วัตถุซึ่งจะอยู่ในส่วนของ Local environment หรือบางครั้งเรียกว่า Lexical environment ตัวแปร/วัตถุดังกล่าวจะมองเห็นหรือเข้าถึงได้เฉพาะภายในฟังก์ชันนั้น ๆ ไม่สามารถมองเห็นหรือเข้าถึงนอกเหนือขอบเขตของฟังก์ชันได้ ตัวอย่างเช่น ในการสร้างเมทริกซ์จะมีการผ่านพารามิเตอร์ที่เป็นเวกเตอร์ไปที่ตัวแปรชื่อ data ดังนี้

```
R> my_matrix <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
R> my_matrix
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

เวกเตอร์ data ไม่สามารถมองเห็นหรือเข้าถึงนอกเหนือขอบเขตของฟังก์ชัน `matrix()` ที่เรียกใช้ได้

6.4.2 ลำดับในการค้นหา Path

ในการเข้าถึงคำสั่ง ตัวแปร/วัตถุ หรือฟังก์ชันที่นอกเหนือจากตัวแปร/วัตถุใน Global environment โปรแกรม **R** จะทำการค้นหาคำสั่ง ตัวแปร/วัตถุ หรือฟังก์ชันดังกล่าวโดยจะค้นหาใน environments ตามลำดับใน path ซึ่งดูได้จากคำสั่ง `search()` ดังนี้

```
R> search()
 [1] ".GlobalEnv"        "package:ggplot2"   "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

จากตัวอย่างข้างต้น ถ้าไม่พบคำสั่งใน Global environment (`".GlobalEnv"`) โปรแกรม **R** จะทำการค้นหาใน Package environment ถัดไปชื่อ `ggplot2` (`"package:ggplot2"`) จนถึง Package environment ชื่อ base (`"package:base"`) ตามลำดับ

ผู้ใช้สามารถตรวจสอบว่าคำสั่งที่ใช้อยู่ใน Environment ไหนได้จากคำสั่ง `environment()` ดังตัวอย่างต่อไปนี้

```
R> environment(plot)
<environment: namespace:graphics>

R> environment(matrix)
<environment: namespace:base>
```

จะเห็นว่าคำสั่ง `plot()` อยู่ในแพ็กเกจชื่อ graphics ส่วนคำสั่ง `matrix()` อยู่ในแพ็กเกจชื่อ base

6.4.3 คำสงวน (Reserved Words)

ในโปรแกรม **R** มีคำสงวน (Reserved words) บางคำที่ผู้ใช้ไม่สามารถนำไปใช้เป็นชื่อตัวแปรที่กำหนดขึ้นเองได้ ได้แก่ คำสงวนต่อไปนี้

- `if` และ `else`
- `for`, `while` และ `in`
- `function`
- `repeat`, `break` และ `next`
- `TRUE` และ `FALSE`
- `Inf` และ `-Inf`
- `NA`, `NaN` และ `NULL`

เนื่องจาก **R** แยกความแตกต่างระหว่างอักษรตัวใหญ่และอักษรตัวเล็ก ดังนั้นการตั้งชื่อตัวแปรชื่อเดียวกับคำสงวนแต่มีอักษรตัวใหญ่และอักษรตัวเล็กไม่เหมือนกันก็เป็นไปได้ แต่ผู้เขียนไม่แนะนำ เพราะจะทำให้ผู้ใช้สับสนได้ง่าย รวมทั้งไม่แนะนำให้กำหนดค่าจริง/เท็จโดยใช้อักษรย่อ `T` และ `F` เช่น

```
R> True <- "True Value"  # not recommend
R> na <- "Not recommend"
R> T <- 12  # not recommend
R> F <- TRUE  # not recommend
R> ls()
[1] "F"         "my_matrix" "my_string" "my_value"  "na"        "T"
[7] "True"
```

ผู้ใช้สามารถลบตัวแปร/วัตถุออกจาก Global environment โดยใช้คำสั่ง **rm()** และระบุพารามิเตอร์ **list = ls()** ดังนี้

```
R> rm(list = ls())
R> ls()
character(0)
```

จากคำสั่งข้างบนจะเห็นว่าเมื่อใช้คำสั่ง **ls()** จะได้ Global environment ที่ว่างเปล่า ซึ่งแสดงว่าไม่มีอักขระอะไรอยู่เลย (character(0))

6.5 การระบุพารามิเตอร์ (Parameter Identification)

การระบุพารามิเตอร์ (Parameter identification) สามารถกำหนดได้หลายลักษณะ โดยใช้ชื่อเต็ม ชื่อย่อ ไม่ใช้ชื่อเลย หรือผสมกันก็ได้ ดังนี้

6.5.1 การระบุพารามิเตอร์แบบเต็ม (Exact Identification)

การระบุพารามิเตอร์แบบเต็ม (Exact identification) มีประโยชน์หลายประการ ได้แก่

- ช่วยลดข้อผิดพลาดในการระบุพารามิเตอร์เมื่อเทียบกับการระบุพารามิเตอร์แบบอื่น
- ลำดับของพารามิเตอร์แต่ละตัวไม่มีความสำคัญ สามารถกำหนดให้พารามิเตอร์ตัวไหนขึ้นก่อน/หลังก็ได้
- เมื่อฟังก์ชันมีพารามิเตอร์หลายตัว ผู้ใช้สามารถระบุเฉพาะพารามิเตอร์ที่ต้องการได้ ส่วนพารามิเตอร์ตัวอื่นก็ใช้ค่าโดยปริยาย

แต่อย่างไรก็ตาม การระบุพารามิเตอร์แบบเต็มก็มีข้อเสียอยู่บ้างตรงที่เสียเวลาและดูยุ่งยากแม้แต่การใช้คำสั่งทั่วไป ผู้ใช้ต้องจำชื่อพารามิเตอร์ และพิมพ์ให้ถูกต้องไม่ว่าจะเป็นอักษรตัวใหญ่หรืออักษรตัวเล็ก (Case-sensitive)

ตัวอย่างในการสร้างเมทริกซ์ สามารถระบุพารามิเตอร์แบบเต็ม ได้ดังตัวอย่างต่อไปนี้

```
R> my_matrix <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE,
                       dimnames = list(c("Row1", "Row2", "Row3"),
                                       c("A", "B", "C")))
R> my_matrix
     A B C
Row1 1 2 3
Row2 4 5 6
```

```
Row3 7 8 9
```

สามารถสลับลำดับของพารามิเตอร์ได้ แต่ต้องใส่ชื่อพารามิเตอร์ (Named parameter) ด้วย โดยผลลัพธ์ไม่เปลี่ยนแปลง ดังนี้

```
R> my_matrix <- matrix(ncol = 3, nrow = 3, data = 1:9, byrow = TRUE,
                       dimnames = list(c("Row1", "Row2", "Row3"),
                                       c("A", "B", "C")),
R> my_matrix
     A B C
Row1 1 2 3
Row2 4 5 6
Row3 7 8 9
```

6.5.2 การระบุพารามิเตอร์แบบย่อ (Abbreviated Identification)

การระบุพารามิเตอร์แบบย่อ (Abbreviation identification) คล้ายกับการระบุพารามิเตอร์แบบเต็ม แตกต่างกันที่ชื่อพารามิเตอร์ จะมีเพียง 2-3 ตัวอักษร ให้สามารถแยกความแตกต่างระหว่างชื่อพารามิเตอร์ได้ก็พอ จากตัวอย่างก่อนหน้านี้ ถ้าเป็นการระบุพารามิเตอร์แบบย่อ จะเขียนได้ดังนี้

```
R> my_matrix <- matrix(1:9, nr = 3, nc = 3, by = TRUE,
                       di = list(c("Row1", "Row2", "Row3"),
                                 c("A", "B", "C")))
R> my_matrix
     A B C
Row1 1 2 3
Row2 4 5 6
Row3 7 8 9
```

การระบุพารามิเตอร์แบบย่อมีข้อดีตรงเหมือนกับการระบุพารามิเตอร์แบบเต็ม แตกต่างกันที่การระบุพารามิเตอร์แบบย่อเขียนเป็นคำสั่งได้สั้นกว่าการระบุพารามิเตอร์แบบเต็ม แต่ก็มีข้อเสียตรง ได้แก่

- ชื่อตัวแปรที่สั้นเกินไปอาจทำให้ผู้ใช้สับสนได้ และยากต่อการจดจำ
- การกำหนดชื่อตัวแปรที่สั้นเกินไปอาจทำให้เกิดข้อผิดพลาดได้ง่าย เช่น

```
R> my_matrix <- matrix(dat = 1:9, nr = 3, nc = 3, by = TRUE,
                       d = list(c("Row1", "Row2", "Row3"),
                                c("A", "B", "C")))
Error in matrix(dat = 1:9, nr = 3, nc = 3, by = TRUE, d = list(c("Row1",  :
  formal parameter "data" matched by multiple actual parameters
```

6.5.3 การระบุพารามิเตอร์ตามตำแหน่ง (Positional Identification)

การระบุพารามิเตอร์ตามตำแหน่ง (Position identification) เป็นการระบุพารามิเตอร์โดยไม่กำหนดชื่อพารามิเตอร์ ฟังก์ชันจะแยกความแตกต่างพารามิเตอร์แต่ละตัวตามลำดับก่อน/หลังแทน ผู้ใช้สามารถดูลำดับก่อน/หลังได้จาก Usage ใน Help ของฟังก์ชันนั้น ๆ หรือใช้คำสั่ง `args()` พร้อมกับระบุชื่อฟังก์ชัน เช่น

```
R> args(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
NULL
```

คำสั่งข้างต้นจะแสดงลำดับของพารามิเตอร์แต่ละตัว เริ่มต้นด้วย data ตามด้วย nrow, ncol, byrow และ dimnames ตามลำดับ

ตัวอย่างก่อนหน้านี้ เขียนเป็นคำสั่งโดยวิธีการระบุพารามิเตอร์ตามตำแหน่ง ได้ดังนี้

```
R> my_matrix <- matrix(1:9, 3, 3, TRUE, list(c("Row1", "Row2", "Row3"),
                                             c("A", "B", "C")))
R> my_matrix
     A B C
Row1 1 2 3
Row2 4 5 6
Row3 7 8 9
```

การระบุพารามิเตอร์ตามตำแหน่งมีข้อดีคือ เขียนคำสั่งได้สั้น กระชับ ไม่ต้องพิมพ์ชื่อพารามิเตอร์ ดังนั้นจึงเหมาะกับคำสั่งที่ไม่ซับซ้อน

ถึงแม้ว่าพารามิเตอร์บางตัวมีค่าโดยปริยายกำหนดไว้ แต่ผู้ใช้ก็จำเป็นต้องระบุพารามิเตอร์ดังกล่าวด้วย เช่น

```
R> my_matrix <- matrix(1:9, 3, 3, list(c("Row1", "Row2", "Row3"),
                                       c("A", "B", "C")))
Error in matrix(1:9, 3, 3, list(c("Row1", "Row2", "Row3"), c("A", "B",  :
  invalid 'byrow' parameter
```

ตัวอย่างข้างบน ผู้ใช้ไม่ได้ระบุค่า **TRUE** หรือ **FALSE** ให้กับพารามิเตอร์ชื่อ byrow ดังนั้นโปรแกรม **R** จึงแสดงข้อผิดพลาดขึ้น สรุปข้อเสียของการระบุพารามิเตอร์ตามตำแหน่ง คือ

- ผู้ใช้ต้องระบุพารามิเตอร์ให้ครบและถูกต้องตามลำดับ
- ผู้ใช้ที่ไม่คุ้นเคยกับฟังก์ชันนั้น ๆ จะต้องจดจำลำดับของพารามิเตอร์ให้ได้ ไม่อย่างนั้นจะเกิดข้อผิดพลาดได้

6.5.4 การระบุพารามิเตอร์แบบผสม (Mixed Identification)

การระบุพารามิเตอร์แบบผสม (Mixed identification) เป็นการระบุพารามิเตอร์ตามตำแหน่งผสมกับการระบุพารามิเตอร์แบบเต็มหรือแบบย่อ เช่น

```
R> my_matrix <- matrix(1:9, 3, 3, dim = list(c("Row1", "Row2", "Row3"),
                                             c("A", "B", "C")))
R> my_matrix
     A B C
Row1 1 2 3
Row2 4 5 6
Row3 7 8 9
```

พารามิเตอร์ 3 ตัวแรกเป็นการระบุตามตำแหน่ง ในขณะเดียวกันใช้การระบุแบบย่อ โดยใช้พารามิเตอร์ชื่อ dim ซึ่งเป็นตัวย่อสามตัวแรกของ dimnames และไม่มีการระบุพารามิเตอร์ byrow

6.5.5 การระบุพารามิเตอร์แบบเปิด (Open Identification)

ฟังก์ชันบางประเภทไม่สามารถกำหนดจำนวนพารามิเตอร์ล่วงหน้าได้แน่นอน จึงต้องมีการระบุอีกแบบหนึ่งเรียกว่า การระบุพารามิเตอร์แบบเปิด (Open identification) ตัวอย่างเช่น ฟังก์ชัน **c()** **data.frame()** และ **list()** ในการระบุพารามิเตอร์แบบเปิดจะใช้สัญลักษณ์สามจุด (...) (Dot-Dot-Dot หรือเรียกว่า Ellipses)

```
R> args(data.frame)
function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
    fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())
NULL
```

พารามิเตอร์ที่ไม่ตรงกับชื่อที่กำหนดไว้ในฟังก์ชัน **data.frame()** สามารถรับพารามิเตอร์ดังกล่าวได้โดยใช้พารามิเตอร์สามจุด (...)

พารามิเตอร์แบบเปิด โดยปกติแบ่งเป็นสองประเภทคือ พารามิเตอร์แบบเปิดที่รับค่าหลักเข้ามาในฟังก์ชัน ตัวอย่างเช่น พารามิเตอร์ในฟังก์ชัน **c()** **data.frame()** และ **list()** พารามิเตอร์แบบเปิดอีกประเภทหนึ่งจะรับค่าเสริมเข้ามาในฟังก์ชัน เช่น พารามิเตอร์ในฟังก์ชัน **plot()** พารามิเตอร์แบบเปิดที่รับค่าเสริมเข้ามาในฟังก์ชันมักจะเป็นพารามิเตอร์ทางเลือก (Optional parameters) เช่น

```
R > args(plot)
function (x, y, ...)
NULL
```

6.6 การสร้างฟังก์ชัน

การสร้างฟังก์ชันมีรูปแบบโดยทั่วไปดังนี้

```
functionName <- function(arg1, arg2, arg3, ...) {
  # code here
  return(returnObject)
}
```

functionName เป็นตัวแปรที่เก็บฟังก์ชันการทำงานที่สร้างขึ้น ตามด้วยเครื่องหมายกำหนดค่า (**<-**) และคำสั่ง **function()** ตามด้วยพารามิเตอร์คั่นด้วยเครื่องหมายจุลภาค (Comma) ถ้าไม่มีพารามิเตอร์ที่ผ่านค่ามาก็ใส่แค่เครื่องหมายวงเล็บ **()** ส่วนตัวชุดคำสั่งจะอยู่ภายในเครื่องหมายวงเล็บปีกกา **{ }** ซึ่งสามารถใช้คำสั่งต่าง ๆ ที่ผ่านมา เช่น **for**, **while**, **if** อยู่ภายในเครื่องหมาย **{ }** ได้ ฟังก์ชันสามารถส่งค่ากลับ (return) ได้โดยการกำหนดวัตถุที่ต้องการส่งกลับด้วยคำสั่ง **return()** เช่น returnObject

เมื่อมีการเรียกใช้ฟังก์ชัน ผู้ใช้จะเรียก `functionName()` พร้อมกับส่งค่าพารามิเตอร์ที่ต้องการมาในเครื่องหมายวงเล็บ `()` เช่น arg1, arg2, arg2, ... ตามลำดับ

ตัวอย่างการสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ในหัวข้อ 6.3.2 เขียนเป็นฟังก์ชันได้ดังนี้

```
R> my_fib <- function() {
     fib1 <- 1
     fib2 <- 1
     repeat {
        temp <- fib1 + fib2
        fib1 <- fib2
        fib2 <- temp
        cat(fib2, ", ", sep="")
        if(fib2 > 100) {
           cat("Stop...")
           break
        }
     }
   }
```

คำสั่งแรกเป็นการกำหนดตัวแปรชื่อ my_fib เก็บฟังก์ชันที่สร้างขึ้น โดยมีชุดคำสั่งอยู่ภายในเครื่องหมาย `{ }` เมื่อต้องการเรียกใช้งานฟังก์ชัน ก่อนอื่นต้องรันฟังก์ชันเพื่อให้ R รู้จักฟังก์ชันที่กำหนดขึ้นมาก่อน หลังจากนั้นผู้ใช้จึงเรียกฟังก์ชันที่สร้างไว้มาทำงานได้ ดังนี้

```
R> my_fib()
2, 3, 5, 8, 13, 21, 34, 55, 89, 144, Stop...
```

6.7 การกำหนดพารามิเตอร์ให้ฟังก์ชัน (Adding Parameters)

แทนที่จะกำหนดค่าตายตัวให้กับการหยุดการทำงานฟังก์ชันเหมือนชุดคำสั่งในหัวข้อก่อนหน้านี้ ผู้ใช้สามารถกำหนดพารามิเตอร์เข้าไปในฟังก์ชันได้ ตัวอย่างเช่น

```
R> my_fib2 <- function(stop_value) {
     fib1 <- 1
     fib2 <- 1
     repeat {
        temp <- fib1 + fib2
        fib1 <- fib2
        fib2 <- temp
        cat(fib2, ", ", sep="")
        if(fib2 > stop_value) {
           cat("Stop...")
           break
        }
     }
   }
```

ชุดคำสั่งข้างบนจะเห็นว่ามีการกำหนดพารามิเตอร์ (Formal parameter) ชื่อ stop_value เพิ่มเข้ามาในคำสั่ง **function()** เมื่อเรียกใช้งานฟังก์ชัน ผู้ใช้ต้องกำหนดค่าพารามิเตอร์ (Actual parameters) บางครั้งเรียกว่าอาร์กิวเมนต์ (Arguments) ให้กับฟังก์ชัน ดังนี้

```
R> my_fib2(stop_value = 50)
2, 3, 5, 8, 13, 21, 34, 55, Stop...

R> my_fib2(5000)  # option without a keyword
2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,
   Stop...
```

คำสั่งแรกเป็นการระบุพารามิเตอร์แบบเต็ม (Exact identification) โดยระบุค่าพารามิเตอร์ชื่อ stopValue เท่ากับ 50 ส่วนคำสั่งที่สองกำหนดค่าพารามิเตอร์ให้กับฟังก์ชันแบบตามตำแหน่ง (Positional identification) โดยกำหนดค่าให้เท่ากับ 5,000

R สามารถกำหนดค่าโดยปริยายให้กับพารามิเตอร์แต่ละตัวได้ จากตัวอย่างนี้ผู้ใช้จะกำหนดค่าโดยปริยายของพารามิเตอร์ stopValue ให้เท่ากับ 10,000 ซึ่งหมายความว่า ถ้าเรียกใช้ฟังก์ชันโดยไม่ส่งค่าพารามิเตอร์เข้ามา ฟังก์ชันนี้จะใช้ค่าพารามิเตอร์ **stop_value = 10000** โดยอัตโนมัติ ดังนี้

```
R> my_fib3 <- function(stop_value = 10000) {
     fib1 <- 1
     fib2 <- 1
     repeat {
       temp <- fib1 + fib2
       fib1 <- fib2
       fib2 <- temp
       cat(fib2, ", ", sep = "")
       if(fib2 > stop_value) {
         cat("Stop...")
         break
       }
     }
   }

R> my_fib3()
2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765,
   10946, Stop...
```

6.8 การส่งค่ากลับ (Return an Object)

จากตัวอย่างในหัวข้อก่อนหน้านี้ เมื่อต้องการผลลัพธ์ที่ได้จากฟังก์ชันไปใช้งานต่อ แทนที่จะพิมพ์ผลลัพธ์ออกทางจอภาพ ผู้ใช้สามารถเก็บค่าดังกล่าวไว้ในตัวแปร เพื่อนำไปใช้งานต่อได้โดยการคืนค่ากลับมาที่ตัวฟังก์ชัน โดยใช้คำสั่ง **return()** แต่ถ้าค่าที่ต้องการส่งค่ากลับอยู่ที่บรรทัดสุดท้าย **R** ไม่จำเป็นต้องมีคำสั่งดังกล่าวก็ได้ ดังนี้

```
R> my_fib4 <- function(stop_value = 10000) {
     fib <- c(1, 1)
     counter <- 2
```

```
     repeat {
        fib <- c(fib, fib[counter-1] + fib[counter])
        counter <- counter + 1
        if(fib[counter] > stop_value) {
           break
        }
     }
     fib
     # equivalent to return(fib)
   }
```

คำสั่งภายในฟังก์ชัน my_fib4 คำสั่งแรกเป็นการสร้างเวกเตอร์ fib และกำหนดค่าให้เท่ากับ 1 สองตัวด้วยคำสั่ง `c(1, 1)` และสร้างตัวแปร counter กำหนดค่าเริ่มต้นให้เท่ากับ 2 แล้วจึงสร้างชุดคำสั่งให้ทำซ้ำภายในคำสั่ง `repeat{}` โดยกำหนดค่า fib ใหม่โดยการแทนด้วยเวกเตอร์ `c(fib, fib[counter-1] + fib[counter])` ซึ่งหมายถึง การเพิ่มสมาชิกตัวใหม่ให้กับเวกเตอร์ fib เดิมด้วยผลรวมของสมาชิกในเวกเตอร์ก่อนหน้านี้สองตัว หลังจากนั้นเพิ่มค่าของตัวแปร counter โดยการบวกด้วย 1 แล้วจึงตรวจสอบว่าสมาชิกในเวกเตอร์ fib ตัวสุดท้ายมากกว่าค่าที่กำหนดไว้ (stop_value) หรือไม่ ถ้ามากกว่าก็หยุดการทำงานด้วยคำสั่ง `break` แล้วคืนค่าเวกเตอร์ fib กลับด้วยคำสั่ง `return(fib)` ไปที่ฟังก์ชันที่เรียกใช้งาน ถ้าไม่มากกว่าก็ทำซ้ำภายในคำสั่ง `repeat{}`

ตัวอย่างการใช้งานฟังก์ชัน my_fib4 แสดงได้ดังนี้

```
R> fibonacci <- my_fib4(50000)
R> fibonacci
 [1]     1     1     2     3     5     8    13    21    34    55
[11]    89   144   233   377   610   987  1597  2584  4181  6765
[21] 10946 17711 28657 46368 75025
```

คำสั่งข้างบนเป็นการเรียกฟังก์ชัน my_fib4 มาใช้สร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) แล้วเก็บไว้ในเวกเตอร์ชื่อ fibonacci หลังจากนั้นต้องการใช้ชุดตัวเลขดังกล่าว 10 ตัวแรกโดยเก็บไว้ในเวกเตอร์ชื่อ my_fib ด้วยการสับเซตเฉพาะสมาชิก 10 แรกที่ต้องการด้วยคำสั่ง `fibonacci[1:10]` ดังนี้

```
R> my_fib <- fibonacci[1:10]
R> my_fib
 [1]  1  1  2  3  5  8 13 21 34 55
```

ในการส่งค่ากลับ ถ้าในตัวฟังก์ชันไม่ระบุด้วยคำสั่ง `return()` โปรแกรม R จะคืนค่าบรรทัดสุดท้ายในตัวฟังก์ชันแทน เช่น

```
R> my_func1 <- function() {
     my_value <- 12
     my_string <- "Hello world"
     my_vector <- 1:10
   }

R> my_func2 <- function() {
     my_value <- 12
     my_string <- "Hello world"
```

```
     my_vector <- 1:10
     return(my_vector)
   }
```

เมื่อเรียกใช้ฟังก์ชันดังกล่าว จะได้ผลลัพธ์เหมือนกัน ดังนี้

```
R> my_vector1 <- my_func1()
R> my_vector1
 [1]  1  2  3  4  5  6  7  8  9 10

R> my_vector2 <- my_func2()
R> my_vector2
 [1]  1  2  3  4  5  6  7  8  9 10
```

ฟังก์ชันจะสิ้นสุดการทำงานเมื่อพบคำสั่ง **return()** ตัวแรก โดยไม่ทำงานในคำสั่งใด ๆ ต่อจากนั้นเลย เช่น

```
R> my_func3 <- function() {
     my_value <- 12
     my_string <- "Hello world"
     return(my_string)  # return and do not execute the rest command(s)
     my_vector <- 1:10
     return(my_vector)
}
```

ฟังก์ชันข้างต้น เมื่อเรียกใช้ จะได้ผลดังนี้

```
R> my_value2 <- my_func3()
R> my_value2
[1] "Hello world"
```

จะเห็นว่าฟังก์ชันดังกล่าวสิ้นสุดการทำงานเมื่อพบคำสั่ง **return( )** ตัวแรก ซึ่งก็คือ **return(my_string)** แล้วออกจากฟังก์ชันทันที โดยไม่ทำงานในคำสั่งถัดไปเลย

R สามารถจัดการกับพารามิเตอร์แบบเปิดโดยใช้สัญลักษณ์สามจุด (...) (Dot-Dot-Dot หรือ Ellipses) โดยการสร้างชุดคำสั่งเพื่อรองรับการทำงานของพารามิเตอร์แบบเปิด ดังตัวอย่างต่อไปนี้

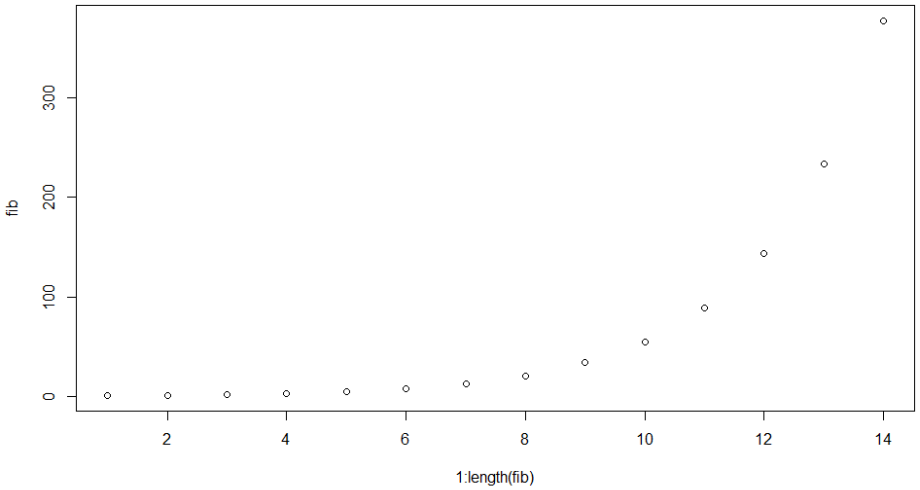
```
R> my_fib5 <- function(stop_value = 10000, show_plot = TRUE, ...) {
     fib <- c(1, 1)
     counter <- 2
     repeat {
        fib <- c(fib, fib[counter-1] + fib[counter])
        counter <- counter + 1
        if(fib[counter] > stop_value) {
           break
        }
     }

     if(show_plot) {
        plot(1:length(fib), fib, ...)  # ... to deal with open-ended parameters
     } else {
        return(fib)
     }
```

```
}
```

เมื่อเรียกใช้ฟังก์ชัน my_fib5 โปรแกรมจะทำการสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ตามที่กำหนด และสร้างกราฟตัวเลขลำดับฟีโบนัชชี แสดงได้ดังรูปต่อไปนี้

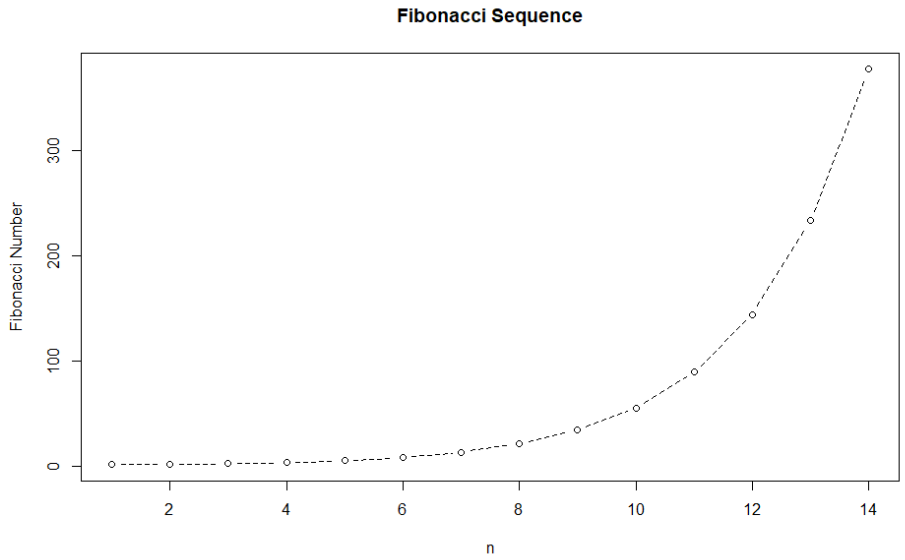
```
R> my_fib5(300)
```



รูปที่ 6-1 ฟังก์ชันแสดงกราฟที่ไม่กำหนดพารามิเตอร์แบบเปิด

กราฟข้างต้นเป็นกราฟที่แสดงโดยใช้ค่าโดยปริยายของคำสั่ง `plot()` ผู้ใช้สามารถกำหนดพารามิเตอร์ต่าง ๆ ให้กับกราฟได้ เช่น ตั้งชื่อกราฟ ตั้งชื่อแกน x-y ลักษณะจุด ชนิดของเส้น เช่น

```
R> my_fib5(300, type = "b", lty = 2, main = "Fibonacci Sequence",
           ylab = "Fibonacci Number", xlab = "n")
```



รูปที่ 6-2 ฟังก์ชันแสดงกราฟที่มีการกำหนดพารามิเตอร์แบบเปิด

6.9 ฟังก์ชันที่ไม่กำหนดชื่อ (Anonymous Functions)

บางครั้งผู้ใช้ต้องการสร้างฟังก์ชันขึ้นมาใช้งานชั่วคราว โดยไม่ต้องการเก็บฟังก์ชันดังกล่าวไว้ในหน่วยความจำของเครื่องคอมพิวเตอร์ โดยมากมักเป็นการผ่านค่าให้กับฟังก์ชันอื่น ๆ เช่น ฟังก์ชัน `apply()` ผู้ใช้สามารถสร้างฟังก์ชันโดยไม่ต้องกำหนดชื่อฟังก์ชันได้ (Anonymous functions) ได้ดังนี้

```
R> my_matrix <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
R> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

R> apply(my_matrix, MARGIN = 2, FUN = function(x) {rep(x, 2)})
   [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]    1    2    3    4
[5,]    5    6    7    8
[6,]    9   10   11   12
```

ฟังก์ชัน `apply()` ข้างต้นทำการสร้างเมทริกซ์เหมือนกับเมทริกซ์ตั้งต้น โดยการสร้างฟังก์ชันที่ไม่มีชื่อผ่านทางพารามิเตอร์ `FUN` แล้วใช้คำสั่งให้ทำซ้ำ 2 ครั้ง `rep(x, 2)`

6.10 ฟังก์ชันเรียกตัวเอง (Recursive Functions)

ฟังก์ชันการสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) สามารถเขียนได้อีกลักษณะหนึ่งโดยการเรียกตัวเอง ได้ดังนี้

```
R> my_fib6 <- function(stop_value) {
     if (stop_value == 1 || stop_value == 2) {
       return(1)
     } else {
       return(my_fib6(stop_value-1) + my_fib6(stop_value-2))
     }
   }
```

ในการเรียกใช้ฟังก์ชัน `my_fib6()` จะต้องผ่านพารามิเตอร์ลำดับที่ของตัวเลขลำดับฟีโบนัชชีเข้าในฟังก์ชัน เช่น ต้องการแสดงตัวเลขฟีโบนัชชีลำดับที่ 10 ผู้ใช้สามารถเรียกใช้ฟังก์ชัน ได้ดังนี้

```
R> my_fib6(10)
[1] 55
```

6.11 การจัดการความผิดปกติ (Exception Handing)

6.11.1 ความผิดพลาด (Errors) และข้อความเตือน (Warnings)

เหมือนกับภาษาโปรแกรมอื่น ๆ **R** สามารถจัดการกับความผิดปกติได้ โดยการแสดงความผิดพลาด (Errors) หรือข้อความเตือน (Warnings) ผู้ใช้สามารถตรวจสอบความผิดพลาดที่เกิดขึ้น แล้วแสดงข้อความความผิดพลาด (Errors) ที่เกิดขึ้นได้โดยใช้คำสั่ง **stop()** เช่น

```
R> show_error <- function(x) {
     if (x <= 0) {
        stop("value is less than or equal to 0\nSTOP...")
     }
     x
   }

R> show_error(-3)
Error in show_error(-3) : value is less than or equal to 0
STOP...

R> show_error(0)
Error in show_error(0) : value is less than or equal to 0
STOP...
```

หรือแสดงข้อความเตือน (Warnings) โดยใช้คำสั่ง **warning()** เช่น

```
R> show_warning <- function(x) {
     if (x <= 0) {
        warning("value is less than or equal to 0, set the value to 1 and
                continute...")
        x <- 1
     }
     x
   }

R> show_warning(-2)
[1] 1
Warning message:
In showWarning(-2) :
  value is less than or equal to 0, set the value to 1 and continute...
```

จะเห็นว่าในการแสดงข้อความความผิดพลาด (Errors) โดยใช้คำสั่ง **stop()** โปรแกรม **R** จะสิ้นสุดการทำงานเมื่อเจอคำสั่ง **stop()** ส่วนการแสดงข้อความเตือน (Warnings) โดยใช้คำสั่ง **warning()** โปรแกรมจะทำงานต่อไปจนจบชุดคำสั่งในฟังก์ชัน

ตัวอย่างการสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ที่ผ่านมา สามารถปรับปรุงชุดคำสั่งให้ตรวจจับความผิดปกติที่อาจจะเกิดขึ้นได้ โดยการแสดงข้อความความผิดพลาด (Errors) หรือข้อความเตือน (Warnings) ดังนี้

```
R> my_fib7 <- function(stop_value) {
```

```
    if (stop_value < 0) {
       warning("Assume a paremeter is positive, so convert a given parameter
               to a positive value")
       stop_value <- stop_value * -1
    } else if (stop_value == 0) {
       stop("Parameter 'stop_value' cannot be 0")
    }

    if (stop_value == 1 || stop_value == 2) {
       return(1)
    } else {
       return(my_fib7(stop_value-1) + my_fib7(stop_value-2))
    }
  }

R> my_fib7(10)
[1] 55
```

เมื่อเรียกใช้ฟังก์ชันโดยผ่านพารามิเตอร์ที่มีค่าเป็นลบ หรือเท่ากับศูนย์ โปรแกรมจะแสดงข้อความเตือน ดังนี้

```
R> my_fib7(-12)
[1] 144
Warning message:
In my_fib7(-12) :
  Assume a paremeter is positive, so convert a given parameter to a positive
   value

R> my_fib7(0)
Error in my_fib7(0) : Parameter 'stop_value' cannot be 0
```

6.11.2 การตรวจจับความผิดพลาดด้วยคำสั่ง try()

โดยปกติเมื่อเกิดความผิดพลาด (Errors) ขึ้น ฟังก์ชันนั้นจะหยุดทำงานรวมไปถึงฟังก์ชันที่ทำการเรียกฟังก์ชันนั้นด้วย (Parent functions) เพื่อป้องกันข้อผิดพลาดดังกล่าวผู้ใช้สามารถใช้คำสั่ง **try()** เพื่อตรวจจับความผิดพลาดที่เกิดขึ้น

ตัวอย่างการสร้างตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ที่ผ่านมา ถ้าผู้ใช้เรียกฟังก์ชันดังกล่าวโดยผ่านพารามิเตอร์ **stop_value = 0** โปรแกรมจะแสดงข้อผิดพลาดที่เกิดขึ้น แต่ถ้าเรียกฟังก์ชันดังกล่าวภายในคำสั่ง **try()** พร้อมกับผ่านพารามิเตอร์ **silent = TRUE** โปรแกรมจะไม่แสดงข้อผิดพลาดทางหน้าจอ แต่จะเก็บข้อผิดพลาดไว้ในตัวแปรที่กำหนด ในที่นี้กำหนดให้เก็บไว้ในตัวแปรวัตถุชื่อ try1 และสามารถเรียกดูข้อผิดพลาดได้ ดังนี้

```
R> my_fib7(stop_value = 10)
[1] 55

R> try1 <- try(my_fib7(stop_value = 0), silent = TRUE)
R> try1
[1] "Error in my_fib7(stop_value = 0) : Parameter 'stop_value' cannot be 0\n"
```

```
attr(, "class")
[1] "try-error"
attr(, "condition")
<simpleError in my_fib7(stop_value = 0): Parameter 'stop_value' cannot be 0>
```

การใช้คำสั่ง `try()` จะช่วยป้องกันฟังก์ชันหยุดทำงาน โดยไม่ส่งผลต่อฟังก์ชันที่ทำการเรียกฟังก์ชันนั้น (Parent functions) ในขณะเดียวกัน ถ้าผู้ใช้ผ่านพารามิเตอร์แล้วไม่เกิดข้อผิดพลาดขึ้น โปรแกรมก็จะทำงานตามปกติ เช่น

```
R> try2 <- try(my_fib7(stop_value = 14), silent = TRUE)
R> try2
[1] 377
```

ตัวอย่างการสร้างตัวเลขลำดับฟีโบนัชชีที่ผ่านมา ผู้ใช้สามารถเรียกใช้ฟังก์ชันดังกล่าวโดยการสร้างฟังก์ชันชื่อว่า `my_fib_vector()` แล้วใช้ดัชนีของเวกเตอร์เรียกฟังก์ชัน `my_fib7()` ดังนี้

```
R> my_fib_vector <- function(nvector) {
     nlength <- length(nvector)
     result <- rep(0, nlength)
     for (i in 1:nlength) {
       result[i] <- my_fib7(nvector[i])
     }
     result
   }
```

ผู้ใช้สามารถเรียกใช้ฟังก์ชันข้างต้นโดยการผ่านพารามิเตอร์ชนิดที่เป็นเวกเตอร์เข้าไปในฟังก์ชันได้ เช่น

```
R> my_vector <- my_fib_vector(15)
R> my_vector
[1] 610

R> my_vector <- my_fib_vector(nvector = c(1, 2, 10, 15, 17))
R> my_vector
[1]    1    1   55  610 1597
```

ฟังก์ชันดังกล่าวสามารถทำงานได้ตามปกติ ยกเว้นเมื่อผ่านพารามิเตอร์ที่เป็น 0 เข้าไปในฟังก์ชัน โปรแกรมจะแจ้งเตือนข้อผิดพลาด ดังนี้

```
R> my_vector <- my_fib_vector(nvector = c(1, 2, 0, 15, 17))
Error in my_fib7(nvector[i]) : Parameter 'stop_value' cannot be 0
```

ฟังก์ชัน `my_fib7()` จะหยุดทำงานเมื่อพบข้อผิดพลาด เนื่องจากพบว่าพารามิเตอร์ที่ส่งเข้ามามีค่าเท่ากับ 0 ส่งผลให้ฟังก์ชัน `my_fib_vector()` ที่เรียกใช้หยุดทำงานไปด้วย ผู้ใช้สามารถป้องกันข้อผิดพลาดที่จะเกิดขึ้นโดยการใช้คำสั่ง `try()` ในการตรวจสอบเพิ่มเติมในตัวฟังก์ชันได้เอง ดังนี้

```
R> my_fib_vector2 <- function(nvector) {
     nlength <- length(nvector)
     result <- rep(0, nlength)
     for (i in 1:nlength) {
       try3 <- try(my_fib7(nvector[i]), silent = TRUE)
       if (class(try3) == "try-error") {
         result[i] <- NA
```

```
        } else {
          result[i] <- try3
        }
      }
      result
    }
```

ชุดคำสั่งข้างต้นเพิ่มการตรวจสอบการใช้งานฟังก์ชัน **my_fib7()** โดยการใช้คำสั่ง **try()**[13] และเก็บข้อผิดพลาดไว้ในวัตถุชื่อ try3 แล้วจึงทำการตรวจสอบว่า try3 เป็นคลาสชื่อ "try-error" หรือไม่ ถ้าเป็นจริงก็กำหนดผลลัพธ์ให้เท่ากับค่า **NA** ให้ แต่ถ้าเป็นเท็จซึ่งก็คือไม่มีข้อผิดพลาดใด ๆ ก็กำหนดค่าเป็นผลลัพธ์ตามปกติ ดังนี้

```
R> my_vector <- my_fib_vector2(nvector = c(1, 2, 10, 0, 17))
R> my_vector
[1]    1    1   55   NA 1597
```

การใช้คำสั่ง **try()** และกำหนดค่า **silent = TRUE** จะไม่แสดงข้อผิดพลาดทางหน้าจอ (Console) ก็จริง แต่ข้อความเตือน (Warning) ยังคงทำงานตามปกติ ดังนี้

```
R> try4 <- try(my_fib7(-10), silent = TRUE)
Warning message:
In my_fib7(-10) :
  Assume a paremeter is positive, so convert a given parameter to a positive
   value

R> try4
[1] 55
```

ถ้าไม่ต้องการให้โปรแกรมแสดงข้อความเตือน (Warning) ใด ๆ เกิดขึ้น ผู้ใช้ต้องใช้ฟังก์ชัน **suppressWarnings()** ดังนี้

```
R> try5 <- suppressWarnings(my_fib7(-10))
R> try5
[1] 55
```

6.12 การตรวจสอบเวลา (Timing) และการแสดงความก้าวหน้า (Progress)

6.12.1 การตรวจสอบเวลา (Timing)

ในการตรวจสอบเวลาปัจจุบันบนเครื่องคอมพิวเตอร์ใช้คำสั่ง **Sys.time()** ดังนี้

```
R> Sys.time()
```

[13] ฟังก์ชัน **try()** เป็นฟังก์ชันในการจัดการข้อผิดพลาดอย่างง่าย ผู้อ่านที่ต้องการจัดการข้อผิดพลาดอย่างละเอียด สามารถใช้ฟังก์ชัน **tryCatch()** ค้นคว้าเพิ่มเติมได้จากพิมพ์ **?tryCatch** ที่ Console

```
[1] "2018-03-29 10:27:06 +07"
```

ในการตรวจสอบว่าโปรแกรมใช้เวลาในการประมวลผลนานเท่าไร ผู้ใช้เรียกคำสั่ง **Sys.time()** แล้วบันทึกเวลาเริ่มต้นในตัวแปร t1 และบันทึกเวลาเริ่มจบในตัวแปร t2 และคำนวณผลต่างของเวลา t2-t1 โดยเขียนเป็นชุดคำสั่ง ได้ดังนี้

```
t1 <- Sys.time()
Sys.sleep(5)
t2 <- Sys.time()
t2-t1
```

คำสั่ง **Sys.sleep()** ทำให้โปรแกรมหยุดประมวลผลชั่วคราว (พารามิเตอร์ที่ผ่านเข้าไปในฟังก์ชัน มีหน่วยเป็นวินาที) เมื่อทำการประมวลผลที่หน้าจอ Console จะแสดงผลดังนี้

```
R> t1 <- Sys.time()
R> Sys.sleep(5)
R> t2 <- Sys.time()
R> t2-t1
Time difference of 5.011541 secs
```

6.12.2 การแสดงความก้าวหน้า (Progress)

การรันโปรแกรม **R** บ่อยครั้งที่ใช้เวลาประมวลผลนาน ผู้ใช้ต้องการทราบความก้าวหน้าของโปรแกรมในระหว่างการประมวลผล ผู้ใช้สามารถเรียกใช้แถบแสดงความก้าวหน้า (Progress bar) ที่อยู่ในเพ็กเกจ utils ได้ การใช้งานแถบแสดงความก้าวหน้า (Progress bar) ประกอบด้วย 3 ขั้นตอนคือ (1) กำหนดค่าเริ่มต้นด้วยคำสั่ง **txtProgressBar()** (2) รายงานความก้าวหน้าด้วยคำสั่ง **setTxtProgressBar()** และ (3) ปิดการใช้งานด้วยคำสั่ง **close()** โดยเขียนเป็นชุดคำสั่งได้ ดังตัวอย่างต่อไปนี้

```
R> progressBarTest <- function(step) {
     progressBar <- txtProgressBar(min = 0, max = step, style = 3, char = "*")
     for (i in 1:step) {
        Sys.sleep(1.0)
        setTxtProgressBar(progressBar, value = i)
     }
     close(progressBar)
   }
```

ชุดคำสั่งข้างต้นเป็นการสร้างฟังก์ชัน **progressBarTest()** โดยผ่านพารามิเตอร์จำนวน step ที่จะแสดงผล คำสั่งภายในฟังก์ชันคำสั่งแรกเป็นการกำหนดค่าเริ่มต้นให้กับตัวแปรวัตถุชื่อ progressBar แล้วจึงสร้างลูป **for** ตามจำนวน step โดยแต่ละ step กำหนดให้โปรแกรมหยุดชั่วคราวเป็นเวลา step ละ 1 วินาที แล้วจึงทำการปรับ (Update) แถบแสดงความก้าวหน้าด้วยคำสั่ง **setTxtProgressBar()** หลังจากทำงานครบทุก step ปิดการใช้งานแถบแสดงความก้าวหน้า (Progress bar) ด้วยคำสั่ง **close()** ตามด้วยชื่อตัวแปรวัตถุ progressBar

ทดสอบการทำงานของฟังก์ชัน **progressBarTest()** ได้ดังนี้

```
R> progressBarTest(10)
  |************************************************************| 100%
```

6.13 การจัดการฟังก์ชันในแพ็กเกจ (Package) และวัตถุในเดต้าเฟรม (data.frame)

6.13.1 การเข้าถึงฟังก์ชันในแพ็กเกจ (Package)

ในการเข้าถึงคำสั่งหรือฟังก์ชัน โปรแกรม **R** จะทำการค้นหาคำสั่งหรือฟังก์ชันดังกล่าวโดยจะค้นหาใน Environments ตามลำดับใน Path ซึ่งดูได้จากคำสั่ง **search()** ดังนี้

```
R> search()
 [1] ".GlobalEnv"        "package:ggplot2"   "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

จากตัวอย่างข้างต้น ถ้าไม่พบคำสั่งหรือฟังก์ชันใน Global environment (**".GlobalEnv"**) โปรแกรม **R** จะทำการค้นหาใน Package environment ชื่อ **ggplot2** (**"package:ggplot2"**) จนถึง Package environment ชื่อ base (**"package:base"**) ตามลำดับ

ถ้าชื่อคำสั่งหรือฟังก์ชันเหมือนกัน **R** จะเรียกใช้คำสั่งที่อยู่ในส่วนของ Global environment (**".GlobalEnv"**) ก่อน แล้วจึงหาคำสั่งหรือฟังก์ชันดังกล่าวในแพ็กเกจอื่น ๆ ต่อไปตามลำดับ ตัวอย่างเช่น ฟังก์ชัน **sum()** อยู่ในแพ็กเกจชื่อ base ในการเรียกใช้ฟังก์ชัน **sum()** โปรแกรม **R** จะหาฟังก์ชัน **sum()** ใน Global environment ก่อน ถ้าไม่พบจึงทำการหาในแพ็กเกจอื่นต่อไป

ปกติฟังก์ชัน **sum()** จะทำการหาผลบวกของสมาชิกทุกตัวในเวกเตอร์ ดังนี้

```
R> my_vector <- c(1:10)
R> sum(my_vector)
[1] 55
```

แต่ถ้ามีการนิยามฟังก์ชัน **sum()** ใหม่เป็นการหาผลบวกของสมาชิกทุกตัวยกกำลังสอง ดังนี้

```
sum <- function(x) {
  result <- 0
  for (i in 1:length(x)) {
    result <- result + x[i]^2
  }
  result
}
```

เมื่อมีการเรียกใช้ฟังก์ชัน **sum()** **R** จะใช้ฟังก์ชัน **sum()** ใน Global environment แทนฟังก์ชัน **sum()** ในแพ็กเกจ Base ดังนี้

```
R> sum(my_vector)
[1] 385
```

ถ้าต้องการใช้ฟังก์ชัน `sum()` ในแพ็กเกจ base จะต้องเรียกโดยใช้ชื่อแพ็กเกจตามด้วยเครื่องหมาย `::` (Double collon) ต่อด้วยชื่อฟังก์ชัน ดังนี้

```
R> base::sum(my_vector)
[1] 55
```

โปรแกรมภาษา **R** เรียกเทคนิคนี้ว่า Masking หมายถึง มีฟังก์ชันบางตัว (ในที่นี้คือ `sum()`) บังอีกฟังก์ชันหนึ่งไว้ (`base::sum()`)

ถ้าต้องการลบฟังก์ชัน `sum()` ออกจาก Global environment ที่นิยามไว้ ให้ใช้คำสั่ง `rm()` ดังนี้

```
R> ls()  # show a list of objects in global environment
[1] "sum"

R> rm(sum)

R> ls()
character(0)
```

หรือถ้าต้องการลบตัวแปร/วัตถุทุกตัวออกจาก Global environment ให้ระบุพารามิเตอร์ `list = ls()` ดังนี้

```
R> rm(list = ls())
R> ls()
character(0)
```

เมื่อมีการเรียกใช้แพ็กเกจหลายแพ็กเกจ อาจทำให้มีวัตถุบางตัวมีชื่อซ้ำกัน **R** จะแจ้งเตือนว่ามีวัตถุบางตัวที่ถูกบังไว้อยู่ (Masked) เช่น มีการเรียกแพ็กเกจชื่อ spatstat ก่อน แล้วจึงเรียกแพ็กเกจชื่อ car ทำให้วัตถุที่ถูกเรียกใช้ล่าสุด (car::ellipse) บังวัตถุก่อนหน้านั้น (spatstat::ellipse) ดังนี้

```
R> install.packages("spatstat")
R> install.packages("car")
R> library(spatstat)
spatstat 1.55-0       (nickname: ‘Stunned Mullet’)
For an introduction to spatstat, type ‘beginner’

R> library("car")
Attaching package: ‘car’

The following objects are masked from ‘package:spatstat’:
    bc, ellipse
```

การเรียกวัตถุ ellipse โดยไม่ระบุชื่อแพ็กเกจจะหมายถึง ellipse ที่อยู่ในแพ็กเกจ car ส่วนถ้าต้องการเรียกวัตถุ ellipse ที่อยู่ในแพ็กเกจชื่อ spatstat เรียกโดยใช้ชื่อแพ็กเกจตามด้วยชื่อวัตถุดังนี้ `spatstat::ellipse`

การเรียกแพ็กเกจเข้ามาใช้งานเพิ่มเติม แล้วชื่อวัตถุในแพ็กเกจดังกล่าวเหมือนกับวัตถุใน Global environment วัตถุในแพ็กเกจที่ถูกเรียกเข้ามาจะถูกบังโดยวัตถุใน Global environment ตัวอย่างเช่น

```
R> cats <- "This is my cats"
R> library("MASS")

Attaching package: ‘MASS’
The following object is masked _by_ ‘.GlobalEnv’:
    cats

The following object is masked from ‘package:spatstat’:
    area

R> cats
[1] "This is my cats"
```

คำสั่งแรกเป็นการสร้างวัตถุ cats เป็นเก็บสตริง "This is my cats" แล้วโหลดแพ็กเกจชื่อ MASS เข้ามา เนื่องจากมีวัตถุ cats ใน Global environment อยู่แล้ว วัตถุ cats ใน MASS จึงถูกบังโดยวัตถุ cats ใน Global environment เมื่อทำการเรียกคำสั่ง cats โปรแกรมเลยแสดงผลเป็นสตริง "This is my cats" ในขณะเดียวกันวัตถุ area ในแพ็กเกจ MASS ก็บังวัตถุ area ในแพ็กเกจ spatstat

6.13.2 การถอดแพ็กเกจออกจากหน่วยความจำ (Unmounting Packages)

เมื่อต้องการแสดงลำดับของแพ็กเกจที่ถูกโหลดเข้ามาในหน่วยความจำของโปรแกรม **R** ที่กำลังทำงานอยู่ ใช้คำสั่ง `search()` ดังนี้

```
R> search()
 [1] ".GlobalEnv"            "package:MASS"          "package:car"
 [4] "package:spatstat"      "package:rpart"         "package:nlme"
 [7] "package:spatstat.data" "tools:rstudio"         "package:stats"
[10] "package:graphics"      "package:grDevices"     "package:utils"
[13] "package:datasets"      "package:methods"       "Autoloads"
[16] "package:base"
```

ถ้าไม่ต้องการใช้งานแพ็กเกจบางแพ็กเกจ ผู้ใช้สามารถถอดแพ็กเกจดังกล่าวออกจากหน่วยความจำได้ โดยใช้คำสั่ง `detach()` กำหนดชื่อแพ็กเกจ และพารามิเตอร์ `unload = TRUE` ดังนี้

```
R> detach("package:car", unload=TRUE)
R> search()
 [1] ".GlobalEnv"            "package:spatstat"      "package:rpart"
 [4] "package:nlme"          "package:spatstat.data" "tools:rstudio"
 [7] "package:stats"         "package:graphics"      "package:grDevices"
[10] "package:utils"         "package:datasets"      "package:methods"
[13] "Autoloads"             "package:base"
```

คำสั่งข้างต้นเป็นการถอดแพ็กเกจ `car` ออกจากหน่วยความจำปัจจุบัน

6.13.3 การเข้าถึงวัตถุในเดต้าเฟรม (Dataframe)

ตามปกติผู้ใช้สามารถเข้าถึงวัตถุในเดต้าเฟรม (Dataframe) โดยการเรียกชื่อเดต้าเฟรมต่อด้วยเครื่องหมาย **$** แล้วตามด้วยชื่อวัตถุในเดต้าเฟรม ได้ดังนี้

```
R> my_data <- data.frame(name = c("Big", "Dan", "June", "James", "Kate"),
                         gender = factor(c("M", "M", "F", "M", "F")),
                         height = c(172, 180, 169, 188, 175),
                         stringsAsFactors = FALSE)
R> my_data
   name gender height
1   Big      M    172
2   Dan      M    180
3  June      F    169
4 James      M    188
5  Kate      F    175

R> my_data$name
[1] "Big"   "Dan"   "June"  "James" "Kate"
```

คำสั่งแรกเป็นการสร้างเดต้าเฟรมชื่อ my_data แล้วแสดงข้อมูลในเดต้าเฟรมประกอบด้วย ชื่อ (name) เพศ (gender) และส่วนสูง (height) ตามลำดับ ผู้ใช้เรียกดูข้อมูลในเดต้าเฟรมโดยการเรียกชื่อเดต้าเฟรมต่อด้วยเครื่องหมาย **$** แล้วตามด้วยชื่อวัตถุในเดต้าเฟรมที่ต้องการ ตัวอย่างเช่น ต้องการเข้าถึงตัวแปร name ผู้ใช้เรียกโดยใช้ชื่อ my_data$name

ผู้ใช้สามารถเพิ่มเดต้าเฟรมเข้าไปใน Path โดยใช้คำสั่ง **attach()** แล้วผ่านพารามิเตอร์ชื่อเดต้าเฟรมที่ต้องการ ได้ดังนี้

```
R> attach(my_data)
R> search()
 [1] ".GlobalEnv"            "my_data"               "package:spatstat"
 [4] "package:rpart"         "package:nlme"          "package:spatstat.data"
 [7] "tools:rstudio"         "package:stats"         "package:graphics"
[10] "package:grDevices"     "package:utils"         "package:datasets"
[13] "package:methods"       "Autoloads"             "package:base"
```

คำสั่ง **attach(my_data)** เป็นการเพิ่มเดต้าเฟรม my_data เข้าไปใน Environment แล้วแสดงด้วยคำสั่ง **search()** จะเห็นว่ามีวัตถุ "my_data" เพิ่มเข้ามาใน Environment ผู้ใช้สามารถเข้าถึงวัตถุในเดต้าเฟรมโดยใช้ชื่อวัตถุในเดต้าเฟรมได้โดยตรง โดยไม่ต้องระบุชื่อเดต้าเฟรม ได้ดังนี้

```
R> name
[1] "Big"   "Dan"   "June"  "James" "Kate"
```

ถ้ามีการเพิ่มเดต้าเฟรมเข้าไปใน Environment โปรแกรม **R** จะกำหนดให้เดต้าเฟรมที่เพิ่มเข้ามาใหม่มีลำดับความสำคัญก่อนเดต้าเฟรมตัวก่อนหน้านี้ แต่หลัง Global environment ดังนี้

```
R> your_data <- data.frame(name = c("Anne", "Jim", "Joe", "Mike", "Mary"),
                           gender = factor(c("F", "M", "M", "M", "F")),
                           weight = c(58, 75, 84, 78, 62),
                           stringsAsFactors = FALSE)
R> your_data
  name gender weight
1 Anne      F     58
2  Jim      M     75
3  Joe      M     84
```

```
4 Mike      M     78
5 Mary      F     62

R> attach(your_data)
The following objects are masked from my_data:

    gender, name

R> search()
 [1] ".GlobalEnv"        "your_data"         "my_data"
 [4] "tools:rstudio"     "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

คำสั่งแรกเป็นการสร้างเดต้าเฟรมชื่อ your_data แล้วแสดงข้อมูลในเดต้าเฟรมประกอบด้วย ชื่อ (name) เพศ (gender) และน้ำหนัก (weight) ตามลำดับ คำสั่งถัดมาเป็นการเพิ่มเดต้าเฟรม your_data เข้าไปใน Environment จะเห็นว่าลำดับแรกคือ **".GlobalEnv"** ตามด้วย **"your_data"** และ **"my_data"** ตามลำดับ และ **R** ยังแจ้งให้ทราบว่าวัตถุ gender และ name ของเดต้าเฟรม your_data บังวัตถุชื่อเดียวกันของเดต้าเฟรม my_data เมื่อใช้คำสั่ง name จะหมายถึงการเข้าถึงวัตถุ name ของเดต้าเฟรม your_data ดังนี้

```
R> name
[1] "Anne" "Jim"  "Joe"  "Mike" "Mary"
```

ผู้ใช้สามารถถอดเดต้าเฟรมที่ไม่ต้องการออกจาก Environment ได้ โดยใช้คำสั่ง **detach()** แล้วกำหนดพารามิเตอร์เป็นชื่อแพ็กเกจที่ต้องการถอด ดังนี้

```
R> detach(my_data)
R> search()
 [1] ".GlobalEnv"        "your_data"         "tools:rstudio"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

6.14 สรุปท้ายบท

บทนี้เป็นการควบคุมทิศทางของโปรแกรม (Flow control) โดยใช้คำสั่งเงื่อนไข **if** ได้แก่ เงื่อนไข **if** อย่างเดียว (Stand-alone if statement) เงื่อนไข **if-else** (if-else statements) เงื่อนไข **if-else** หลายตัว (Multiple if-else statements) เงื่อนไข **ifelse** สำหรับตรวจสอบสมาชิกในเวกเตอร์ (ifelse element-wise statement) และคำสั่ง **switch()** การวนลูปแบบ **for** การวนลูปแบบ **while** กล่าวถึงการวนลูปที่เข้าถึงสมาชิกต่าง ๆ ของเวกเตอร์ เมทริกซ์ หรือลิสต์ได้โดยไม่ต้องกำหนดกลไกในการวนลูปเอง (Implicit loops) โดยใช้คำสั่ง **apply() lapply() sapply() tapply()** และ **mapply()** นอกจากนี้ยังมีคำสั่ง **break** และ **next** สำหรับกำหนดเงื่อนไขการออกจากลูป และคำสั่ง **repeat**

ขอบเขตของตัวแปร/วัตถุต่าง ๆ ในโปรแกรม **R** โดยแบ่งเป็น Environments 3 ส่วน ได้แก่ (1) Global environment เป็นตัวแปร/วัตถุต่าง ๆ ที่ผู้ใช้กำหนดขึ้น (2) Package Environment เป็นตัวแปร/วัตถุต่าง ๆ ของแพ็กเกจ (3) Local Environment เป็นตัวแปร/วัตถุที่อยู่ภายในฟังก์ชัน กล่าวถึงลำดับในการค้นหา Path คำสงวน (Reserved words) ที่ใช้สำหรับในโปรแกรม **R** การระบุพารามิเตอร์ (Parameter Identification) โดยการระบุพารามิเตอร์แบบเต็ม (Exact Identification) การระบุพารามิเตอร์แบบย่อ (Abbreviated Identification) การระบุพารามิเตอร์ตามตำแหน่ง (Positional Identification) การระบุพารามิเตอร์แบบผสม (Mixed Identification) และการระบุพารามิเตอร์แบบเปิด (Open Identification)

การสร้างฟังก์ชัน การกำหนดพารามิเตอร์ให้ฟังก์ชัน (Adding parameters) การส่งค่ากลับ (Return an object) ฟังก์ชันที่ไม่กำหนดชื่อ (Anonymous functions) และฟังก์ชันที่เรียกตัวเอง (Recursive functions) การจัดการความผิดปกติ (Exception handing) ได้แก่ การแสดงความผิดพลาด (Errors) ที่เกิดขึ้นโดยใช้คำสั่ง `stop()` หรือข้อความเตือน (Warnings) ที่เกิดขึ้นโดยใช้คำสั่ง `warning()` การตรวจจับความผิดพลาดด้วยคำสั่ง `try()` การตรวจสอบเวลา (Timing) และการแสดงความก้าวหน้า (Progress) การเข้าถึงฟังก์ชันในแพ็กเกจ (Packages) การถอดแพ็กเกจออกจากหน่วยความจำ (Unmounting packages) และการเข้าถึงวัตถุในเดต้าเฟรม (data.frame)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	**คำอธิบาย**
`if() { }`	คำสั่งเงื่อนไข if
`if() { } else { }`	คำสั่งเงื่อนไข if-else
`ifelse`	คำสั่งเงื่อนไข if-else สำหรับสมาชิกแต่ละตัว (Element-wise)
`switch()`	คำสั่งเงื่อนไข switch
`for() { }`	คำสั่งวนลูปแบบ for
`while() { }`	คำสั่งวนลูปแบบ while
`apply()`	คำสั่ง implicit loop ตามแนวแถวหรือคอลัมน์
`lapply()`	คำสั่ง implicit loop ตามสมาชิกแต่ละตัว
`sapply()`	คำสั่ง implicit loop ตามสมาชิกแต่ละตัว แต่คืนค่ากลับในรูปเวกเตอร์หรือเมทริกซ์
`tapply()`	คำสั่ง implicit loop ตามแฟกเตอร์
`mapply()`	คำสั่ง implicit loop ใช้กับลิสต์หรือเวกเตอร์หลายตัวพร้อม ๆ กัน
`break`	ออกจากจากการวนลูป (Loop)

คำสั่ง	คำอธิบาย
next	ข้ามไปทำคำสั่งในรอบ (Iteration) ถัดไป
repeat{ }	ทำซ้ำจนกว่าจะพบคำสั่ง break
ls()	แสดงตัวแปร/วัตถุที่อยู่ใน environment
search()	แสดงรายละเอียดของ path
environment()	แสดงคุณสมบัติของฟังก์ชันว่าอยู่ใน environment ไหน
rm()	ลบตัวแปร/วัตถุใน workspace
args()	แสดงพารามิเตอร์ของฟังก์ชัน
function() { }	การสร้างฟังก์ชัน
return()	ส่งค่ากลับจากฟังก์ชัน
...	Ellipsis แทนการส่งพารามิเตอร์แบบเปิด
warning()	แสดงข้อความเตือน (Warnings)
stop()	แสดงข้อความความผิดพลาด (Errors)
try()	ตรวจจับความผิดพลาด
Sys.time()	ตรวจสอบเวลาปัจจุบันบนเครื่องคอมพิวเตอร์
Sys.sleep()	สั่งให้โปรแกรมหยุดประมวลผลชั่วคราว
txtProgressBar()	คำสั่งเริ่มต้น Progress bar
setTxtProgressBar()	คำสั่งเพิ่มค่าใน Progress bar
close()	ปิด Progress bar
attach()	เพิ่มวัตถุใน Path
detach()	ถอดแพ็กเกจ/วัตถุออกจากหน่วยความจำ

6.15 แบบฝึกหัดท้ายบท

1. ให้สร้างเวกเตอร์ดังต่อไปนี้

```
my_vector <- 1:5
your_vector <- c(7, 0, 9, 2, 5)
```

1.1 ผลลัพธ์ต่อไปนี้คืออะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

```
if (my_vector[1] >= 1 && your_vector[1] >= 5) {
  cat("The test condition is TRUE.")
}
```

1.2 ผลลัพธ์ต่อไปนี้คืออะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

```
if (!is.na(your_vector[6])) {
  cat("The test condition is TRUE.")
}
```

1.3 ผลลัพธ์ต่อไปนี้คืออะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

```
if ((my_vector[2] + your_vector[3]) == 10) {
  cat("The test condition is TRUE.")
} else {
  cat("The test condition is FALSE.")
}
```

1.4 ผลลัพธ์ต่อไปนี้คืออะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

```
if (!is.na(your_vector[6])) {
  cat("The test condition is TRUE.")
} else {
  cat("The test condition is FALSE.")
}
```

1.5 ผลลัพธ์ต่อไปนี้คืออะไร? (ประมาณด้วยตัวเองก่อน แล้วค่อยตรวจสอบด้วย R)

```
if (my_vector[1] >= 3) {
  cat("The test condition 1 is TRUE.")
} else if (my_vector[2] >= 3) {
  cat("The test condition 2 is TRUE.")
} else if (my_vector[3] >= 3) {
  cat("The test condition 3 is TRUE.")
} else if (my_vector[4] >= 3) {
  cat("The test condition 4 is TRUE.")
} else {
  cat("The all test condition is FALSE.")
}
```

2. จากเวกเตอร์ในข้อ 1. ให้สร้างเวกเตอร์ใหม่ ถ้าผลรวมของ my_vector และ your_vector มากกว่า 3 ให้สมาชิกในเวกเตอร์ใหม่เท่ากับ my_vector ยกกำลังสอง ถ้าไม่เป็นไปตามเงื่อนไขให้สมาชิกในเวกเตอร์ใหม่เท่ากับ your_vector ยกกำลังสอง
3. ให้สร้างชุดคำสั่งเพื่อตรวจสอบเกรดของนักศึกษา ซึ่งมีระดับเกรด (grade) เท่ากับ A, B+, B, C+, C, D+, D และ F โดยมีระดับคะแนน (point) เท่ากับ 4.0, 3.5, 3.0, 2.5, 2.0, 1.5, 1.0 และ 0.0 ตามลำดับ ตรวจสอบชุดคำสั่งที่สร้างขึ้น โดยกำหนดให้ตัวแปร grade เท่ากับ B+ รันชุดคำสั่งดังกล่าวแล้วตรวจสอบตัวแปร point ว่าเท่ากับ 3.5 หรือไม่?
4. ให้สร้างชุดคำสั่งในข้อ 3. โดยใช้คำสั่ง **switch()** แบบย่อ ตรวจสอบชุดคำสั่งที่สร้างขึ้น โดยกำหนดให้ตัวแปร grade เท่ากับ 2 รันชุดคำสั่งดังกล่าว แล้วตรวจสอบตัวแปร point ว่าเท่ากับ 3.5 หรือไม่?
5. ให้สร้างสูตรคูณตั้งแต่ 2x1, 2x2, ..., 12x12 โดยแสดงผลดังตัวอย่างต่อไปนี้

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
...
2 x 10 = 20
2 x 11 = 22
2 x 12 = 24
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
...
12 x 11 = 132
12 x 12 = 144
```

6. ใช้คำสั่ง while ในการคำนวณหาค่า Factorial แล้วตรวจสอบความถูกต้องว่า

   6.1 ค่า **10!** เท่ากับ 3,628,800 หรือไม่?

   6.2 ค่า **1!** เท่ากับ 1 หรือไม่?

   6.3 ค่า **0!** เท่ากับ 1 หรือไม่?

7. จากลิสต์นี้

```
my_list <- list(mat1 = matrix(c("a", "b", "c", "d", "e", "f"),
                              nrow = 2, ncol = 3),
                mat2 = matrix(1:12, nrow = 3, ncol = 2),
                mat3 = matrix(c(T, T, T, F, F, F), nrow = 3, ncol = 2))
```

ให้เขียนชุดคำสั่งด้วย Implicit loops ให้ได้ผลลัพธ์เหมือนกับการใช้คำสั่ง **for** ดังนี้

```
for (i in 1:length(my_list)) {
  my_list[[i]] <- t(my_list[[i]])
}
My_list
```

8. ให้ค่าผลรวมตามแนวคอลัมน์ของเมทริกซ์ชื่อ mat2 ในลิสต์จากข้อ 7.
9. ข้อมูลจากชุดข้อมูลที่มีอยู่ในโปรแกรม **R** (Build-in data sets) ชื่อว่า mtcars ให้หาค่าเฉลี่ยของจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (ตัวแปรชื่อ mpg) แยกตามประเภทระบบเกียร์ (ตัวแปรชื่อ am) (หมายเหตุ: 0 = Automatic, 1 = Manual)
10. จากลิสต์คะแนนแต่ละห้องเรียน ดังนี้

```
Score_list <- list(class1 = c(85, 70, 91, 78, 82, 86),
                   class2 = c(77, 85, 89, 93, 84),
                   class3 = c(68, 66, 70, 72, 76, 58, 64))
```

หาค่าสถิติเบื้องต้น ได้แก่ Mean, Median, Min, Max, 1st quantile, 3rd quantile ของ class1, class2 และ class3

11. ชุดคำสั่งจากหัวข้อ 6.3.1 แสดงได้ดังนี้

```
R> my_value <- 20
R> my_vector <- c(3, 2, 1, 0, -1, -2)
R> result <- rep(NA, length(my_vector))
R> result
[1] NA NA NA NA NA NA

R> for (i in 1:length(my_vector)) {
     value <- my_value/my_vector[i]
     if (is.finite(value)) {
        result[i] <- value
     } else {
        break  # break and exit for loop
     }
   }
R> result
[1]  6.666667 10.000000 20.000000        NA        NA        NA
```

ให้เขียนโปรแกรมที่สามารถทำงานได้เหมือนชุดคำสั่งข้างบนด้วยคำสั่ง **while** และ **break**

12. ชุดคำสั่งจากข้อ 11. ให้เขียนชุดคำสั่งดังกล่าวด้วยคำสั่ง **repeat** ให้โปรแกรมสามารถทำงานได้เหมือนชุดคำสั่งดังกล่าว
13. ชุดคำสั่งจากหัวข้อ 6.3.1 แสดงได้ดังนี้

```
R> result <- rep(NA, length(my_vector))
R> result
[1] NA NA NA NA NA NA

R> for (i in 1:length(my_vector)) {
     value <- my_value/my_vector[i]
     if (is.finite(value)) {
        result[i] <- value
     } else {
        next  # skip this iteration and continue the for loop
     }
   }
R> result
```

```
[1]  6.666667  10.000000  20.000000     NA -20.000000 -10.000000
```

ให้เขียนโปรแกรมที่สามารถทำงานได้เหมือนชุดคำสั่งข้างบนด้วยคำสั่ง `while` และ `next`

14. วัตถุต่อไปนี้อยู่ใน enviroments อะไร?

    14.1 `data.frame`
    14.2 `read.csv`
    14.3 `array`
    14.4 `matrix`
    14.5 `png`
    14.6 `my_vector <- 1:10`
    14.7 `my_matrix <- matrix(1:12, nrow = 3, ncol = 4)`

15. ให้ใช้คำสั่ง `ls()` ในการตรวจสอบว่าฟังก์ชัน `spineplot()` อยู่ในแพ็กเกจชื่อว่า graphics

16. ให้สร้างเลขลำดับดังนี้ -10.0, -9.5, -9.0, ..., 9.5, 10.0 โดยใช้คำสั่ง `seq()`

    16.1 ใช้การระบุพารามิเตอร์แบบเต็ม
    16.2 ใช้การระบุพารามิเตอร์แบบย่อ
    16.3 ใช้การระบุพารามิเตอร์ตามตำแหน่ง
    16.4 ใช้การระบุพารามิเตอร์แบบผสม

17. ให้หาว่าคำสั่งต่อไปนี้เป็นการระบุพารามิเตอร์แบบเต็ม แบบย่อ แบบตามตำแหน่ง หรือแบบผสม ถ้าเป็นการระบุพารามิเตอร์แบบผสมให้หาว่าพารามิเตอร์ตัวไหนเป็นการระบุแบบไหน?

    17.1 `matrix(1:12, nrow = 3, ncol = 4)`
    17.2 `array(data = 1:24, dim = c(2, 3, 4))`
    17.3 `rep(c(1, 2, 3), 5)`
    17.4 `sort(decreasing = TRUE, c(1, 9, 5, 7, 3, 8))`
    17.5 `which(c(T, T, F, F, T), arr = T)`

18. ให้สร้างฟังก์ชัน `your_fib()` ทำงานเหมือนกับฟังก์ชัน `my_fib4()` ในหัวข้อ 6.8 โดยรับพารามิเตอร์ 2 ตัว ตัวแรกระบุค่าพารามิเตอร์ชื่อ stop_value เพื่อตรวจสอบจำนวนตัวเลขที่ได้จากฟังก์ชันและหยุดการคำนวณต่อ พารามิเตอร์ตัวที่สองเป็นค่าตรรกะชื่อ show_console ถ้าค่าดังกล่าวเท่ากับ **TRUE** ให้พิมพ์ตัวเลขลำดับฟีโบนัชชี (Fibonacci sequence) ออกทาง Console ถ้าเท่ากับ **FALSE** ให้คืนค่าเป็นเวกเตอร์กลับไปที่ฟังก์ชันเพื่อนำไปใช้งานต่อ โดยให้ค่าโดยปริยายของ show_console มีค่าเป็น **FALSE** แล้วตรวจสอบความถูกต้องด้วยคำสั่งต่อไปนี้

    18.1
    ```
    R> fib1 <- your_fib(stop_value = 10000, show_console = FALSE)
    R> fib1
    ```
    18.2
    ```
    R> fib2 <- your_fib(10000, FALSE)
    R> fib2
    ```
    18.3
    ```
    R> your_fib(stop = 10000)
    ```

18.4 `R> your_fib(stop = 10000, T)`

18.5 `R> your_fib(10000, TRUE)`

18.6 `R> your_fib(10000)`

19. ให้เขียนฟังก์ชัน `my_factorial()` ในการคำนวณหาค่า factorial โดยรับพารามิเตอร์ 1 ตัว ชื่อ `x` โดยสมมติว่าพารามิเตอร์ที่นำเข้าเป็นเลขจำนวนเต็มบวก แล้วตรวจสอบความถูกต้องว่า

    19.1 `my_factorial(x = 10)` เท่ากับ 3,628,800 หรือไม่?

    19.2 `my_factorial(1)` เท่ากับ 1 หรือไม่?

    19.3 `my_factorial(x = 0)` เท่ากับ 1 หรือไม่?

20. ให้เขียนฟังก์ชัน `my_factorial2()` ในการคำนวณหาค่า Factorial โดยรับพารามิเตอร์ 1 ตัวชื่อ `x` เหมือนกับข้อ 2. แต่มีการตรวจสอบว่าพารามิเตอร์ที่ส่งเข้ามาเป็นบวกหรือไม่ ถ้ามีค่าเป็นบวกให้ทำการคำนวณตามปกติ ถ้ามีค่าติดลบให้คืนค่ากลับมาเป็น `NaN` แล้วตรวจสอบความถูกต้องด้วยคำสั่งต่อไปนี้

    20.1 `my_factorial2(x = -10)` เท่ากับ `NaN` หรือไม่?

    20.2 `my_factorial2(10)` เท่ากับ 3,628,800 หรือไม่?

    20.3 `my_factorial2(x = 1)` เท่ากับ 1 หรือไม่?

    20.4 `my_factorial2(0)` เท่ากับ 1 หรือไม่?

21. ให้เขียนฟังก์ชัน `my_quadratic()` คำนวณค่าฟังก์ชันจากสมการ $y = ax^2 + bx + c$ โดยมีพารามิเตอร์ ได้แก่ a, b, c, show_plot และพารามิเตอร์แบบเปิดเพื่อรับค่ารายละเอียดในการพล็อตลักษณะต่าง ๆ โดยสมมติว่าพารามิเตอร์ x ที่นำเข้าเป็นเลขจำนวนเต็มบวก โดยให้ค่าโดยปริยายของ show_plot เท่ากับ `TRUE` ถ้าค่า show_plot เป็น `TRUE` ให้ทำการพล็อตโดยมีค่า x ต่ำสุดเท่ากับ 0 และสูงสุดเท่ากับ x ส่วนค่า y ต่ำสุดเท่ากับ 0 และสูงสุดเท่ากับ y ถ้าค่า show_plot เป็น `FALSE` ให้คืนค่ากลับมาที่ฟังก์ชันเพื่อนำไปใช้งานต่อไป โดยไม่ต้องพล็อตกราฟใด ๆ แล้วตรวจสอบความถูกต้องด้วยคำสั่งต่อไปนี้

    21.1 `my_quadratic(a = 0.1, b = 0.2, c = 0.3, x = 10, show_plot = FALSE)`

    21.2 `my_quadratic(0.1, 0.2, 0.3, 10, show = FALSE)`

    21.3 `my_quadratic(0.1, 0.2, 0.3, 10, show = TRUE)`

    21.4 `my_quadratic(a = 0.1, b = 0.2, c = 0.3, x = 10, type = "b", lty =2, main = "Quadratic Equation", ylab = "Y", xlab = "X")`

22. ให้ประยุกต์ใช้ฟังก์ชันที่ไม่กำหนดชื่อ (Anonymous functions) ร่วมกับฟังก์ชัน `lapply()` ในการใส่เครื่องหมาย `!!!` ต่อท้ายสตริงทุกตัวในลิสต์ต่อไปนี้

```
my_list <- list(var1 = "Hello world",
                var2 = c("This is R","It is awesome"),
```

```
                var3 = c("Learning R is good for your future"))
```

23. ให้เขียนฟังก์ชัน `my_factorial3()` ในการคำนวณหาค่า Factorial โดยรับพารามิเตอร์ 1 ตัวชื่อ x เหมือนกับข้อ 19. และข้อ 20. แต่มีการตรวจสอบว่าพารามิเตอร์ที่ส่งเข้ามาเป็นบวกหรือไม่ ถ้ามีค่าเป็นบวกให้ทำการคำนวณตามปกติ ถ้ามีค่าติดลบแสดงข้อความความผิดพลาด (Errors) ที่เกิดขึ้น แล้วตรวจสอบความถูกต้องด้วยคำสั่งต่อไปนี้

    23.1 `my_factorial3(x = -10)` แสดงข้อความความผิดพลาด (Errors)

    23.2 `my_factorial3(x = -5)` แล้วแสดงข้อความความผิดพลาด (Errors) เก็บไว้ในตัวแปร โดยไม่ให้โปรแกรมหยุดทำงาน

    23.3 `my_factorial3(10)` เท่ากับ 3,628,800 หรือไม่?

    23.4 `my_factorial3(x = 1)` เท่ากับ 1 หรือไม่?

    23.5 `my_factorial3(0)` เท่ากับ 1 หรือไม่?

24. ให้ตรวจสอบเวลาว่าโปรแกรมในข้อ 23. ใช้เวลาในการประมวลผลนานเท่าไร?
25. ให้เขียนโปรแกรมแสดงความก้าวหน้าในระหว่างการประมวลผล โดยกำหนดให้โปรแกรมหยุดชั่วคราวเป็น step ๆ ละ 0.1 วินาที ให้แถบแสดงความก้าวหน้าใช้ style = 1 และอักขระ `"."` ทดสอบชุดคำสั่งโดยกำหนดให้พารามิเตอร์เท่ากับ 50 และ 100 ตามลำดับ
26. จากข้อมูลที่ติดตั้งมาพร้อมกับแพ็กเกจ `ggplot2` ชื่อว่า mpg ให้หาค่าเฉลี่ยจำนวนระยะทางต่อน้ำมันเชื้อเพลิงสำหรับรถยนต์ที่วิ่งในเขตเมือง (ตัวแปรชื่อ cty) แยกตามประเภทรถ (ตัวแปรชื่อ class) โดยใช้คำสั่ง `tapply()` โดยให้เข้าถึงตัวแปรใน mpg ได้โดยตรง โดยไม่ต้องอ้างถึง mpg ทุกครั้ง (ไม่ใช่เครื่องหมาย `$`)

บทที่ 7

R Markdown และ R Notebook

การเขียนคำสั่งที่ผ่านมาในบทก่อนหน้านี้เป็นการเขียนผ่าน console เมื่อต้องการรันเพียงไม่กี่คำสั่ง และไม่ต้องการนำคำสั่งดังกล่าวกลับมาใช้ใหม่ หรือการเขียนโดยใช้ R Script ถ้าหากต้องการนำคำสั่งดังกล่าวกลับมาใช้ใหม่ ถ้าผู้ใช้ต้องการเขียนคำอธิบาย ใส่สัญลักษณ์ สูตร สมการ เครื่องหมาย รูปภาพ หรือ Comments ต่าง ๆ ได้เหมือนการเขียน HTML ร่วมกับการเขียนโค้ดอยู่บน Platform เดียวกัน RStudio สามารถเขียนในรูปแบบ R Markdown และ R Notebook ได้

ตัวอย่างการเขียน R Markdown แสดงได้ดังนี้

````
---
title: "Example"
author: "Sutthipong Meeyai"
date: "11/11/2024"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for
   authoring HTML, PDF, and MS Word documents. For more details on using R
   Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both
   content as well as the output of any embedded R code chunks within the
   document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Inline code 2 + 2 =  `r 2 + 2`

Note that the `echo = FALSE` parameter was added to the code chunk to prevent
   printing of the R code that generated the plot.
````

ผลที่ได้จากโค้ดดังกล่าวแสดงได้ดังรูปต่อไปนี้

Example

Sutthipong Meeyai

11/11/2024

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

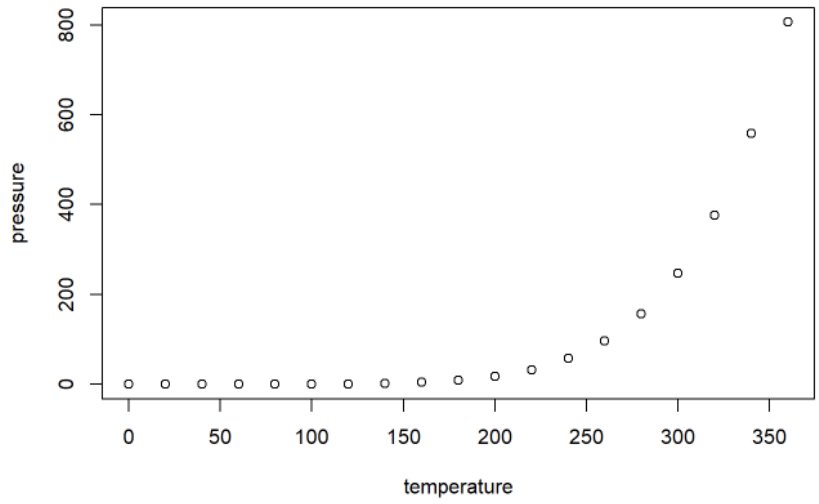
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

Including Plots

You can also embed plots, for example:



Inline code 2 + 2 = 4

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

รูปที่ 7-1 R Markdown results

ส่วนแรกเรียกว่า YAML header เป็นการให้ข้อมูลโดยทั่วไปของเอกสาร เช่น ชื่อเอกสาร ผู้แต่ง วันที่ ปรับปรุง ผลลัพธ์ที่ต้องการแสดง โดยอยู่ภายในเครื่องหมาย --- ดังนี้

```
---
title: "Example"
```

```
author: "Sutthipong Meeyai"
date: "11/4/2021"
output: html_document
---
```

ส่วนในการเขียนโค้ดจะอยู่ภายใน Chunks (Code chunks) สร้างด้วยการกด Ctrl + Alt + i โดยอยู่ภายในเครื่องหมาย ```{r name} และ ``` (เครื่องหมาย ` เรียกว่า Backtick สร้างด้วยการกด Alt + 96) และสามารถกำหนดชื่อ Chunk ได้ดังนี้

```
```{r cars}
summary(cars)
```
```

โค้ดด้านบนเป็นสร้าง Chunk ชื่อ cars และมีคำสั่ง `summary()` อยู่ภายใน Chunk ดังกล่าว

การเขียนโค้ดที่อยู่ภายในข้อความ (Inline code) โค้ดจะอยู่ภายในเครื่องหมาย ` และตามด้วย `r` ดังนี้

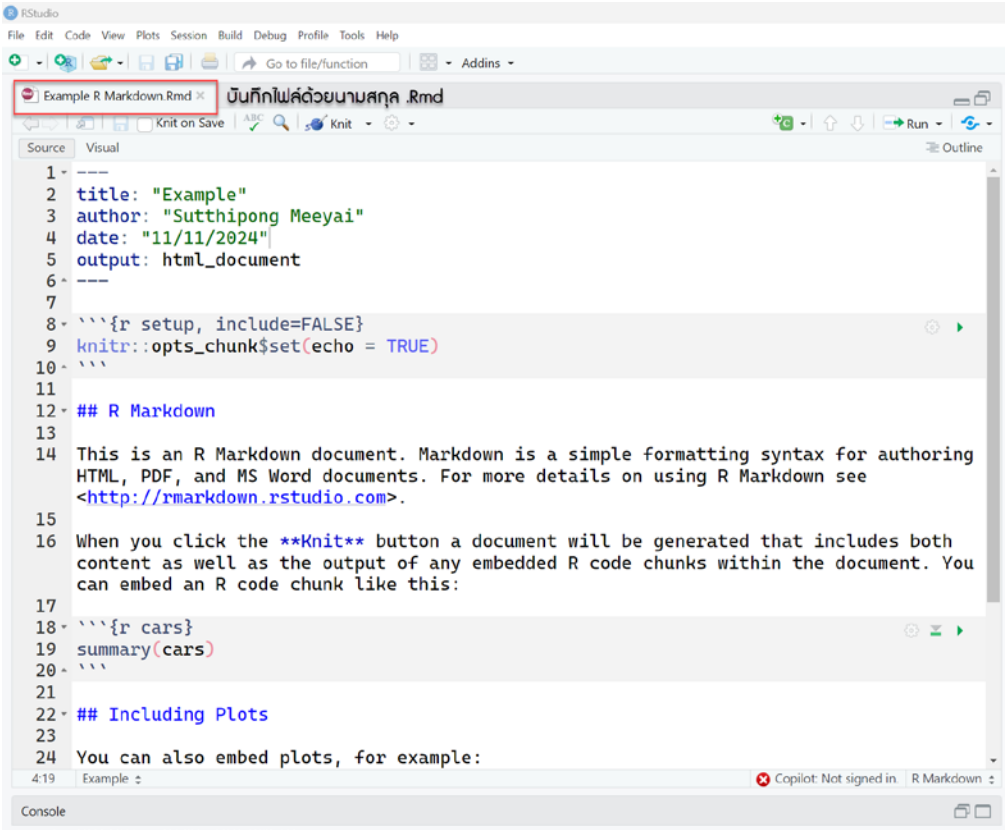
```
Inline code 2 + 2 =  `r 2 + 2`
```

คำอธิบายจะใช้ข้อความ (Text) ผสมกับเครื่องหมายต่าง ๆ เช่น `# _bold_ **Knit**` เพื่อแสดงรูปแบบตัวอักษรในลักษณะต่าง ๆ ดังนี้

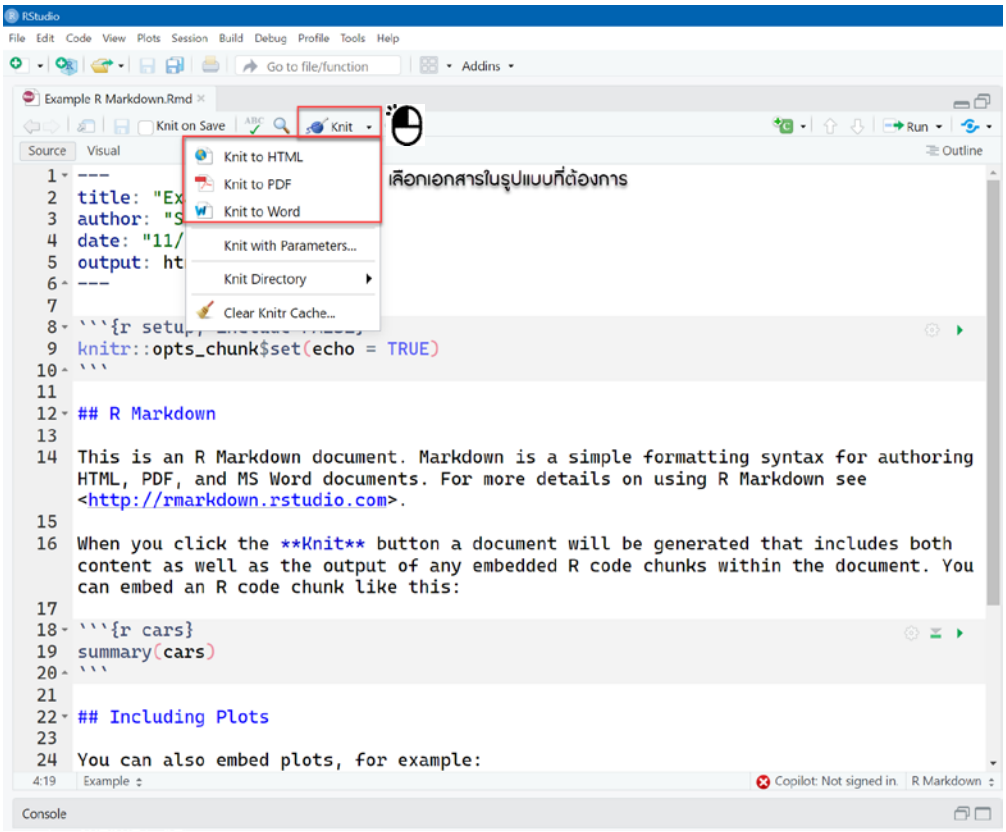
```
## R Markdown

When you click the **Knit** button a document will be generated that includes both
   content as well as the output of any embedded R code chunks within the
   document. You can embed an R code chunk like this:
```

ไฟล์ R Markdown และ R Notebook จะบันทึกด้วยนามสกุล .Rmd ซึ่งสามารถสร้างเอกสารในรูปแบบต่าง ๆ ได้ เช่น pdf, microsoft word, HTML เป็นต้น การสร้างไฟล์เอกสารดังกล่าวจะต้องทำการ knit ไฟล์ .Rmd ไปเป็นไฟล์เอกสารประเภทต่าง ๆ ตัวอย่างไฟล์ R Markdown และการ knit ไฟล์ .Rmd แสดงดังรูปที่ 7-2 และรูปที่ 7-3



รูปที่ 7-2 ตัวอย่างไฟล์ R Markdown



รูปที่ 7-3 การ knit ไฟล์ R Markdown

R Notebook เป็นเอกสารประเภทหนึ่งของ R Markdown ซึ่ง R Notebook จะแสดงผลแต่ละ Chunk ได้เป็นอิสระต่อกัน ส่วน R Markdown จะต้องทำการ knit เพื่อแสดงผลเอกสารพร้อมกันทุก Chunks

R Markdown และ R Notebook เหมาะกับ (1) การผลิตเอกสารที่ต้องการสื่อสารกับผู้บริหาร/ผู้มีอำนาจตัดสินใจ (Decision makers) ซึ่งสนใจผลสรุป ไม่ใช่โค้ดที่อยู่ในเอกสารแต่เพียงอย่างเดียว (2) เอกสารที่ต้องการสื่อสารกับผู้ร่วมงานอื่น หรือผู้สร้างเอกสารเองที่จะกลับมาใช้อนาคต และ (3) สร้างสิ่งแวดล้อมในการทำงานนอกเหนือจากสิ่งที่ทำ (โค้ดที่เขียน) ยังต้องการสื่อถึงสิ่งที่คิด (คำอธิบายต่าง ๆ) อย่างเป็นขั้นตอนชัดเจน

RStudio ได้สร้างไฟล์สรุปการใช้งาน R Markdown ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมากสำหรับผู้ใช้ **R** (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว หรือในเมนู Help ใน RStudio)

7.1 การจัดรูปแบบเอกสารด้วยภาษา Markdown

Markdown เป็นเหมือนภาษาในการจัดรูปแบบเอกสารที่ใช้การโค้ดไม่มากนัก แต่สามารถแสดงรูปแบบต่าง ๆ ได้เหมือนกับ HTML

คำสั่งพื้นฐานประกอบด้วย

- คำสั่งในการแสดงผลเป็น Heading ขนาดต่าง ๆ ใช้เครื่องหมาย **# ## ###** ตามลำดับ
- การขึ้นบรรทัดใหม่ใช้ช่องว่าง 2 ตัว (Two spaces)
- อักษรตัวเอียงจะอยู่ภายในเครื่องหมาย * หรือ **_**
- อักษรตัวหนาจะอยู่ภายในเครื่องหมาย ** หรือ **__**
- ตัวยก (Superscript) จะอยู่ภายในเครื่องหมาย **^**
- การขีดคล่อมตัวอักษรจะอยู่ภายในเครื่องหมาย ~
- การสร้าง link ไปยัง website ต่าง ๆ จะอยู่ภายในเครื่องหมาย **[…]** และตามด้วย url ที่ต้องการ

Syntax	Outputs
Plain text End a line with two spaces to start a new paragraph. *italics* and _italics_ **bold** and __bold__ superscript^2^ ~~strikethrough~~ [link](www.rstudio.com) # Header 1 ## Header 2 ### Header 3 #### Header 4 ##### Header 5 ###### Header 6	Plain text End a line with two spaces to start a new paragraph. *italics* and *italics* **bold** and **bold** superscript^2 ~~strikethrough~~ link **Header 1** **Header 2** **Header 3** **Header 4** **Header 5** **Header 6**

ที่มา: Rmarkdown Cheat Sheet

รูปที่ 7-4 R Markdown formats

เครื่องหมายที่สำคัญประกอบด้วย

- การสร้างสมการที่อยู่ภายในบรรทัดนั้น (Inline equation) จะอยู่ภายในเครื่องหมาย $
- การใส่ภาพจะต้องกำหนด path และชื่อไฟล์รูปภาพต่อจากเครื่องหมาย `![]`
- การขีดเส้นแบ่งใช้เครื่องหมาย ***
- การสร้าง Block quote ใช้เครื่องหมาย >
- การสร้าง List แบบไม่มีเลขลำดับ (Unordered list) นำหน้าด้วยเครื่องหมาย *
- Sub-list นำหน้าด้วยเครื่องหมาย +
- การสร้าง List แบบมีเลขลำดับ (Ordered list) นำหน้าด้วยตัวเลขตามด้วยจุด . และเว้นวรรค
- การสร้างตารางใช้เส้นแนวนอนและเส้นแนวตั้งดังรูปต่อไปนี้

Syntax

```
endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***

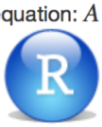
> block quote

* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

Outputs

endash: –
emdash: —
ellipsis: …
inline equation: $A = \pi * r^2$
image: 
horizontal rule (or slide break):

> block quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list
2. item 2
   - sub-item 1
   - sub-item 2

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4

ที่มา: Rmarkdown Cheat Sheet

รูปที่ 7-5 R Markdown formats (cont.)

7.2 Code Chunk Options

Code chunks สามารถกำหนดรูปแบบการแสดงผลในแต่ละ Chunk โดยการกำหนด Options แบบต่าง ๆ Options ที่สำคัญได้แก่

- `eval` เป็นการ evaluate โค้ดใน Chunk และแสดงผลลัพธ์ (ค่าโดยปริยายเท่ากับ `TRUE`) ดังนี้

`eval = TRUE`

```
dim(iris)
```

```
## [1] 150   5
```

`eval = FALSE`

```
dim(iris)
```

- `echo` เป็นการแสดงโค้ดใน Chunk (ค่าโดยปริยายเท่ากับ `TRUE`) ดังนี้

`echo = TRUE`

```
dim(iris)
```

```
## [1] 150   5
```

`echo = FALSE`

```
## [1] 150   5
```

- `warning` มีการแสดงการเตือน (Warning) (ค่าโดยปริยายเท่ากับ `TRUE`)

- **error** มีการแสดงข้อผิดพลาด (Errors) (ค่าโดยปริยายเท่ากับ **FALSE**)
- **tidy** มีการ reformat โค้ดให้เรียบง่ายขึ้น (ค่าโดยปริยายเท่ากับ **FALSE**)
- **comment** การแสดงเครื่องหมายหน้าผลลัพธ์ (ค่าโดยปริยายเท่ากับ **"##"**)
- **fig.width** ค่าความกว้างของรูปที่พล็อตหน่วยเป็นนิ้ว (ค่าโดยปริยายเท่ากับ 7)
- **fig.height** ค่าความยาวของรูปที่พล็อตหน่วยเป็นนิ้ว (ค่าโดยปริยายเท่ากับ 7)

7.3 การสร้างสูตรและเครื่องหมายทางคณิตศาสตร์ด้วย Latex

R Markdown สามารถสร้างสูตรและเครื่องหมายทางคณิตศาสตร์ 2 แบบ คือ

1. การแทรกภายในบรรทัด (Inline) จะอยู่ภายในเครื่องหมาย **$**
2. การแยกบรรทัด (Separate line) จะอยู่ภายในเครื่องหมาย **$$**

ดังตัวอย่างต่อไปนี้

```
Here is an in line equation: $\displaystyle \sum_{n=1}^{\infty}\frac{1}{n}$
or $\sum_{n=1}^{\infty}\frac{1}{n}$

Here is a separate line equation:
$$
x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}\
$$
```

Here is an in line equation: $\displaystyle \sum_{n=1}^{\infty}\frac{1}{n}$ or $\sum_{n=1}^{\infty}\frac{1}{n}$

Here is a separate line equation:

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

รูปที่ 7-6 Inline equation and separate line equation

คำสั่งแรกเป็นการสร้างสมการโดยการแทรกในบรรทัด (Inline) นั้น ส่วนคำสั่งที่สองเป็นการสร้างสมการแยกบรรทัด (Separate line) ออกมา

การสร้างสูตรและเครื่องหมายทางคณิตศาสตร์จะใช้คำสั่งจาก Latex ซึ่งเป็นภาษาโปรแกรมในการจัดเรียงพิมพ์ (Type-setting program) ที่สามารถปรับแต่งรูปแบบเอกสารโดยการป้อนข้อความที่ต้องการ พร้อมกับคำสั่งในการเรียงพิมพ์ โดยมีลักษณะเป็น Mark-up language คือ มีการกำกับข้อความส่วนต่าง ๆ ด้วยลักษณะที่ต้องการให้เรียงพิมพ์คล้ายกับภาษา HTML

ผู้ใช้สามารถค้นหา Cheat sheet สรุปการใช้งาน Latex ได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว การสร้างสูตรและเครื่องหมายทางคณิตศาสตร์พื้นฐานทำได้โดยการใช้คำสั่ง ดังนี้

รายละเอียด	คำสั่ง	ผลลัพธ์
addition	`+`	$+$
subtraction	`-`	$-$
plus or minus	`\pm`	$\pm$
multiplication (times)	`\times`	$\times$
multiplication (dot)	`\cdot`	$\cdot$
division symbol	`\div`	$\div$
division (slash)	`/`	$/$
circle plus	`\oplus`	$\oplus$
circle times	`\otimes`	$\otimes$
equal	`=`	$=$
not equal	`\ne`	$\neq$
less than	`<`	$<$
greater than	`>`	$>$
less than or equal to	`\le`	$\leq$
greater than or equal to	`\ge`	$\geq$
approximately equal to	`\approx`	$\approx$
infinity	`\infty`	∞
dots	`1,2,3,\ldots`	$1, 2, 3, \ldots$
dots	`1+2+3+\cdots`	$1 + 2 + 3 + \cdots$
fraction	`\frac{a}{b}`	$\frac{a}{b}$
square root	`\sqrt{x}`	$\sqrt{x}$
nth root	`\sqrt[n]{x}`	$\sqrt[n]{x}$
exponentiation	`a^b`	a^b
subscript	`a_b`	a_b
absolute value	`|x|`	$\|x\|$
natural log	`\ln(x)`	$\ln(x)$
logarithms	`\log_{a}b`	$\log_a b$
exponential function	`e^x=\exp(x)`	$e^x = \exp(x)$
degree	`\deg(f)`	$\deg(f)$

รูปที่ 7-7 Basic mathematics symbols

รายละเอียด	คำสั่ง	ผลลัพธ์
maps to	`\to`	$\to$
composition	`\circ`	$\circ$

piecewise function	`|x|=`
	`\begin{cases}`
	`x & x\ge 0\\`
	`-x & x<0`
	`\end{cases}`
		$\|x\| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$

รูปที่ 7-8 Basic function symbols

คำสั่ง	ผลลัพธ์	คำสั่ง	ผลลัพธ์
\alpha	α	\tau	τ
\beta	β	\theta	θ
\chi	χ	\upsilon	υ
\delta	δ	\xi	ξ
\epsilon	ϵ	\zeta	ζ
\varepsilon	ε	\Delta	Δ
\eta	η	\Gamma	Γ
\gamma	γ	\Lambda	Λ
\iota	ι	\Omega	Ω
\kappa	κ	\Phi	Φ
\lambda	λ	\Pi	Π
\mu	μ	\Psi	Ψ
\nu	ν	\Sigma	Σ
\omega	ω	\Theta	Θ
\phi	ϕ	\Upsilon	Υ
\varphi	φ	\Xi	Ξ
\pi	π	\aleph	$\aleph$
\psi	ψ	\beth	$\beth$
\rho	ρ	\daleth	$\daleth$
\sigma	σ	\gimel	$\gimel$

รูปที่ 7-9 Greek letters

รายละเอียด	คำสั่ง	ผลลัพธ์
derivative	\frac{df}{dx}	$\frac{df}{dx}$
derivative	\f'	f'
partial derivative	\frac{\partial f}{\partial x}	$\frac{\partial f}{\partial x}$
integral	\int	$\int$
double integral	\iint	$\iint$
triple integral	\iiint	$\iiint$
limits	\lim_{x\to \infty}	$\lim_{x\to\infty}$
summation	\sum_{n=1}^{\infty}a_n	$\sum_{n=1}^{\infty} a_n$
product	\prod_{n=1}^{\infty}a_n	$\prod_{n=1}^{\infty} a_n$

รูปที่ 7-10 Calculus symbols

รายละเอียด	คำสั่ง	ผลลัพธ์
vector	`\vec{v}`	$\vec{v}$
vector	`\mathbf{v}`	$\mathbf{v}$
norm	`||\vec{v}||`	$\|\|\vec{v}\|\|$

matrix	`\left[`
	`\begin{array}{ccc}`
	`1 & 2 & 3 \\`
	`4 & 5 & 6\\`
	`7 & 8 & 0`
	`\end{array}`
	`\right]`
		$\left[\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6\\ 7 & 8 & 0 \end{array}\right]$

determinant	`\left|`
	`\begin{array}{ccc}`
	`1 & 2 & 3 \\`
	`4 & 5 & 6 \\`
	`7 & 8 & 0`
	`\end{array}`
	`\right|`
		$\left\|\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{array}\right\|$

determinant	`\det(A)`	$\det(A)$
trace	`\operatorname{tr}(A)`	$\operatorname{tr}(A)$
dimension	`\dim(V)`	$\dim(V)$

รูปที่ 7-11 Linear algebra symbols

รายละเอียด	คำสั่ง	ผลลัพธ์
angle	`\angle ABC`	$\angle ABC$
degree	`90^{\circ}`	90°
triangle	`\triangle ABC`	$\triangle ABC$
segment	`\overline{AB}`	$\overline{AB}$
sine	`\sin`	$\sin$
cosine	`\cos`	$\cos$
tangent	`\tan`	$\tan$
cotangent	`\cot`	$\cot$
secant	`\sec`	$\sec$
cosecant	`\csc`	$\csc$
inverse sine	`\arcsin`	$\arcsin$
inverse cosine	`\arccos`	$\arccos$
inverse tangent	`\arctan`	$\arctan$

รูปที่ 7-12 Geometry and trigonometry symbols

รายละเอียด	คำสั่ง	ผลลัพธ์
set brackets	`\{1,2,3\}`	$\{1,2,3\}$
element of	`\in`	$\in$
not an element of	`\not\in`	$\notin$
subset of	`\subset`	$\subset$
subset of	`\subseteq`	$\subseteq$
not a subset of	`\not\subset`	$\not\subset$
contains	`\supset`	$\supset$
contains	`\supseteq`	$\supseteq$
union	`\cup`	$\cup$
intersection	`\cap`	$\cap$
big union	`\bigcup_{n=1}^{10}A_n`	$\bigcup_{n=1}^{10} A_n$
big intersection	`\bigcap_{n=1}^{10}A_n`	$\bigcap_{n=1}^{10} A_n$
empty set	`\emptyset`	$\emptyset$
power set	`\mathcal{P}`	$\mathcal{P}$
minimum	`\min`	$\min$
maximum	`\max`	$\max$
supremum	`\sup`	$\sup$
infimum	`\inf`	$\inf$
limit superior	`\limsup`	$\limsup$
limit inferior	`\liminf`	$\liminf$
closure	`\overline{A}`	$\overline{A}$

รูปที่ 7-13 Set theory symbols

รายละเอียด	คำสั่ง	ผลลัพธ์
not	`\neg`	$\neg$
and	`\land`	$\land$
or	`\lor`	$\lor$
if...then	`\implies`	$\implies$
if and only if	`\iff`	$\iff$
logical equivalence	`\equiv`	$\equiv$
therefore	`\therefore`	$\therefore$
there exists	`\exists`	$\exists$
for all	`\forall`	$\forall$

รูปที่ 7-14 Logic symbols

7.4 สรุปท้ายบท

บทนี้เป็นการแนะนำการใช้ R Markdown และ R Notebook ซึ่งเป็นเอกสารที่สามารถใส่คำอธิบาย สัญลักษณ์ สูตร สมการ เครื่องหมาย รูปภาพ หรือ Comments ต่าง ๆ ได้เหมือนการเขียน HTML ร่วมกับการเขียนโค้ดอยู่บน Platform เดียวกัน เนื้อหาประกอบด้วยการจัดรูปแบบเอกสารด้วยภาษา Markdown การแสดงรูปแบบต่าง ๆ ด้วย Code chunk options การสร้างสูตรและเครื่องหมายทางคณิตศาสตร์ด้วย Latex

7.5 แบบฝึกหัดท้ายบท

1. สร้างไฟล์ใหม่ที่เป็น R Markdown และ R Notebook โดยไปที่ File > New Flie > R Markdown และ R Notebook จากนั้นทำการคลิก Knit หรือ Preview เพื่อดูผลลัพธ์และเปรียบเทียบผลลัพธ์ระหว่าง R Markdown และ R Notebook ว่าเหมือนหรือแตกต่างกันอย่างไร
2. สร้างไฟล์ R Markdown เพื่อทดสอบเขียนข้อความหรือเพิ่มข้อมูลต่าง ๆ ดังต่อไปนี้

   2.1 เขียนข้อความที่เป็นตัวหนาสำหรับ Header 1

   2.2 เขียนข้อความที่เป็นตัวเอียงสำหรับ Header 2

   2.3 เขียนข้อความที่เป็นตัวหนาและตัวเอียงสำหรับ Header 3

   2.4 เพิ่มรูปภาพ 1 รูป โดยสามารถใช้ URL ของรูปภาพจากอินเทอร์เน็ต (เช่น https://www.r-project.org/logo/Rlogo.png)

   2.5 เพิ่มลิงก์ไปยังเว็บไซต์ https://www.r-project.org/

   2.6 เพิ่มสมการที่เกี่ยวข้องกับการหาค่าเฉลี่ย $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ พร้อมเขียนข้อความอธิบายความหมายของแต่ละตัวแปร

ส่วนที่ 2
การสร้างภาพข้อมูล (Data Visualization)

บทที่ 8
ggplot2 เบื้องต้น

บทนี้จะกล่าวถึงแพ็กเกจที่นิยมใช้ในการพล็อตมากที่สุดตัวหนึ่งคือ **ggplot2** เป็นแพ็กเกจที่ถูกสร้างโดย Hadley Wickham ซึ่งออกแบบมาเพื่อสนับสนุนงานพล็อตโดยเฉพาะ (Hadley Wickham, 2016) ออกแบบโดยใช้ไวยากรณ์ด้านกราฟิก (Grammar of Graphics) (Wilkinson, Wills, Rope, Norton, & Dubbs, 2006) มีองค์ประกอบในการพล็อตหลายลักษณะแยกเป็นอิสระต่อกัน ทำให้ **ggplot2** มีความยืดหยุ่นสูง สามารถรองรับการพล็อตที่ซับซ้อนได้ดี ผู้ใช้ยังสามารถสร้างรูปแบบการพล็อตตามความต้องการได้อีกด้วย

เพื่อจะได้เห็นภาพการทำงานของแพ็กเกจนี้จะใช้ข้อมูลที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า mtcars เป็นข้อมูลการใช้น้ำมันของรถยนต์ในช่วงปี ค.ศ. 1973 ถึง 1974 จาก 1974 Motor Trend US magazine ดังนี้

```
R> str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

ข้อมูล mtcars ประกอบด้วย

- mpg คือ จำนวนระยะทางต่อน้ำมันเชื้อเพลิง (miles per gallon)
- cyl คือ จำนวนกระบอกสูบ
- displ คือ ความจุกระบอกสูบ หน่วยเป็นลูกบาศก์นิ้ว
- hp คือ แรงม้า (Housepower)
- drat คือ อัตราทดเกียร์
- wt คือ น้ำหนัก (1000 ปอนด์)
- qsec คือ ระยะเวลาที่ใช้ในการเร่งที่ระยะทาง ¼ ไมล์
- vs คือ เครื่องยนต์ (0 = V-shaped, 1 = straight)
- am คือ เกียร์ (0 = automatic, 1 = manual)
- gear คือ จำนวนเกียร์
- carb คือ จำนวน carburetors

8.1 องค์ประกอบหลัก (Key Components)

การใช้ **ggplot2** จะประกอบด้วยข้อมูล 3 ส่วน คือ

- Data: ข้อมูลที่ต้องการพล็อต ซึ่งก็คือ เดต้าเฟรม (data.frame)
- Aesthetics: แสดงข้อมูลตัวแปร X และตัวแปร Y ที่ต้องการพล็อต และยังใช้ควบคุมรายละเอียดในการพล็อต เช่น สี ขนาด รูปร่างของวัตถุที่ต้องการพล็อต เป็นต้น
- Geometry: ชนิดของกราฟที่ต้องการแสดงผล มักสร้างด้วยฟังก์ชัน **geom_xxx()**[14]

คำสั่งหลักที่ใช้ในการพล็อต ได้แก่ **qplot()** สำหรับการพล็อตอย่างง่าย และ **ggplot()** สำหรับการพล็อตที่มีรายละเอียดปลีกย่อยมากขึ้น เหมาะกับงานพล็อตที่ซับซ้อน ในเอกสารฉบับนี้จะเน้นที่การใช้ **ggplot()** นอกจากนี้ยังมีคำสั่งที่สำคัญที่ใช้หลังการพล็อต ได้แก่ **last_plot()** สำหรับการคืนค่าการพล็อตล่าสุด และ **ggsave()** สำหรับบันทึกงานพล็อตเป็นไฟล์เพื่อใช้งานต่อไป

8.2 คำสั่ง qplot()

qplot() เป็นคำสั่งที่คล้ายกับการพล็อตด้วย base **R** ซึ่งสั่งพล็อตได้ง่าย รวดเร็ว รูปแบบการใช้เบื้องต้น ประกอบด้วยดังนี้

```
qplot(x, y = NULL, data, geom = "auto")
```

โดยที่ x, y คือ ค่าที่ต้องการพล็อต, data คือ เดต้าเฟรม และ geom คือ เวกเตอร์ประเภทกราฟที่ต้องการพล็อต ค่าโดยปริยายของ geom เท่ากับ "point" ถ้ามีการกำหนดค่า x และ y และเท่ากับ "histogram" ถ้ากำหนดค่า x เพียงอย่างเดียว นอกจากนั้นอาจจะเพิ่ม option ด้วยพารามิเตอร์อื่น ๆ เช่น main, xlab, ylab สำหรับกำหนดชื่อเรื่อง ชื่อแกน x และชื่อแกน y ตามลำดับ

8.2.1 Scatter plots

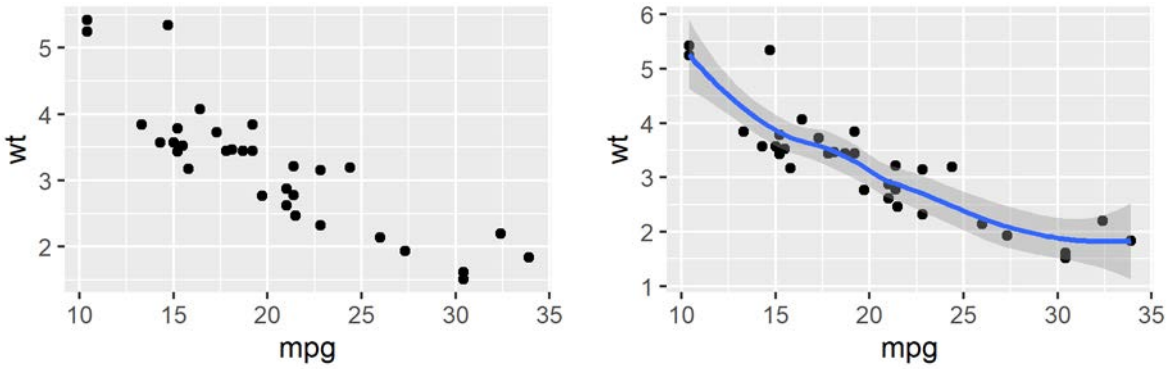
ถ้าต้องการพล็อตข้อมูล mtcars แสดงระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) กับน้ำหนักรถ เขียนได้ดังนี้

```
# Load package
R> library(tidyverse)
# Basic scatter plot
R> qplot(x = mpg, y = wt, data = mtcars)

# Scatter plot with smoothed line
```

[14] **geom_xxx()** หมายถึง ฟังก์ชันในกลุ่ม geom เช่น geom_point(), geom_line() เป็นต้น

```
R> qplot(x = mpg, y = wt, data = mtcars, geom = c("point","smooth"))
```



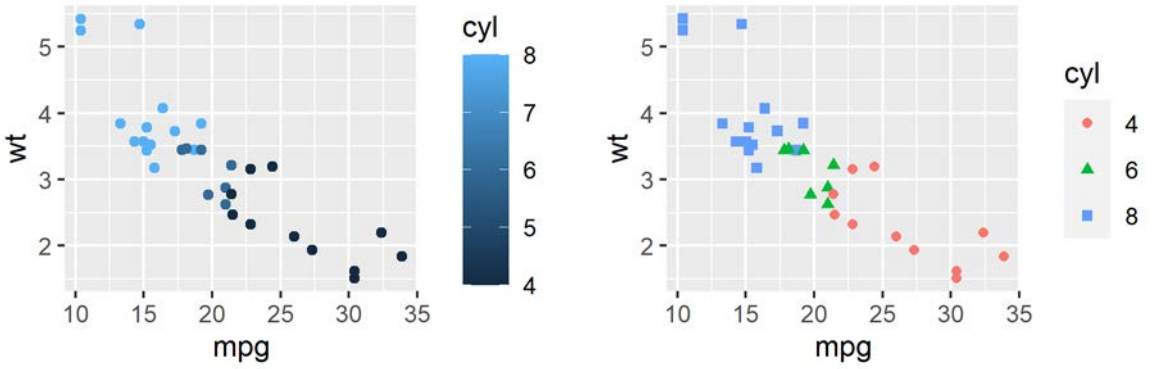
รูปที่ 8-1 Scatter plots: mpg vs. wt

รูปแรกแสดง Scatter plots แกน X เป็นระยะทางต่อน้ำมันเชื้อเพลิง (mpg) แกน Y เป็นน้ำหนัก (wt) รูปที่สองแสดง Scatter plots พร้อมกับเส้น Smooth และแถบสีเทาแสดงช่วงความเชื่อมั่น (Confidence interval) รอบเส้น Smooth

การเปลี่ยนสีและรูปร่างของจุดตามจำนวนกระบอกสูบ เขียนได้ดังนี้

```
# Change the color by a continuous numeric variable
R> qplot(mpg, wt, data = mtcars, color = cyl)

# Change color and shape by groups (factor)
R> mtcars$cyl <- factor(mtcars$cyl)
R> qplot(mpg, wt, data = mtcars, color = cyl, shape = cyl)
```



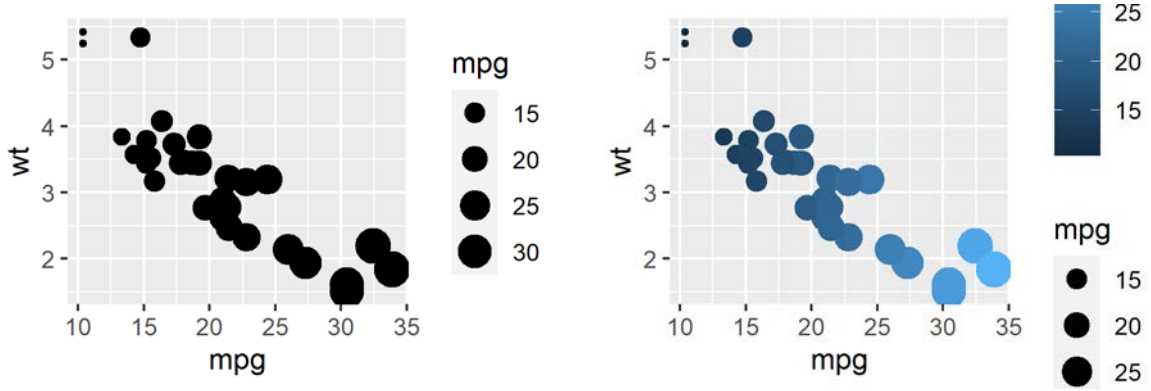
รูปที่ 8-2 Scatter plots: mpg vs. wt by color and shape

ทั้ง 2 รูปเป็นข้อมูลชุดเดียวกับรูปก่อนหน้านี้ รูปแรกแสดงสีของจุดตามจำนวนกระบอกสูบ (cyl) โดยแสดงเป็นสีน้ำเงินความเข้มค่อย ๆ ลดลงตามจำนวนกระบอกสูบ รูปที่สองแสดงสีและรูปร่างของจุดตามจำนวนกระบอกสูบ ถ้า cyl เท่ากับ 4 จะแสดงเป็นจุดวงกลมสีชมพู ถ้า cyl เท่ากับ 6 จะแสดงเป็นจุดสามเหลี่ยมสีเขียว และถ้า cyl เท่ากับ 8 จะแสดงเป็นจุดสี่เหลี่ยมสีฟ้า

การเปลี่ยนขนาดและสีของจุดตามจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (mpg) เขียนได้ดังนี้

```
# Change the size of points according to the values of a continuous variable
R> qplot(mpg, wt, data = mtcars, size = mpg)

# Change color and shape by a continuous variable
R> qplot(mpg, wt, data = mtcars, size = mpg, color = mpg)
```



รูปที่ 8-3 Scatter plots: mpg vs. wt by color and size

ทั้ง 2 รูปเป็นข้อมูลชุดเดียวกับรูปก่อนหน้านี้ แต่รูปแรกแสดงขนาดของจุดตามระยะทางต่อน้ำมันเชื้อเพลิง (mpg) ถ้า mpg สูงขึ้นจะแสดงขนาดของจุดที่ใหญ่ขึ้นตามลำดับ รูปที่สองแสดงทั้งขนาดและสีของจุดตาม mpg ถ้า mpg สูงขึ้นจะแสดงสีที่อ่อนลง และขนาดของจุดที่ใหญ่ขึ้นตามลำดับ

8.2.2 Box plots, histogram and density plots

สร้างข้อมูลเดต้าเฟรม ประกอบด้วยเพศ และน้ำหนัก เพื่อใช้ในการพล็อตต่อไป ดังนี้

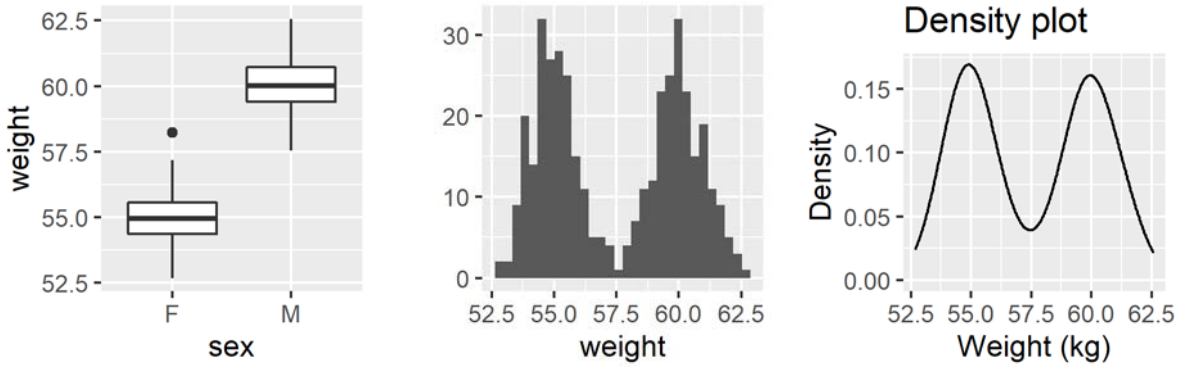
```
R> set.seed(123)
R> weight_data = data.frame(
     sex = factor(rep(c("F", "M"), each = 200)),
     weight = c(rnorm(200, 55), rnorm(200, 60)))
```

สร้าง Box plot, histogram and density plots ได้ดังนี้

```
# Basic box plot from data frame
# Change fill color by sex
R> qplot(sex, weight, data = weight_data, geom = "boxplot")

# Basic histogram
R> qplot(weight, data = weight_data, geom = "histogram")

# Density plot with main titles and axis labels
R> qplot(weight, data = weight_data, geom = "density",
      xlab = "Weight (kg)", ylab = "Density", main = "Density plot")
```



รูปที่ 8-4 Box plot, histogram, and density plots

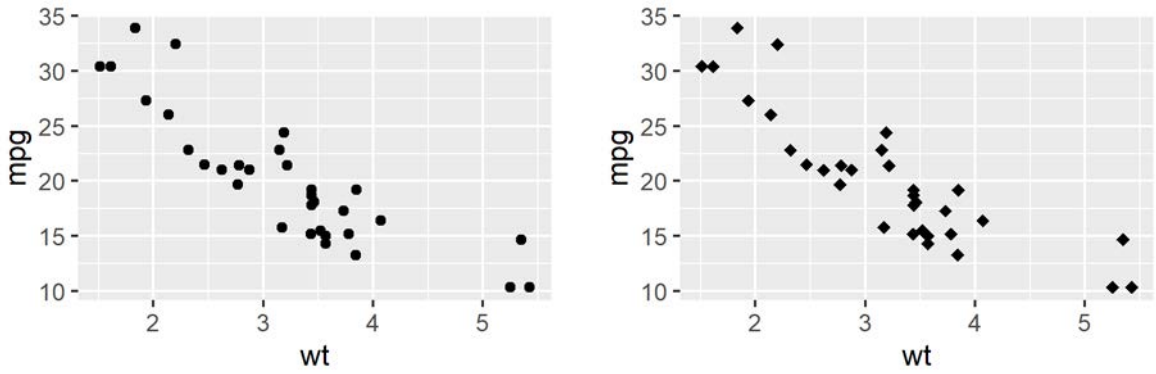
รูปแรกเป็น Box plots แกน X แสดงเพศ แกน Y แสดงน้ำหนัก รูปที่สองแสดงฮิสโตแกรม แกน X แสดงน้ำหนัก แกน Y เป็นจำนวนข้อมูล (Count) รูปสุดท้ายแสดง Density plots แกน X แสดงน้ำหนัก แกน Y เป็นสัดส่วนของข้อมูล

8.3 คำสั่ง ggplot()

`ggplot()` เป็นการพล็อตที่สามารถเพิ่มรายละเอียดปลีกย่อยได้มากขึ้น เหมาะกับงานพล็อตที่ซับซ้อน การพล็อตด้วยแพ็กเกจ `ggplot2` ประกอบด้วยข้อมูล 3 ส่วน ได้แก่ (1) Data คือข้อมูลที่ต้องการพล็อต ซึ่งก็คือ เดต้าเฟรม (data.frame) (2) Aesthetics แสดงข้อมูลตัวแปร X และตัวแปร Y ที่ต้องการพล็อต ซึ่งจะใช้ฟังก์ชัน `aes()` ในการระบุตัวแปร และ (3) Geometry กำหนดชนิดของกราฟที่ต้องการแสดงผล ตัวอย่างการสร้าง Scatter plots เขียนได้ดังนี้

```
# Load package
R> library(tidyverse)
# Basic scatter plot
R> ggplot(data = mtcars, aes(x = wt, y = mpg)) +
     geom_point()

# Change the point size, and shape
# Tidyverse style guide recommends omit often use arguments, i.e. data
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(size = 2.0, shape = 18)
```



รูปที่ 8-5 Scatter plots: wt vs. mpg

รูปแรกเป็น Scatter plots ที่แสดงข้อมูลแต่ละตัวด้วยจุดวงกลม รูปที่สองเป็นข้อมูลชุดเดียวกันแต่แสดงข้อมูลแต่ละตัวด้วยเครื่องหมาย (◆)

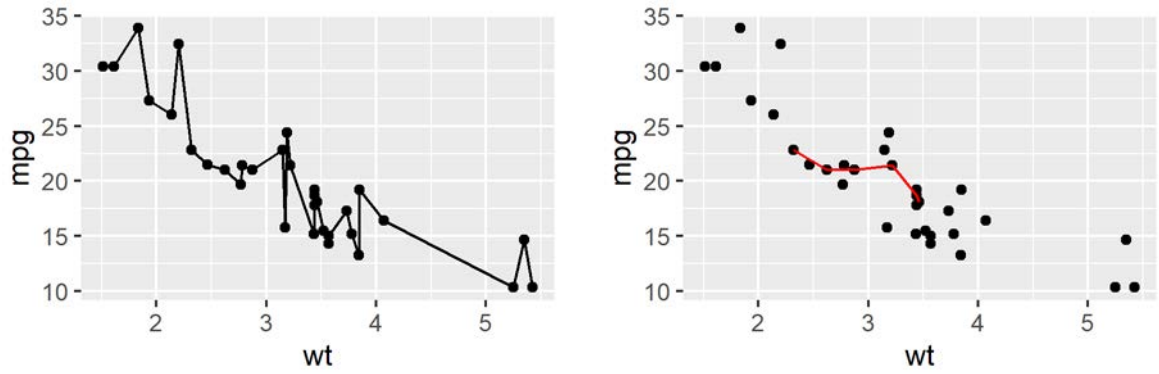
สามารถกำหนดข้อมูล Aesthetics เข้ามาเป็นข้อความ (String) ได้ด้วยฟังก์ชัน **aes_string()** ดังนี้

```
# Using aes_string() instead of aes()
R> ggplot(mtcars, aes_string(x = "wt", y = "mpg")) +
     geom_point(size = 2.0, shape = 18)
```

Geometry แสดงชนิดของกราฟที่ต้องการพล็อตจะแสดงผลเป็น Layer และใช้เครื่องหมายบวก + ในการเชื่อมคำสั่งต่อเนื่อง (Chain) แต่ละ Layer เข้ากับคำสั่ง **ggplot()** ทำให้สามารถพล็อตต่อเนื่องหลาย Layers และสามารถพล็อตแต่ละ Layer ด้วยข้อมูลที่แตกต่างกัน ได้ดังนี้

```
# Multiple layers
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point() +   # draw points
     geom_line()      # draw a line

# Multiple layers with different data
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point() +   # draw points
     geom_line(data = head(mtcars),
               color = "red") # draw a line with different data
```



รูปที่ 8-6 Scatter and line plots with different data

รูปแรกแสดง Scatter plots และเส้นตรงเชื่อมแต่ละจุด รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงเส้นตรงสีแดงเชื่อมเป็นบางจุด

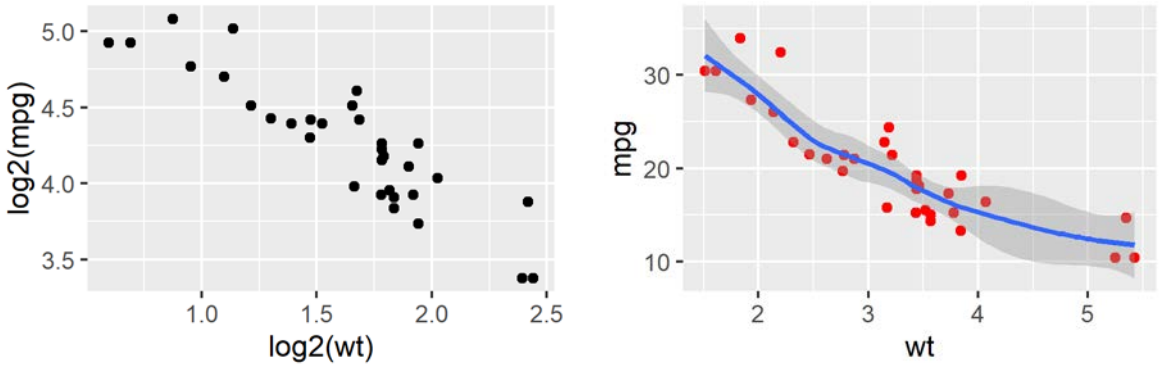
สามารถคำนวณค่าทางคณิตศาสตร์พื้นฐานภายในฟังก์ชัน `aes()` ได้ เช่น

```
# Log2 transformation in the aes()
R> ggplot(mtcars, aes(x = log2(wt), y = log2(mpg))) +
     geom_point()
```

ฟังก์ชัน `aes_string()` มีประโยชน์มากในการผ่านอากิวเมนต์เข้าไปในฟังก์ชัน ดังนี้

```
# Helper function for creating a scatter plot
# data: data.frame
# x_name, y_name: specify x and y variables, respectively
R> ggpoints <- function (data, x_name, y_name) {
       p <- ggplot(data = data, aes_string(x_name, y_name)) +
            geom_point(color = "red") +
            geom_smooth()
        p
  }

R> ggpoints(mtcars, x_name = "wt", y_name = "mpg")
```



รูปที่ 8-7 Scatter and line plots: log2(wt) vs. log2(mpg) and plots using function

รูปแรกแสดง Scatter plots แต่แกน X และแกน Y เป็นค่า Log ฐาน 2 รูปที่สองเป็นข้อมูลชุดเดียวกันแต่เป็นการสร้างกราฟโดยเรียกใช้ฟังก์ชันที่ผ่านอากิวเมนต์ข้อความเข้าไปในฟังก์ชัน

คำสั่ง **ggplot()** สามารถสร้างตัวแปรสำหรับเก็บวัตถุในการพล็อตเพื่อใช้ในการแสดงผลต่อไปได้ เช่น ตัวแปร **p** จากตัวอย่างข้างต้น

8.4 การบันทึกข้อมูล ggplots

การบันทึกไฟล์จาก **ggplot** ใช้คำสั่ง **print()** ได้ดังนี้

```
# Print the plot to a pdf file
R> pdf("my_plot.pdf")
R> my_plot <- ggplot(mtcars, aes(wt, mpg)) + geom_point()
R> print(my_plot)
R> dev.off()

# Print the plot to a png file
R> png("my_plot.png")
R> print(my_plot)
R> dev.off()
```

บันทึกไฟล์จาก **ggplot** หลังจากการพล็อตตามปกติได้ด้วยคำสั่ง **ggsave()** ดังนี้

```
# Create a plot: displayed on the screen (by default)
R> ggplot(mtcars, aes(wt, mpg)) +
     geom_point()

# Save the plot to a pdf
R> ggsave("my_plot.pdf")

# OR save it to png file
R> ggsave("my_plot.png", width = 3, height = 2)
```

8.5 สรุปท้ายบท

บทนี้เป็นการแนะนำการใช้แพ็กเกจ **ggplot2** ซึ่งเป็นแพ็กเกจที่นิยมใช้ในการพล็อตมากที่สุดตัวหนึ่ง โดยกล่าวถึงองค์ประกอบหลัก (Key Components) 3 ส่วน คือ Data, Aesthetics และ Geometry การใช้คำสั่ง **qplot()** ซึ่งเป็นคำสั่งในการพล็อตที่คล้ายกับการพล็อตด้วย base **R** สั่งพล็อตได้ง่าย รวดเร็ว การใช้คำสั่ง **ggplot()** เป็นการพล็อตที่สามารถเพิ่มรายละเอียดปลีกย่อยได้มากขึ้น เหมาะกับงานพล็อตที่ซับซ้อน และการบันทึกไฟล์จากการพล็อต

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`qplot()`	คำสั่งในการพล็อตอย่างง่าย
`ggplot()`	คำสั่งในการพล็อต
`print()`	คำสั่งในการสั่งพิมพ์
`pdf()`	คำสั่งในการสร้างไฟล์ pdf
`png()`	คำสั่งในการสร้างไฟล์ png
`dev.off()`	คำสั่งในการปิดไฟล์ Device ที่สร้างไว้ในการสั่งพิมพ์กราฟิก
`ggsave()`	คำสั่งในการบันทึกไฟล์กราฟิก
`rnorm()`	คำสั่งในการสร้างเลขสุ่มจาก Normal Distribution

8.6 แบบฝึกหัดท้ายบท

1. สร้าง Scatter plots โดยใช้ข้อมูลเดต้าเฟรมที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า iris ระหว่างตัวแปร Sepal.Length และ Petal.Length พร้อมแบ่งสีตามลักษณะของ Species

   1.1 ใช้คำสั่ง `qplot()`

   1.2 ใช้คำสั่ง `ggplot()`

2. สร้าง Box plots โดยใช้ข้อมูลเดต้าเฟรมที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า iris ด้วยคำสั่ง `qplot()` ในแต่ละตัวแปรดังต่อไปนี้ พร้อมแบ่งสีตามลักษณะของ Species

   2.1 สร้าง Box plots ของตัวแปร Sepal.Length

   2.2 สร้าง Box plots ของตัวแปร Sepal.Width

   2.3 สร้าง Box plots ของตัวแปร Petal.Length

   2.4 สร้าง Box plots ของตัวแปร Petal.Width

3. สร้าง Histograms โดยใช้ข้อมูลเดต้าเฟรมที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า iris ด้วยคำสั่ง `qplot()` ในแต่ละตัวแปรดังต่อไปนี้ พร้อมแบ่งสีตามลักษณะของ Species

   3.1 สร้าง Histograms ของตัวแปร Sepal.Length

   3.2 สร้าง Histograms ของตัวแปร Sepal.Width

   3.3 สร้าง Histograms ของตัวแปร Petal.Length

   3.4 สร้าง Histograms ของตัวแปร Petal.Width

4. สร้าง Density plots โดยใช้ข้อมูลเดต้าเฟรมที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า iris ด้วยคำสั่ง `qplot()` ในแต่ละตัวแปรดังต่อไปนี้ พร้อมแบ่งสีตามลักษณะของ Species

   4.1 สร้าง Density plots ของตัวแปร Sepal.Length

   4.2 สร้าง Density plots ของตัวแปร Sepal.Width

   4.3 สร้าง Density plots ของตัวแปร Petal.Length

   4.4 สร้าง Density plots ของตัวแปร Petal.Width

5. สร้าง Scatter plots โดยใช้ข้อมูลเดต้าเฟรมที่ติดตั้งมาพร้อมกับ **R** ชื่อว่า iris ระหว่างตัวแปร Sepal.Width และ Petal.Width พร้อมทั้งแสดงเส้น smooth line และแบ่งสีตามลักษณะของ Species จากนั้นทำการบันทึกเป็น file รูปภาพ (.png)

บทที่ 9
การพล็อตกราฟ 1 ตัวแปร (X): ตัวแปรต่อเนื่อง หรือตัวแปรแบ่งกลุ่ม

ในบทนี้จะกล่าวถึงการพล็อตกราฟที่มีตัวแปรต่อเนื่อง (Continuous variable) หรือตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) เพียง 1 ตัวแปร เท่านั้น Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

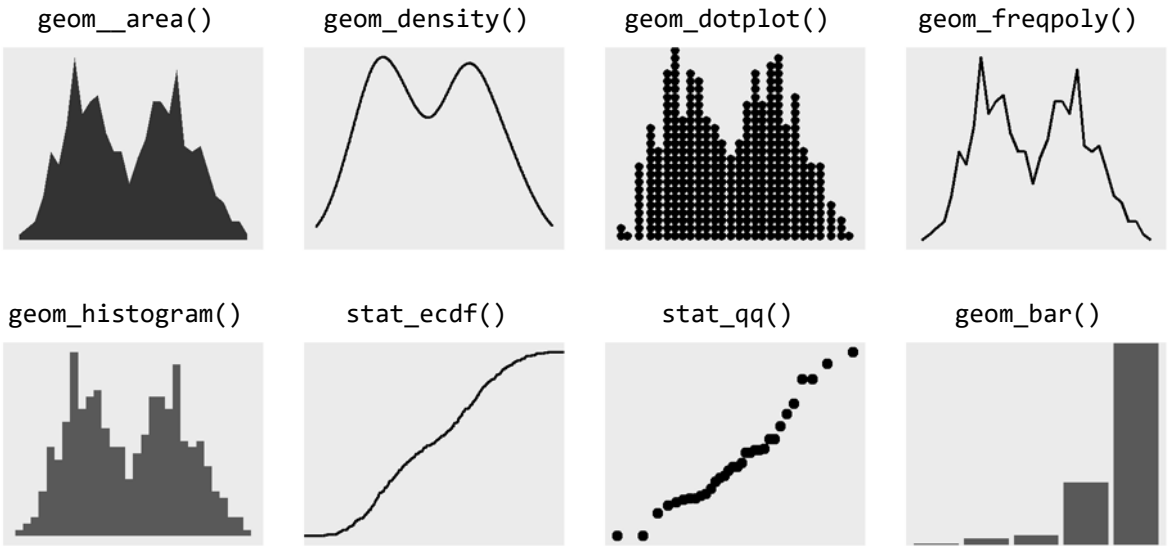
Geometry สำหรับตัวแปรต่อเนื่อง 1 ตัวแปร ได้แก่

- `geom_area()` สำหรับพล็อตกราฟพื้นที่ (Area plot)
- `geom_density()` สำหรับพล็อตกราฟความหนาแน่น (Density plot)
- `geom_dotplot()` สำหรับพล็อตกราฟจุดแสดงความหนาแน่น (Dot plot)
- `geom_freqpoly()` สำหรับพล็อตกราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency polygon plot)
- `geom_histogram()` สำหรับพล็อตฮีสโตแกรม (Histogram plot)
- `stat_ecdf()` สำหรับพล็อตกราฟความถี่สะสม (Cumulative density function plot)
- `stat_qq()` สำหรับพล็อตกราฟ Quantile–Quantile (Q-Q plots)

Geometry สำหรับตัวแปรแบ่งกลุ่ม 1 ตัวแปร ได้แก่

- `geom_bar()` สำหรับพล็อตกราฟแท่ง (Bar plot)

ตัวอย่างการพล็อตดังกล่าว แสดงดังรูปต่อไปนี้



รูปที่ 9-1 One continuous or discrete variable plots

ข้อมูลที่จะใช้ในการพล็อตเป็นเดต้าเฟรม ประกอบด้วยเพศ และน้ำหนัก ดังนี้

```
# Load package
R> library(tidyverse)

# Generate data.frame
R> set.seed(234)
R> weight_data = data.frame(
     sex = factor(rep(c("F", "M"), each = 200)),
     weight = c(rnorm(200, 55), rnorm(200, 60)))

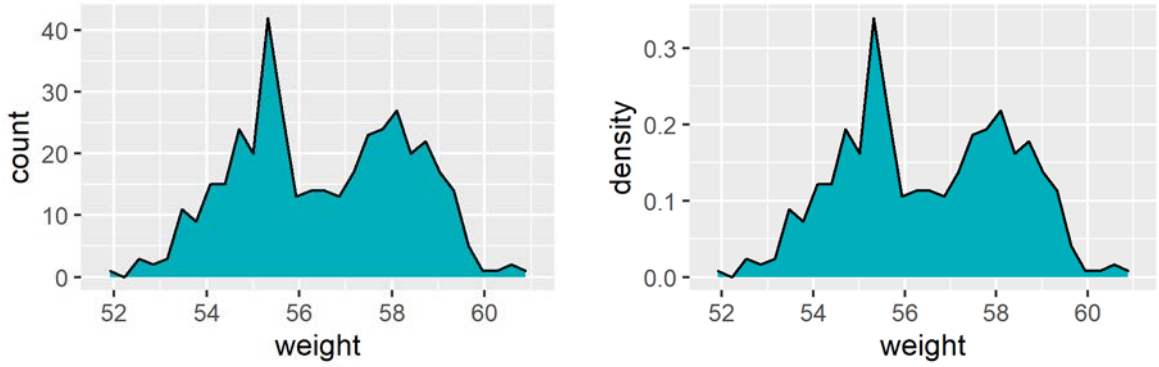
# Save an object for further plot
R> a <- ggplot(weight_data, aes(x = weight))
```

9.1 กราฟพื้นที่ (Area Plots)

กราฟพื้นที่ (Area plots) เป็นการแสดงการกระจายตัวตามจำนวนข้อมูลหรือความถี่ข้อมูลโดยใช้เส้นและพื้นที่ใต้กราฟ การพล็อตกราฟพื้นที่ใช้คำสั่ง `geom_area()` หรือ `stat_bin()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size` ตัวอย่างการพล็อตกราฟพื้นที่แสดงได้ดังนี้

```
# Area plot -- y axis is (default) count
R> a + geom_area(stat = "bin", color = "black", fill = "#00AFBB")

# Area plot -- y axis is density
R> a + geom_area(aes(y = ..density..), stat = "bin", color = "black",
              fill = "#00AFBB")
```



รูปที่ 9-2 Area plots -- y axis is count and density

รูปแรกเป็นกราฟพื้นที่แสดงแกน X เป็นน้ำหนัก แกน Y เป็นจำนวนข้อมูล (Count) รูปที่สองเป็นข้อมูลเดียวกันแต่แกน Y แสดงเป็นสัดส่วนของข้อมูลหรือ density แทน

การเปรียบเทียบระหว่างกราฟพื้นที่ (Area plot) และกราฟแท่ง (Bar plot) แบ่งตามกลุ่มข้อมูลโดยใช้ข้อมูลชื่อ diamonds แสดงได้ดังนี้

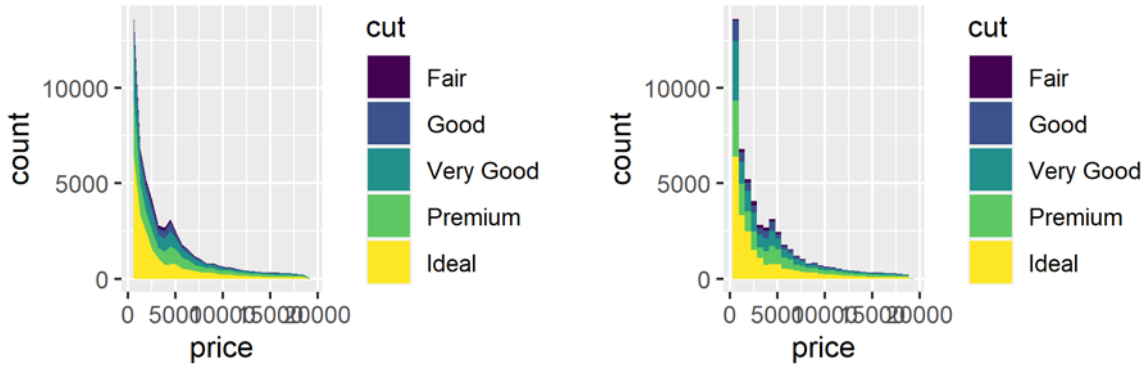
```
# Load the data
R> data("diamonds")
```

```
R> diamonds
# A tibble: 53,940 × 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
# i 53,930 more rows
# i Use `print(n = ...)` to see more rows

R> p <- ggplot(diamonds, aes(x = price, fill = cut))

# Area plot
R> p + geom_area(stat = "bin")

# Bar plot
R> p + geom_bar(stat = "bin")
```



รูปที่ 9-3 Area and bar plots

รูปแรกเป็นกราฟพื้นที่แกน X เป็นราคา แกน Y เป็นจำนวนข้อมูล (Count) โดยแยกสีตามประเภทการ cut รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงด้วยกราฟแท่ง (Bar plot)

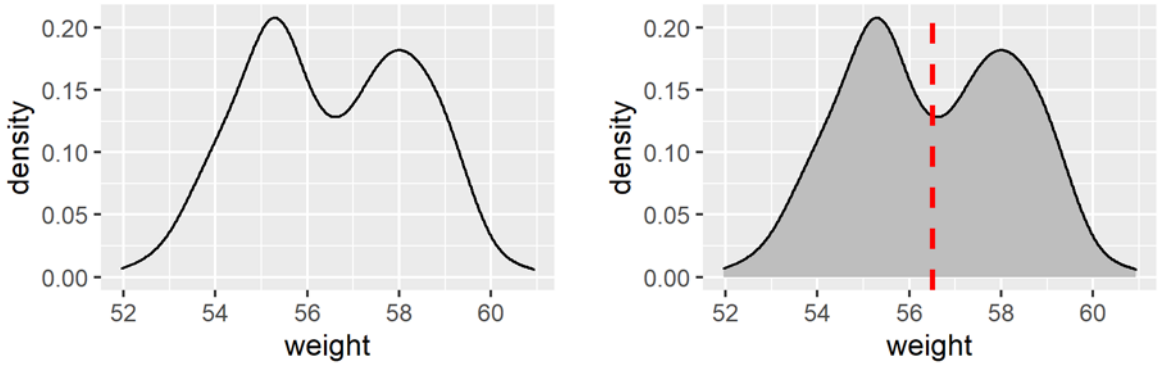
9.2 กราฟความหนาแน่น (Density Plots)

กราฟความหนาแน่น (Density plots) เป็นกราฟที่แสดง Kernel probability density ใช้คำสั่ง `geom_density()` หรือ `stat_ density()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size` ตัวอย่างการพล็อตความหนาแน่น แสดงได้ดังนี้

```
# Basic density plot
```

```
R> a + geom_density()

# Change line color and fill color, add mean line
R> a + geom_density(color = "black", fill = "grey") +
     geom_vline(aes(xintercept = mean(weight)),
                color = "red", linetype = "dashed", size = 1)
```



รูปที่ 9-4 Density plots

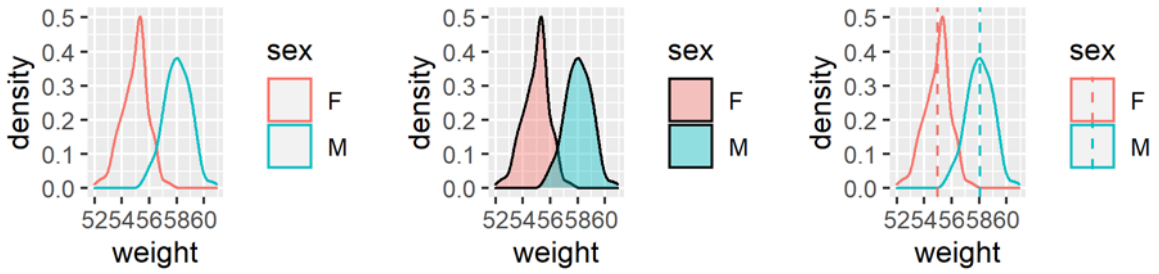
รูปแรกเป็นกราฟความหนาแน่น แกน X เป็นน้ำหนัก แกน Y เป็นสัดส่วนของข้อมูล รูปที่สองเป็นข้อมูลเดียวกันแต่มีการเพิ่มสีเทาบริเวณพื้นที่ใต้กราฟ และเพิ่มเส้นประแนวตั้งสีแดงแสดงค่าเฉลี่ย (Mean)

การเปลี่ยนสีตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `fill` หรือ `color` ดังนี้

```
# Change line colors by sex
R> a + geom_density(aes(color = sex))

# Change fill color by sex
# Use semi-transparent fill: alpha = 0.4
R> a + geom_density(aes(fill = sex), alpha = 0.4)

# Add mean lines and color by sex
R> mu <- weight_data |>
     group_by(sex) |>
     summarise(group_mean = mean(weight))
R> a + geom_density(aes(color = sex), alpha = 0.4)+
     geom_vline(mu, aes(xintercept = group_mean, color = sex),
                linetype = "dashed")
```



รูปที่ 9-5 Density plots -- color by groups

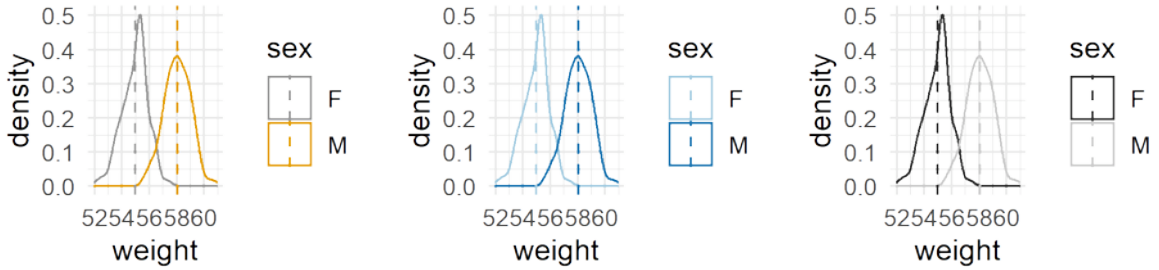
รูปแรกเป็นกราฟความหนาแน่นที่แสดงสีตามเพศ โดยเพศหญิงแสดงเป็นเส้นสีชมพู เพศชายแสดงเป็นเส้นสีฟ้า รูปที่สองเป็นข้อมูลเดียวกันแต่เปลี่ยนเป็นแสดงสีพื้นที่ใต้กราฟแทน รูปสุดท้ายเหมือนกับรูปแรกแต่เพิ่มเส้นค่าเฉลี่ยในแนวตั้งแทน

สามารถเปลี่ยนสีเส้นด้วยคำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` (ดูรายละเอียดการใช้สีเพิ่มเติมจาก https://colorbrewer2.org) ดังนี้

```
# Add mean line and color manually
R> a2 <- a + geom_density(aes(color = sex)) +
     geom_vline(data = mu, aes(xintercept = group_mean, color = sex),
                linetype = "dashed") +
     theme_minimal()
R> a2 + scale_color_manual(values = c("#999999", "#E69F00"))

# Use brewer color palettes
R> a2 + scale_color_brewer(palette = "Paired")

# Use grey scale
R> a2 + scale_color_grey()
```



รูปที่ 9-6 Density plots -- line colors

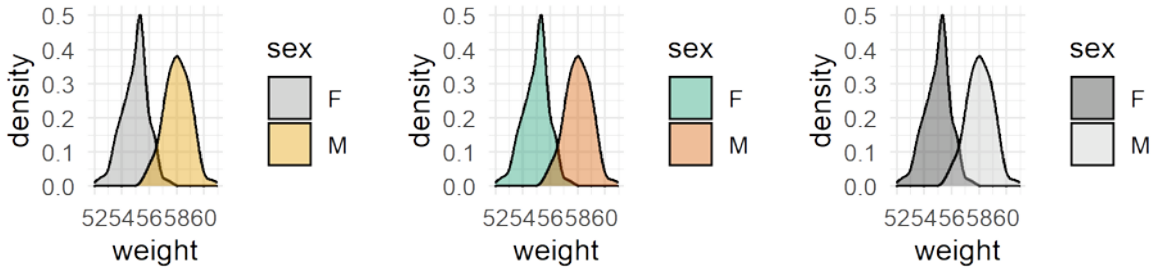
รูปแรกเป็นกราฟความหนาแน่นที่แสดงสีตามเพศ โดยเพศหญิงแสดงเป็นเส้นสีเทา เพศชายแสดงเป็นเส้นสีเหลือง รูปที่สองเป็นข้อมูลเดียวกันแต่เปลี่ยนเพศหญิงแสดงเป็นเส้นสีฟ้าอ่อน เพศชายแสดงเป็นเส้นสีฟ้าเข้ม รูปสุดท้ายเหมือนกับ 2 รูปแรกแต่เปลี่ยนเพศหญิงแสดงเป็นเส้นสีเทาเข้ม เพศชายแสดงเป็นเส้นสีเทาอ่อน

สามารถเติมสี (Fill) ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_fill_grey()` ดังนี้

```
# Fill manually
R> a3 <- a + geom_density(aes(fill = sex), alpha = 0.4) +
     theme_minimal()
R> a3 + scale_fill_manual(values = c("#999999", "#E69F00"))

# Use brewer color palettes
R> a3 + scale_fill_brewer(palette = "Dark2") +
     theme_minimal()
```

```
# Use grey scale
R> a3 + scale_fill_grey() +
     theme_minimal()
```



รูปที่ 9-7 Density plots -- fill colors

รูปแรกเป็นกราฟความหนาแน่นที่แสดงสีพื้นที่ใต้กราฟตามเพศ โดยเพศหญิงแสดงเป็นสีเทา เพศชายแสดงเป็นสีเหลือง รูปที่สองเป็นข้อมูลเดียวกันแต่เปลี่ยนเพศหญิงแสดงเป็นสีเขียวอ่อน เพศชายแสดงเป็นเส้นสีชมพู รูปสุดท้ายเหมือนกับทั้ง 2 รูปแรกแต่เปลี่ยนเพศหญิงแสดงเป็นสีเทาเข้ม เพศชายแสดงเป็นสีเทาอ่อน

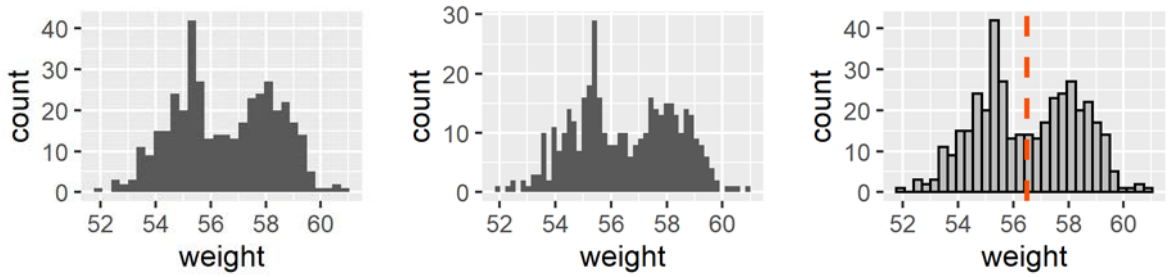
9.3 ฮีสโตแกรม (Histograms)

ฮิสโตแกรม (Histogram) เป็นกราฟแท่งที่บอกถึงความถี่ที่เกิดขึ้นในแต่ละอันตรภาคชั้น การพล็อตฮีสโตแกรม (Histogram plots) ใช้คำสั่ง `geom_histogram()` หรือ `stat_bin()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size` ค่าโดยปริยายของ `bins = 30` และแกน Y เท่ากับ count ถ้าต้องการให้แกน Y แสดงความหนาแน่นใช้คำสั่ง `aes(y = ..density..)` เช่นเดียวกับการพล็อตในหัวข้อก่อนหน้านี้ ตัวอย่างการพล็อตฮีสโตแกรม แสดงได้ดังนี้

```
# Basic histogram
R> a + geom_histogram()

# Change the number of bins
R> a + geom_histogram(bins = 50)

# Change line color and fill color, add mean line
R> a + geom_histogram(color = "black", fill = "grey")+
     geom_vline(aes(xintercept = mean(weight)),
                color = "#FC4E07", linetype = "dashed", size = 1)
```



รูปที่ 9-8 Histograms

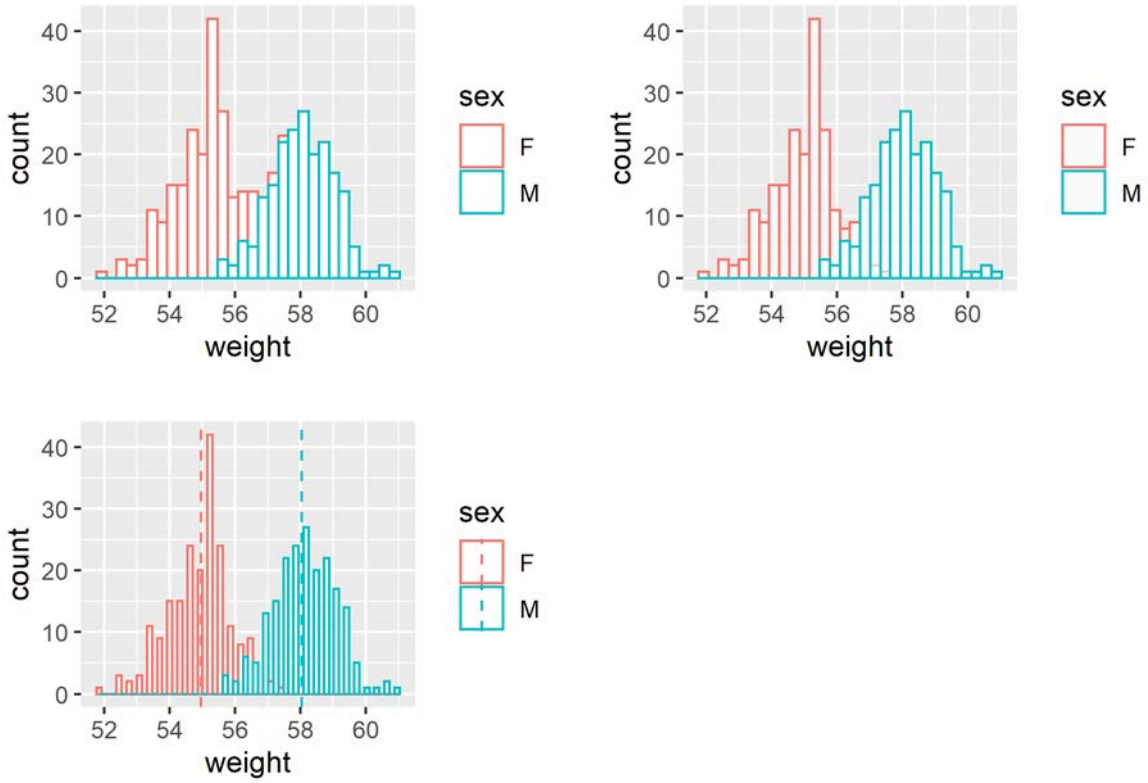
รูปแรกเป็นฮิสโตแกรมที่แกน X แสดงน้ำหนัก แกน Y เป็นจำนวนข้อมูล (Count) ค่าโดยปริยายกำหนดให้จำนวนแท่งกราฟเท่ากับ 30 แท่ง รูปที่สองเป็นข้อมูลเดียวกันแต่เพิ่มจำนวนแท่งกราฟให้แสดงเป็น 50 แท่ง รูปสุดท้ายเหมือนกับ 2 รูปแรกแต่แสดงแท่งเป็นสีเทากรอบสีดำ และเพิ่มเส้นประแนวตั้งสีแดงแสดงค่าเฉลี่ย (Mean)

การเปลี่ยนสีตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `fill` หรือ `color` กำหนดตำแหน่งของแท่งกราฟ (Bin) ด้วยคำสั่ง `position` เท่ากับ (1) `"identity"` หมายถึงแท่งกราฟแต่ละ Bin มีการซ้อนทับกัน (2) `"stack"` หมายถึงแท่งกราฟแต่ละ Bin มีการวางซ้อนกันในแนวตั้ง (3) `"dodge"` หมายถึงแท่งกราฟแต่ละ Bin วางตัวโดยไม่ซ้อนทับกัน ค่าโดยปริยายกำหนดให้ `position = "stack"` ดังนี้

```
# Change line colors by sex
# By default position = "stack"
R> a + geom_histogram(aes(color = sex), fill = "white")

# Position adjustment: "identity" (overlaid)
R> a + geom_histogram(aes(color = sex), fill = "white", alpha = 0.6,
                      position = "identity")

# Position adjustment: "dodge" (Interleaved)
# Add mean lines and color by sex
R> a + geom_histogram(aes(color = sex), fill = "white",
                      position = "dodge") +
     geom_vline(data = mu, aes(xintercept = group_mean, color = sex),
                linetype = "dashed")
```



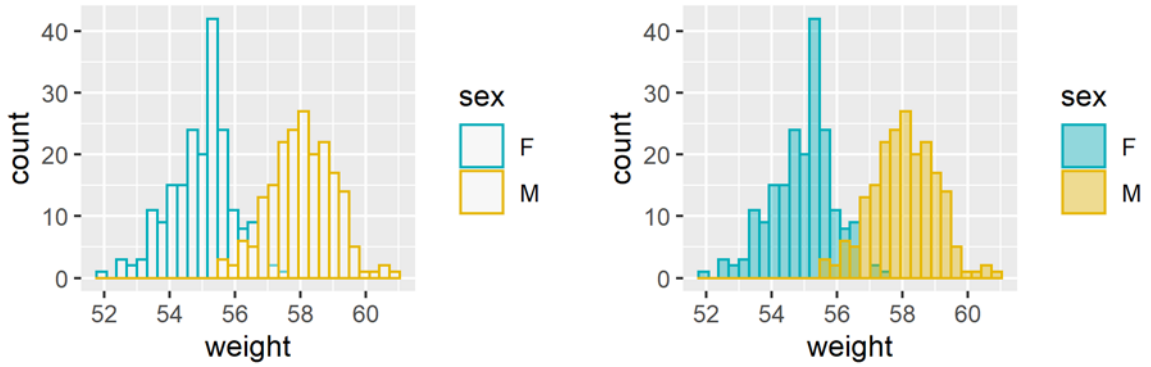
รูปที่ 9-9 Histogram: color by groups and positions by stack, identity and dodge

รูปซ้ายบนเป็นฮิสโตแกรมที่แสดงสีแท่งกราฟตามเพศ โดยเพศหญิงแสดงเป็นเส้นสีชมพู เพศชายแสดงเป็นเส้นสีฟ้า ส่วนกราฟแต่ละแท่งวางซ้อนกันในแนวตั้ง (Stack) รูปขวาบนเป็นข้อมูลเดียวกันแต่กราฟแต่ละแท่งมีการซ้อนทับกัน (Overlaid) รูปล่างเป็นข้อมูลเดียวกันแต่กราฟแต่ละแท่งวางตัวโดยไม่ซ้อนทับกัน (Interleaved)

สามารถเปลี่ยนสีเส้นด้วยคำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` และสามารถเติมสี (Fill) ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_fill_grey()` ดังนี้

```
# Use classic theme and change legend position to "top"
# Change outline color manually
R> a + geom_histogram(aes(color = sex), fill = "white",
                      alpha = 0.4, position = "identity") +
     scale_color_manual(values = c("#00AFBB", "#E7B800"))

# Change fill and outline color manually
R> a + geom_histogram(aes(color = sex, fill = sex),
                      alpha = 0.4, position = "identity") +
     scale_fill_manual(values = c("#00AFBB", "#E7B800")) +
     scale_color_manual(values = c("#00AFBB", "#E7B800"))
```



รูปที่ 9-10 Histogram: color and fill color by groups

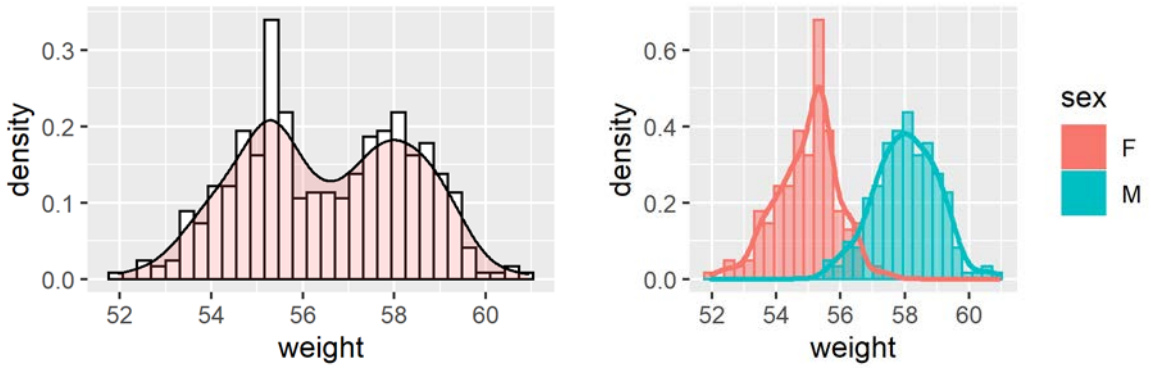
รูปแรกเป็นฮิสโตแกรมที่แสดงสีแท่งกราฟตามเพศ โดยเพศหญิงแสดงเป็นเส้นกรอบสีฟ้า เพศชายแสดงเป็นเส้นกรอบสีเหลือง และกราฟแต่ละแท่งมีการซ้อนทับกัน (Overlaid) รูปที่สองเป็นข้อมูลเดียวกันแต่เป็นการเติมสี (Fill) แท่งกราฟดังกล่าวแทน

9.4 ฮิสโตแกรมและกราฟความหนาแน่น (Combine Histogram and Density Plots)

การพล็อตฮิสโตแกรมและกราฟความหนาแน่นร่วมกันต้องใช้ฟังก์ชัน `geom_histogram()` ด้วยการกำหนด `aes(y = ..density..)` เพื่อให้แกน y แสดงความหนาแน่น (Density) แทนที่จะแสดงเป็นจำนวนข้อมูล (Count) ดังนี้

```
# Histogram with density plot
R> a + geom_histogram(aes(y = ..density..), color = "black", fill = "white") +
     geom_density(alpha = 0.2, fill = "#FF6666")

# Color by groups
R> a + geom_histogram(aes(y = ..density.., color = sex, fill = sex),
                      alpha = 0.5, position = "identity") +
     geom_density(aes(color = sex), size = 1)
```



รูปที่ 9-11 Combine histogram and density plots

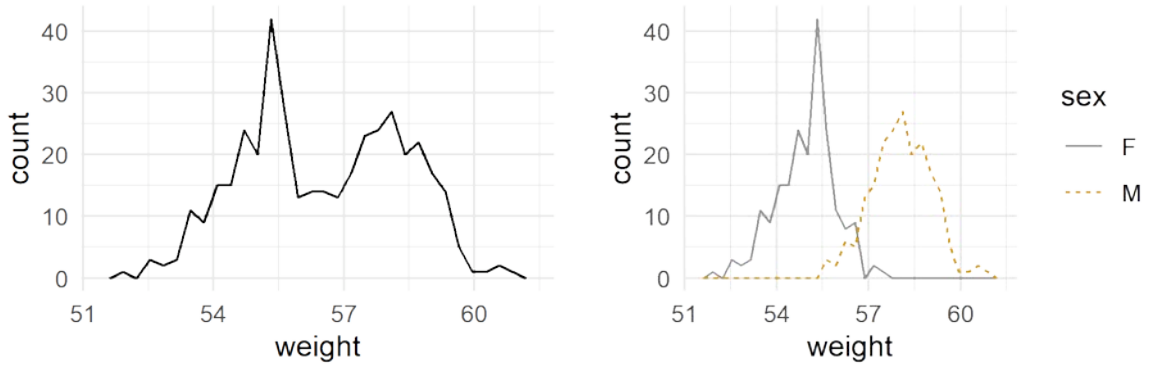
รูปแรกเป็นฮิสโตแกรมและความหนาแน่นบนรูปเดียวกัน โดยแท่งกราฟใช้กรอบสีดำ ส่วนพื้นที่ใต้กราฟความหนาแน่นแสดงเป็นสีชมพู รูปที่สองเป็นฮิสโตแกรมและความหนาแน่นบนรูปเดียวกัน แต่แสดงสีตามเพศ โดยเพศหญิงแสดงด้วยสีชมพู เพศชายแสดงด้วยสีฟ้า และกราฟแต่ละแท่งมีการซ้อนทับกัน (Overlaid)

9.5 กราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency Polygon Plots)

กราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency polygon plots) คล้ายกับฮีสโตแกรม แต่มีความแตกต่างกันโดยที่ฮีสโตแกรมใช้แท่งกราฟ แต่กราฟความถี่แบบพื้นที่หลายเหลี่ยมใช้เส้นในการแสดงผล การพล็อตกราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency polygon plots) ใช้คำสั่ง `geom_freqpoly()` หรือ `stat_bin()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, linetype` และ `size` ถ้าต้องการให้แกน y เป็นความหนาแน่นใช้คำสั่ง `aes(y = ..density..)` เช่นเดียวกับการพล็อตในหัวข้อก่อนหน้านี้ ตัวอย่างการพล็อตแสดงได้ดังนี้

```
# Frequency polygon plot
R> a + geom_freqpoly(bins = 30) +
     theme_minimal()

# Change color and linetype by sex
# Use custom color palettes
R> a + geom_freqpoly(aes(color = sex, linetype = sex)) +
     scale_color_manual(values = c("#999999", "#E69F00")) +
     theme_minimal()
```



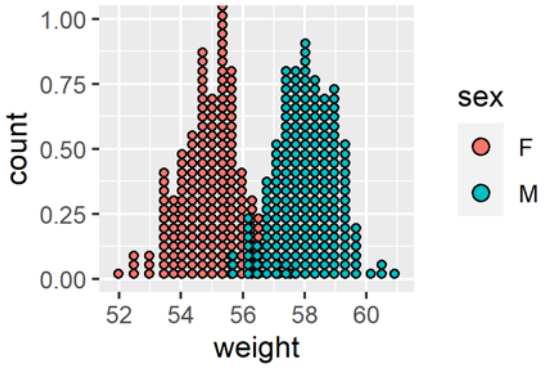
รูปที่ 9-12 Frequency polygon plots

รูปแรกเป็นกราฟความถี่แบบพื้นที่หลายเหลี่ยมคล้ายกับฮีสโตแกรมแต่ใช้เส้นในการแสดงผล รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงสีและประเภทของเส้นตามเพศ โดยเพศหญิงแสดงด้วยเส้นทึบสีเทา เพศชายแสดงด้วยเส้นประสีเหลือง

9.6 กราฟจุดแสดงความหนาแน่น (Dot Plots)

กราฟจุดแสดงความหนาแน่น (Dot Plots) เป็นกราฟที่บอกถึงความถี่ที่เกิดขึ้นในแต่ละอันตรภาคชั้น โดยใช้จุดแสดงความถี่ที่เกิดขึ้น การพล็อตกราฟจุดแสดงความหนาแน่นใช้คำสั่ง `geom_dotplot()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill` และ `dotsize` ตัวอย่างการพล็อตแสดงได้ดังนี้

```
# Dot plot
# change fill and color by sex
R> a + geom_dotplot(aes(fill = sex))
```



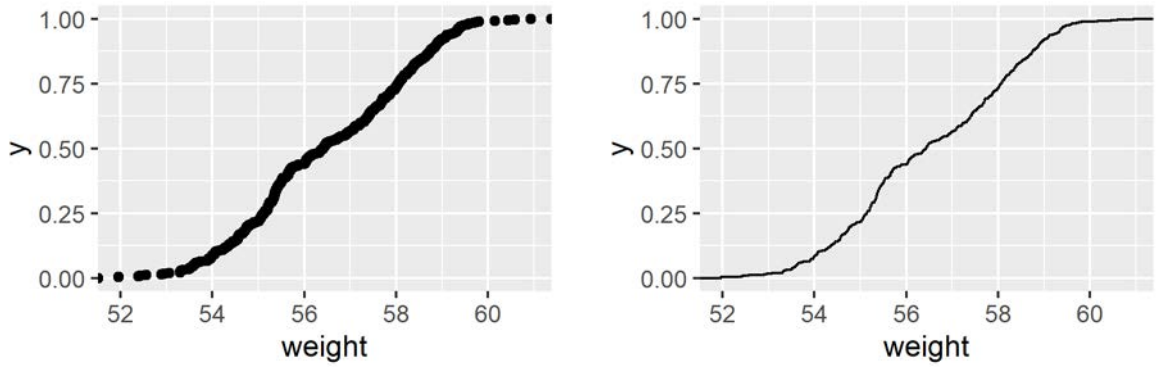
รูปที่ 9-13 Dot plot

รูปด้านบนแสดงความหนาแน่น (Density) ของข้อมูลน้ำหนัก โดยใช้จุดเรียงต่อกันในแนวตั้ง และใช้สีจุดแยกตามเพศ โดยเพศหญิงแสดงด้วยจุดสีชมพู เพศชายแสดงด้วยจุดสีฟ้า

9.7 กราฟความถี่สะสม (Cumulative Density Function Plots)

กราฟความถี่สะสม (Cumulative density function plots) ใช้แสดงค่าความถี่สะสม โดยค่าในแกน Y แสดงค่าสัดส่วนตั้งแต่ 0.0–1.0 การพล็อตกราฟความถี่สะสมใช้คำสั่ง `stat_ecdf()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size` ตัวอย่างการพล็อตแบบพื้นที่แสดงได้ดังนี้

```
# Empirical cumulative density function plot
R> a + stat_ecdf(geom = "point")
R> a + stat_ecdf(geom = "step")
```



รูปที่ 9-14 Cumulative density function plots

ทั้ง 2 รูปแกน X แสดงน้ำหนัก แกน Y เป็นความถี่สะสม รูปแรกแสดงด้วยจุดวงกลม ส่วนรูปที่สองเป็นข้อมูลเดียวกันแต่แสดงด้วยเส้นต่อเนื่องกันตามลำดับ

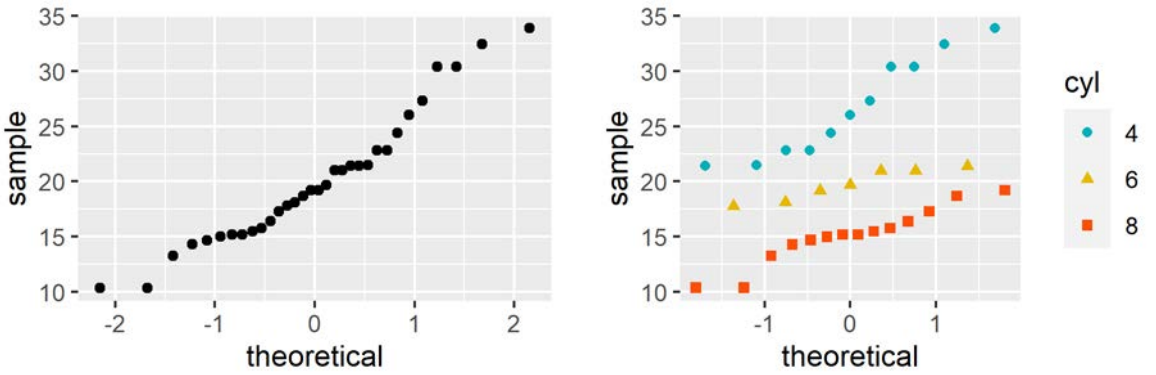
9.8 กราฟ Quantile – Quantile (Q-Q plots)

กราฟ Quantile–Quantile (Q-Q plots) ใช้ตรวจสอบว่าข้อมูลมีการกระจายแบบปกติ (Normal distribution) หรือไม่ การพล็อตกราฟ Q-Q ใช้คำสั่ง `stat_qq()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติม ได้แก่ `alpha, color, shape` และ `size` ตัวอย่างการพล็อต แสดงได้ดังนี้

```
R> data(mtcars)
# Convert cyl column from a numeric to a factor variable
R> mtcars$cyl <- as.factor(mtcars$cyl)

R> p <- ggplot(mtcars, aes(sample = mpg))
R> p + stat_qq()

# Change point shapes by groups
# Use custom color palettes
R> p + stat_qq(aes(shape = cyl, color = cyl)) +
     scale_color_manual(values = c("#00AFBB", "#E7B800", "#FC4E07"))
```



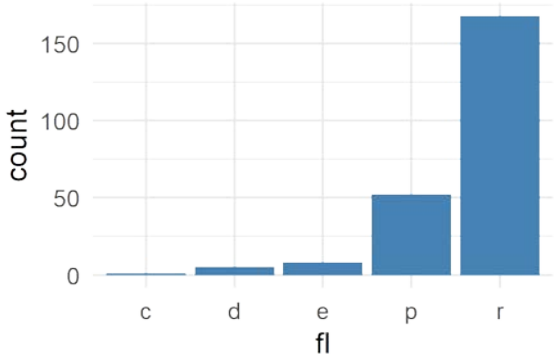
รูปที่ 9-15 Quantile–Quantile plots (Q-Q plots)

รูปแรกเป็นกราฟ Quantile–Quantile ถ้าแต่ละจุดอยู่ในแนวเดียวกันเหมือนเป็นเส้นตรงแสดงว่าข้อมูลกระจายแบบปกติ (Normal distribution) รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงสีและรูปร่างจุดตามจำนวนกระบอกสูบ (cyl) ถ้า cyl เท่ากับ 4 จะแสดงเป็นจุดวงกลมสีฟ้า ถ้า cyl เท่ากับ 6 จะแสดงเป็นจุดสามเหลี่ยมสีเหลือง และถ้า cyl เท่ากับ 8 จะแสดงเป็นจุดสี่เหลี่ยมสีแดง

9.9 กราฟแท่ง (Bar Plots)

กราฟแท่ง (Bar Plots) คือ กราฟที่ประกอบไปด้วยแกนตั้งและแกนนอน และรูปสี่เหลี่ยมมุมฉากที่มีความสูงแตกต่างกันขึ้นอยู่กับขนาดของข้อมูล เรียกรูปสี่เหลี่ยมแต่ละรูปนี้ว่าแท่ง (bar) การพล็อตกราฟแท่งใช้คำสั่ง `geom_bar()` หรือ `stat_count()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype,` และ `size` ตัวอย่างการพล็อตแสดงได้ดังนี้

```
R> data(mpg)
R> ggplot(mpg, aes(fl)) +
     geom_bar(fill = "steelblue") +
     theme_minimal()
```



รูปที่ 9-16 Bar plots

รูปด้านบนแสดงกราฟแท่ง (Bar plots) แกน X แสดงประเภทของน้ำมันเชื้อเพลิงที่ใช้ (fl) แกน Y แสดงจำนวนข้อมูล (Count) ตามประเภทน้ำมันเชื้อเพลิงที่ใช้

9.10 สรุปท้ายบท

ในบทนี้จะกล่าวถึงการพล็อตกราฟที่มีตัวแปรต่อเนื่อง (Continuous variable) หรือตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) เพียง 1 ตัวแปร เท่านั้น กราฟที่ใช้ในการพล็อตดังกล่าวประกอบด้วย กราฟพื้นที่ (Area plots) กราฟความหนาแน่น (Density plots) กราฟจุดแสดงความหนาแน่น (Dot plots) กราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency polygon plots) ฮีสโตแกรม (Histogram) กราฟความถี่สะสม (Cumulative density function plots) กราฟ Quantile–Quantile (Q-Q plots) กราฟแท่ง (Bar plots)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`ggplot()`	คำสั่งในการพล็อต
`geom_area()`	Geometry สำหรับพล็อตกราฟพื้นที่ (Area plots)
`geom_density()`	Geometry สำหรับพล็อตกราฟความหนาแน่น (Density plots)
`geom_dotplot()`	Geometry สำหรับพล็อตกราฟจุดแสดงความหนาแน่น (Dot plots)
`geom_freqpoly()`	Geometry สำหรับพล็อตกราฟความถี่แบบพื้นที่หลายเหลี่ยม (Frequency polygon plots)
`geom_histogram()`	Geometry สำหรับพล็อตฮีสโตแกรม (Histogram)
`geom_ecdf()`	Geometry สำหรับพล็อตกราฟความถี่สะสม (Empirical cumulative density function)
`geom_qq()`	Geometry สำหรับพล็อตกราฟ Quantile – Quantile (Q-Q plots)
`geom_bar()`	Geometry สำหรับพล็อตกราฟแท่ง (Bar chart)
`geom_vline()`	Geometry สำหรับพล็อตเส้นในแนวตั้ง (Vertical line plots)
`scale_color_manual()`	คำสั่งในการกำหนดสีแบบ Manual
`scale_color_brewer()`	คำสั่งในการกำหนดสีด้วยจานสี (Palettes)
`scale_color_grey()`	คำสั่งในการกำหนดสีด้วยจานสีเทา
`scale_fill_manual()`	คำสั่งในการเติมสีแบบ Manual
`scale_fill_brewer()`	คำสั่งในการเติมสีด้วยจานสี (Palettes)
`scale_fill_grey()`	คำสั่งในการเติมสีด้วยจานสีเทา
`theme_minimal()`	คำสั่งในการกำหนดธีมแบบ Minimal
`scale_shape_manual()`	คำสั่งในการกำหนดรูปร่างแบบ Manual
`scale_size_manual()`	คำสั่งในการกำหนดขนาดแบบ Manual

9.11 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ dplyr ชื่อว่า starwars ด้วยตัวแปร height และ mass สร้างกราฟต่าง ๆ ดังต่อไปนี้

   1.1 สร้างกราฟพื้นที่

   1.2 สร้างกราฟแท่ง

   1.3 สร้างกราฟความหนาแน่น

   1.4 สร้างกราฟฮีสโตแกรมพร้อมกราฟความหนาแน่นและสร้างเส้นค่าเฉลี่ย

   1.5 สร้างกราฟความถี่แบบพื้นที่หลายเหลี่ยม

   1.6 สร้างกราฟจุดแสดงความหนาแน่น

   1.7 สร้างกราฟความถี่สะสม

   1.8 สร้างกราฟ Q-Q plot

2. สร้างกราฟแท่ง โดยใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ dplyr ชื่อว่า starwars ด้วยตัวแปร sex
3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า swiss สร้างกราฟต่าง ๆ ดังต่อไปนี้ พร้อมกำหนดค่าพารามิเตอร์ของกราฟเพิ่มเติมเพื่อปรับปรุงกราฟให้สวยงาม

   3.1 สร้างกราฟพื้นที่ ด้วยตัวแปร Fertility

   3.2 สร้างกราฟแท่ง ด้วยตัวแปร Agriculture

   3.3 สร้างกราฟความหนาแน่น ด้วยตัวแปร Examination

   3.4 สร้างกราฟฮีสโตแกรมพร้อมกราฟความหนาแน่นและสร้างเส้นค่าเฉลี่ย ด้วยตัวแปร Catholic

   3.5 สร้างกราฟความถี่แบบพื้นที่หลายเหลี่ยม ด้วยตัวแปร Examination

   3.6 สร้างกราฟจุดแสดงความหนาแน่น ด้วยตัวแปร Infant.Mortality

   3.7 สร้างกราฟความถี่สะสม ด้วยตัวแปร Fertility

   3.8 สร้างกราฟ Q-Q plot ด้วยตัวแปร Agriculture

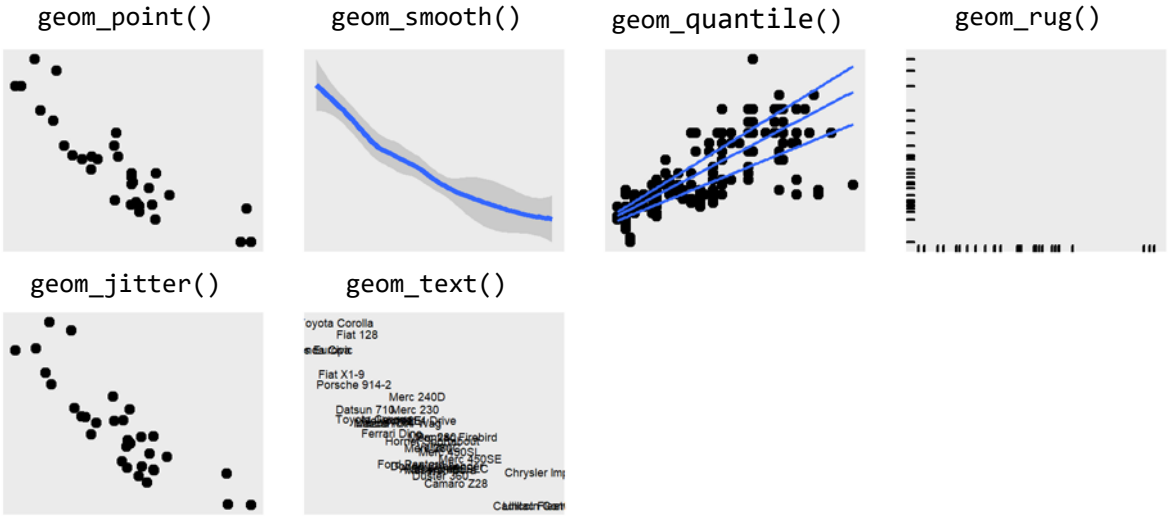
บทที่ 10
การพล็อตกราฟ 2 ตัวแปร (X และ Y): ตัวแปรต่อเนื่อง หรือตัวแปรแบ่งกลุ่ม

ในบทนี้จะกล่าวถึงการพล็อตกราฟที่มีตัวแปร 2 ตัวแปร ทั้งตัวแปรต่อเนื่อง (Continuous variable) หรือตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

สำหรับตัวแปรต่อเนื่อง 2 ตัวแปร ได้แก่

- `geom_point()` สำหรับพล็อต Scatter plots
- `geom_smooth()` สำหรับพล็อตเส้น Smooth เช่น Regression line
- `geom_quantile()` สำหรับพล็อตเส้น Quantile
- `geom_rug()` สำหรับพล็อต Marginal rug
- `geom_jitter()` สำหรับพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกัน
- `geom_text()` สำหรับการพล็อตข้อความ

ตัวอย่างการพล็อตดังกล่าว แสดงดังรูปต่อไปนี้



รูปที่ 10-1 Two continuous or discrete variable plots

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล mtcars ที่ติดตั้งมาพร้อมกับ base R ดังนี้

```
R> b <- ggplot(mtcars, aes(x = wt, y = mpg))
```

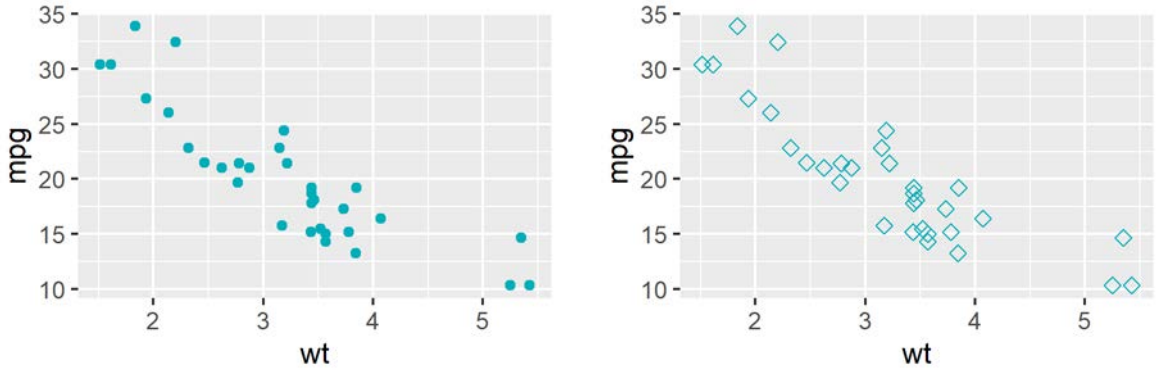
10.1 Scatter Plots

Scatter plots ใช้แสดงความสัมพันธ์ระหว่างตัวแปรต่อเนื่อง (Continunous variables) สองตัว ข้อมูลแต่ละตัวแทนด้วยจุด (Point) อาจจะใช้เส้นเพื่อแสดงแนวโน้มข้อมูลจากแบบจำลองทางสถิติ (Statistical models) ประกอบด้วยก็ได้

การพล็อตจุดกระจายตามแกน x และแกน y ใช้คำสั่ง **geom_point()** โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, fill, shape** และ **size** ตัวอย่าง Scatter plots แสดงได้ดังนี้

```
# Basic scatter plots
R> b + geom_point(color = "#00AFBB")

# Change the point size, and shape
R> b + geom_point(color = "#00AFBB", size = 2, shape = 23)
```

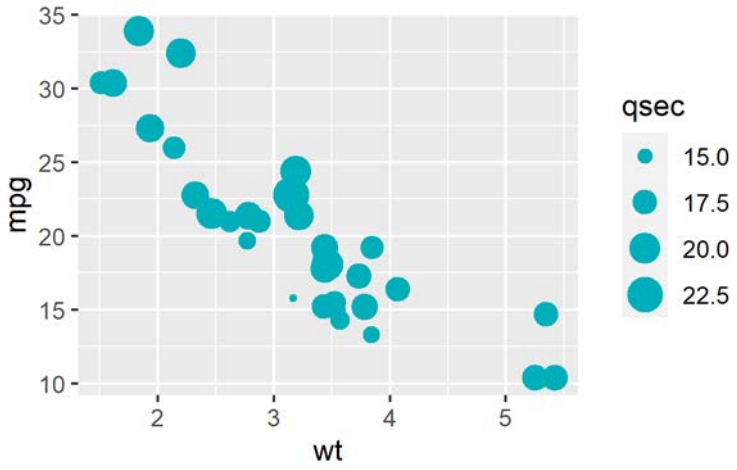


รูปที่ 10-2 Scatter plots

รูปด้านบนทั้ง 2 รูปแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) รูปแรกแสดงข้อมูลโดยใช้จุดวงกลมทึบสีฟ้า รูปที่สองใช้เครื่องหมาย (◇) กรอบสีฟ้าแทน

แสดงขนาดของจุดตามขนาดของตัวแปรต่อเนื่อง โดยใช้คำสั่ง **size** ได้ดังนี้

```
# Control point size by continuous variable values
R> b + geom_point(aes(size = qsec), color = "#00AFBB")
```



รูปที่ 10-3 Scatter plots by point size control

รูปด้านบนแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) โดยใช้จุดทึบสีฟ้าแทนข้อมูลแต่ละตัว และใช้จุดที่มีขนาดแตกต่างกันแสดงเวลาที่ใช้เร่งความเร็วภายในระยะทาง ¼ ไมล์ (qsec) โดยจุดที่มีขนาดใหญ่ขึ้นหมายถึงใช้เวลามากขึ้น ตามลำดับ

การใช้ Scatter plots เพื่อแสดงข้อมูลแต่ละกลุ่ม โดยแบ่งตามสี และรูปร่าง ได้ดังนี้

```
# Change point shapes by the levels of cyl
R> b + geom_point(aes(shape = cyl))

# Change point shapes and colors
R> b + geom_point(aes(shape = cyl, color = cyl))
```

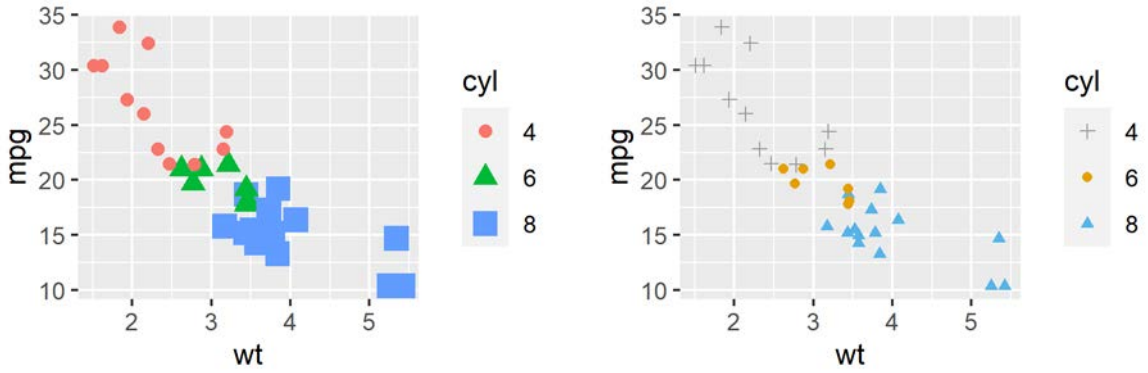
รูปที่ 10-4 Scatter plots with multiple groups

รูปด้านบนทั้ง 2 รูปแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) และใช้รูปร่างของจุดที่แตกต่างกันแสดงจำนวนกระบอกสูบ ถ้า cyl เท่ากับ 4 จะแสดงเป็นจุดวงกลม ถ้า cyl เท่ากับ 6 จะแสดงเป็นจุดสามเหลี่ยม และถ้า cyl เท่ากับ 8 จะแสดงเป็นจุดสี่เหลี่ยม รูปขวามือเพิ่มสีเพื่อแสดงความแตกต่างดังกล่าว ถ้า cyl เท่ากับ 4 จะแสดงเป็นด้วยสีชมพู ถ้า cyl เท่ากับ 6 จะแสดงเป็นสีเขียว และถ้า cyl เท่ากับ 8 จะแสดงเป็นสีฟ้า

การแสดงข้อมูลแต่ละกลุ่มตามสี รูปร่าง หรือขนาด โดยการปรับด้วยผู้ใช้เอง ทำได้โดยใช้คำสั่ง `scale_shape_manual()` สำหรับการปรับรูปร่าง `scale_color_manual()` สำหรับการปรับสี และ `scale_size_manual()` สำหรับการปรับขนาด ได้ดังนี้

```
# Change the point sizes manually
R> b + geom_point(aes(color = cyl, shape = cyl, size = cyl)) +
     scale_size_manual(values = c(2, 3, 4))

# Change point shapes and colors manually
R> b + geom_point(aes(color = cyl, shape = cyl)) +
     scale_shape_manual(values = c(3, 16, 17)) +
     scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```



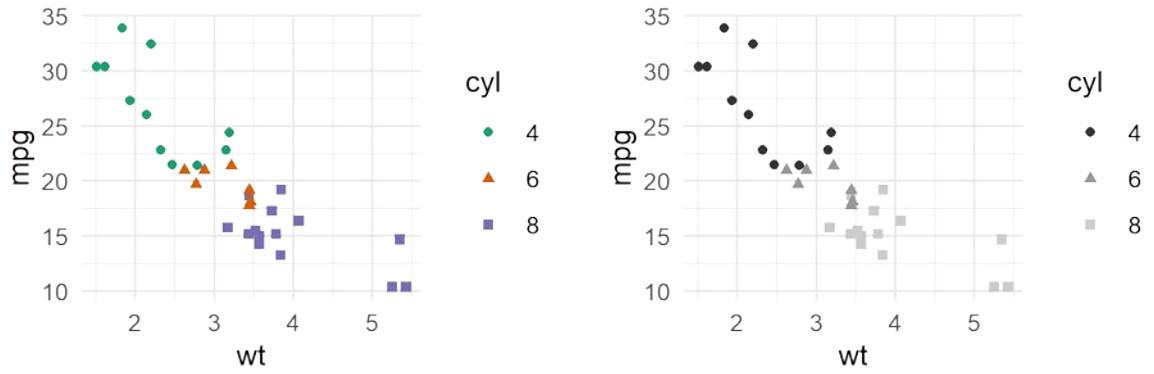
รูปที่ 10-5 Scatter plots with multiple groups by manual control

รูปด้านบนทั้ง 2 รูปแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) และใช้รูปร่างของจุดและสีที่แตกต่างกันแสดงจำนวนกระบอกสูบ รูปแรกเพิ่มขนาดที่แตกต่างกันแสดงจำนวนกระบอกสูบ ถ้าจำนวนกระบอกสูบเพิ่มขึ้นก็จะแสดงด้วยสัญลักษณ์ที่มีขนาดใหญ่ขึ้น

การแสดงสีแต่ละกลุ่มสามารถใช้สีจากจานสี (Palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` ได้ดังนี้

```
# use brewer color palettes
R> b + geom_point(aes(color = cyl, shape = cyl)) +
     scale_color_brewer(palette = "Dark2") +
     theme_minimal()

# Use grey scale
R> b + geom_point(aes(color = cyl, shape = cyl)) +
     scale_color_grey() +
     theme_minimal()
```



รูปที่ 10-6 Scatter plots with multiple groups by color control with pattetes

รูปด้านบนทั้ง 2 รูปแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) และใช้รูปร่างของจุดและสีที่แตกต่างกันแสดงจำนวนกระบอกสูบ เพิ่มเติมคือรูปขวามือใช้สีเทาที่ความเข้มแตกต่างกันแสดงจำนวนกระบอกสูบ

10.2 การพล็อตเส้น Smooth

เส้น Smooth คือ เส้นแสดงแนวโน้มข้อมูล การพล็อตเส้น Smooth ใช้คำสั่ง `geom_smooth()` หรือ `stat_smooth()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, shape, linetype,` และ `size` รูปแบบการพล็อตอย่างง่าย ประกอบด้วย

```
geom_smooth(method = "auto", se = TRUE, level = 0.95)
```

- `method` คือ วิธีการที่ใช้ในการแสดงเส้น Smooth ค่าที่ใช้ ได้แก่ `lm, glm, gam, loess, rlm` ค่าโดยปริยายคือ `method = loess` หมายถึง ใช้เทคนิคการสร้างแบบจำลองโดยใช้วิธีการ locally weighted polynomial ส่วน `method = lm` เป็นการพล็อตแบบจำลองเชิงเส้น (Linear model)
- `se` เป็นค่าตรรกะ ถ้า `se = TRUE` จะแสดงช่วงความเชื่อมั่น (Confidence interval) โดยรอบเส้น Smooth
- `level` คือ ระดับความเชื่อมั่น (Level of confidence) ค่าโดยปริยายเท่ากับ 0.95

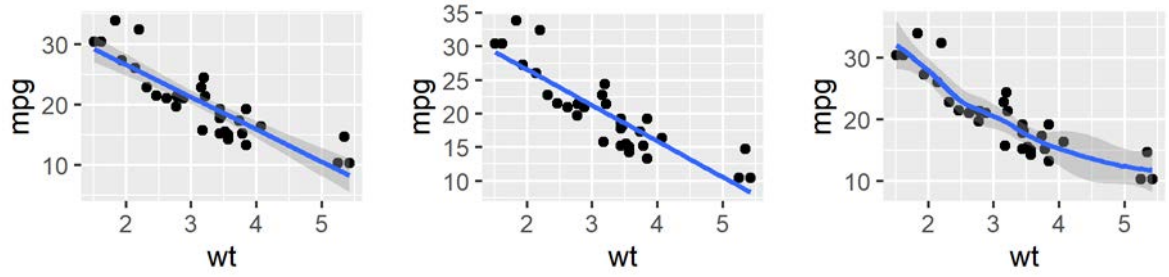
การแสดงเส้น Smooth เพิ่มใน Scatter plots ทำได้ดังนี้

```
# Point + regression line
R> b + geom_point() +
     geom_smooth(method = lm)

# Remove the confidence interval
R> b + geom_point() +
     geom_smooth(method = lm, se = FALSE)

# loess method: local regression fitting
R> b + geom_point() +
```

```
geom_smooth()
```



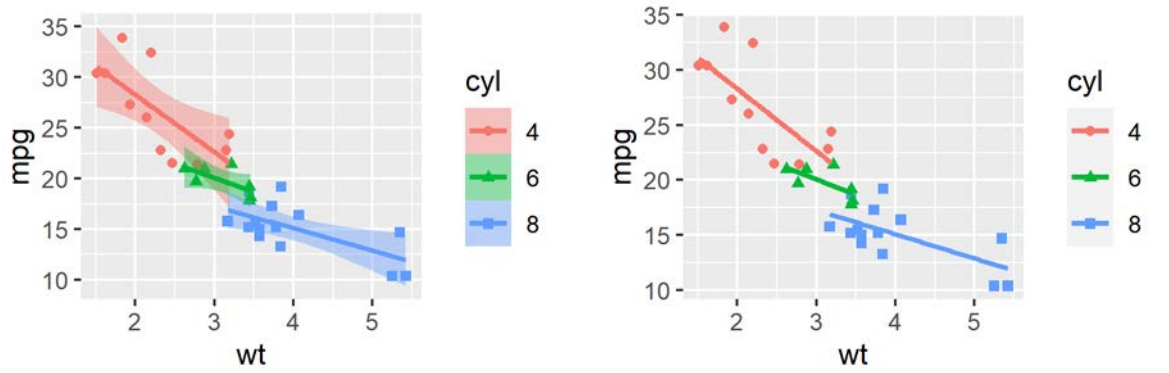
รูปที่ 10-7 Smooth lines

ทั้ง 3 รูปด้านบนแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) เพิ่มเติมคือ รูปแรกแสดงเส้น Smooth เป็นเส้นตรง และแถบสีเทาแสดงช่วงความเชื่อมั่น (Confidence interval) รอบเส้น Smooth รูปที่สองแสดงแต่เส้น Smooth อย่างเดียวไม่แสดงช่วงความเชื่อมั่น รูปสุดท้ายใช้เส้นโค้งแสดงเส้น Smooth ด้วย loess method พร้อมกับแถบสีเทาแสดงช่วงความเชื่อมั่น

การเปลี่ยนสีและรูปร่างตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีและรูปร่างในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` หรือ `shape` ดังนี้

```
# Change color and shape by groups (cyl)
R> b + geom_point(aes(color = cyl, shape = cyl)) +
     geom_smooth(aes(color = cyl, fill = cyl), method = lm)

# Remove confidence intervals
R> b + geom_point(aes(color = cyl, shape = cyl)) +
     geom_smooth(aes(color = cyl), method = lm, se = FALSE)
```



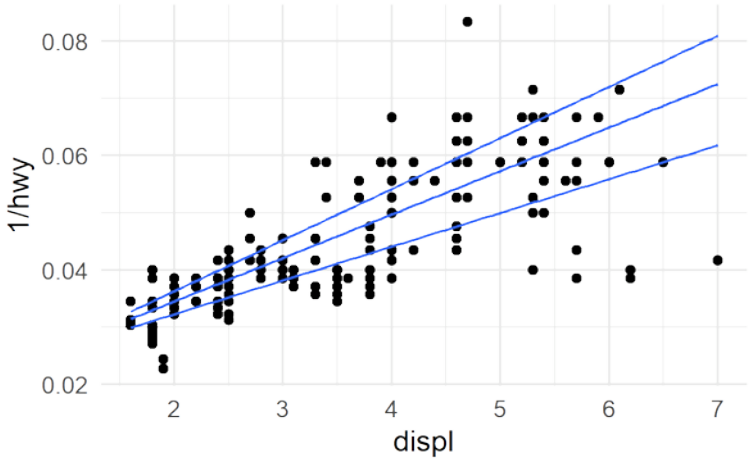
รูปที่ 10-8 Smooth lines with color and shape by groups

ทั้ง 2 รูปด้านบนแสดง Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) โดยใช้สีและรูปร่างของจุดแสดงความแตกต่างของจำนวนกระบอกสูบ ถ้า cyl เท่ากับ 4 จะแสดงเป็นจุดวงกลมสีชมพู ถ้า cyl เท่ากับ 6 จะแสดงเป็นจุดสามเหลี่ยมสีเขียว และถ้า cyl เท่ากับ 8 จะแสดงเป็นจุดสี่เหลี่ยมสีฟ้า และแสดงเส้น Smooth เป็นเส้นตรงโดยใช้สีเดียวกับจุดที่แสดง ทั้ง 2 รูปแตกต่างกันที่รูปแรกมีแถบสีแสดง

ช่วงความเชื่อมั่น (Confidence interval) รอบเส้น Smooth ส่วนรูปที่สองแสดงแต่เส้น Smooth อย่างเดียว ไม่แสดงช่วงความเชื่อมั่น

การเพิ่มเส้น Quantile จากสมการ Quantile regression ทำได้ด้วยคำสั่ง **geom_quantile()** ดังนี้

```
# Quantile regression
R> ggplot(mpg, aes(displ, 1 / hwy)) +
     geom_point() +
     geom_quantile() +
     theme_minimal()
```



รูปที่ 10-9 Quantile regression plots

รูปด้านบนแสดง Quantile regression plots ระหว่างความจุกระบอกสูบ (displ) และส่วนกลับของระยะทางบนทางหลวงต่อน้ำมันเชื้อเพลิง (1/hwy) โดยใช้เส้นตรงแสดงความสัมพันธ์ดังกล่าว แต่ละเส้นแยกตาม Quantile ตามลำดับ

10.3 การพล็อต Marginal Rug

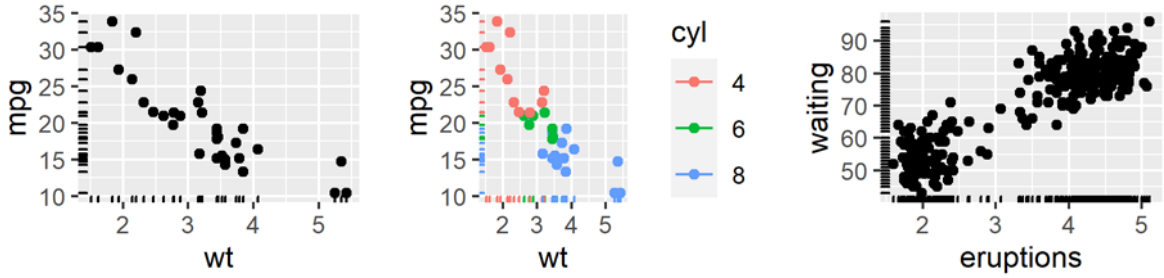
Marginal rug เป็นการแสดงการกระจายตัว (Distribution) ของข้อมูล 2 มิติโดยแสดงอยู่ในรูป 1 มิติ (1d marginal distribution) ในแกน X และ Y การพล็อต Marginal rug ใช้คำสั่ง **geom_rug()** โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color** และ **size** ดังนี้

```
# Add marginal rugs
R> b + geom_point() +
     geom_rug()

# Change colors by groups
R> b + geom_point(aes(color = cyl)) +
     geom_rug(aes(color = cyl))

# Add marginal rugs using faithful data
R> data(faithful)
```

```
R> ggplot(faithful, aes(x = eruptions, y = waiting)) +
      geom_point() +
      geom_rug()
```



รูปที่ 10-10 Marginal rug plots

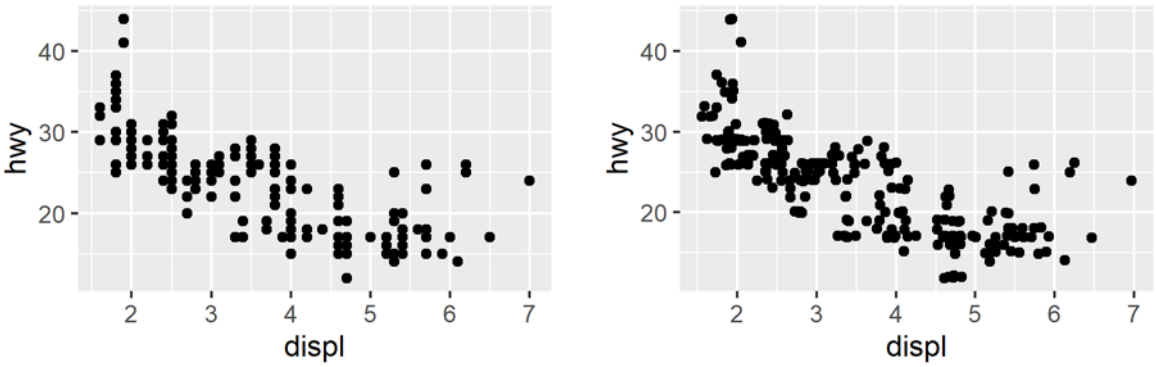
ทั้ง 3 รูปด้านบน แสดง Marginal rug plots ซึ่งแสดงการกระจายตัว (Distribution) ของข้อมูล 2 มิติให้แสดงอยู่ในแกน X และ Y แทน รูปที่หนึ่งและสองแกน X แสดงการกระจายตัวของน้ำหนัก (wt) และแกน Y แสดงการกระจายตัวของระยะทางต่อน้ำมันเชื้อเพลิง (mpg) รูปสุดท้ายแกน X แสดงการกระจายตัวของ eruptions และแกน Y แสดงการกระจายตัวของ waiting

10.4 การพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter Plots)

สำหรับการพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกันให้เลื่อนออกมาทั้งแนวตั้งและแนวนอน เพื่อให้ผู้อ่านมองเห็นจุดดังกล่าวได้ ทำได้โดยใช้คำสั่ง `geom_jitter()` และ `position_jitter()` มีพารามิเตอร์ที่สามารถกำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, shape` และ `size` และกำหนดตำแหน่งจุดที่ซ้อนทับให้เลื่อนออกมาทั้งแนวตั้งและแนวนอนด้วยคำสั่ง `position_jitter()` โดยระบุพารามิเตอร์ `width` และ `height` ตัวอย่างการพล็อต แสดงได้ดังนี้

```
R> p <- ggplot(mpg, aes(displ, hwy))
# Default scatter plots
R> p + geom_point()

# Use jitter to reduce overplotting
R> p + geom_jitter(position = position_jitter(width = 0.15, height = 0.15))
```



รูปที่ 10-11 Jitter plots

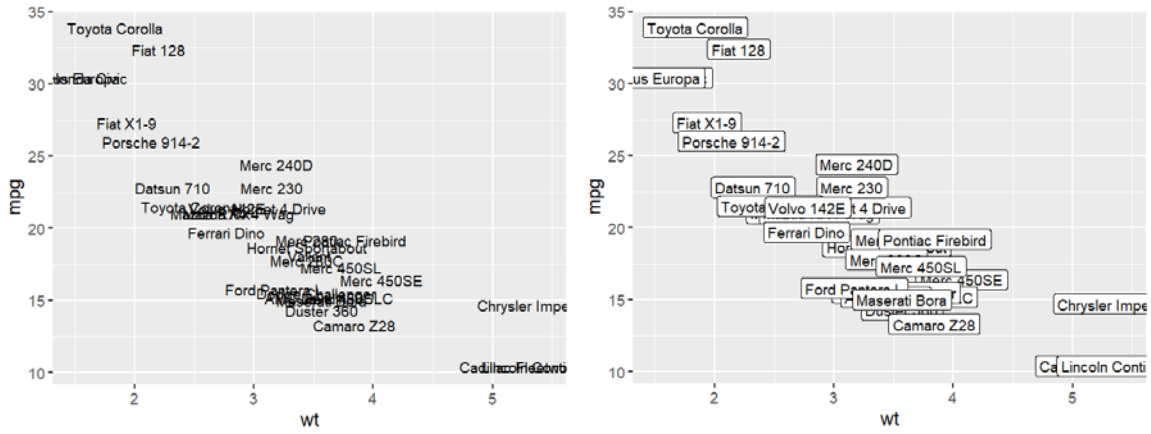
รูปด้านบนแสดง Scatter plots ระหว่างความจุกระบอกสูบ (displ) และระยะทางบนทางหลวงต่อน้ำมันเชื้อเพลิง (hwy) รูปแรกเป็นการพล็อตโดยใช้จุดแสดงตำแหน่งข้อมูลตามแกน X และแกน Y ส่วนรูปที่สองจะพล็อตโดยใช้ `geom_jitter()` ซึ่งจะทำการย้ายจุดที่ซ้อนทับกันให้เลื่อนออกมาทั้งแนวตั้งและแนวนอน ทำให้มองเห็นจุดที่เคยซ้อนทับกันได้

10.5 การพล็อตข้อความ (Textual Annotations) และป้ายหรือฉลาก (Text Label)

การพล็อตข้อความ (Textual annotations) ใช้คำสั่ง `geom_text()` ส่วนการพล็อตป้ายหรือฉลากใช้คำสั่ง `geom_label()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `label, alpha, angle, color, family, fontface, hjust, lineheight, size` และ `vjust` พารามิเตอร์ `label` เป็นการกำหนดตัวอักษรหรือข้อความที่จะแสดงผล ตัวอย่างเช่น

```
# Text annotation
R> b + geom_text(aes(label = rownames(mtcars)), size = 3)

# Text label
R> b + geom_label(aes(label = rownames(mtcars)), size = 3)
```



รูปที่ 10-12 Text plot vs Text Label plot

รูปด้านบนรูปแรกแสดงข้อความชื่อข้อมูลแต่ละจุด คล้ายกับ Scatter plots ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันเชื้อเพลิง (mpg) แต่ใช้ข้อความแสดงแทน ส่วนรูปที่สองเหมือนกับรูปแรกแต่ต่างกันที่รูปแรกมีเฉพาะข้อความส่วนรูปที่สองมีทั้งข้อความและกล่องข้อความแสดงความเป็นป้ายหรือฉลาก

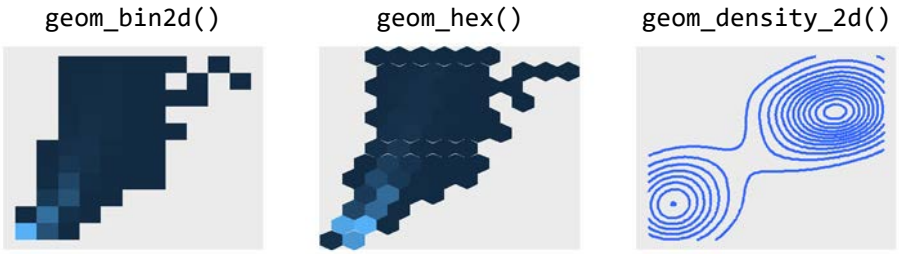
10.6 กราฟการกระจายสำหรับตัวแปรต่อเนื่อง 2 ตัวแปร (Continuous Bivariate Distribution Plots)

ในหัวข้อนี้จะกล่าวถึงการพล็อตกราฟการกระจายสำหรับตัวแปรต่อเนื่อง 2 ตัวแปร (Continuous bivariate distribution plots) Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

- `geom_bin_2d()` สำหรับพล็อตกราฟ Heatmap

- `geom_hex()` สำหรับพล็อตกราฟ Hexagon bining
- `geom_density_2d()` สำหรับพล็อตกราฟเส้น Contour

ตัวอย่างการพล็อตแสดงดังรูปต่อไปนี้



รูปที่ 10-13 Continuous Bivariate Distribution Plots

ข้อมูลที่ใช้คือ ข้อมูลชื่อ diamonds ที่ติดตั้งมาพร้อมกับแพ็กเกจ `ggplot2`

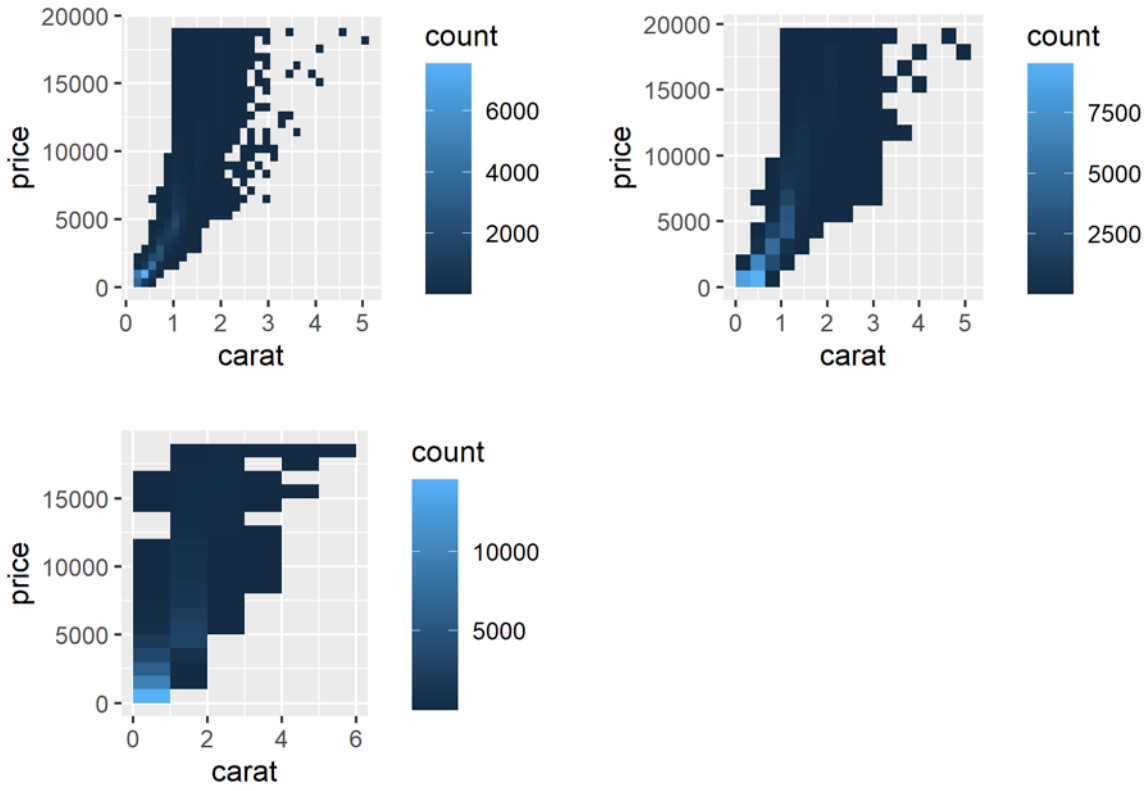
```
R> data(diamonds)
R> c <- ggplot(diamonds, aes(carat, price))
```

การสร้าง Heatmap จะทำการพล็อต Scatter ในลักษณะสี่เหลี่ยมจัตุรัสย่อย ๆ (Rectangular bins) โดยใช้คำสั่ง `geom_bin_2d()` `stat_bin_2d()` หรือ `stat_summary_2d()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `max, xmin, ymax, ymin, alpha, color, fill, linetype,` และ `size` ดังนี้

```
# Default plots
R> c + geom_bin_2d()

# Change the number of bins
R> c + geom_bin_2d(bins = 15)

# Or specify the width of bins
R> c + geom_bin_2d(binwidth = c(1, 1000))
```



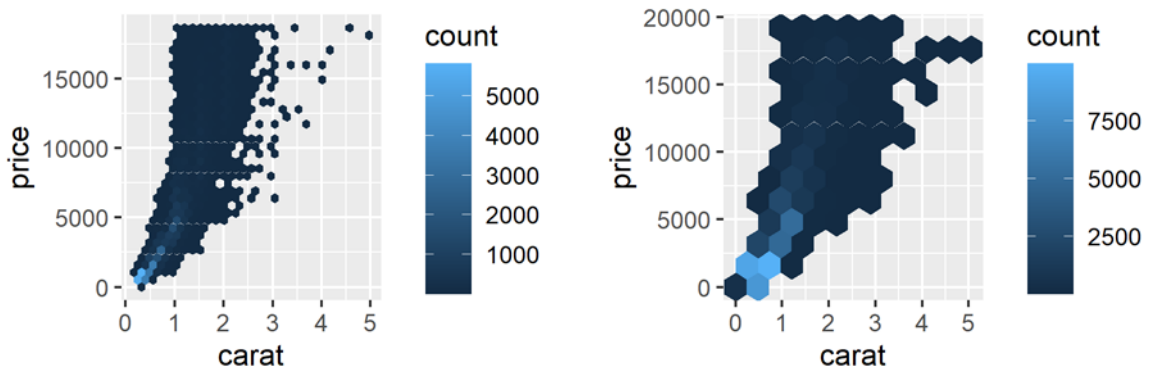
รูปที่ 10-14 Heatmaps by geom_bin_2d()

ทั้ง 3 รูปด้านบนเป็นการสร้าง Heatmap โดยทำการพล็อตคล้ายกับ Scatter แต่แสดงเป็นรูปสี่เหลี่ยมจัตุรัสย่อย ๆ (Rectangular bins) รูปซ้ายบนใช้จำนวน Bins เท่ากับ 30 ตามค่าโดยปริยาย รูปขวาบนใช้จำนวน Bins เท่ากับ 15 รูปซ้ายล่างแสดงโดยการกำหนดเวกเตอร์ความกว้างของ Bins เท่ากับ 1000 ด้วยคำสั่ง `c(1, 1000)`

คำสั่ง `geom_hex()` การสร้าง Heatmap ในลักษณะหกเหลี่ยมย่อย ๆ (Hexagon) ดังนี้

```
# install.packages("hexbin")
R> require(hexbin)
# Default plots
R> c + geom_hex()

# Change the number of bins
R> c + geom_hex(bins = 10)
```



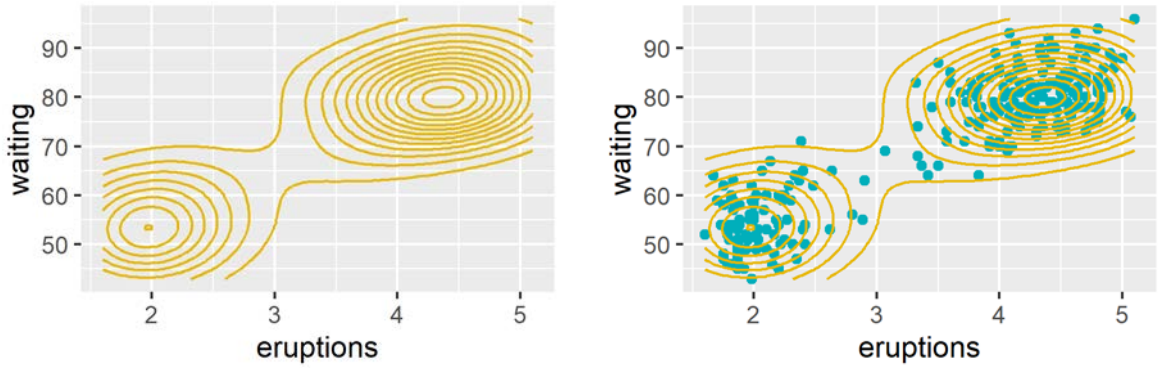
รูปที่ 10-15 Heatmaps by geom_hex()

ทั้ง 2 รูปด้านบนเป็นการสร้าง Heatmap โดยทำการพล็อตคล้ายกับ Scatter แต่เป็นการแสดงเป็นลักษณะหกเหลี่ยมย่อย ๆ รูปแรกใช้จำนวน Bins เท่ากับ 30 ตามค่าโดยปริยาย รูปที่สองใช้จำนวน Bins เท่ากับ 10

อีกวิธีหนึ่งในการพล็อตกราฟการกระจายสำหรับตัวแปรต่อเนื่อง 2 ตัวแปร คือการสร้างเส้น Contour โดยใช้คำสั่ง `geom_density_2d()` และ `stat_density_2d()` ดังนี้

```
R> data("faithful")
# Scatter plots
R> sp <- ggplot(faithful, aes(x = eruptions, y = waiting))
# Default plots
R> sp + geom_density_2d(color = "#E7B800")

# Add points
R> sp + geom_point(color = "#00AFBB") +
     geom_density_2d(color = "#E7B800")
```



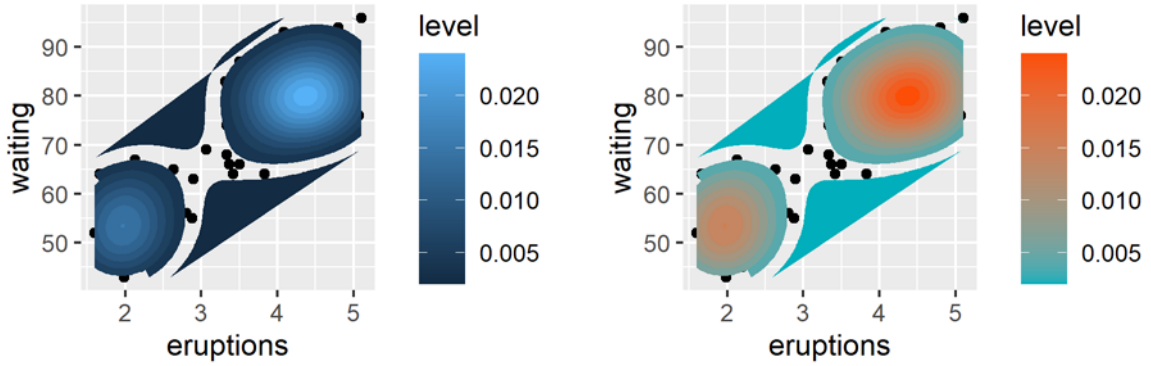
รูปที่ 10-16 Contour plots

รูปแรกเป็นการสร้างเส้น Contour ระหว่างแกน X คือ eruptions และแกน Y คือ waiting ส่วนรูปที่สองเป็นข้อมูลเดียวกันแต่แสดง Scatter plots พร้อมกับเส้น Contour

```
# Use stat_density_2d with geom = "polygon"
R> sp + geom_point() +
```

```
      stat_density_2d(aes(fill = ..level..), geom = "polygon")

# Change the gradient color
R> sp + geom_point() +
      stat_density_2d(aes(fill = ..level..), geom = "polygon") +
      scale_fill_gradient(low =  "#00AFBB", high= "#FC4E07")
```



รูปที่ 10-17 Contour plots with gradient fill

ทั้ง 2 รูปด้านบนเป็นการสร้างเส้น Contours ระหว่างแกน X คือ eruptions และแกน Y คือ waiting พร้อมกับ Scatter plots โดยรูปแรกแสดงสีของเส้น Contour โดยใช้สีน้ำเงินและค่อย ๆ จางลงเมื่อ level สูงขึ้น รูปที่สองแสดงสีของเส้น Contour โดยใช้สีฟ้าเมื่อ level ต่ำ ๆ และเปลี่ยนเป็นสีแดงเมื่อ level สูงขึ้นตามลำดับ

10.7 การพล็อตกราฟแสดงตัวแปรต่อเนื่อง

การพล็อตกราฟแสดงตัวแปรต่อเนื่องด้วยเส้นประเภทต่าง ๆ สามารถพล็อตได้ด้วยคำสั่งดังนี้

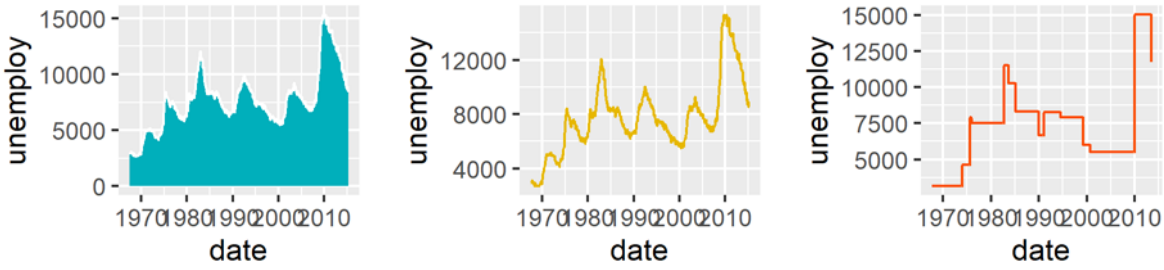
- `geom_area()` สำหรับพล็อตกราฟพื้นที่ (Area plots)
- `geom_line()` สำหรับพล็อตกราฟเส้น (Line plots)
- `geom_step()` สำหรับพล็อตกราฟขั้นบันได (Step plots)

```
data(economics)
R> d <- ggplot(economics, aes(x = date, y = unemploy))

# Area plots
R> d + geom_area(fill = "#00AFBB", color = "white")

# Line plots: connecting observations, ordered by x
R> d + geom_line(color = "#E7B800")

# Connecting observations by stairs
# a subset of economics data set is used
R> set.seed(1234)
R> ss <- economics[sample(1:nrow(economics), 15), ]
R> ggplot(ss, aes(x = date, y = unemploy)) +
      geom_step(color = "#FC4E07")
```



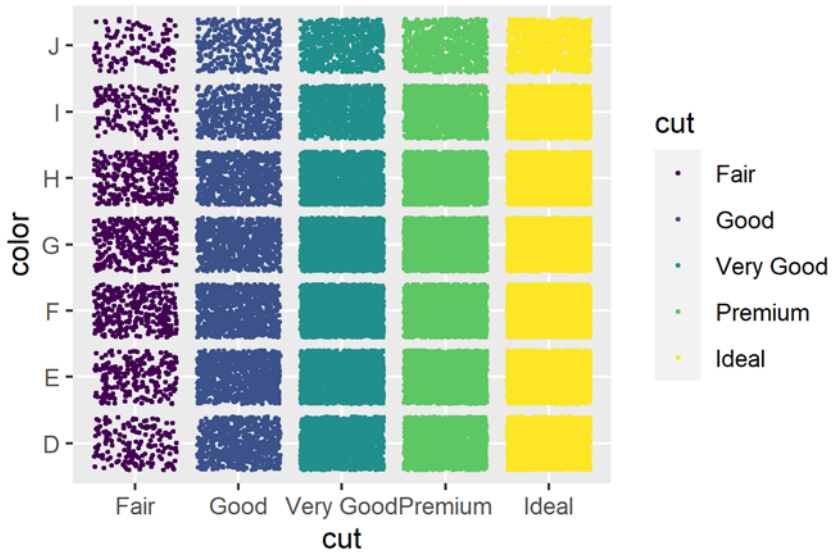
รูปที่ 10-18 Continuous variable plots with area, line, step plots

ทั้ง 3 รูปด้านบนแสดงจำนวนคนว่างงานตามช่วงเวลาต่าง ๆ รูปแรกเป็นการพล็อตด้วยกราฟพื้นที่ (Area plots) และแสดงสีของพื้นที่ใต้กราฟ รูปที่สองเป็นการพล็อตด้วยกราฟเส้น (Line plots) รูปสุดท้ายเป็นการพล็อตด้วยกราฟขั้นบันได (Step plots)

10.8 กราฟตัวแปรแบ่งกลุ่ม 2 ตัวแปร (Two Variables: Discrete X, Discrete Y)

การพล็อตสำหรับตัวแปรแบ่งกลุ่ม/ไม่ต่อเนื่องทั้ง 2 ตัวแปร (Two Variables: Discrete X, Discrete Y) ใช้คำสั่ง `geom_jitter()` และ `position_jitter()` มีพารามิเตอร์ที่สามารถกำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, shape` และ `size` และกำหนดตำแหน่งจุดที่ซ้อนทับให้เลื่อนออกมาทั้งแนวตั้งและแนวนอนด้วยคำสั่ง `position_jitter()` โดยระบุพารามิเตอร์ `width` และ `height` ตัวอย่างการพล็อต แสดงได้ดังนี้

```
R> data("diamonds")
R> ggplot(diamonds, aes(cut, color)) +
     geom_jitter(aes(color = cut), size = 0.5)
```



รูปที่ 10-19 Discrete X and discrete Y plots

รูปด้านบนแสดงการ cut ประเภทต่าง ๆ และ color แบบต่าง ๆ ของข้อมูล diamonds

10.9 สรุปท้ายบท

บทนี้กล่าวถึงการพล็อตกราฟที่มีตัวแปร 2 ตัวแปร ทั้งตัวแปรต่อเนื่อง (Continuous variable) หรือตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) กราฟที่ใช้ในการพล็อตดังกล่าวประกอบด้วย Scatter plots, เส้น Smooth, เส้น Quantile, Marginal rug, การพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter plots), การพล็อตข้อความและป้ายข้อความ, กราฟการกระจายสำหรับตัวแปรต่อเนื่อง 2 ตัวแปร (Continuous bivariate distribution plots), การพล็อตกราฟแสดงตัวแปรต่อเนื่อง และกราฟตัวแปรแบ่งกลุ่ม 2 ตัวแปร (Two variables: discrete X, discrete Y)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`ggplot()`	คำสั่งในการพล็อต
`geom_point()`	Geometry สำหรับพล็อต Scatter plots
`geom_smooth()`	Geometry สำหรับพล็อตเส้น Smooth
`geom_quantile()`	Geometry สำหรับพล็อตเส้น Quantile
`geom_rug()`	Geometry สำหรับพล็อต Marginal rug
`geom_jitter()`	Geometry สำหรับพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกัน
`geom_text()`	Geometry สำหรับการพล็อตข้อความ
`geom_label()`	Geometry สำหรับการพล็อตป้ายข้อความ
`geom_bin_2d()`	Geometry สำหรับพล็อตกราฟ Heatmap
`geom_hex()`	Geometry สำหรับพล็อตกราฟ Hexagon bining
`geom_density_2d()`	Geometry สำหรับพล็อตกราฟเส้น Contour
`geom_area()`	Geometry สำหรับพล็อตกราฟพื้นที่ (Area plots)
`geom_line()`	Geometry สำหรับพล็อตกราฟเส้น (Line plots)
`geom_step()`	Geometry สำหรับพล็อตกราฟขั้นบันได (Step plots)
`scale_color_manual()`	คำสั่งในการกำหนดสีแบบ Manual
`scale_color_brewer()`	คำสั่งในการกำหนดสีด้วยจานสี (Palettes)
`scale_color_grey()`	คำสั่งในการกำหนดสีด้วยจานสีเทา
`scale_fill_manual()`	คำสั่งในการเติมสีแบบ Manual
`scale_fill_brewer()`	คำสั่งในการเติมสีด้วยจานสี (Palettes)
`scale_fill_grey()`	คำสั่งในการเติมสีด้วยจานสีเทา
`scale_shape_manual()`	คำสั่งในการกำหนดรูปร่างแบบ Manual
`scale_size_manual()`	คำสั่งในการกำหนดขนาดแบบ Manual

10.10 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า swiss สร้างกราฟต่าง ๆ ดังต่อไปนี้

   1.1 สร้างกราฟจุด Scatter plot เพื่อดูความสัมพันธ์ระหว่างตัวแปร Fertility และ Agriculture

   1.2 จากข้อ 1.1 พล็อตเส้น Smooth เพิ่มเติมในลักษณะแบบจำลองเชิงเส้น

   1.3 จากข้อ 1.2 พล็อต Marginal Rug เพิ่มเติม

   1.4 สร้างกราฟที่ต้องการแสดงจุดที่ซ้อนทับกันให้เลื่อนออกมาทั้งแนวตั้งและแนวนอน เพื่อดูความสัมพันธ์ระหว่างตัวแปร Examination และ Education

   1.5 จากข้อ 1.1 เปลี่ยนเป็นการพล็อตข้อความแทนจุด ด้วยชื่อจังหวัด ซึ่งแทนด้วยชื่อของแต่ละแถวในเดต้าเฟรม

   1.6 จากข้อ 1.5 เปลี่ยนจากข้อความเป็นพล็อตป้ายหรือฉลาก

2. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า pressure สร้างกราฟต่าง ๆ ดังต่อไปนี้

   2.1 สร้างกราฟจุด Scatter plot เพื่อดูความสัมพันธ์ระหว่างตัวแปร temperature และ pressure

   2.2 จากข้อ 2.1 พล็อตเส้น Smooth เพิ่มเติม

   2.3 จากข้อ 2.2 พล็อต Marginal Rug เพิ่มเติม

   2.4 จากข้อ 2.3 พล็อตกราฟเส้น Contour ให้เป็นเส้นสีแดง เพิ่มเติม

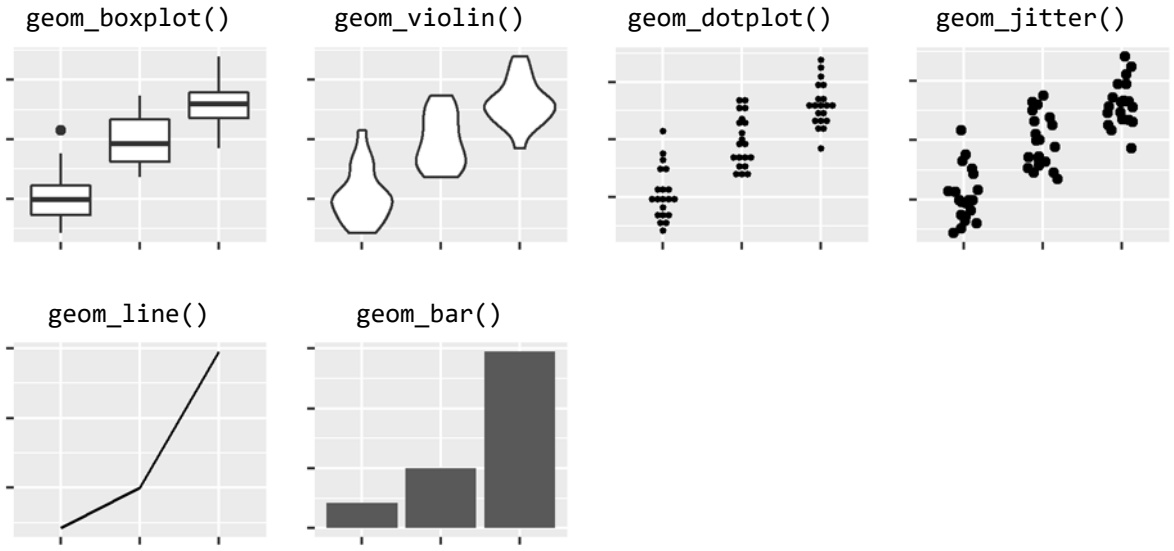
3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ dplyr ชื่อว่า starwars สร้างกราฟตัวแปรแบ่งกลุ่ม 2 ตัวแปรโดยใช้คำสั่ง **`geom_jitter()`** เพื่อแสดงความสัมพันธ์ระหว่างตัวแปร sex และตัวแปร hair_color โดยให้มีการแบ่งสีของจุดตามประเภทของ sex

บทที่ 11
การพล็อตกราฟ 2 ตัวแปร (X and Y): X ตัวแปรแบ่งกลุ่ม และ Y ตัวแปรต่อเนื่อง

ในบทนี้จะกล่าวถึงการพล็อตกราฟที่มีแกน X เป็นตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) และแกน Y เป็นตัวแปรต่อเนื่อง (Continuous variable) Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

- `geom_boxplot()` สำหรับพล็อตกราฟการกระจายตัวแบบ Box plots
- `geom_violin()` สำหรับพล็อตกราฟความหนาแน่นแบบ Violin plots
- `geom_dotplot()` สำหรับพล็อตกราฟจุดแสดงความหนาแน่น (Dot plots)
- `geom_jitter()` สำหรับพล็อตที่ต้องการแสดงจุดที่ซ้อนทับกัน
- `geom_line()` สำหรับพล็อตกราฟเส้น
- `geom_bar()` สำหรับพล็อตกราฟแท่ง

ตัวอย่างการพล็อตดังกล่าว แสดงดังรูปต่อไปนี้



รูปที่ 11-1 X discrete variable and Y continuous variable plots

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base R ดังนี้

```
R> data("ToothGrowth")
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> e <- ggplot(ToothGrowth, aes(x = dose, y = len))
```

11.1 กราฟการกระจายตัวแบบ Box Plots

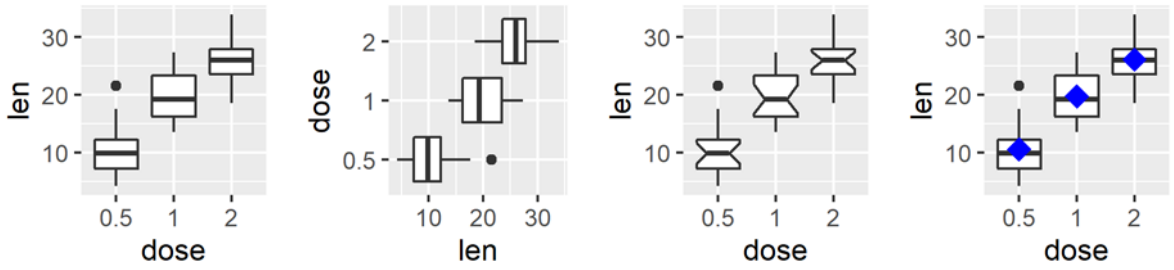
Box plots เป็นการพล็อตแสดงการกระจายตัวข้อมูลในแนวแกน Y ส่วนแกน X เป็นตัวแปรแบ่งกลุ่ม (สลับแกน X และ Y ได้) การพล็อตใช้คำสั่ง **geom_boxplot()** หรือ **stat_boxplot()** โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, linetype, fill, shape** และ **size** ตัวอย่าง Box plots แสดงได้ดังนี้

```
# Basic box plots
R> e + geom_boxplot()

# Rotate the box plots
R> e + geom_boxplot() +
     coord_flip()

# Notched box plots
R> e + geom_boxplot(notch = TRUE)

# Box plots with mean points
R> e + geom_boxplot() +
     stat_summary(fun = mean, geom = "point",
                  shape = 18, size = 4, color = "blue")
```



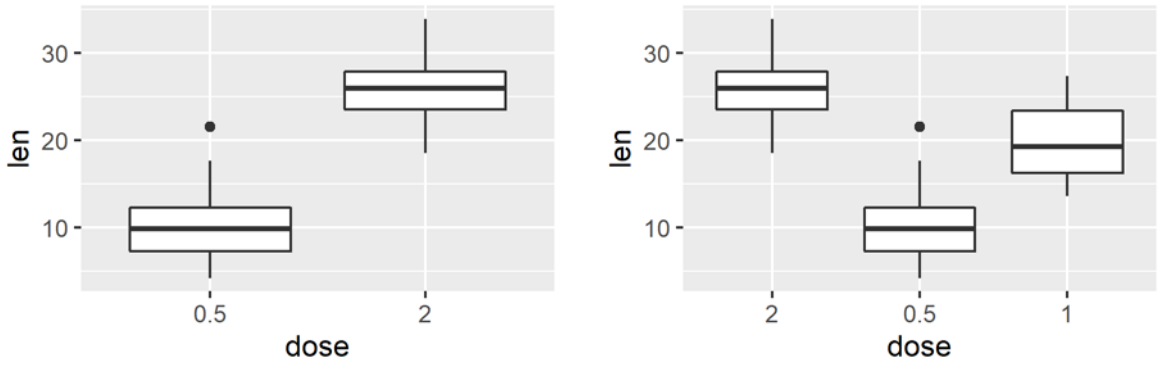
รูปที่ 11-2 Box plots

รูปแรกเป็นการพล็อตแสดงตัวแปร dose ในแกน X ส่วนแกน Y เป็นตัวแปร len รูปที่สองเป็นการพล็อตแบบสลับแกน X และแกน Y รูปถัดมาเป็นการแสดง Box plots แบบบาก (Notch) แทนการใช้ Box และรูปสุดท้ายเป็นการเพิ่มค่าเฉลี่ย (Mean) เข้ามาใน Box plots

สามารถแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน **scale_x_discrete()** โดยผ่านพารามิเตอร์ **limits** ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อตตามลำดับ ดังนี้

```
# Choose which items to display: group "0.5" and "2"
R> e + geom_boxplot() +
     scale_x_discrete(limits = c("0.5", "2"))

# Change the default order of items
R> e + geom_boxplot() +
     scale_x_discrete(limits = c("2", "0.5", "1"))
```



รูปที่ 11-3 Box plots with order by groups

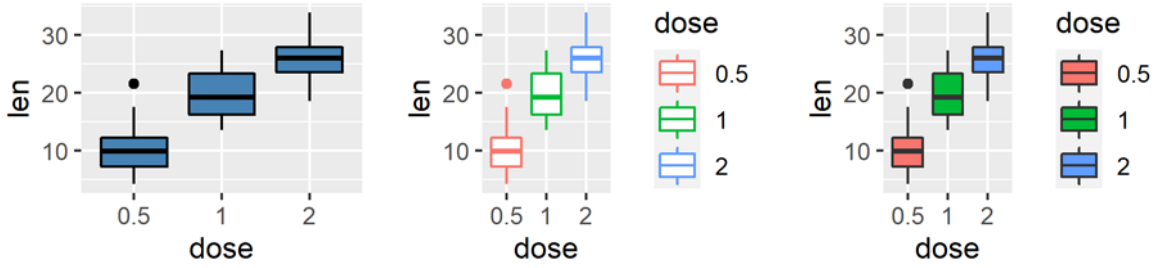
รูปแรกแสดง Box plots แกน X แสดงตัวแปร dose เท่ากับ 0.5 และ 2 ส่วนรูปที่สองแกน X แสดงตัวแปร dose เท่ากับ 2, 0.5 และ 1 ตามลำดับ

การเปลี่ยนสีแต่ละ Box ตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` หรือ `fill` ดังนี้

```
# Use single colors
R> e + geom_boxplot(color = "black", fill = "steelblue")

# Change outline colors by dose (groups)
R> e + geom_boxplot(aes(color = dose))

# Change fill color by dose (groups)
R> e + geom_boxplot(aes(fill = dose))
```



รูปที่ 11-4 Box plots with colors by groups

รูปแรกแสดง Box plots ด้วยสีน้ำเงินและเส้นกรอบสีดำ รูปที่สองแสดง Box plots ด้วยเส้นกรอบสีตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีชมพู, สีเขียว, และสีฟ้า ตามลำดับ รูปสุดท้ายเป็นข้อมูลเดียวกันแต่แสดงสี Box ดังกล่าวเหมือนกับรูปกลางยกเว้นกรอบสีดำ

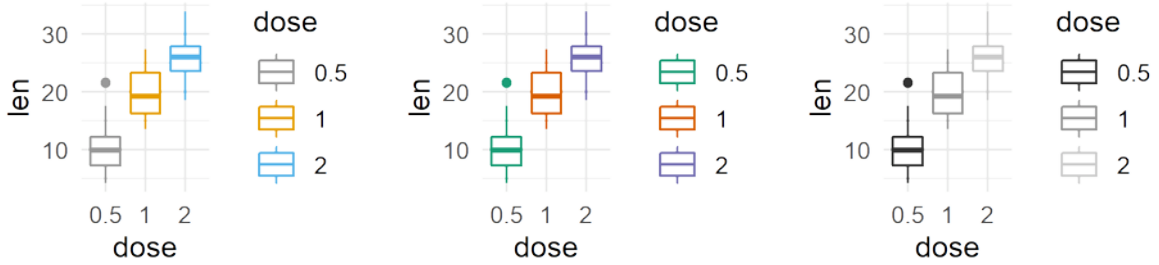
สามารถเปลี่ยนสีเส้นกรอบ Box ด้วยคำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` ดังนี้

```
# Use custom color palettes
R> e2 <- e + geom_boxplot(aes(color = dose)) + theme_minimal()
```

```
R> e2 + scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
R> e2 + scale_color_brewer(palette = "Dark2")

# Use grey scale
R> e2 + scale_color_grey()
```



รูปที่ 11-5 Box plots with manually color by groups

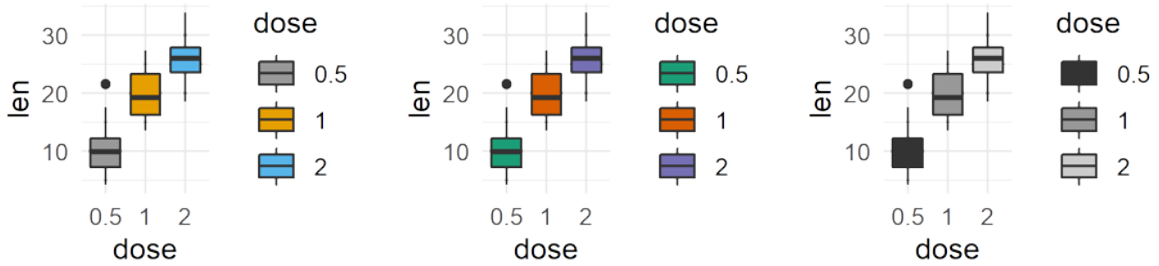
ทั้ง 3 รูปด้านบนเป็น Box plots ที่แสดงสีเส้นกรอบตามที่กำหนด รูปแรกเป็นการกำหนดสีแบบ manual รูปที่สองเป็นการใช้จานสี (Palette) รูปสุดท้ายเป็นการใช้โทนสีเทา

และเติมสี Box (Fill) ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_fill_grey()` ดังนี้

```
# Use custom color palettes
R> e3 <- e + geom_boxplot(aes(fill = dose)) +
  theme_minimal()
R> e3 + scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
R> e3 + scale_fill_brewer(palette = "Dark2")

# Use grey scale
R> e3 + scale_fill_grey()
```



รูปที่ 11-6 Box plots with manually fill colors by groups

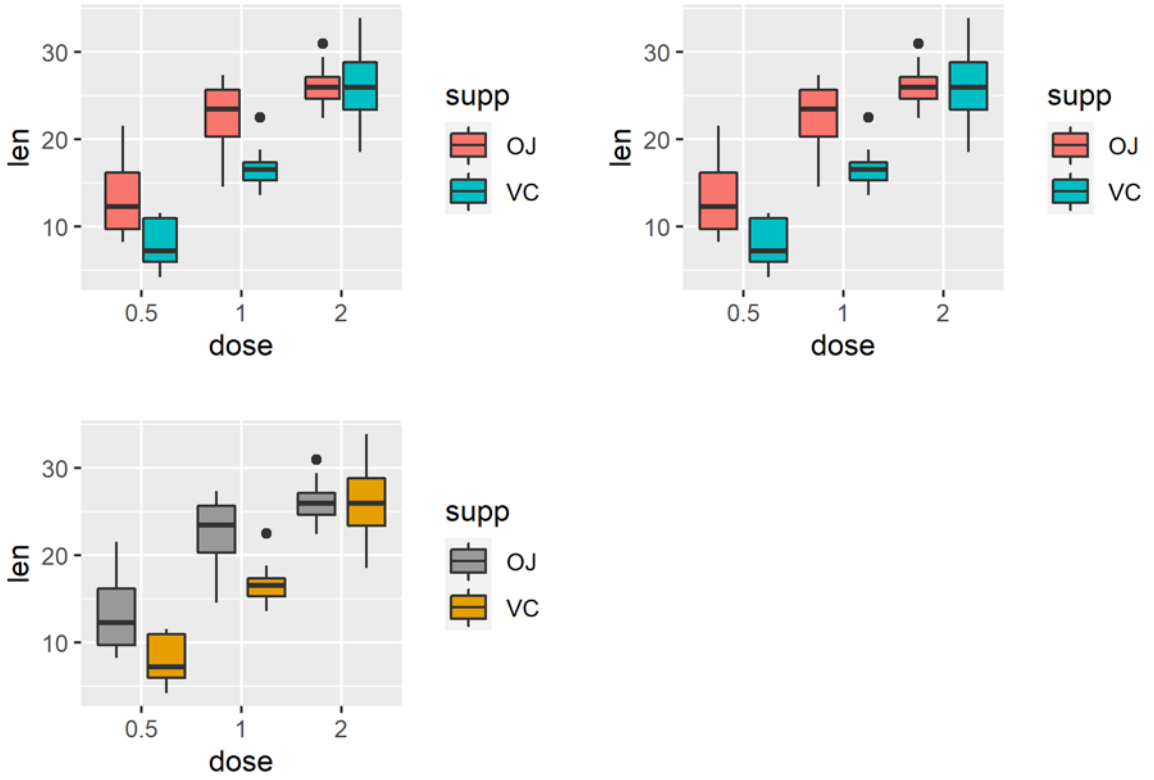
ทั้ง 3 รูปด้านบนเป็น Box plots ที่แสดงสี Box (Fill) ตามที่กำหนด รูปแรกเป็นการกำหนดสีแบบ manual รูปที่สองเป็นการใช้จานสี (Palette) รูปสุดท้ายเป็นการใช้โทนสีเทา

แสดง Box plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ได้ดังนี้

```
# Change box plots colors by groups
R> e + geom_boxplot(aes(fill = supp))

# Change the position
R> e + geom_boxplot(aes(fill = supp), position = position_dodge(1))

# Change fill colors
R> e + geom_boxplot(aes(fill = supp), position = position_dodge(1)) +
     scale_fill_manual(values = c("#999999", "#E69F00"))
```



รูปที่ 11-7 Box plots with multiple groups

ทั้ง 3 รูปด้านบนเป็น Box plots ที่แสดงสี Box (Fill) ด้วยตัวแปร supp และแกน X ด้วยตัวแปร dose รูปซ้ายบนเป็นการพล็อตด้วยค่าโดยปริยาย ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า รูปขวาบนเป็นข้อมูลเดียวกับรูปซ้ายบนแต่มีระยะด้านข้างระหว่าง Box ห่างกันเพิ่มขึ้น รูปล่างเป็นการพล็อตโดยการกำหนดสีแบบ Manual

11.2 กราฟความหนาแน่นแบบไวโอลิน (Violin Plots)

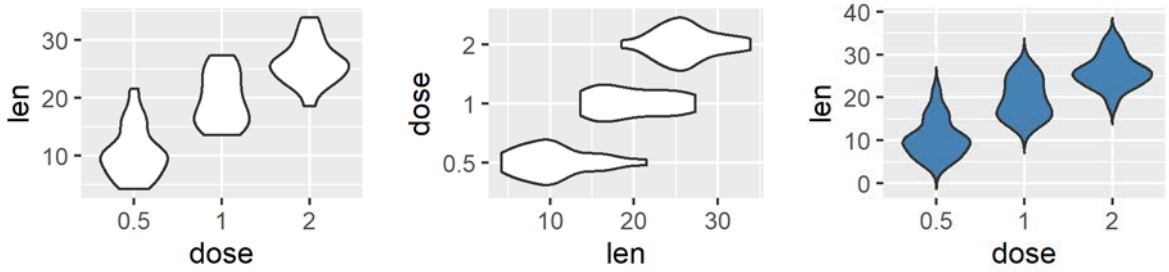
Violin plots คล้ายกับ Box plots ซึ่งแสดงการกระจายตัวของข้อมูล แต่ Violin plots แสดงเป็น Kernel probability density แทน การพล็อตใช้คำสั่ง `geom_violin()` หรือ `stat_ydensity()` โดยมี

พารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, linetype, fill** และ **size** ตัวอย่าง Violin plots แสดงได้ดังนี้

```
# Basic violin plots
R> e + geom_violin()

# Rotate the violin plots
R> e + geom_violin() +
     coord_flip()

# Set trim argument to FALSE
R> e + geom_violin(trim = FALSE, fill = "steelblue")
```



รูปที่ 11-8 Violin plots

รูปแรกเป็นการแสดงตัวแปร dose ในแกน X ส่วนแกน Y เป็นตัวแปร len รูปที่สองเป็นการพล็อตแบบสลับแกน X และแกน Y รูปสุดท้ายไม่มีการ Trim ค่าที่ส่วนหางของกราฟ

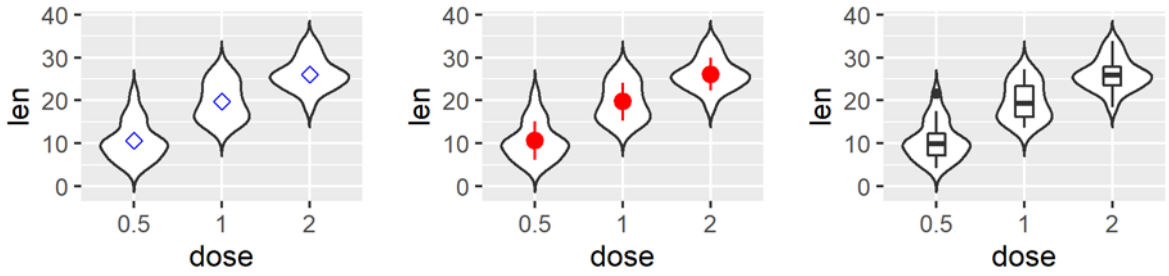
เพิ่มการแสดงสีด้วยพารามิเตอร์ **fill** และแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน **scale_x_discrete()** โดยผ่านพารามิเตอร์ **limits** ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ เช่นเดียวกับหัวข้อก่อนหน้านี้

ฟังก์ชัน **stat_summary()** สามารถแสดงค่าเฉลี่ยทางคณิตศาสตร์ เช่น Mean, Median และค่าอื่นได้ดังนี้

```
# Add mean or median point: use fun = mean or fun = median
R> e + geom_violin(trim = FALSE) +
     stat_summary(fun = mean, geom = "point",
                  shape = 23, size = 2, color = "blue")

# Add mean points +/- SD
# Use geom = "pointrange" or geom = "crossbar"
R> e + geom_violin(trim = FALSE) +
     stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                  geom = "pointrange", color = "red")

# Combine with box plots to add median and quartiles
R> e + geom_violin(trim = FALSE) +
     geom_boxplot(width = 0.2)
```



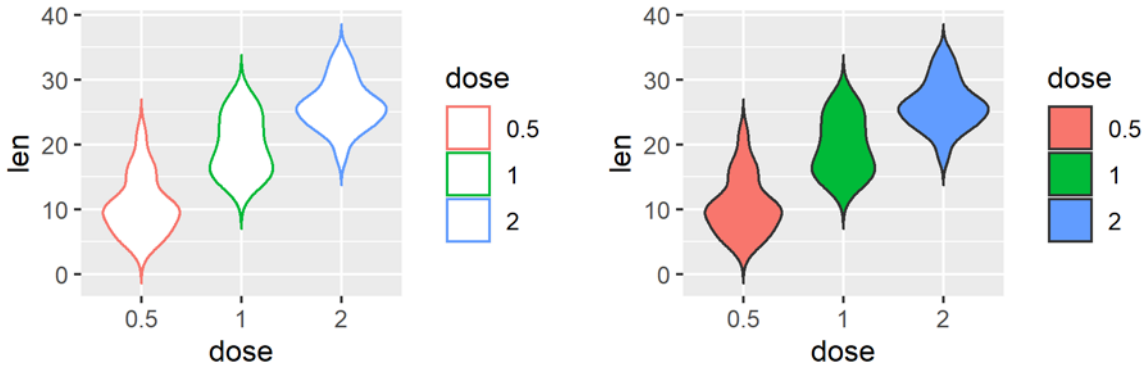
รูปที่ 11-9 Violin plots with statistics

รูปแรกเป็นการแสดงค่า mean พร้อมกับข้อมูล density ของตัวแปร len ในแกน Y ส่วนแกน X เป็นตัวแปร dose รูปที่สองเป็นการแสดงค่า mean และค่า sd รูปสุดท้ายเป็นการสร้าง Box plots ภายใน Violin plots

การเปลี่ยนสีแต่ละกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` หรือ `fill` ดังนี้

```
# Change outline colors by dose (groups)
R> e + geom_violin(aes(color = dose), trim = FALSE)

# Change fill color by dose (groups)
R> e + geom_violin(aes(fill = dose), trim = FALSE)
```



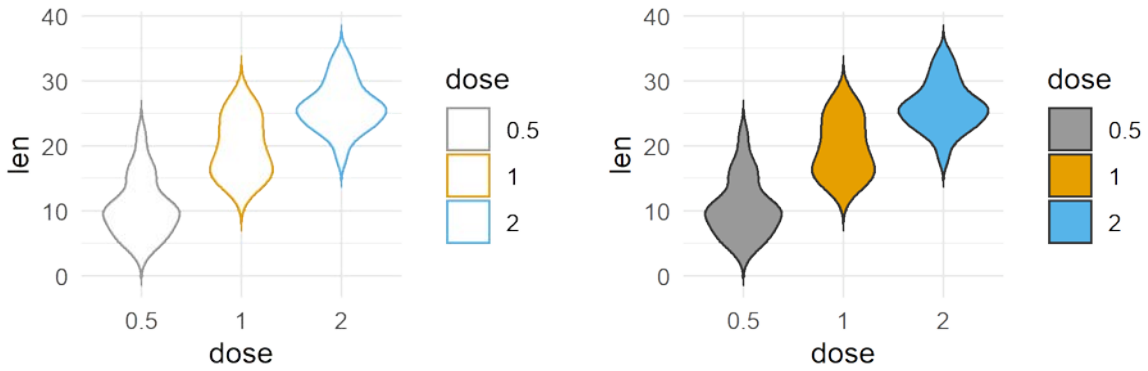
รูปที่ 11-10 Violin plots with color by groups

ทั้ง 2 รูปเป็น Violin plots แสดงสีตามข้อมูลตัวแปร dose รูปแรกแสดงเส้นกรอบสีตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีชมพู, สีเขียว, และสีฟ้า ตามลำดับ รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงสี Violin เหมือนกับรูปแรกยกเว้นกรอบสีดำ

สามารถเปลี่ยนสีเส้นกรอบด้วยคำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` หรือเติมสี (Fill) Violin ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_fill_grey()` ดังนี้

```
# Change manually outline colors
R> e2 <- e + geom_violin(aes(color = dose), trim = FALSE) +
     theme_minimal()
R> e2 + scale_color_manual(values=c("#999999", "#E69F00", "#56B4E9"))

# Change manually fill colors
R> e3 <- e + geom_violin(aes(fill = dose), trim = FALSE) +
     theme_minimal()
R> e3 + scale_fill_manual(values=c("#999999", "#E69F00", "#56B4E9"))
```



รูปที่ 11-11 Violin plots with manual colors by groups

ทั้ง 2 รูปเป็น Violin plots แสดงสีตามข้อมูลในตัวแปร dose ด้วยการกำหนดสีแบบ manual รูปแรกแสดงด้วยเส้นกรอบสีตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีเทา, สีเหลือง, และสีฟ้า ตามลำดับ รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงสี (Fill) ที่ตัว Violin แทน

แสดง Violin plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ได้ดังนี้

```
# Change colors by groups
R> e + geom_violin(aes(fill = supp), trim = FALSE)

# Change fill colors
R> e + geom_violin(aes(fill = supp), trim = FALSE) +
     scale_fill_manual(values = c("#999999", "#E69F00"))
```

รูปที่ 11-12 Violin plots with multiple groups

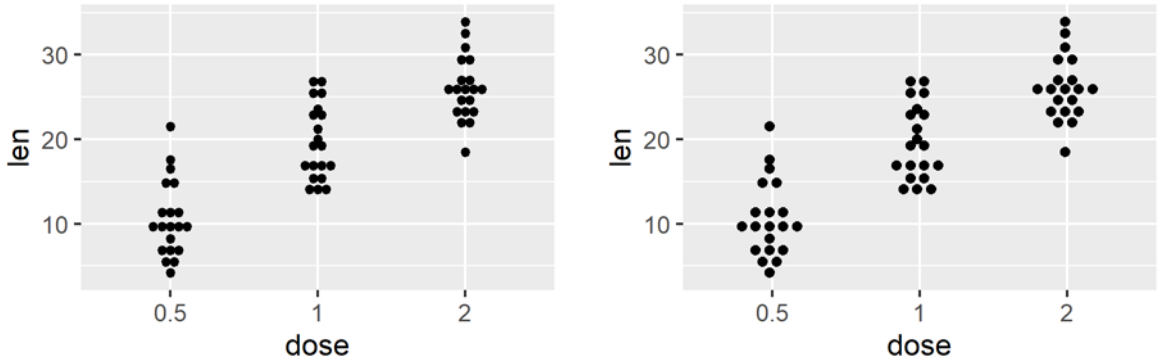
ทั้ง 2 รูปเป็น Violin plots ที่แสดงสี (Fill) ด้วยตัวแปร supp และแสดงแกน X ด้วยตัวแปร dose รูปแรกเป็นการพล็อตด้วยค่าโดยปริยาย ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า รูปที่สองเป็นข้อมูลเดียวกับรูปแรกแต่พล็อตโดยการกำหนดสีแบบ Manual

11.3 กราฟจุดแสดงความหนาแน่น (Dot Plots)

Dot plots เป็นการพล็อตจุดแสดงความหนาแน่น (Density) ของข้อมูลในแนวแกน Y ส่วนแกน X เป็นตัวแปรแบ่งกลุ่ม/ไม่ต่อเนื่อง การพล็อตใช้คำสั่ง **geom_dotplot()** หรือ **stat_summary()** โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, dotsize** และ **fill** ตัวอย่าง Dot plots แสดงได้ดังนี้

```
# Basic plots
R> e + geom_dotplot(binaxis = "y", stackdir = "center")

# Change dotsize and stack ratio
R> e + geom_dotplot(binaxis = "y", stackdir = "center",
                    stackratio = 1.5, dotsize = 1.1)
```



รูปที่ 11-13 Dot plots

ทั้ง 2 รูปเป็น Dot plots ที่แสดงความหนาแน่นของตัวแปร len แยกตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ตามลำดับ รูปแรกเป็นการพล็อตด้วยค่าโดยปริยาย ส่วนรูปที่สองเป็นการพล็อตด้วยการเพิ่มระยะห่างด้านข้างระหว่างแต่ละจุด และเพิ่มขนาดจุด

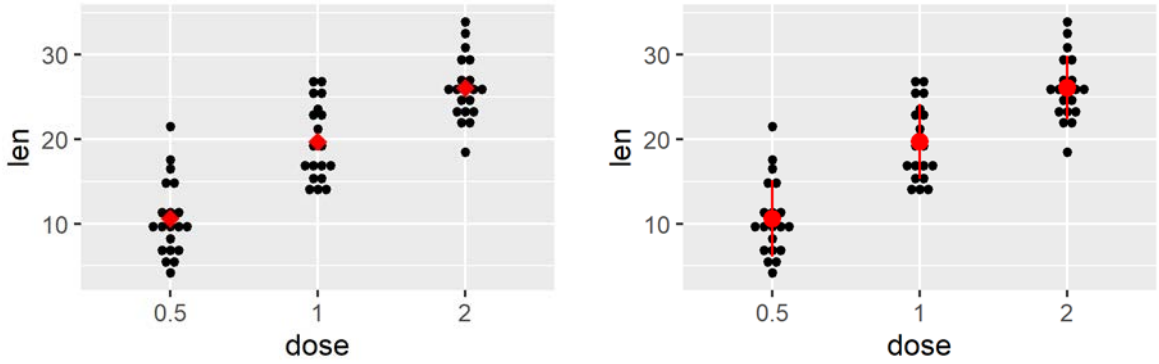
สามารถแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน **scale_x_discrete()** โดยผ่านพารามิเตอร์ **limits** ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ เช่นเดียวกับหัวข้อก่อนหน้านี้

ฟังก์ชัน **stat_summary()** สามารถแสดงค่าเฉลี่ยทางคณิตศาสตร์ เช่น Mean, Median และค่าอื่นได้ดังนี้

```
# Add mean or median point: use fun = mean or fun = median
R> e + geom_dotplot(binaxis = "y", stackdir = "center") +
     stat_summary(fun = mean, geom = "point",
```

```
                  shape = 18, size = 3, color = "red")

# Add mean points +/- SD
# Use geom = "pointrange" or geom = "crossbar"
R> e + geom_dotplot(binaxis = "y", stackdir = "center") +
     stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                  geom = "pointrange", color = "red")
```



รูปที่ 11-14 Dot plots with statistics

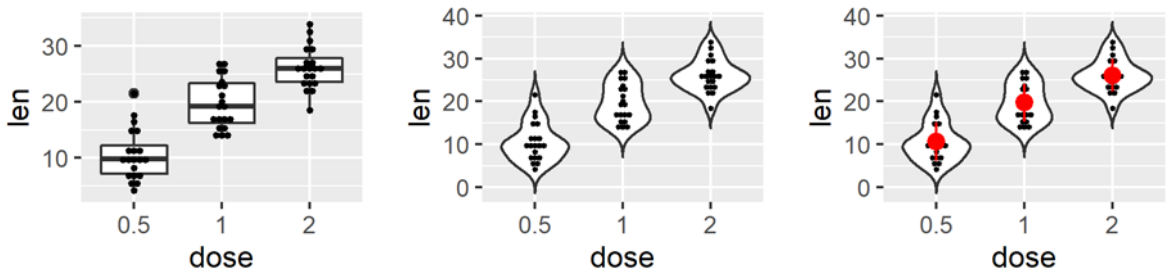
ทั้ง 2 รูปเป็น Dot plots ที่แสดงความหนาแน่นของตัวแปร len แยกตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ตามลำดับ รูปแรกแสดงค่าเฉลี่ยประกอบ ส่วนรูปที่สองแสดงทั้งค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐาน

สามารถแสดง Dot plots ร่วมกับ Box plots และ Violin plots รวมทั้งเพิ่มการแสดงค่าเฉลี่ยทางคณิตศาสตร์ ได้ดังนี้

```
# Combine with box plots
R> e + geom_boxplot() +
     geom_dotplot(binaxis = "y", stackdir = "center")

# Combine with violin plots
R> e + geom_violin(trim = FALSE) +
     geom_dotplot(binaxis = "y", stackdir = "center")

# Dot plots + violin plots + stat summary
R> e + geom_violin(trim = FALSE) +
     geom_dotplot(binaxis = "y", stackdir = "center") +
     stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1),
                  geom = "pointrange", color = "red")
```



รูปที่ 11-15 Dot plots with box plots and violin plots

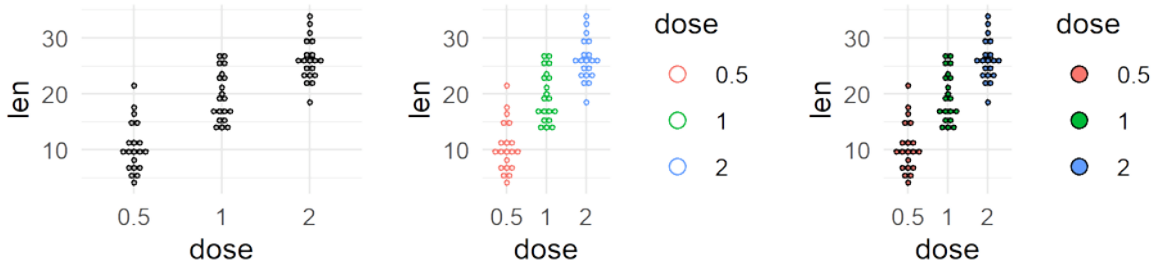
รูปแรกเป็นการแสดง Dot plots ร่วมกับ Box plots รูปที่สองเป็นการแสดง Dot plots ร่วมกับ Violin plots รูปสุดท้ายเป็นการแสดงค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐานร่วมกับ Dot plots และ Violin plots

การเปลี่ยนสีแต่ละกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน **aes()** โดยใช้พารามิเตอร์ **color** หรือ **fill** ดังนี้

```
# Use single colors
R> e + geom_dotplot(binaxis = "y", stackdir = "center", color = "black",
                        fill = "lightgrey") +
     theme_minimal()

# Change outline colors by dose (groups)
R> e + geom_dotplot(aes(color = dose), binaxis = "y", stackdir = "center",
                        fill = "white") +
     theme_minimal()

# Change fill color by dose (groups)
R> e + geom_dotplot(aes(fill = dose), binaxis = "y", stackdir = "center") +
     theme_minimal()
```



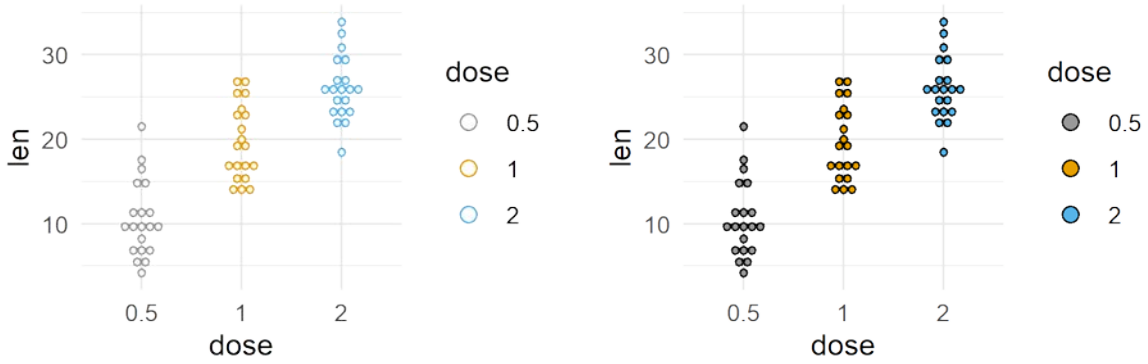
รูปที่ 11-16 Dot plots with color by groups

ทั้ง 3 รูปด้านบนแสดง Dot plots โดยแกน Y แสดงความหนาแน่นของตัวแปร len ส่วนแกน X แสดงตัวแปร dose รูปที่สองแสดงด้วยเส้นกรอบสีตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีชมพู, สีเขียว และสีฟ้า ตามลำดับ รูปสุดท้ายเป็นข้อมูลเดียวกันแต่แสดงสีจุด (Fill) เหมือนกับรูปที่สองยกเว้นกรอบสีดำ

สามารถเปลี่ยนสีเส้นด้วยคำสั่ง **scale_color_manual()** หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง **scale_color_brewer()** และ **scale_color_grey()** หรือเติมสี (Fill) จุดด้วยคำสั่ง **scale_fill_manual()** หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง **scale_fill_brewer()** และ **scale_fill_grey()** ดังนี้

```
# Change manually outline colors
# Use custom color palettes
R> e2 <- e + geom_dotplot(aes(color = dose), binaxis = "y",
                              stackdir = "center", fill = "white") +
     theme_minimal()
R> e2 + scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```

```
# Change manually fill colors
# Use custom color palettes
R> e3 <- e + geom_dotplot(aes(fill = dose), binaxis = "y",
                          stackdir="center") +
     theme_minimal()
R> e3 + scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```



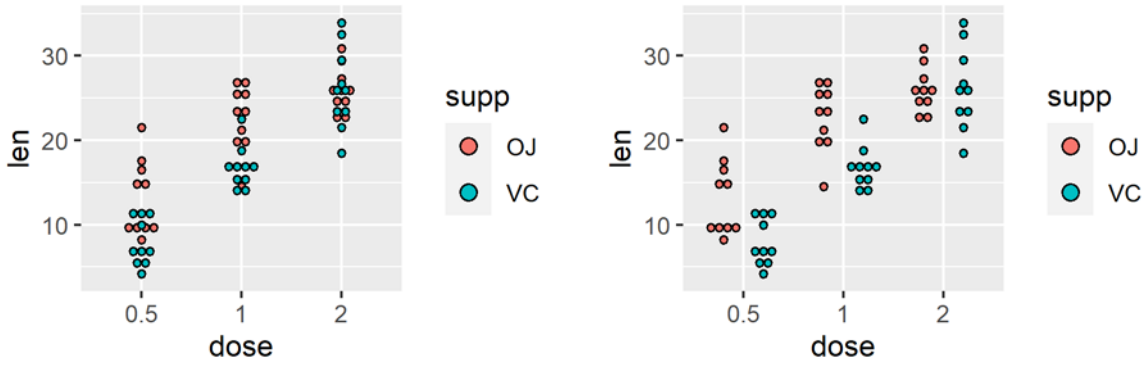
รูปที่ 11-17 Dot plots with manual color by groups

ทั้ง 2 รูปเป็น Dot plots แสดงสีตามข้อมูลในตัวแปร dose ด้วยการกำหนดสีแบบ manual รูปแรกแสดงด้วยเส้นกรอบสีตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีเทา, สีเหลือง, และสีฟ้า ตามลำดับ รูปที่สองเป็นข้อมูลเดียวกันแต่แสดงสี (Fill) ที่จุดแทน

แสดง Dot plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ได้ดังนี้

```
# Change colors by groups
R> e + geom_dotplot(aes(fill = supp), binaxis = "y", stackdir = "center")

# Change the position : interval between dot plots of the same group
R> e + geom_dotplot(aes(fill = supp), binaxis = "y", stackdir = "center",
                    position = position_dodge(0.8))
```



รูปที่ 11-18 Dot plots with multiple groups

ทั้ง 2 รูปเป็น Dot plots ที่แสดงสี (Fill) ด้วยตัวแปร supp และแสดงแกน X ด้วยตัวแปร dose รูปแรกเป็นการพล็อตด้วยค่าโดยปริยาย ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า รูปที่สองเป็น

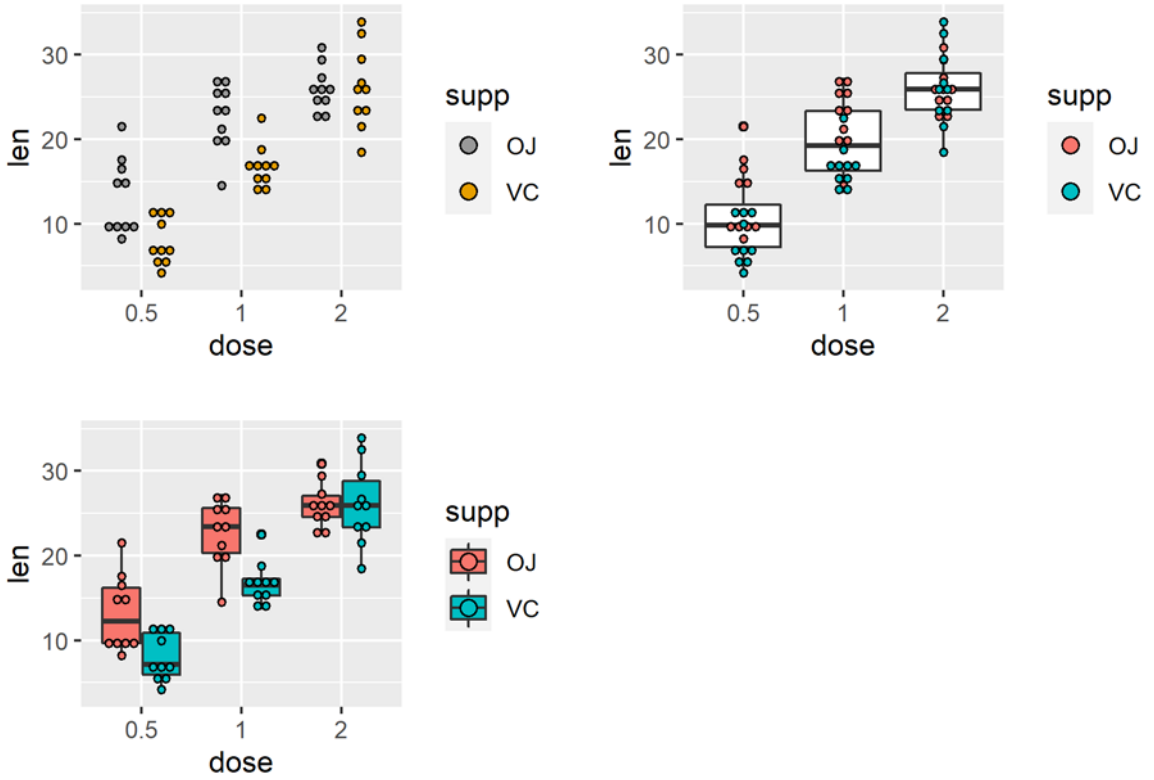
ข้อมูลเดียวกับรูปแรกแต่พล็อตโดยการแยกจุดที่พล็อตข้อมูล OJ และ VC ออกมาด้านข้างให้มองเห็นชัดเจนขึ้น

Dot plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) สามารถเปลี่ยนสีและเพิ่มรายละเอียดอื่น ๆ เช่น เพิ่ม Box plots ได้ดังนี้

```
# Change colors
R> e + geom_dotplot(aes(fill = supp), binaxis = "y", stackdir = "center",
                        position = position_dodge(0.8)) +
     scale_fill_manual(values = c("#999999", "#E69F00"))

# Add box plots
R> e + geom_boxplot(fill = "white") +
     geom_dotplot(aes(fill = supp), binaxis = "y", stackdir = "center")

# Change the position
R> e + geom_boxplot(aes(fill = supp), position = position_dodge(0.8)) +
     geom_dotplot(aes(fill = supp), binaxis = "y", stackdir = "center",
                      position = position_dodge(0.8))
```



รูปที่ 11-19 Dot plots with multiple groups and other details

ทั้ง 3 รูปเป็น Dot plots ที่แสดงสี (Fill) ด้วยตัวแปร supp และแสดงแกน X ด้วยตัวแปร dose รูปซ้ายบนเป็นการพล็อตโดยการกำหนดสีแบบ Manual ซึ่งแสดงข้อมูล OJ ด้วยสีเทาและข้อมูล VC ด้วยสีเหลือง และแยกจุดที่พล็อตข้อมูล OJ และ VC ออกมาด้านข้าง รูปขวาบนเป็นข้อมูลเดียวกับรูปซ้ายบนแต่พล็อตโดยการใช้ค่าโดยปริยาย ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า และมีการเพิ่ม Box plots ของ

ข้อมูลเข้าไปด้วยโดยไม่แยกประเภทตามตัวแปร dose รูปล่างเป็นการแสดง Box plots และ Dot plots โดยการใช้ค่าโดยปริยาย ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้าและแยกข้อมูล OJ และ VC ออกมาด้านข้างให้มองเห็นชัดเจนขึ้น

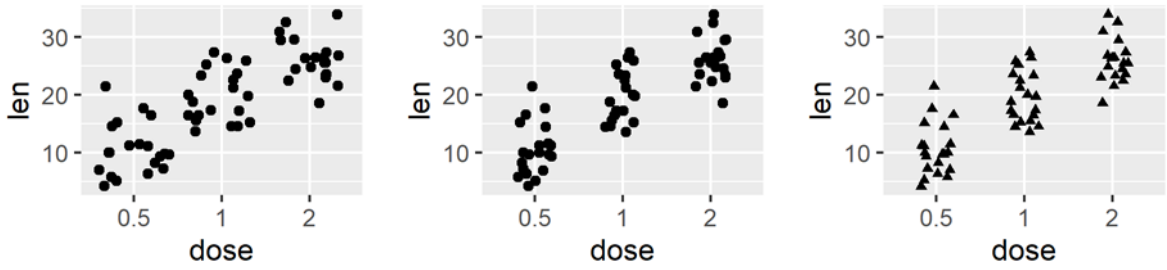
11.4 กราฟที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter Plots)

Jitter plots หรือบางครั้งเรียกว่า Strip charts มีลักษณะเหมือนกับ Scatter plots แต่แสดงเพียง 1 มิติ (One dimension) เพื่อแสดงจุดที่ซ้อนทับกันให้มองเห็นได้ง่ายขึ้น เหมาะกับการพล็อตที่มีจำนวนข้อมูลไม่มากนัก การพล็อตใช้คำสั่ง `geom_jitter()` หรือ `stat_summary()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, shape, fill` และ `size` ดังนี้

```
# Basic plots
R> e + geom_jitter()

# Change the position
# 0.2 : degree of jitter in x direction
R> e + geom_jitter(position = position_jitter(0.2))

# Change point shapes and size
R> e + geom_jitter(position = position_jitter(0.2),
                   shape = 17, size = 1.2)
```



รูปที่ 11-20 Jitter plots

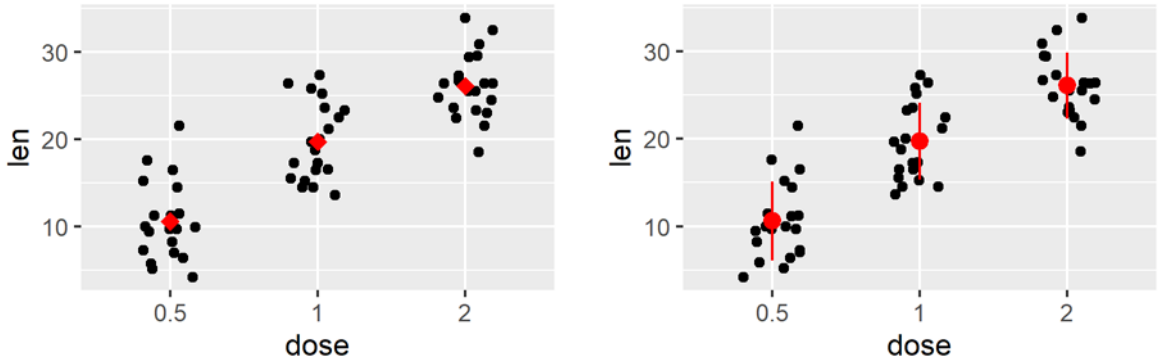
รูปที่หนึ่งและสองแสดง Jitter plots ด้วยจุดวงกลมที่มีระยะห่างระหว่างจุดแตกต่างกัน ส่วนรูปสุดท้ายแสดงด้วยจุดสามเหลี่ยม

สามารถแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน `scale_x_discrete()` โดยผ่านพารามิเตอร์ `limits` ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ เช่นเดียวกับหัวข้อก่อนหน้านี้

ฟังก์ชัน `stat_summary()` สามารถแสดงค่าเฉลี่ยทางคณิตศาสตร์ เช่น Mean, Median และค่าอื่นได้ดังนี้

```
# Add mean or median point: use fun = mean or fun = median
R> e + geom_jitter(position = position_jitter(0.2)) +
     stat_summary(fun = mean, geom = "point",
                  shape = 18, size = 3, color = "red")
```

```
# Add mean points +/- SD
# Use geom = "pointrange" or geom = "crossbar"
R> e + geom_jitter(position = position_jitter(0.2))+
     stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                  geom = "pointrange", color = "red")
```



รูปที่ 11-21 Jitter plots with statistics

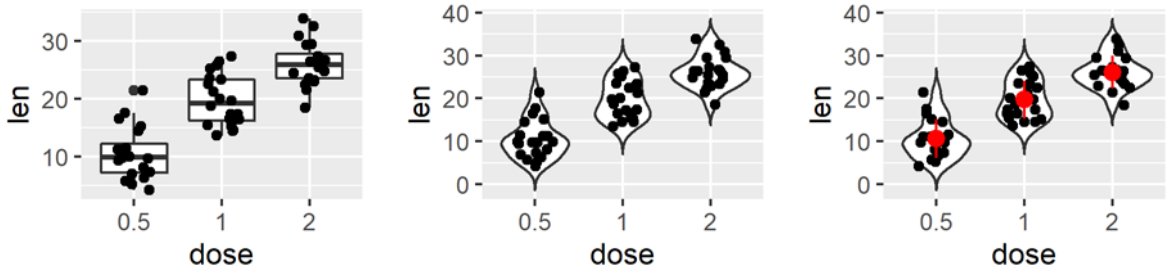
ทั้ง 2 รูปแสดง Jitter plots รูปแรกแสดงค่าเฉลี่ยประกอบ ส่วนรูปที่สองแสดงทั้งค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐาน

สามารถแสดง Jitter plots ร่วมกับ Box plots และ Violin plots รวมทั้งเพิ่มการแสดงค่าเฉลี่ยทางคณิตศาสตร์ ได้ดังนี้

```
# Combine with box plots
R> e + geom_boxplot() +
     geom_jitter(position = position_jitter(0.2))

# Combine with violin plots
R> e + geom_violin(trim = FALSE) +
     geom_jitter(position = position_jitter(0.2))

# Strip charts + violin plots + stat summary
R> e + geom_violin(trim = FALSE) +
     geom_jitter(position = position_jitter(0.2)) +
     stat_summary(fun.data = "mean_sdl", fun.args = list(mult=1),
                  geom = "pointrange", color = "red")
```



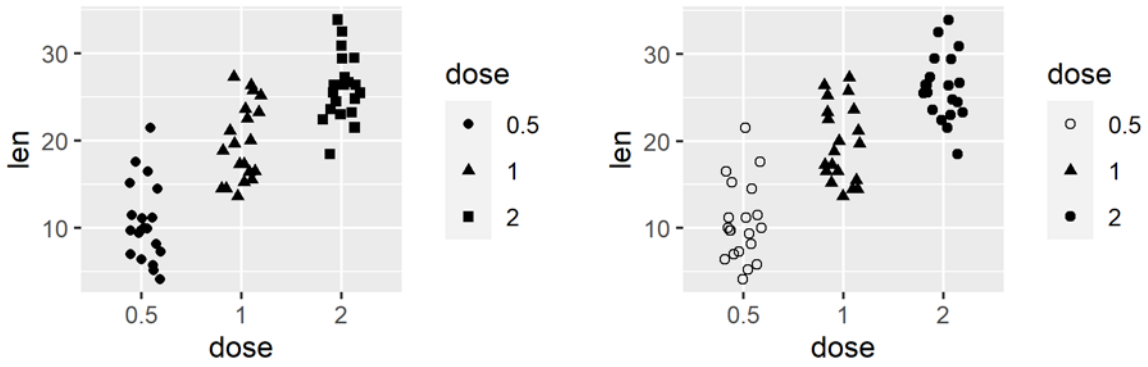
รูปที่ 11-22 Jitter plots with box plots and violin plots

รูปแรกแสดง Jitter plots พร้อมกับ Box plots รูปที่สองแสดง Jitter plots พร้อมกับ Violin plots รูปสุดท้ายแสดงทั้งค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐานประกอบ

การเปลี่ยนรูปร่างจุด (Point shapes) ตามแต่ละกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงรูปร่างด้วยฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `shape` รวมทั้งสามารถกำหนดรูปร่างจุดตามที่ต้องการด้วยคำสั่ง `scale_shape_manual()` ดังนี้

```
# Change point shapes by groups
R> e + geom_jitter(aes(shape = dose), position = position_jitter(0.2))

# Change point shapes manually
R> e + geom_jitter(aes(shape = dose), position = position_jitter(0.2)) +
     scale_shape_manual(values = c(1, 17, 19))
```



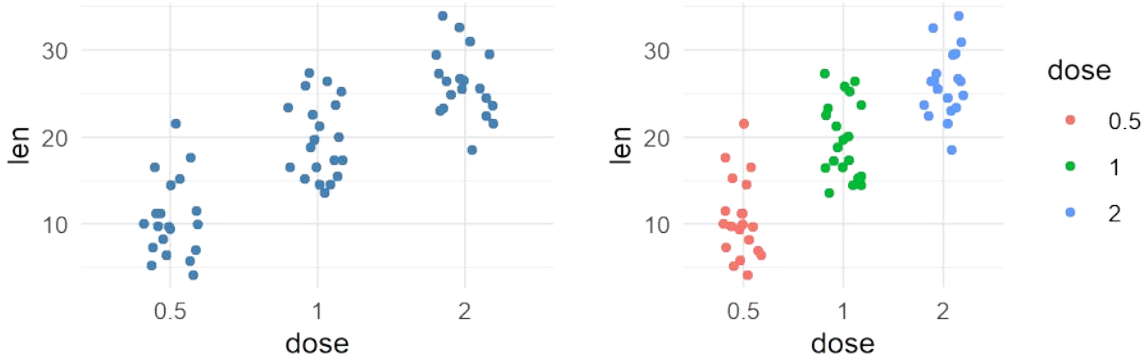
รูปที่ 11-23 Jitter plots with differrent point shapes

ทั้ง 2 รูปแสดง Jitter plots โดยแสดงจุดที่มีรูปร่างแตกต่างกับตามประเภทตัวแปร dose (การกำหนดรูปร่างแบบต่าง ๆ ดูรายละเอียดเพิ่มเติมในหัวข้อ 12.5)

การเปลี่ยนสีจุดตามแต่ละกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` ดังนี้

```
# Use single colors
R> e + geom_jitter(position = position_jitter(0.2), color = "steelblue") +
     theme_minimal()

# Change point colors by dose (groups)
R> e + geom_jitter(aes(color = dose), position = position_jitter(0.2)) +
     theme_minimal()
```



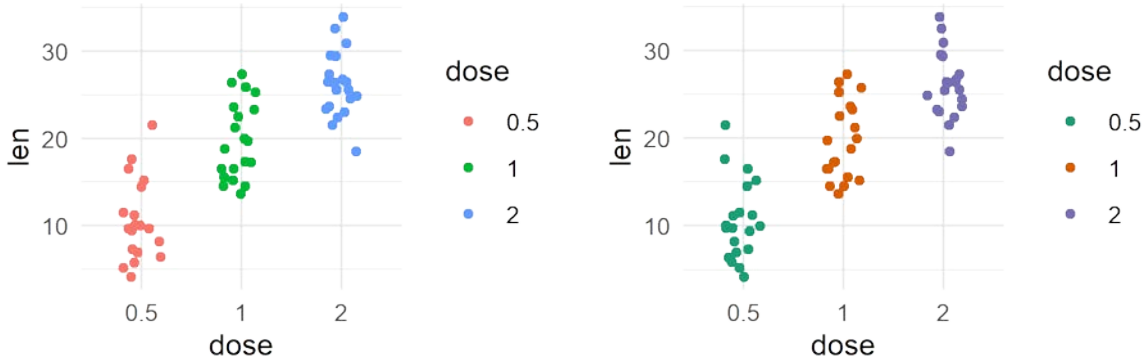
รูปที่ 11-24 Jitter plots with color by groups

ทั้ง 2 รูปแสดง Jitter plots รูปแรกแสดงด้วยจุดกลมสีน้ำเงินตามตัวแปร dose ในแกน X รูปที่สองแสดงสีแตกต่างกันตามตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีชมพู, สีเขียว, และสีฟ้า ตามลำดับ

สามารถเปลี่ยนสีจุดด้วยคำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` หรือเติมสีจุด (Fill) ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_fill_grey()` ดังนี้

```
# Use custom color palettes
R> e3 <- e + geom_jitter(aes(color = dose), position = position_jitter(0.2)) +
     theme_minimal()
R> e3 + scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))

# Use brewer color palettes
R> e3 + scale_color_brewer(palette = "Dark2")
```



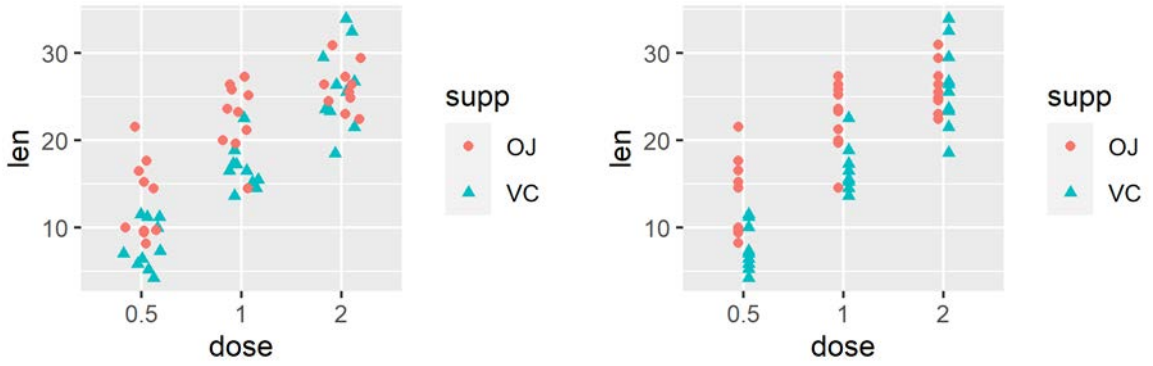
รูปที่ 11-25 Jitter plots with manual colors by groups

ทั้ง 2 รูปแสดง Jitter plots ด้วยสีแตกต่างกันตามตัวแปร dose รูปแรกแสดงสีแบบ manual ด้วยสีชมพู, สีเขียว, และสีฟ้า เมื่อ dose เท่ากับ 0.5, 1 และ 2 ตามลำดับ รูปที่สองแสดงสีจากจานสี (palettes)

แสดง Jitter plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ได้ดังนี้

```
# Change colors and shapes by groups
R> e + geom_jitter(aes(color = supp, shape = supp),
                       position = position_jitter(0.2))

# Change the position : interval between dot plots of the same group
R> e + geom_jitter(aes(color = supp, shape = supp),
                       position = position_dodge(0.2))
```



รูปที่ 11-26 Jitter plots with multiple groups

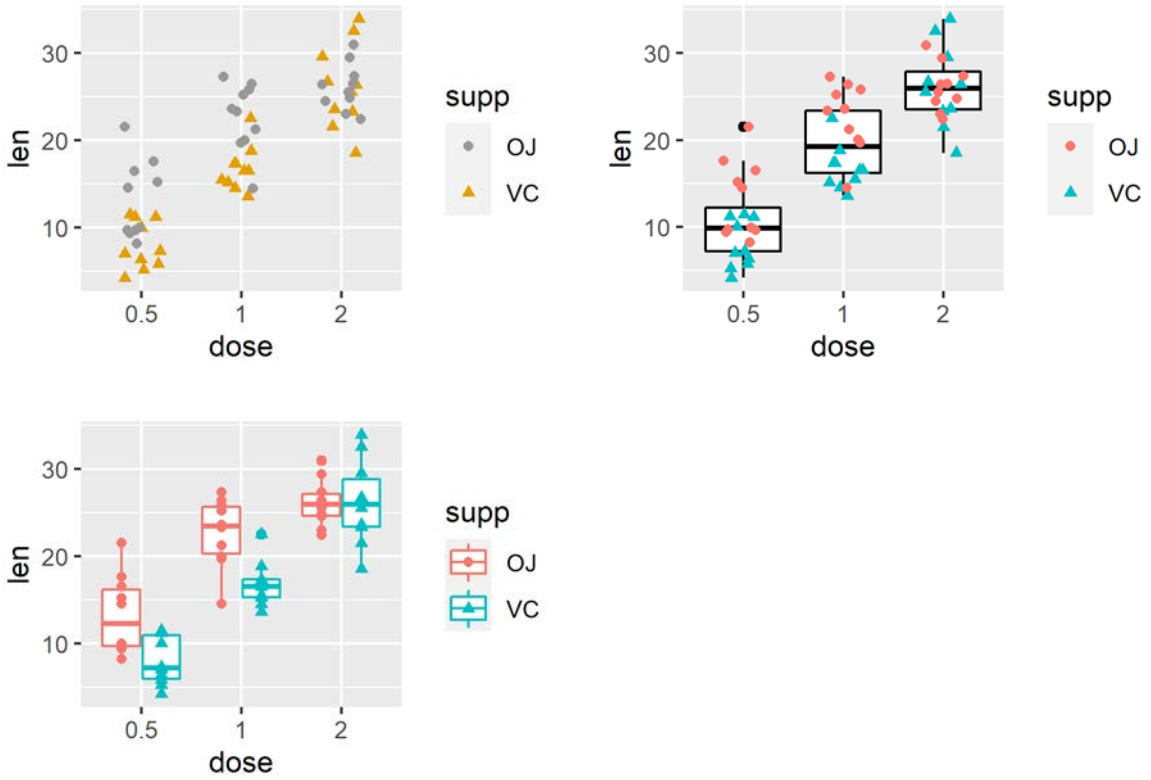
ทั้ง 2 รูปเป็น jitter plots ที่แสดงสี (Fill) ด้วยตัวแปร supp ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า รูปแรกแสดงตำแหน่งจุดด้วยอากิวเมนต์ `position_jitter(0.2)` รูปที่สองแสดงตำแหน่งจุดด้วยอากิวเมนต์ `position_dodge(0.2)`

Jitter plots ด้วยตัวแปรหลายกลุ่ม (Multiple groups) สามารถเปลี่ยนสี และเพิ่มรายละเอียดอื่น ๆ เช่น เพิ่ม Box plots ได้ดังนี้

```
# Change colors
R> e + geom_jitter(aes(color = supp, shape = supp),
                       position = position_jitter(0.2))+
     scale_color_manual(values = c("#999999", "#E69F00"))

# Add box plots
R> e + geom_boxplot(color = "black") +
     geom_jitter(aes(color = supp, shape = supp),
                 position = position_jitter(0.2))

# Change the position
R> e + geom_boxplot(aes(color = supp), position = position_dodge(0.8)) +
     geom_jitter(aes(color = supp, shape = supp),
                 position = position_dodge(0.8))
```



รูปที่ 11-27 Jitter plots with multiple groups and other details

ทั้ง 3 รูปเป็น Jitter plots ที่แสดงสี (Fill) ด้วยตัวแปร supp และแสดงแกน X ด้วยตัวแปร dose รูปซ้ายบนเป็นการพล็อตโดยการกำหนดสีแบบ Manual ซึ่งแสดงข้อมูล OJ ด้วยสีเทาและข้อมูล VC ด้วยสีเหลือง รูปขวาบนเป็นข้อมูลเดียวกันแต่แสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้า และมีการเพิ่ม Box plots ของข้อมูลเข้าไปด้วยโดยไม่แยกประเภทตามตัวแปรใด ๆ รูปล่างเป็นการแสดง Box plots และ Jitter plots ซึ่งแสดงข้อมูล OJ ด้วยสีชมพูและข้อมูล VC ด้วยสีฟ้าและแยกข้อมูล OJ และ VC ออกมาด้านข้างให้มองเห็นชัดเจนขึ้น

11.5 กราฟเส้น (Line Plots)

กราฟเส้นเป็นการพล็อตเส้นเชื่อมต่อกันระหว่างจุดต่อจุดต่อเนื่องกันตามลำดับ เพื่อที่จะให้มองเห็นการกระเพื่อมขึ้นลง (Fluctuation) หรือแนวโน้ม (Trend) ของกราฟ หรือข้อมูลใด ๆ ที่มีการเปลี่ยนแปลงไป ตัวแปร X อาจจะเป็นอนุกรมเวลา (Time series) ข้อความ (Texts) ตัวเลขไม่ต่อเนื่อง (Discrete numeric values) หรือตัวเลขต่อเนื่องกัน (Continuous numeric values) ก็ได้ ส่วนตัวแปร Y เป็นตัวแปรต่อเนื่อง (Continuous variable) กราฟเส้นอาจจะใช้กับตัวแปรแบบไม่ต่อเนื่อง (Discrete variable) บนแกน x ได้เมื่อตัวแปรดังกล่าวมีลำดับ (Order) เช่น เล็ก กลาง ใหญ่ เป็นต้น แต่ไม่เหมาะกับตัวแปรไม่ต่อเนื่องที่ไม่มีลำดับ (Unorder) Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

- `geom_line()` สำหรับพล็อตกราฟเส้น

- `geom_step()` สำหรับพล็อตกราฟเส้นแบบขั้นบันได
- `geom_path()` สำหรับพล็อตกราฟเส้นแบบ path

โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, linetype,` และ `size` ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูลที่ดัดแปลงจาก ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

```
R> df <- data.frame(dose = c("D0.5", "D1", "D2"),
                    len = c(4.2, 10, 29.5))

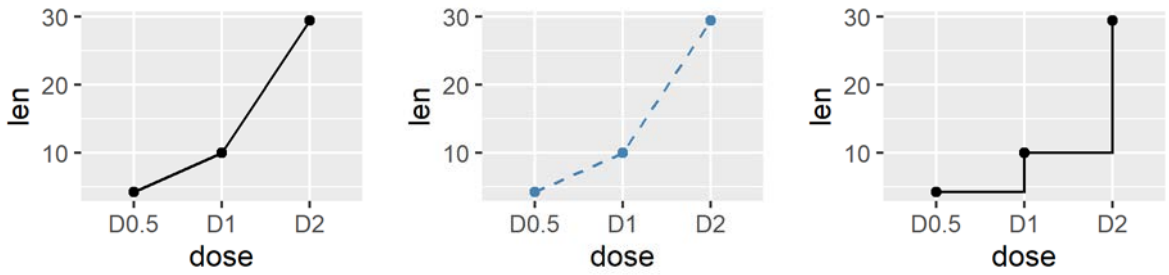
R> df2 <- data.frame(supp = rep(c("VC", "OJ"), each = 3),
                     dose = rep(c("D0.5", "D1", "D2"), 2),
                     len = c(6.8, 15, 33, 4.2, 10, 29.5))
```

ตัวอย่างกราฟเส้น แสดงได้ดังนี้

```
R> p <- ggplot(df, aes(x = dose, y = len, group = 1))
# Basic line plots with points
R> p + geom_line() +
     geom_point()

# Change line type and color
R> p + geom_line(linetype = "dashed", color = "steelblue") +
     geom_point(color = "steelblue")

# Use geom_step()
R> p + geom_step() +
     geom_point()
```



รูปที่ 11-28 Line plots

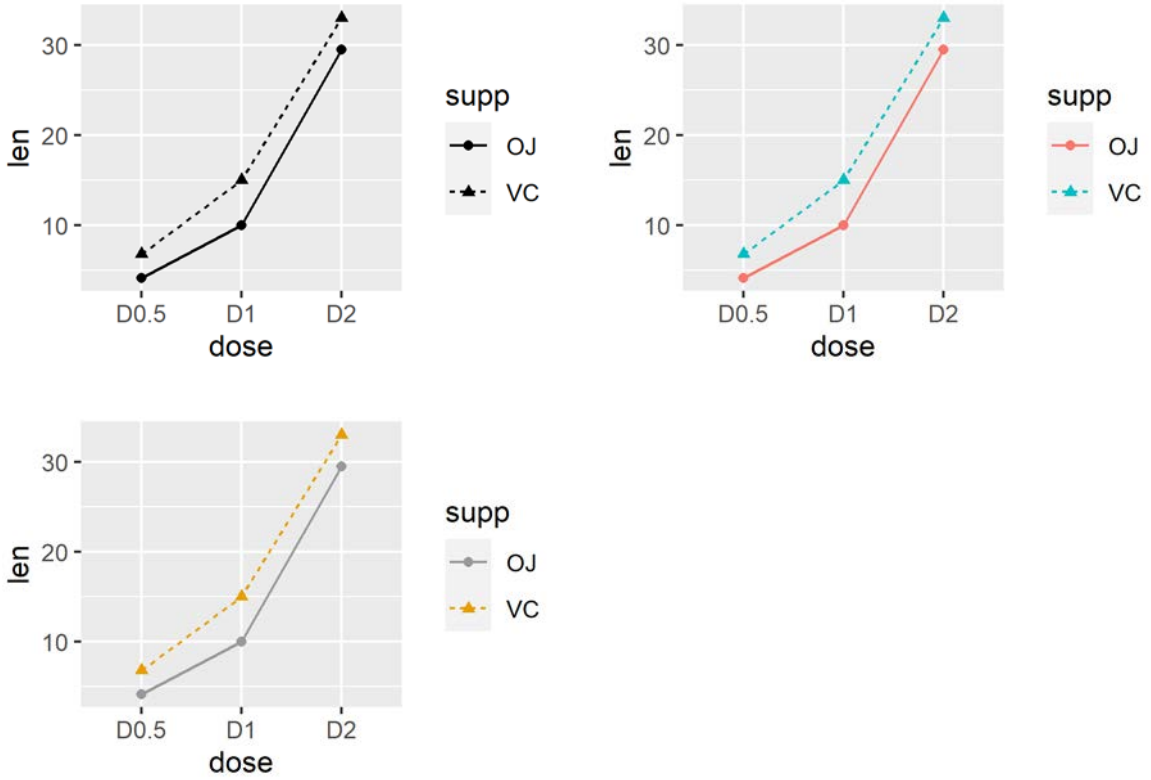
รูปแรกแสดงเส้นตรง (Lines) ต่อเนื่องระหว่างจุดแต่ละจุดด้วยข้อมูล X และ Y ตามลำดับ รูปที่สองแสดงเส้นประ (Dash) สีน้ำเงินเข้ม ส่วนรูปสุดท้ายแสดงเส้นแบบขั้นบันได (Step)

การเปลี่ยนชนิดเส้นและสีตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงชนิดเส้นและสีในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `linetype` หรือ `color` ดังนี้

```
R> p <- ggplot(df2, aes(x = dose, y = len, group = supp))
# Change line types and point shapes by groups
R> p + geom_line(aes(linetype = supp)) +
     geom_point(aes(shape = supp))
```

```
# Change line types, point shapes and colors
R> p + geom_line(aes(linetype = supp, color = supp)) +
     geom_point(aes(shape = supp, color = supp))

# Change color manually: custom color
R> p + geom_line(aes(linetype = supp, color = supp)) +
     geom_point(aes(shape = supp, color = supp)) +
     scale_color_manual(values = c("#999999", "#E69F00"))
```



รูปที่ 11-29 Line plots by groups

รูปซ้ายบนแสดงกราฟเส้นตามกลุ่มตัวแปร supp ซึ่งประกอบด้วย OJ และ VC แสดงด้วยเส้นประและเส้นตรง ตามลำดับ รูปขวาบนเป็นข้อมูลชุดเดียวกันแต่เพิ่มสีให้แตกต่างตามกลุ่มตัวแปร supp ซึ่งประกอบด้วย OJ แสดงด้วยเส้นสีชมพู และ VC แสดงด้วยเส้นสีฟ้า รูปล่างซ้ายเป็นข้อมูลชุดเดียวกันแต่เปลี่ยนสีให้ OJ แสดงด้วยเส้นสีเทา และ VC แสดงด้วยเส้นสีเหลือง

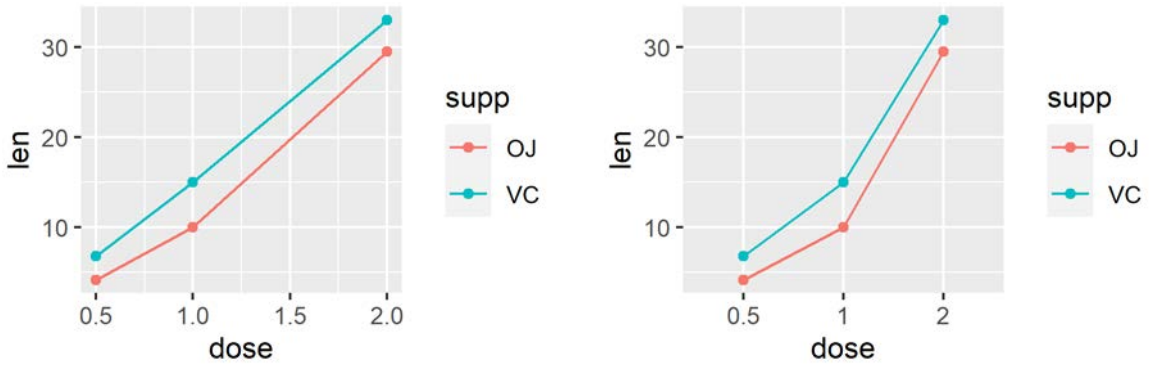
ถ้าข้อมูลตัวแปรในแกน X เป็นตัวเลข สามารถกำหนดให้แสดงผลเป็นตัวเลขต่อเนื่อง หรือเป็นเหมือนตัวแปรไม่ต่อเนื่อง ได้ดังนี้

```
# Create some data
R> df3 <- data.frame(supp = rep(c("VC", "OJ"), each = 3),
                     dose = rep(c("0.5", "1", "2"),2),
                     len = c(6.8, 15, 33, 4.2, 10, 29.5))

# x-axis treated as continuous variable
R> df3$dose <- as.numeric(df3$dose)
R> ggplot(df3, aes(x = dose, y = len, group = supp, color = supp)) +
```

```
      geom_line() +
      geom_point()

# x-axis treated as discrete variable
R> df2$dose <- as.factor(df3$dose)
R> ggplot(df2, aes(x = dose, y = len, group = supp, color = supp)) +
      geom_line() +
      geom_point()
```



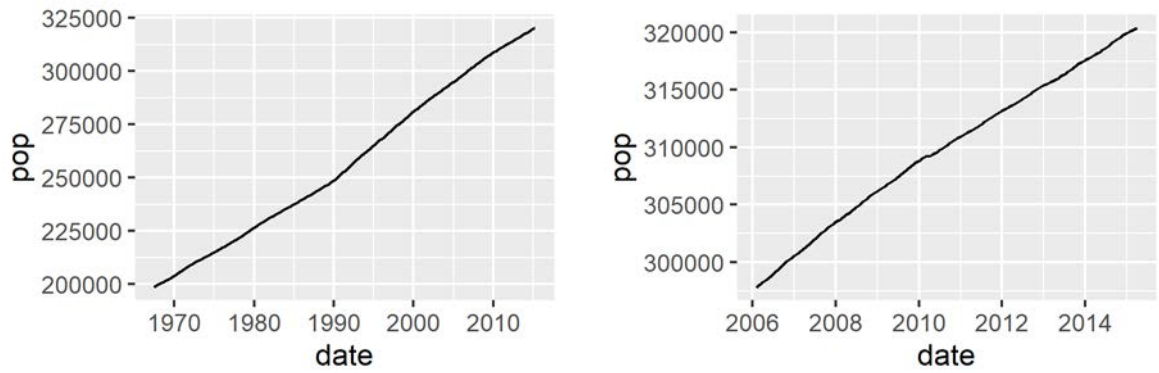
รูปที่ 11-30 Line plots by X-axis as continuous and discrete variables

รูปแรกแสดงแกน X เป็นตัวเลขต่อเนื่อง ส่วนรูปที่สองแสดงแกน X เป็นตัวเลขไม่ต่อเนื่องเสมือนเป็นตัวแปรแบ่งกลุ่ม ค่าแรกคือ 0.5 ตามด้วย 1 และ 2 ตามลำดับ

กราฟเส้นสามารถแสดงแกน X เป็นอนุกรมเวลา (Time series) โดยจะใช้ข้อมูลชื่อ economics ซึ่งติดตั้งมาพร้อมกับแพ็กเกจ `ggplot2` ได้ดังนี้

```
# Basic line plots
R> ggplot(economics, aes(x = date, y = pop))+
      geom_line()

# Plot a subset of the data
R> ss <- subset(economics, date > as.Date("2006-1-1"))
R> ggplot(ss, aes(x = date, y = pop)) +
      geom_line()
```

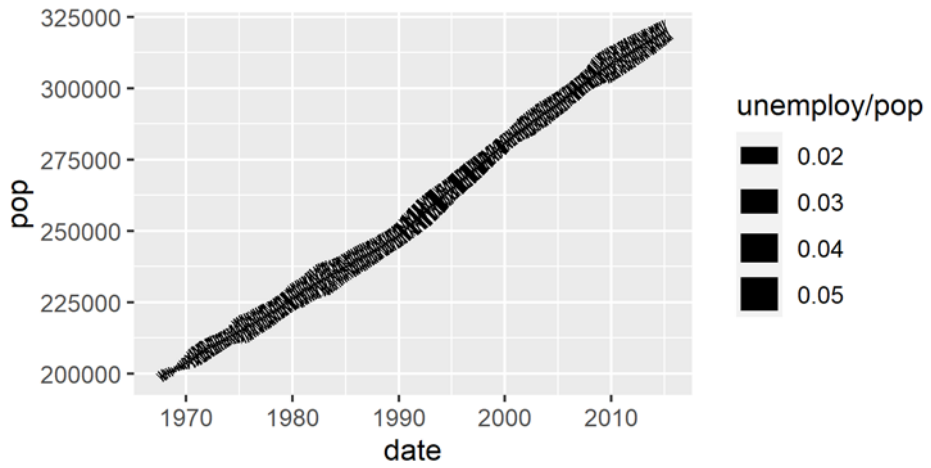


รูปที่ 11-31 Line plots with time series

รูปแรกแสดงแกน X เป็นปีตั้งแต่ประมาณปี 1967 ถึงปี 2015 ส่วนรูปที่สองแสดงแกน X เป็นปีตั้งแต่ปี 2006 เป็นต้นไป

การเปลี่ยนขนาดเส้นทำได้โดยระบุขนาดที่ต้องการในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `size` ดังนี้

```
# Change line size
R> ggplot(economics, aes(x = date, y = pop, size = unemploy/pop)) +
     geom_line()
```

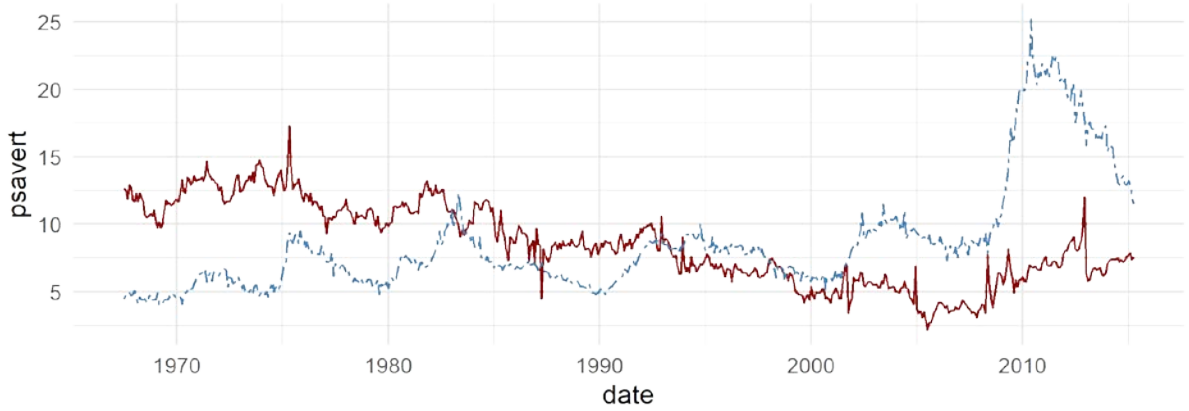


รูปที่ 11-32 Line plots with different line sizes

รูปด้านบนแสดงความหนาของเส้นตามสัดส่วนของ unemploy/pop

สามารถพล็อตกราฟเส้นหลายเส้นอยู่ในกราฟเดียวกัน โดยการพล็อตข้อมูลแต่ละชุดต่อเนื่องกันด้วยคำสั่ง `geom_line()` ได้ดังนี้

```
# Solution 1
R> ggplot(economics, aes(x = date)) +
     geom_line(aes(y = psavert), color = "darkred") +
     geom_line(aes(y = uempmed), color = "steelblue", linetype = "twodash") +
     theme_minimal()
```

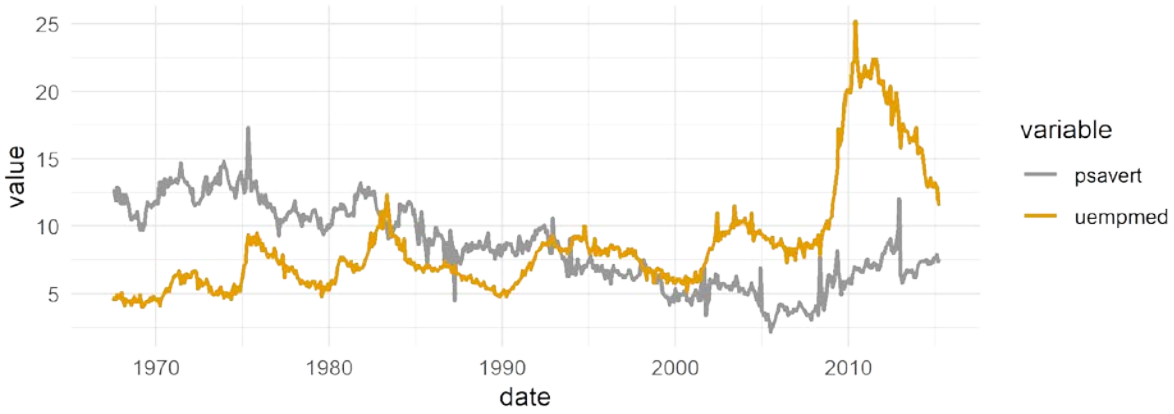


รูปที่ 11-33 Line plots with multiple time series data

รูปด้านบนแสดงข้อมูล 2 ชุด ชุดแรกเป็นเส้นสีแดงเข้มแสดงข้อมูล psavert ชุดที่สองเป็นเส้นน้ำเงินแสดงข้อมูล uempmed

วิธีการพล็อตกราฟเส้นด้วยข้อมูลหลายชุดสามารถทำได้อีกวิธีด้วยการสร้างตัวแปรขึ้นมาใหม่ด้วยคำสั่ง `melt()` และทำการพล็อตโดยการระบุตัวแปรที่สร้างขึ้นมาใหม่ในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` ได้ดังนี้

```
# Solution 2: melt by date.
R> require(reshape2)
R> df <- melt(economics[, c("date", "psavert", "uempmed")], id = "date")
R> ggplot(df, aes(x = date, y = value)) +
     geom_line(aes(color = variable), size = 1) +
     scale_color_manual(values = c("#999999", "#E69F00")) +
     theme_minimal()
```

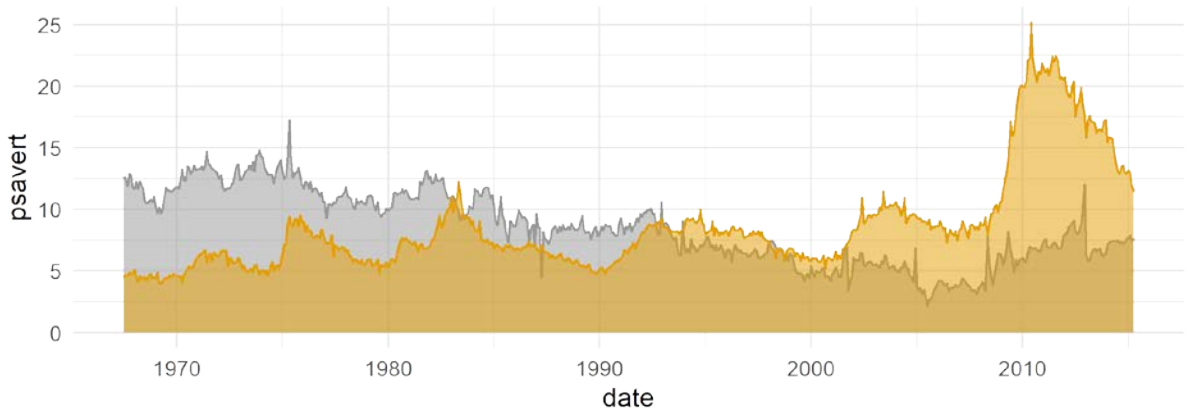


รูปที่ 11-34 Line plots with multiple time series data

รูปด้านบนแสดงข้อมูล 2 ชุดเหมือนกับรูปก่อนหน้านี้ ชุดแรกเป็นเส้นสีเทาแสดงข้อมูล psavert ชุดที่สองเป็นเส้นสีเหลืองแสดงข้อมูล uempmed

อีกวิธีการหนึ่งในการพล็อตกราฟเส้นคือ ใช้การพล็อตแบบพื้นที่ (Area plots) สำหรับข้อมูลแต่ละชุดโดยกำหนดพารามิเตอร์ `fill, color` และ `alpha` ได้ดังนี้

```
# Area plots
R> ggplot(economics, aes(x = date)) +
     geom_area(aes(y = psavert), fill = "#999999",
               color = "#999999", alpha = 0.5) +
     geom_area(aes(y = uempmed), fill = "#E69F00",
               color = "#E69F00", alpha = 0.5) +
     theme_minimal()
```



รูปที่ 11-35 Area plots with multiple time series data

รูปด้านบนแสดงข้อมูล 2 ชุด ชุดแรกเป็นพื้นที่สีเทาแสดงข้อมูล psavert ชุดที่สองเป็นพื้นที่สีเหลืองแสดงข้อมูล uempmed โดยมีการแสดงให้โปร่งใสด้วยอากิวเมนต์ `alpha = 0.5`

11.6 กราฟแท่ง (Bar Plots)

กราฟแท่ง (Bar Plots) คือ กราฟที่ประกอบไปด้วยแกนตั้งและแกนนอน และรูปสี่เหลี่ยมมุมฉากที่มีความสูงแตกต่างกันขึ้นอยู่กับขนาดของข้อมูล เรียกรูปสี่เหลี่ยมแต่ละรูปนี้ว่าแท่ง (bar) กราฟแท่งนิยมใช้แสดงตัวเลข (Numeric values) ในแกน y และแสดงข้อมูลแต่ละกลุ่ม (Sub groups) ในแกน x เช่น กราฟแท่งแสดงราคาสินค้าที่แตกต่างกัน 3-4 ประเภท กราฟแท่งไม่เหมาะกับแสดงราคาที่เปลี่ยนแปลงไปตามเวลา ซึ่งเวลาถือว่าเป็นตัวแปรต่อเนื่อง (Continuous variables)

การพล็อตกราฟแท่ง (Bar plots) ใช้คำสั่ง `geom_bar()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size`

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูลที่ดัดแปลงจาก ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

```
R> df <- data.frame(dose = c("D0.5", "D1", "D2"),
                    len = c(4.2, 10, 29.5))

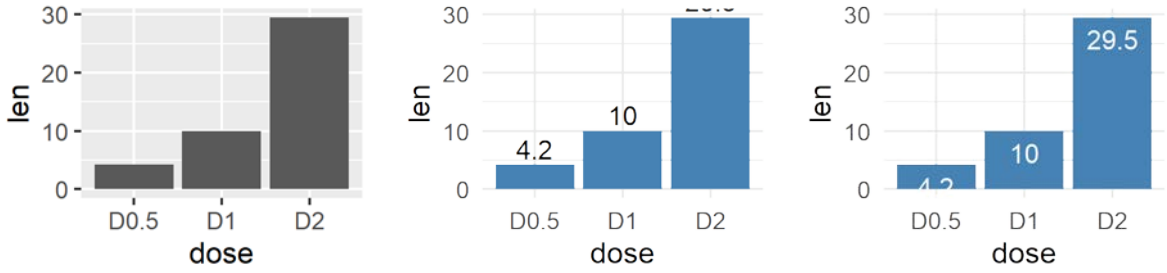
R> df2 <- data.frame(supp = rep(c("VC", "OJ"), each = 3),
                     dose = rep(c("D0.5", "D1", "D2"), 2),
                    len = c(6.8, 15, 33, 4.2, 10, 29.5))
R> f <- ggplot(df, aes(x = dose, y = len))

# Basic bar plots
R> f + geom_bar(stat = "identity")

# Change fill color and add labels at the top (vjust = -0.3)
R> f + geom_bar(stat = "identity", fill = "steelblue") +
     geom_text(aes(label = len), vjust = -0.3, size = 3.5) +
     theme_minimal()

# Label inside bars, vjust = 1.6
```

```
R> f + geom_bar(stat = "identity", fill = "steelblue") +
     geom_text(aes(label = len), vjust = 1.6, color = "white", size = 3.5) +
     theme_minimal()
```



รูปที่ 11-36 Bar plots

รูปแรกแสดงแท่งกราฟสีเทา รูปที่สองแสดงแท่งกราฟสีน้ำเงินและตัวเลขสีดำแสดงข้อมูลประกอบอยู่เหนือกราฟแต่ละแท่ง รูปสุดท้ายคล้ายกับรูปที่สองแต่แสดงตัวเลขสีขาวอยู่ด้านบนภายในกราฟแต่ละแท่ง

เปลี่ยนขนาดของแท่งกราฟด้วยพารามิเตอร์ **width** และแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน **scale_x_discrete()** โดยผ่านพารามิเตอร์ **limits** ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ

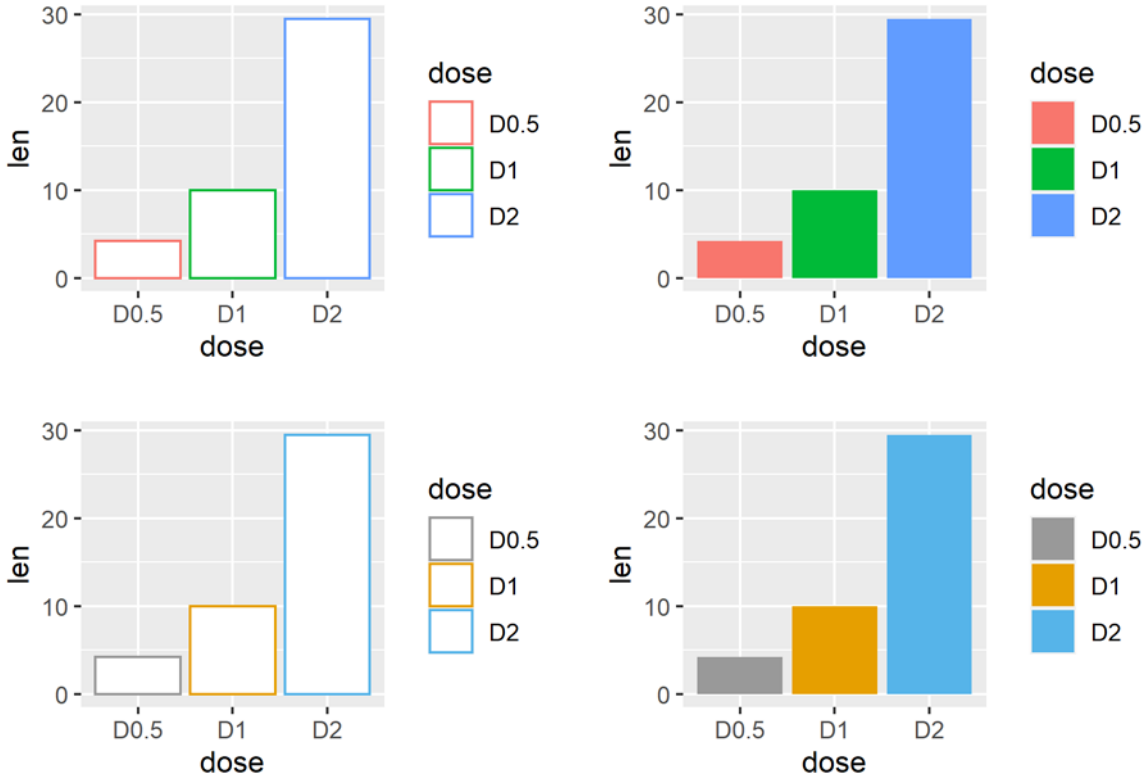
การเปลี่ยนสีกราฟแต่ละแท่งตามกลุ่มข้อมูล ทำได้โดยระบุตัวแปรที่ต้องการแสดงสีในฟังก์ชัน **aes()** โดยใช้พารามิเตอร์ **color** หรือ **fill** รวมทั้งสามารถเปลี่ยนสีเส้นด้วยคำสั่ง **scale_color_manual()** หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง **scale_color_brewer()** และ **scale_color_grey()** หรือเติมสี (Fill) ด้วยคำสั่ง **scale_fill_manual()** หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง **scale_fill_brewer()** และ **scale_fill_grey()** ดังนี้

```
# Change bar plots line colors by groups
R> f + geom_bar(aes(color = dose), stat = "identity", fill = "white")

# Change bar plots fill colors by groups
R> f + geom_bar(aes(fill = dose), stat = "identity")

# Change outline color manually: custom color
R> f + geom_bar(aes(color = dose), stat = "identity", fill = "white") +
     scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9"))

# Change fill color manually: custom color
R> f + geom_bar(aes(fill = dose), stat = "identity") +
     scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```



รูปที่ 11-37 Bar plots with color by groups

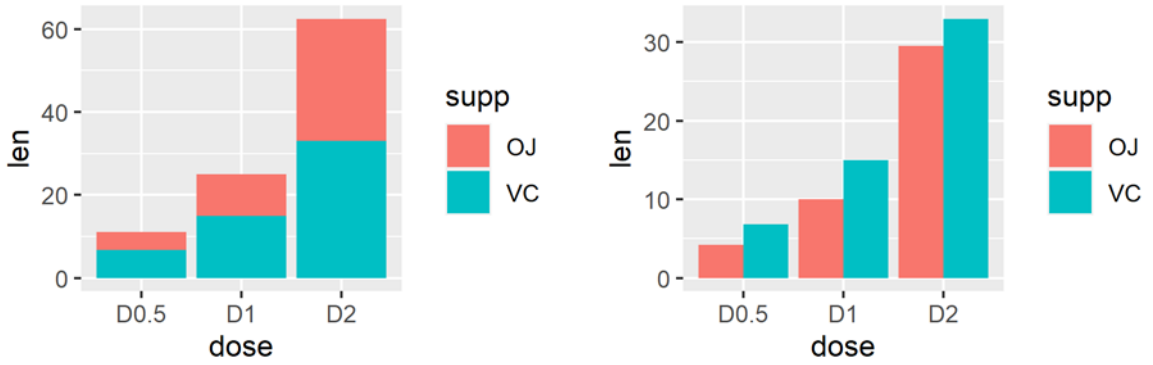
ทั้ง 4 รูปบนแสดงกราฟแท่งโดยแสดงสีตามข้อมูลตัวแปร dose รูปซ้ายบนและซ้ายล่างแสดงสีที่กรอบส่วนภายในแต่ละแท่งกราฟเป็นสีขาว รูปขวาบนและขวาล่างแสดงสี (Fill) แต่ละแท่งกราฟตามตัวแปร dose

แสดงกราฟแท่งด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ได้ดังนี้

```
R> g <- ggplot(df2, aes(x = dose, y = len, fill = supp))

# Stacked bar plots
R> g + geom_bar(stat = "identity")

# Use position = position_dodge()
R> g + geom_bar(stat = "identity", position = position_dodge())
```

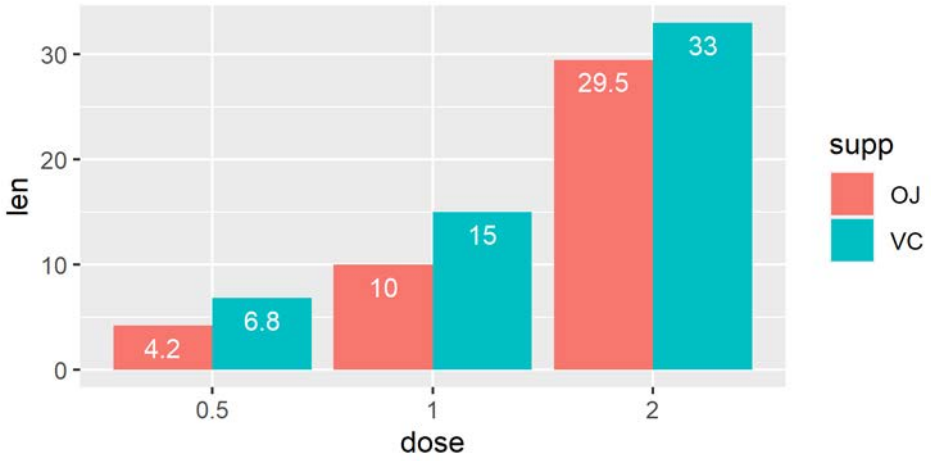


รูปที่ 11-38 Bar plots with multiple groups

รูปด้านบนเป็นกราฟแท่ง แกน X แสดงตัวแปร dose และแสดงสีตามตัวแปร supp ข้อมูล OJ แสดงด้วยสีชมพู ข้อมูล VC แสดงด้วยสีฟ้า ความแตกต่างคือ รูปแรกแสดงตำแหน่งแท่งกราฟวางซ้อนกันในแนวตั้ง รูปที่สองแสดงตำแหน่งแท่งกราฟวางชิดกันในแนวนอน

เพิ่มข้อความ (Texts) ด้วยคำสั่ง `geom_text()` โดยการระบุข้อความที่ต้องการแสดงในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `label` รวมทั้งกำหนดสีข้อความ และตำแหน่งที่ต้องการแสดงผล ได้ดังนี้

```
R> ggplot(df2, aes(x = dose, y = len, fill = supp)) +
     geom_bar(stat = "identity", position = position_dodge()) +
     geom_text(aes(label = len), vjust = 1.6, color = "white",
               position = position_dodge(0.9), size = 3.5)
```



รูปที่ 11-39 Bar plots with texts

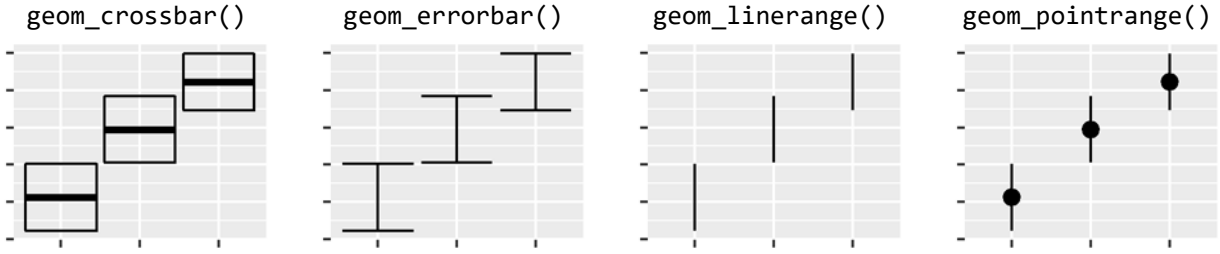
รูปด้านบนเป็นกราฟแท่ง แกน X แสดงตัวแปร dose และแสดงสีตามข้อมูลตัวแปร supp ข้อมูล OJ แสดงด้วยสีชมพู ข้อมูล VC แสดงด้วยสีฟ้า แสดงตำแหน่งแท่งกราฟวางชิดกันในแนวนอน และมีข้อความตัวเลขสีขาวอยู่ด้านบนภายในกราฟแต่ละแท่ง

11.7 การแสดงค่าความคลาดเคลื่อน (Visualizing Errors)

การแสดงค่าความคลาดเคลื่อน (Visualizing errors) เป็นเครื่องมือ/แผนภูมิเพื่อแสดงความแปรปรวนของข้อมูลและระบุความแตกต่างระหว่างค่าที่รายงานและมูลค่าที่แท้จริง Geometry ที่สามารถพล็อตได้ประกอบด้วย

- `geom_crossbar()` สำหรับพล็อตกราฟ Cross bars
- `geom_errorbar()` สำหรับพล็อตกราฟ Error bars
- `geom_errorbarh()` สำหรับพล็อตกราฟ Error bars ในแนวนอน
- `geom_linerange()` สำหรับพล็อตกราฟ Errors เป็นเส้นแนวตั้ง
- `geom_pointrange()` สำหรับพล็อตกราฟ Errors เป็นเส้นแนวตั้งพร้อมกับจุดแสดงค่ากลาง

แสดงดังรูปต่อไปนี้



รูปที่ 11-40 Error plots

การแสดงค่าความคลาดเคลื่อน (Visualizing Errors) จะใช้ข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** และหาค่าเฉลี่ย (Mean) และค่าส่วนเบี่ยงเบนมาตรฐาน (Standard deviation) ได้ดังนี้

```
# ToothGrowth data set
R> df <- ToothGrowth
R> df$dose <- as.factor(df$dose)

R> df2 <- df |>
     group_by(dose) |>
     summarise(
        sd = sd(len),
        len = mean(len)
     )
R> f <- ggplot(df2, aes(x = dose, y = len,
                        ymin = len-sd, ymax = len+sd))
```

11.7.1 การพล็อต Cross Bars

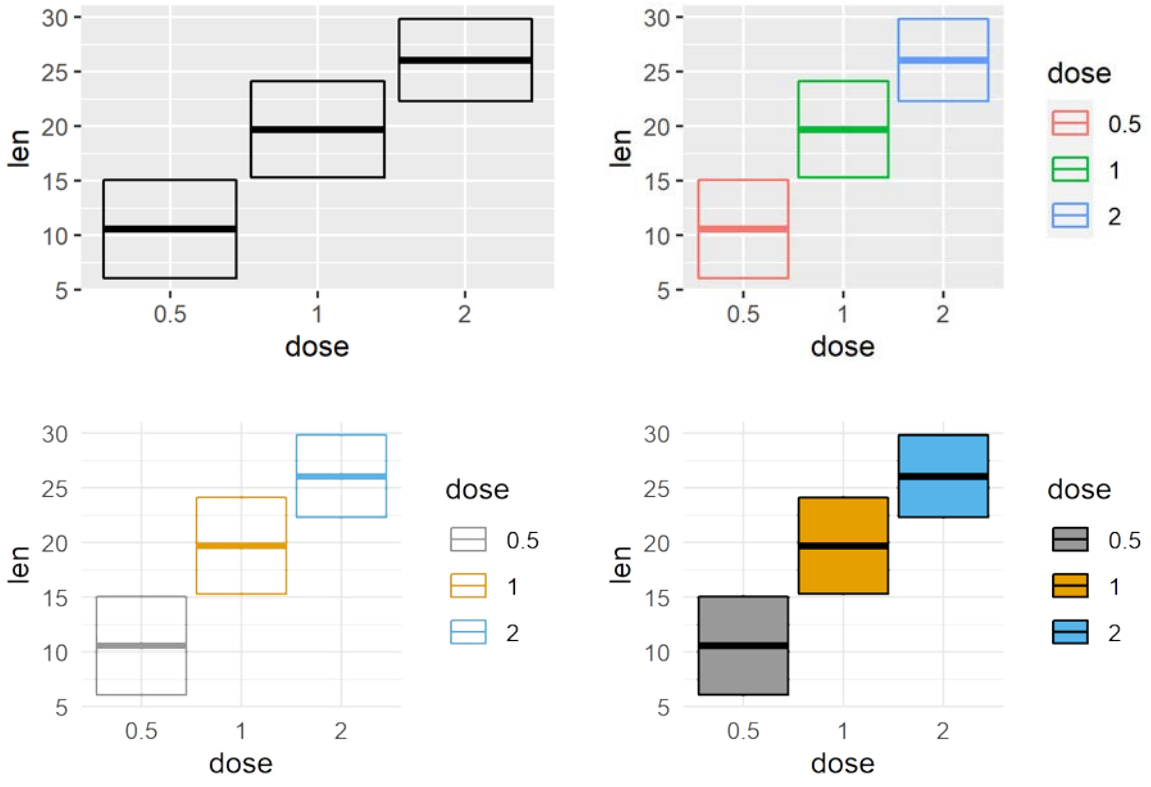
การพล็อต Cross bars ใช้คำสั่ง `geom_crossbar()` หรือ `stat_summary()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, fill, linetype` และ `size` ตัวอย่าง Cross bar plots แสดงได้ดังนี้

```
# Default plots
R> f + geom_crossbar()

# color by groups
R> f + geom_crossbar(aes(color = dose))

# Change color manually
R> f + geom_crossbar(aes(color = dose)) +
     scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9")) +
     theme_minimal()

# Fill by groups and change color manually
R> f + geom_crossbar(aes(fill = dose)) +
     scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9")) +
     theme_minimal()
```



รูปที่ 11-41 Cross bar plots

รูปซ้ายบนเป็นการพล็อต Cross bars แกน X แสดงตัวแปร dose ส่วนแกน Y เป็นตัวแปร len รูปขวาบนเป็นการแสดงสีตามข้อมูลตัวแปร dose สองรูปล่างเป็นการแสดงสีตามที่กำหนด

การพล็อต Cross bars ด้วยตัวแปรหลายกลุ่ม (Multiple groups) แสดงได้โดยการสร้างเดต้าเฟรมที่มีการแบ่งกลุ่มโดยตัวแปร 2 ตัวแปร ดังนี้

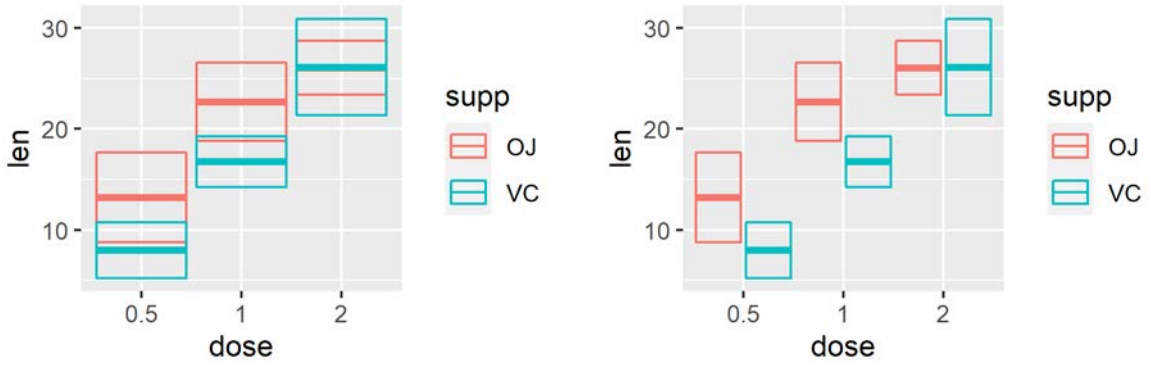
```
R> df3 <- df |>
  group_by(supp, dose) |>
  summarise(
    sd = sd(len),
    len = mean(len)
  )

R> f <- ggplot(df3, aes(x = dose, y = len,
                        ymin = len-sd, ymax = len+sd))
```

แสดงการพล็อต Cross bars ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม ดังนี้

```
# Default plots
R> f + geom_crossbar(aes(color = supp))

# Use position_dodge() to avoid overlap
R> f + geom_crossbar(aes(color = supp),
                     position = position_dodge(1))
```



รูปที่ 11-42 Cross bar plots by groups

รูปแรกเป็นการพล็อต Cross bars แกน X แสดงตัวแปร dose ส่วนแกน Y เป็นตัวแปร len แสดงสีตามข้อมูลตัวแปร dose ข้อมูล OJ และ VC อยู่ในแนวตั้งเดียวกัน รูปที่สองแสดงข้อมูล OJ และ VC ออกมาด้านข้างไม่ซ้อนทับกัน

การพล็อต Cross bars สามารถพล็อตอีกวิธีด้วยคำสั่ง `stat_summary()` โดยวิธีการนี้จะทำการคำนวณค่าเฉลี่ย (Mean) และค่าส่วนเบี่ยงเบนมาตรฐาน (Standard deviation) ให้อัตโนมัติ ดังนี้

```
# Alternative
R> f <- ggplot(df, aes(x = dose, y = len, color = supp))

# Use geom_crossbar()
R> f + stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                    geom = "crossbar", width = 0.6,
                    position = position_dodge(0.8))
```

11.7.2 การพล็อต Error Bars

การพล็อต Error bars ใช้คำสั่ง `geom_errorbar()` หรือ `stat_summary()` โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ `alpha, color, linetype, width` และ `size` ตัวอย่างการพล็อต Error bars แสดงได้ดังนี้

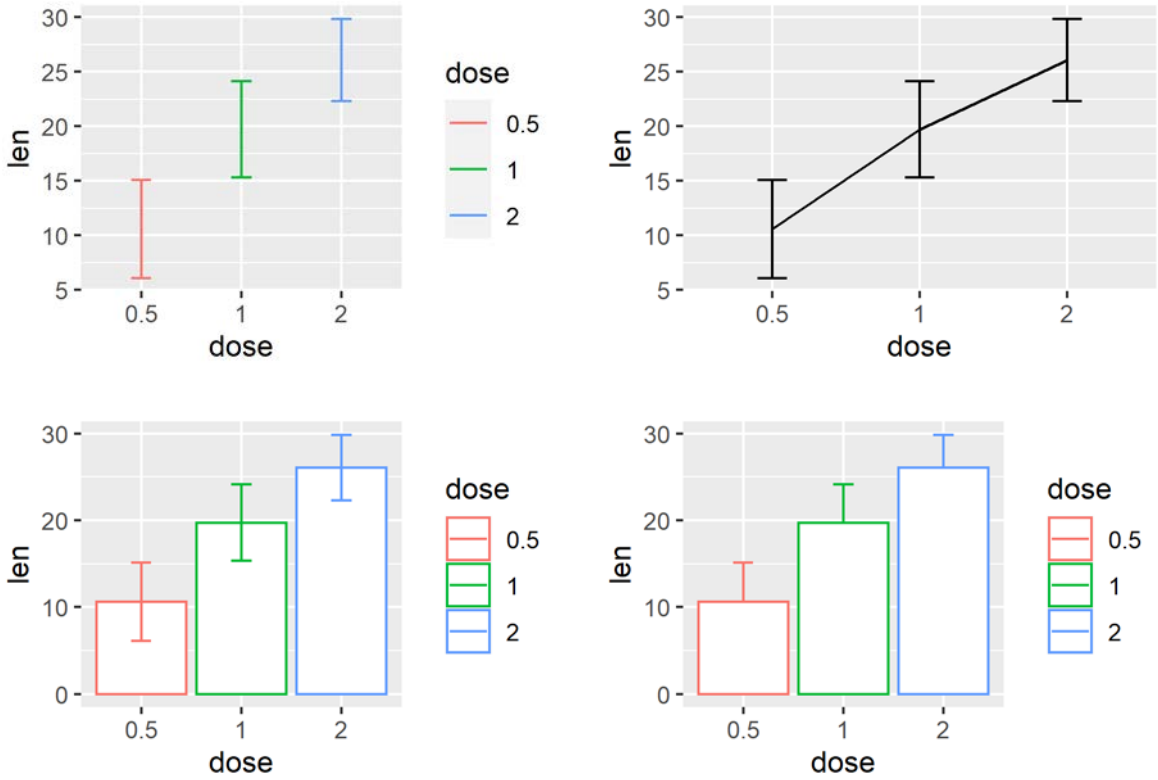
```
R> f <- ggplot(df2, aes(x = dose, y = len,
                        ymin = len-sd, ymax = len+sd))

# Error bars colored by groups
R> f + geom_errorbar(aes(color = dose), width = 0.2)

# Combine with line plots
R> f + geom_line(aes(group = 1)) +
     geom_errorbar(width = 0.2)

# Combine with bar plots, color by groups
R> f + geom_bar(aes(color = dose), stat = "identity", fill = "white") +
     geom_errorbar(aes(color = dose), width = 0.2)

# Keep only upper error bars
R> f + geom_bar(aes(color = dose), stat = "identity", fill = "white") +
     geom_errorbar(aes(color = dose, ymin = len), width = 0.2)
```



รูปที่ 11-43 Error bar plots

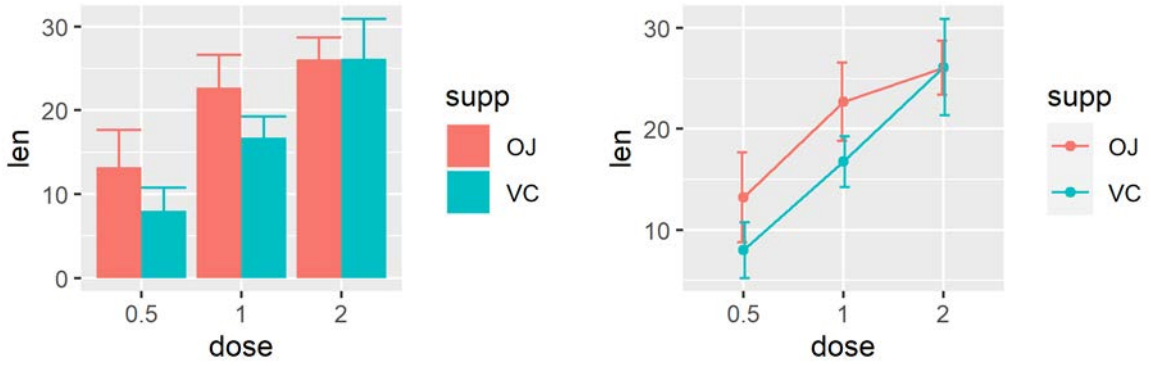
รูปซ้ายบนเป็นการพล็อต Error bars แกน Y แสดงตัวแปร len ส่วนแกน X เป็นตัวแปร dose แสดงสีตามข้อมูลตัวแปร dose รูปขวาบนเป็นการพล็อต Error bars พร้อมกับกราฟเส้นตรงเชื่อมระหว่างค่าเฉลี่ย สองรูปล่างเป็นการพล็อต Error bars พร้อมกับกราฟแท่งและแสดงสีตามข้อมูลตัวแปร dose

แสดงกราฟ Error bars ด้วยตัวแปรหลายกลุ่ม (Multiple groups) โดยแกน X แสดงตัวแปร dose ส่วนตัวแปร supp แสดงสีข้อมูลที่แตกต่างกันแต่ละกลุ่ม และการพล็อตร่วมกับกราฟแท่ง กราฟเส้นและจุด ได้ดังนี้

```
R> f <- ggplot(df3, aes(x = dose, y = len,
                        ymin = len-sd, ymax = len+sd))

# Bar plots with error bars
R> f + geom_bar(aes(fill = supp), stat = "identity",
                position = "dodge") +
     geom_errorbar(aes(color = supp), position = "dodge")

# Line plots with error bars
R> f + geom_line(aes(group = supp, color = supp)) +
     geom_point(aes(color = supp))+
     geom_errorbar(aes(color = supp), width = 0.2,
                   position = position_dodge(0.05))
```



รูปที่ 11-44 Error bar plots with multiple groups

รูปแรกเป็นการพล็อต Error bars พร้อมกับกราฟแท่งและแสดงสีตามข้อมูลตัวแปร dose รูปที่สองเป็นการพล็อต Error bars พร้อมกับกราฟเส้นตรงเชื่อมระหว่างค่าเฉลี่ยและแสดงสีตามข้อมูลตัวแปร dose

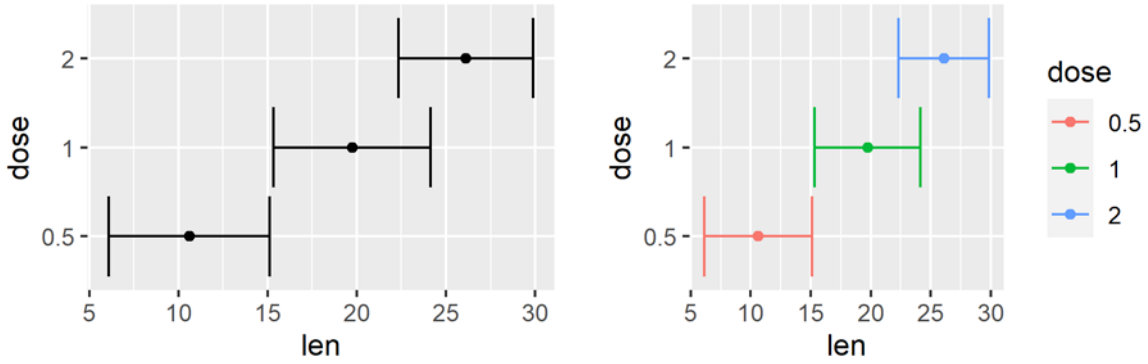
11.7.3 การพล็อต Horizontal Error Bars

การพล็อตกราฟ Error bars ในแนวนอน ใช้คำสั่ง **geom_errorbarh()** โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, linetype, size** และ **height** ตัวอย่างการพล็อตกราฟ Horizontal error bars แสดงได้ดังนี้

```
R> f <- ggplot(df2, aes(x = len, y = dose ,
                        xmin = len-sd, xmax = len+sd))
```

```
# Horizontal error bars
R> f + geom_errorbarh() + geom_point()

# Horizontal error bars by groups
R> f + geom_errorbarh(aes(color = dose)) +
     geom_point(aes(color = dose))
```



รูปที่ 11-45 Horizontal error bar plots by groups

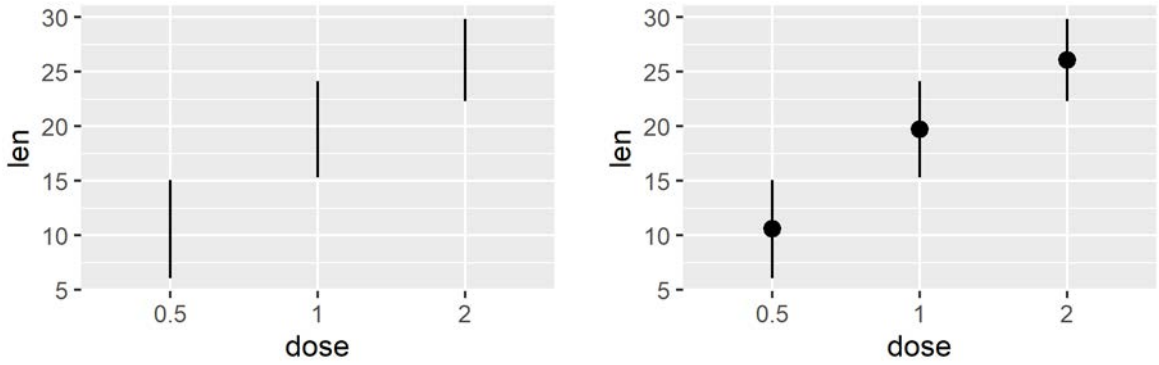
รูปแรกเป็นการพล็อต Horizontal error bars แกน X แสดงตัวแปร len ส่วนแกน Y เป็นตัวแปร dose รูปที่สองเป็นการพล็อต Horizontal error bars พร้อมกับแสดงสีตามข้อมูลตัวแปร dose

11.7.4 การพล็อต Line Range และ Point Range

Line range และ Point range เป็นการแสดงช่วงกว้าง (Interval) ของข้อมูล โดย Line range แสดงเป็นเส้นในแนวตั้ง (Vertical line) ส่วน Point range เป็นเส้นในแนวตั้ง (Vertical line) พร้อมกับจุดที่บริเวณกึ่งกลางของช่วงข้อมูล การพล็อตกราฟ Line range และ point range ใช้คำสั่ง **geom_linerange()** และ **geom_pointrange()** ตามลำดับ โดยมีพารามิเตอร์ที่กำหนดค่าเพิ่มเติมได้แก่ **alpha, color, linetype, size, shape** และ **fill** ตัวอย่างการพล็อตกราฟ Line range และ Point range แสดงได้ดังนี้

```
R> f <- ggplot(df2, aes(x = dose, y = len,
                        ymin = len-sd, ymax = len+sd))
# Line range
R> f + geom_linerange()

# Point range
R> f + geom_pointrange()
```



รูปที่ 11-46 Line range and point range plots

รูปแรก Line range plots แกน X เป็นตัวแปร dose ส่วนแกน Y เป็นตัวแปร len รูปที่สองแสดง Point range plots ซึ่งแสดงจุดที่บริเวณกึ่งกลางของช่วงข้อมูล

11.7.5 การพล็อตจุดแสดงความหนาแน่น (Dot Plots) และ Error Bars

การพล็อตจุดแสดงความหนาแน่น (Dot plots) และ Error bars สามารถใช้คำสั่งที่กล่าวถึงในหัวข้อก่อนหน้านี้ผสมผสานกันได้ ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูลที่ดัดแปลงจาก ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

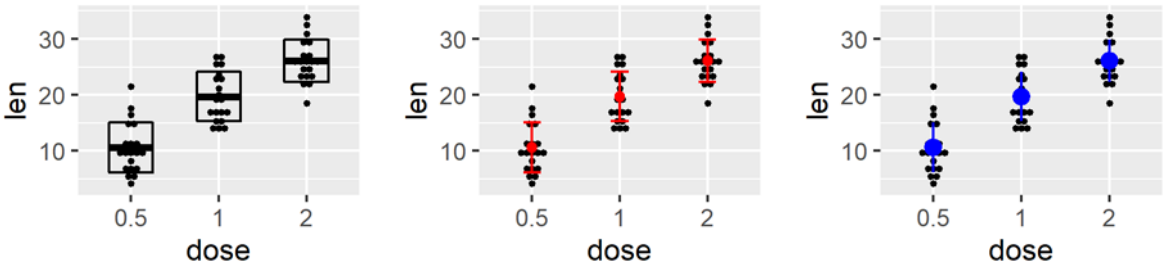
```
R> g <- ggplot(df, aes(x = dose, y = len)) +
     geom_dotplot(binaxis = 'y', stackdir = 'center')
```

สามารถพล็อตอีกวิธีด้วยคำสั่ง `stat_summary()` โดยวิธีการนี้จะทำการคำนวณค่าเฉลี่ย (Mean) และค่าส่วนเบี่ยงเบนมาตรฐาน (Standard deviation) ให้อัตโนมัติโดยใช้พารามิเตอร์ `fun.data = "mean_sdl"` ดังนี้

```
# use geom = "crossbar"
R> g + stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                    geom = "crossbar", width = 0.5)

# Use geom = "errorbar"
R> g + stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                    geom = "errorbar", color = "red", width = 0.2) +
     stat_summary(fun = mean, geom = "point", color = "red")

# Use geom = "pointrange"
R> g + stat_summary(fun.data = "mean_sdl", fun.args = list(mult = 1),
                    geom = "pointrange", color = "blue")
```



รูปที่ 11-47 Dot plots and error bar plots

รูปแรกแสดง Dot plots พร้อมกับ Cross bars รูปที่สองแสดง Dot plots พร้อมกับ Error bars รูปสุดท้ายแสดง Dot plots พร้อมกับ Point range

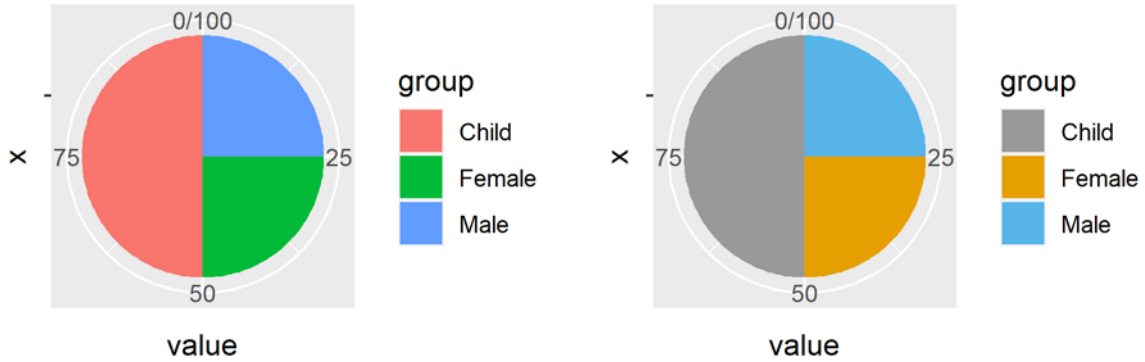
11.8 การพล็อตกราฟวงกลม (Pie Charts)

การพล็อตกราฟวงกลม (Pie charts) ใช้คำสั่ง `coord_polar()` กราฟวงกลมเหมือนกับการพล็อตกราฟแท่ง (Bar plots) แต่เปลี่ยนไปใช้ Polar coordinates แทน ตัวอย่างการพล็อตกราฟวงกลม แสดงได้ดังนี้

```
R> df <- data.frame(
      group = c("Male", "Female", "Child"),
      value = c(25, 25, 50))

# default plots
R> p <- ggplot(df, aes(x = "", y = value, fill = group)) +
      geom_bar(width = 1, stat = "identity") +
      coord_polar("y", start = 0)
R> p

# Use custom fill color palettes
R> p + scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```



รูปที่ 11-48 Pie charts

รูปแรกแสดงกราฟวงกลมแบ่งกลุ่มข้อมูลที่แสดงผลตามตัวแปร group รูปที่สองเป็นข้อมูลเดียวกันแต่กำหนดสีแบบ Manual

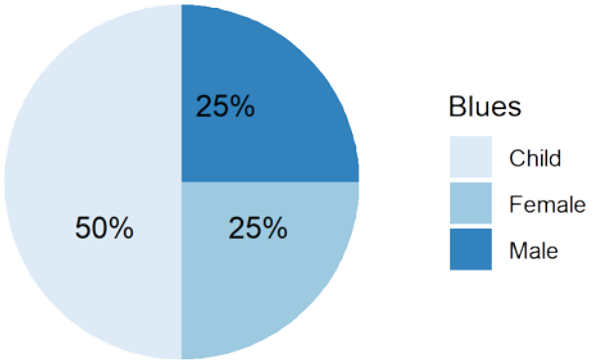
สามารถแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน `scale_x_discrete()` โดยผ่านพารามิเตอร์ `limits` ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ เช่นเดียวกับหัวข้อก่อนหน้านี้

การสร้างธีมขึ้นมาใช้เอง อาจจะสร้างได้โดยการปรับจากธีมเดิมด้วยคำสั่ง `theme()` ได้ดังนี้

```
R> blank_theme <- theme_minimal()+
     theme(
       axis.title.x = element_blank(),
       axis.title.y = element_blank(),
       axis.text.x  = element_blank(),
       panel.border = element_blank(),
       panel.grid = element_blank(),
       axis.ticks = element_blank(),
       plot.title = element_text(size = 14, face = "bold")
     )
```

การพล็อตกราฟวงกลม (Pie charts) โดยใช้ธีมที่สร้างมา และให้แสดงตัวเลขเป็นร้อยละโดยใช้แพ็กเกจ scales และกำหนดให้พารามิเตอร์ `label = percent()` ดังนี้

```
# Apply blank theme
R> require(scales)
R> p + scale_fill_brewer("Blues") +
     blank_theme +
     geom_text(aes(y = value/3 + c(0, cumsum(value)[-length(value)]),
                   label = percent(value/100)), size = 4)
```



รูปที่ 11-49 Pie chart with a customized theme

รูปด้านบนแสดงกราฟวงกลมแบ่งกลุ่มข้อมูลที่แสดงผลตามตัวแปร group ด้วยการใช้จานสี (Palette) พร้อมกับแสดงข้อมูลเป็นเปอร์เซ็นต์

การเปลี่ยนสีกราฟ Pie charts ตามกลุ่มข้อมูล ใช้คำสั่ง `scale_color_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_color_brewer()` และ `scale_color_grey()` หรือเติมสี (Fill) ด้วยคำสั่ง `scale_fill_manual()` หรือใช้สีจากจานสี (palettes) ด้วยคำสั่ง `scale_fill_brewer()`

และ `scale_fill_grey()` เหมือนกับหัวข้อที่กล่าวมาข้างต้น (ดูรายละเอียดการใช้สีเพิ่มเติมจาก https://colorbrewer2.org)

11.9 สรุปท้ายบท

บทนี้กล่าวถึงการพล็อตกราฟที่มีแกน X เป็นตัวแปรแบ่งกลุ่ม/ตัวแปรไม่ต่อเนื่อง (Discrete variable) และแกน Y เป็นตัวแปรต่อเนื่อง (Continuous variable) กราฟที่ใช้ในการพล็อตประกอบด้วยกราฟการกระจายตัวแบบ Box plots, กราฟความหนาแน่นแบบ Violins plots, กราฟจุดแสดงความหนาแน่น (Dot plots), กราฟที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter plots), กราฟเส้น (Line plots), กราฟแท่ง (Bar plots), การแสดงค่าความคลาดเคลื่อน (Visualizing errors) ได้แก่ Cross bars, Error bars, Horizontal error bars, Line range, Point range และกราฟวงกลม (Pie Charts)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`ggplot()`	คำสั่งในการพล็อต
`geom_boxplot()`	Geometry สำหรับพล็อต Box plots
`geom_violin()`	Geometry สำหรับพล็อต Violins plots
`geom_dotplot()`	Geometry สำหรับพล็อตกราฟจุดแสดงความหนาแน่น (Dot plots)
`geom_jitter()`	Geometry สำหรับพล็อตกราฟที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter plots)
`geom_line()`	Geometry สำหรับพล็อตกราฟเส้น
`geom_bar()`	Geometry สำหรับพล็อตกราฟแท่ง
`geom_text()`	Geometry สำหรับเพิ่มข้อความ
`coord_flip()`	คำสั่งในการสลับแกน X และแกน Y
`scale_x_discrete()`	คำสั่งในการกำหนดลำดับในการพล็อตตามแกน X
`stat_summary()`	คำสั่งในการแสดงค่าเฉลี่ยทางคณิตศาสตร์ เช่น Mean, Median
`geom_crossbar()`	Geometry สำหรับพล็อตกราฟ cross bar
`geom_errorbar()`	Geometry สำหรับพล็อตกราฟ error bar
`geom_errorbarh()`	Geometry สำหรับพล็อตกราฟ error bar ในแนวนอน
`geom_linerange()`	Geometry สำหรับพล็อตกราฟ error เป็นเส้นแนวตั้ง
`geom_pointrange()`	Geometry สำหรับพล็อตกราฟ error เป็นเส้นแนวตั้งพร้อมกับจุดแสดงค่ากลาง

คำสั่ง	คำอธิบาย
`scale_color_manual()`	คำสั่งในการกำหนดสีแบบ Manual
`scale_color_brewer()`	คำสั่งในการกำหนดสีด้วยจานสี (Palettes)
`scale_color_grey()`	คำสั่งในการกำหนดสีด้วยจานสีเทา
`scale_fill_manual()`	คำสั่งในการเติมสีแบบ Manual
`scale_fill_brewer()`	คำสั่งในการเติมสีด้วยจานสี (Palettes)
`scale_fill_grey()`	คำสั่งในการเติมสีด้วยจานสีเทา
`scale_shape_manual()`	คำสั่งในการกำหนดรูปร่างแบบ Manual
`scale_size_manual()`	คำสั่งในการกำหนดขนาดแบบ Manual

11.10 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า mtcars สร้างกราฟต่าง ๆ ดังต่อไปนี้

   1.1 สร้างกราฟการกระจายตัวแบบ Box Plot สำหรับตัวแปร X (กลุ่มของเกียร์ gear) และ Y (mpg ระยะทางต่อแกลลอน)

   1.2 จากข้อ 1.1 เพิ่มคำสั่ง `scale_fill_manual()` เพื่อเปลี่ยนสีของ Box Plots ตามกลุ่มของ gear และใช้คำอธิบาย (Legend) ที่แสดงสีของแต่ละกลุ่ม

   1.3 จากข้อ 1.2 เพิ่มค่าเฉลี่ยของ mpg บน Box Plots โดยใช้คำสั่ง `stat_summary()`

   1.4 จากข้อ 1.3 ใช้คำสั่ง `coord_flip()` เพื่อหมุนแกน X และ Y

   1.5 สร้างกราฟความหนาแน่นแบบไวโอลิน (Violin Plots) สำหรับตัวแปร X (กลุ่มของเกียร์ gear) และ Y (mpg ระยะทางต่อแกลลอน) และเปลี่ยนสีของ Box Plots ตามกลุ่มของ gear และใช้คำอธิบาย (Legend) ที่แสดงสีของแต่ละกลุ่ม

   1.6 สร้างกราฟ Dotplot สำหรับตัวแปร gear และ mpg โดยแสดงจุดข้อมูลที่ซ้อนกันในแต่ละกลุ่ม

   1.7 จากข้อ 1.6 ใช้คำสั่ง `geom_text()` เพื่อเพิ่มข้อความแสดงค่าเฉลี่ยและค่ามัธยฐานของแต่ละกลุ่มในตำแหน่งที่เหมาะสม

   1.8 ใช้คำสั่ง `geom_bar()` เพื่อสร้าง Bar Chart ของจำนวนรถในแต่ละ gear พร้อมปรับความกว้างและสีของแท่งกราฟให้สวยงาม

2. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า iris สร้างกราฟต่าง ๆ ดังต่อไปนี้

   2.1 สร้างกราฟความหนาแน่นแบบไวโอลิน (Violin Plot) สำหรับ Species และ Sepal.Length

   2.2 จากข้อ 2.1 เพิ่มกราฟแสดงจุดที่ซ้อนทับกันบน Violin Plot และแสดงการกระจายตัวของข้อมูลโดยใช้สีที่แตกต่างกันในแต่ละกลุ่ม

   2.3 จากข้อ 2.2 ใช้คำสั่ง `scale_fill_brewer()` เพื่อเพิ่มสีใน Violin Plot โดยเลือกโทนสีที่เหมาะสมสำหรับการเปรียบเทียบระหว่างกลุ่ม

   2.4 จากข้อ 2.3 สร้างกราฟการกระจายตัวแบบ Box Plot ซ้อนทับบน Violin Plot

3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ gapminder ชื่อว่า gapminder โดยเลือกตัวอย่างมา 12 ประเทศ ได้แก่ “Thailand”, “Singapore”, “Japan”, “United States”, “Argentina”, “Mexico”, “Finland”, “Germany”, “United Kingdom”, “Egypt”, “Morocco” และ “South Africa” สร้าง Line Plot ของปี (year) และข้อมูล GDP ต่อหัว (gdpPercap) โดยแยกสีเป็นของแต่ละประเทศ และเพิ่มจุดบนเส้น

บทที่ 12
พารามิเตอร์ต่าง ๆ ในกราฟ (Graphical Parameters)

บทนี้จะกล่าวถึงกราฟิกพื้นฐาน (Graphical primatives) ในการพล็อต ได้แก่ รูปหลายเหลี่ยม (Polygon), เส้นต่อเนื่อง (Path), กราฟ Ribbon, บางส่วนของเส้น (Segment), เส้นโค้ง, รูปสี่เหลี่ยม รวมทั้งพารามิเตอร์ที่ใช้แสดงค่าต่าง ๆ ในกราฟ ได้แก่ ชื่อเรื่อง (Main title) ชื่อแกน (Axis labels) สัญลักษณ์ (Legend) ตำแหน่งและรูปแบบต่าง ๆ ของสัญลักษณ์ (Legend position and appearance) สี (Color) รูปร่างจุด สี และขนาด (Point shapes, colors and size) ชนิดเส้น (Line types) การกำหนดขอบเขตแกน (Axis limits) การแปลงแกนด้วย log และ sqrt (Axis transformations: log and sqrt) การกำหนดแกนในหน่วยวัน (Date Axis) การกำหนดรูปแบบการแสดงผลบนแกน (Axis ticks : customize tick marks and labels, reorder and select items) ธีมและสีพื้น (Themes and background colors) ข้อความ (Text annotations) การเพิ่มเส้นตรงในกราฟ (Add straight lines to a plot: horizontal, vertical and regression lines) การหมุนกราฟ (Rotate a plot: flip and reverse) การพล็อตกราฟย่อย (Facets) การจัดวางตำแหน่งองค์ประกอบในกราฟ (Position adjustments) และ Coordinate systems

12.1 กราฟิกพื้นฐาน (Graphical Primatives)

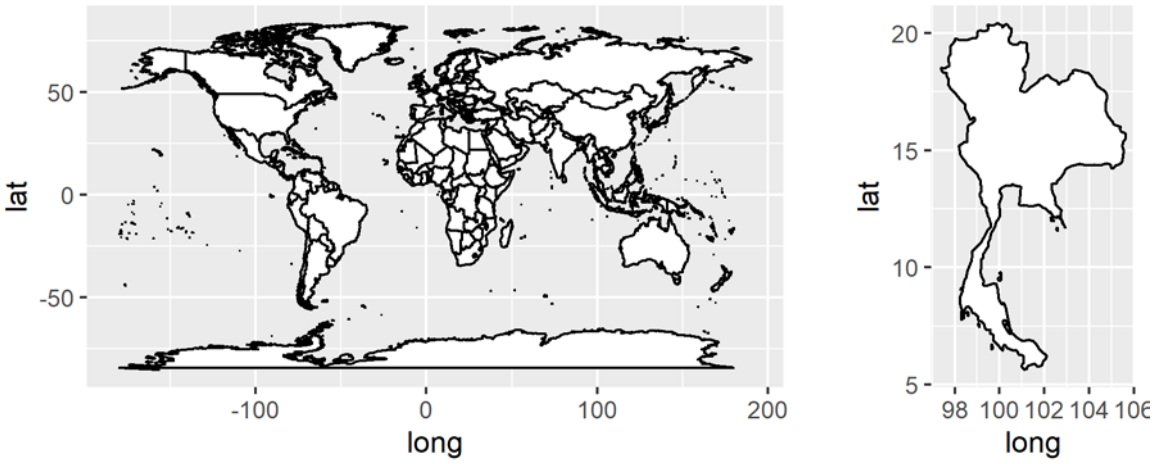
หัวข้อนี้จะกล่าวถึงกราฟิกพื้นฐาน (Graphical Primatives) ในการพล็อต Geometry ที่สามารถพล็อตได้ประกอบด้วย

- **geom_polygon()** สำหรับพล็อตรูปหลายเหลี่ยม (Polygon)
- **geom_path()** สำหรับพล็อตเส้นต่อเนื่อง (Path)
- **geom_ribbon()** สำหรับพล็อตกราฟ Ribbon
- **geom_segment()** สำหรับพล็อตบางส่วนของเส้น (Segment)
- **geom_curve()** สำหรับพล็อตเส้นโค้ง
- **geom_rect()** สำหรับพล็อตรูปสี่เหลี่ยม

การพล็อตแผนที่โลกและแผนที่ประเทศไทยด้วยคำสั่ง **geom_polygon()** แสดงได้ดังนี้

```
R> require(maps)
R> world <- map_data("world")
R> ggplot(world, aes(x = long, y = lat, group = group)) +
     geom_polygon(fill = "white", colour = "black")

R> thai <- map_data("world", region = "Thai")
R> ggplot(thai, aes(x = long, y = lat, group = group)) +
     geom_polygon(fill = "white", color = "black")
```



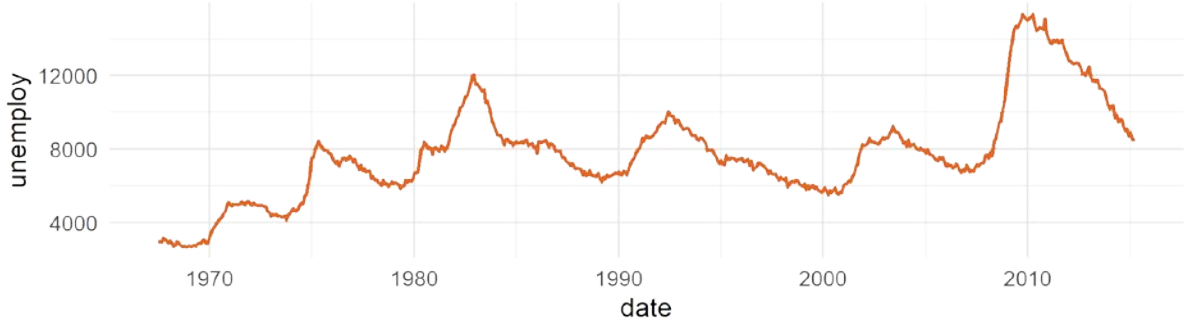
รูปที่ 12-1 Polygon plots

รูปแรกแสดงการพล็อตแผนที่โลกด้วยรูปหลายเหลี่ยม (Polygon) แกน X เป็นลองจิจูด (Longitude) แกน Y เป็นละติจูด (Latitude) รูปที่สองแสดงการพล็อตแผนที่ประเทศไทย

การพล็อตกราฟ Time series โดยใช้คำสั่ง `geom_path()` กำหนดให้พารามิเตอร์ `size` และ `color` ดังนี้

```
R> h <- ggplot(economics, aes(date, unemploy))

# Path
R> h + geom_path(size = 0.8, color = "#E46726") +
  theme_minimal()
```



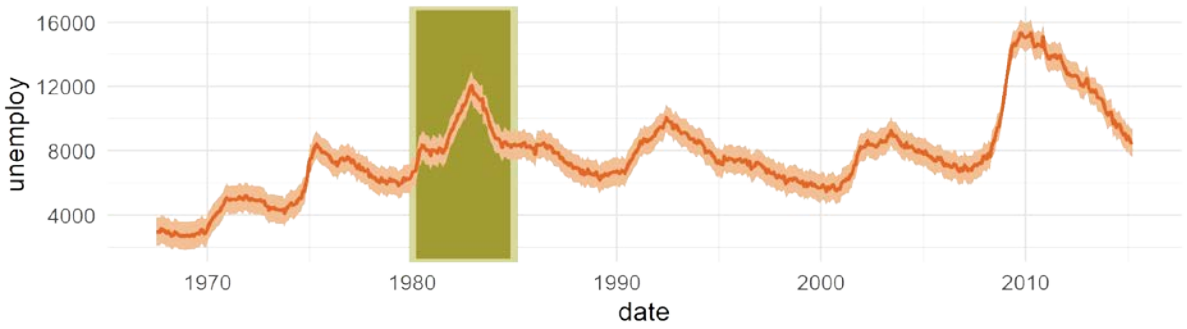
รูปที่ 12-2 Time series plots with path

รูปด้านบนเป็นการพล็อตอนุกรมเวลา (Time series) แกน X เป็นปีตั้งแต่ประมาณปี 1967 ถึงปี 2015 แกน Y เป็นจำนวนคนว่างงาน (undemploy)

การพล็อตกราฟ Time series โดยใช้คำสั่ง `geom_ rectangle()`, `geom_ribbon()` และ `geom_path()` และกำหนดให้พารามิเตอร์ `xmin`, `xmax`, `ymin`, `ymax`, `fill`, `size` และ `color` ดังนี้

```
# Combine path, ribbon and rectangle
```

```
R> h + geom_rect(aes(xmin = as.Date("1980-01-01"), ymin = -Inf,
                     xmax = as.Date("1985-01-01"), ymax = Inf),
                 fill = "#A29B32", color = "#D8DA9E", size = 1.5) +
     geom_ribbon(aes(ymin = unemploy-900, ymax = unemploy+900),
                 fill = "#F3BF94") +
     geom_path(size = 0.8, color = "#E46726") +
     theme_minimal()
```



รูปที่ 12-3 Time series plots with path, ribbon and rectangle

รูปด้านบนเป็นการพล็อตอนุกรมเวลา (Time series) เหมือนรูปก่อนหน้านี้ เพิ่มเติมคือมีการพล็อตแถบ (Ribbon) และ Path

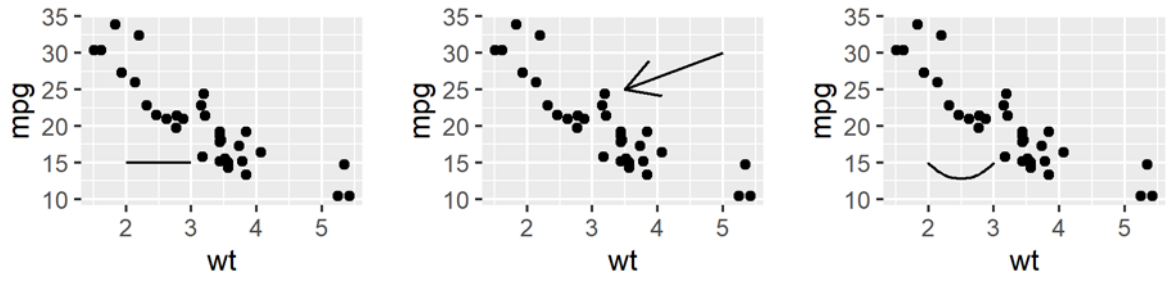
การเพิ่ม Line segments เข้าไปในกราฟที่พล็อตไว้ก่อนแล้ว ใช้คำสั่ง `geom_segment()` หรือ `geom_curve()` ทำได้โดยระบุรายละเอียดที่ต้องการพล็อตในฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `x, y, xend,` หรือ `yend` ดังนี้

```
# Create a scatter plots
R> i <- ggplot(mtcars, aes(wt, mpg)) + geom_point()

# Add segment
R> i + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15))

# Add arrow
R> require(grid)
R> i + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
                    arrow = arrow(length = unit(0.5, "cm")))

# Add curves
R> i + geom_curve(aes(x = 2, y = 15, xend = 3, yend = 15))
```



รูปที่ 12-4 Scatter plots with segment and curve

ทั้ง 3 รูปด้านบนเป็น Scatter plots โดยมีการพล็อตบางส่วนของเส้นตรง หัวลูกศร และเส้นโค้งเพิ่มเข้าไปในกราฟดังกล่าว

12.2 ชื่อเรื่อง (Main Title) ชื่อแกน (Axis Labels) และสัญลักษณ์ (Legend)

คำสั่งที่ใช้ในการพล็อตชื่อเรื่อง (Main title) ชื่อแกน (Axis labels) และสัญลักษณ์ (Legend Title) ประกอบด้วย

- `ggtitle()` สำหรับพล็อตชื่อเรื่อง (Main title)
- `xlab()` สำหรับพล็อตชื่อแกน X
- `ylab()` สำหรับพล็อตชื่อแกน Y
- `labs()` สำหรับพล็อต Main title ชื่อแกน X และชื่อแกน Y

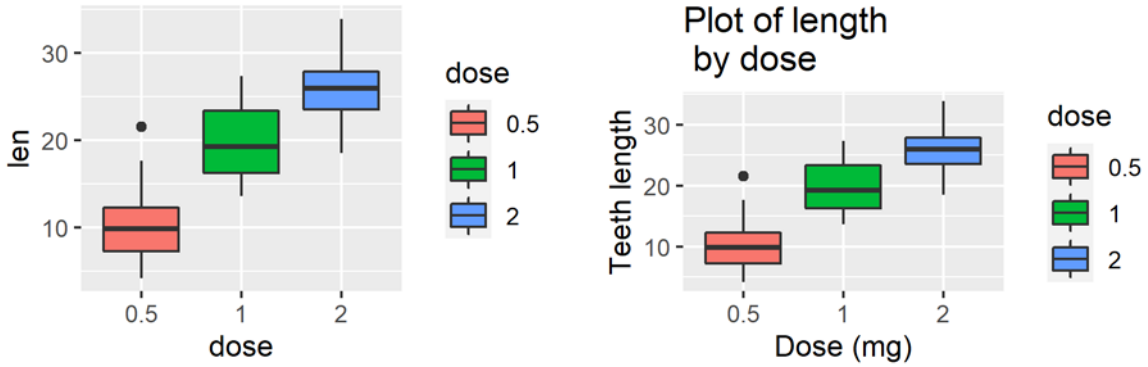
ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base R ดังนี้

```
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len, fill = dose)) +
     geom_boxplot()
```

การพล็อตชื่อเรื่อง (Main title) ชื่อแกน (Axis labels) ใช้คำสั่ง `ggtille(), xlab(), ylab()` และ `labs()` ดังนี้

```
# Default plots
R> print(p)

# Change title and axis labels
R> p1 <- p + labs(title = "Plot of length \n by dose",
              x = "Dose (mg)", y = "Teeth length")
R> p1
# Alternative
R> p + ggtitle("Plot of length \n by dose") +
  xlab("Dose (mg)") + ylab("Teeth length")
```



รูปที่ 12-5 Box plots with main title, and axis labels

รูปแรกเป็น Box plots ที่ไม่มีชื่อเรื่อง (Main title) และใช้ชื่อตัวแปรแสดงชื่อแกน X และแกน Y รูปที่สองเพิ่มชื่อเรื่อง (Main title) และกำหนดชื่อให้แกน X และแกน Y

สามารถแสดงขนาด สี ฟอนต์ของชื่อเรื่อง ชื่อแกน X และชื่อแกน Y ได้ด้วยคำสั่ง **theme()** โดยชื่อเรื่อง (Main title) ใช้พารามิเตอร์ **plot.title** ชื่อแกน X ใช้พารามิเตอร์ **plot.title.x** และชื่อแกน Y ใช้พารามิเตอร์ **plot.title.y** การแสดงขนาด สี ฟอนต์ใช้คำสั่ง **element_text()** พร้อมกับระบุพารามิเตอร์ **color, size** และ **face** แต่ถ้าไม่ต้องการแสดง Labels ใช้คำสั่ง **element_blank()** ดังนี้

```
# Change the appearance of labels
# Values for face are one of "plain", "italic", "bold" and "bold.italic"
R> p1 + theme(
     plot.title = element_text(color = "red", size = 12, face = "bold.italic"),
     axis.title.x = element_text(color = "blue", size = 12, face = "bold"),
     axis.title.y = element_text(color = "#993333", size = 12, face = "bold")
   )

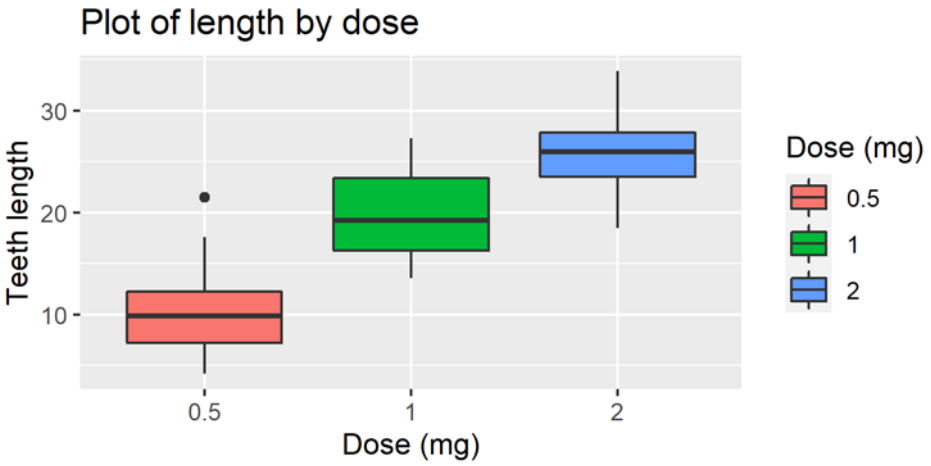
# Hide labels
R> p1 + theme(plot.title = element_blank(),
              axis.title.x = element_blank(),
              axis.title.y = element_blank())
```

รูปที่ 12-6 Box plots with colors, sizes and fonts

รูปแรกเป็น Box plots ที่มีการกำหนดขนาด สี ฟอนต์ของชื่อเรื่อง (Main title) ชื่อแกน X และแกน Y รูปที่สองไม่แสดงชื่อเรื่อง (Main title) หรือชื่อแกน X และแกน Y

การแสดงชื่อหัวข้อสัญลักษณ์ (Legend title) ใช้คำสั่ง **labs()** และพารามิเตอร์ **fill** ดังนี้

```
R> p + labs(title = "Plot of length by dose", x = "Dose (mg)", y = "Teeth length",
              fill = "Dose (mg)")
```



รูปที่ 12-7 Box plots with legend titles

รูปด้านบนเป็น Box plots ที่มีการกำหนดชื่อหัวข้อสัญลักษณ์ (Legend title) เพิ่มเติมนอกเหนือจากชื่อเรื่อง (Main title) ชื่อแกน X และแกน Y

12.3 ตำแหน่งและรูปแบบต่าง ๆ ของสัญลักษณ์ (Legend Position and Appearance)

ตำแหน่งและรูปแบบต่าง ๆ ของสัญลักษณ์ (Legend Position and Appearance) ใช้ในการปรับแต่งกราฟให้อยู่ในรูปแบบที่ผู้ใช้ต้องการ ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

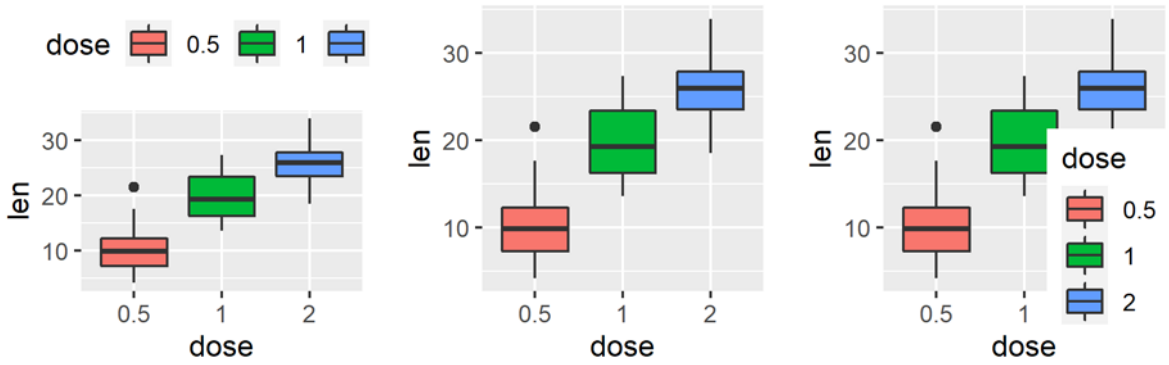
```
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len, fill = dose)) +
     geom_boxplot()
```

การเปลี่ยนตำแหน่งสัญลักษณ์ (Legend position) ใช้คำสั่ง `theme()` ตามด้วยพารามิเตอร์ `legend.position` ให้อยู่ด้านซ้าย ขวา บน ล่าง หรือไม่แสดงสัญลักษณ์ ด้วยพารามิเตอร์ `"left", "right", "top", "bottom"` หรือ `"none"` ตามลำดับ ดังนี้

```
# Change legend position: "left","right", "top", "bottom", "none"
R> p + theme(legend.position = "top")

# Remove legends
R> p + theme(legend.position = "none")

# Legend position as numeric vector c(x, y)
R> p + theme(legend.position = c(0.8, 0.2))
```



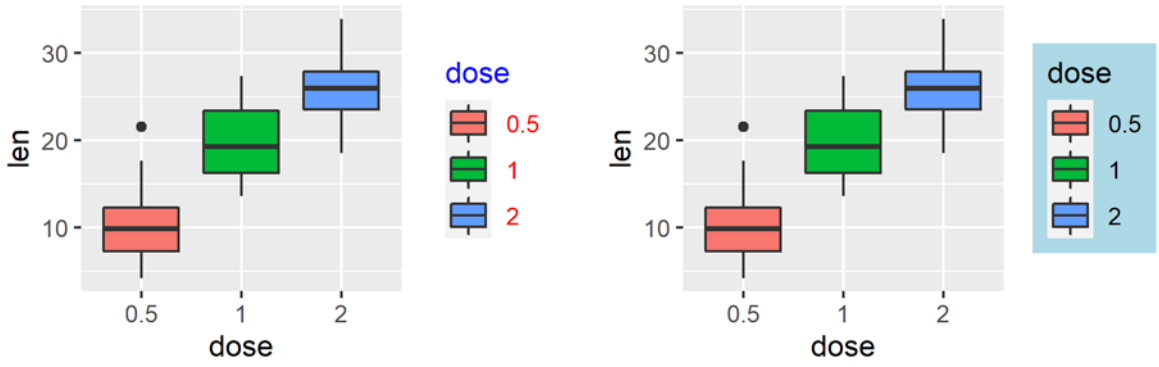
รูปที่ 12-8 Box plots with legend positions

รูปที่หนึ่งเป็นการวางตำแหน่งสัญลักษณ์ (Legend position) ให้อยู่ด้านบน รูปที่สองไม่แสดงสัญลักษณ์ (Legend) รูปสุดท้ายเป็นการวางตำแหน่งสัญลักษณ์ (Legend position) โดยการกำหนด coordinate X, Y ด้วยเวกเตอร์ `c(X, Y)` ตัวแปร X และตัวแปร Y มีค่าระหว่าง 0 ถึง 1 ค่า `c(0, 0)` หมายถึง ตำแหน่งล่างซ้าย `c(1, 1)` หมายถึง ตำแหน่งบนขวา

การเปลี่ยนรูปแบบของสัญลักษณ์ (Legend appearance) ใช้คำสั่ง `theme()` โดยระบุพารามิเตอร์ `legend.title` และ `legend.text` ให้เท่ากับ `element_text()` พร้อมกับระบุพารามิเตอร์ `color, size` และ `face` และ `legend.background` ให้เท่ากับ `element_rect()` พร้อมกับระบุพารามิเตอร์ `fill` ดังนี้

```
# Change the appearance of legend title and labels
R> p + theme(legend.title = element_text(color = "blue"),
             legend.text = element_text(color = "red"))

# Change legend box background color
R> p + theme(legend.background = element_rect(fill = "lightblue"))
```



รูปที่ 12-9 Box plots with legend appearance: title, text and background

รูปแรกเป็น Box plots ที่มีการกำหนดสีให้ชื่อหัวข้อสัญลักษณ์ (Legend title) ให้มีสีน้ำเงินและชื่อสัญลักษณ์ (Legend lable) ให้มีสีแดง รูปที่สองเป็นการกำหนดสีพื้น (Legend background) ให้เป็นสีฟ้า

สามารถแสดงค่าที่พล็อตตามลำดับที่ต้องการได้ด้วยฟังก์ชัน `scale_x_discrete()` โดยผ่านพารามิเตอร์ `limits` ซึ่งกำหนดให้เท่ากับเวกเตอร์ตัวแปร X ที่ต้องการพล็อต ตามลำดับ ฟังก์ชัน `scale_fill_discrete()` โดยระบุพารามิเตอร์ `name` และ `labels` เป็นการกำหนดชื่อหัวข้อสัญลักษณ์ (Legend title) และชื่อสัญลักษณ์ (Legend labels) ดังนี้

```
# Change the order of legend items
R> p + scale_x_discrete(limits = c("2", "0.5", "1"))

# Set legend title and labels
R> p + scale_fill_discrete(name = "Dose", labels = c("A", "B", "C"))
```

รูปที่ 12-10 Box plots with legend appearance: order of legend, legend title and labels

รูปแรกเป็น Box plots ที่แกน X แสดงตัวแปร dose เริ่มจาก 2, 0.5 และ 1 ตามลำดับ รูปที่สองเป็น Box plots ที่แกน X แสดงตัวแปร dose เริ่มจาก 0.5, 1 และ 2 ตามลำดับ นอกจากนี้ยังกำหนดชื่อหัวข้อสัญลักษณ์ (Legend title) และชื่อสัญลักษณ์ (Legend labels) เอง

สามารถแสดง Scatter plots ด้วยการควบคุมรายละเอียดในการพล็อต เช่น สี ขนาด รูปร่างของวัตถุที่ต้องการพล็อตพร้อมกัน (Multiple aesthetics หรือ guides) เช่น ต้องการแสดงสีด้วยตัวแปร cyl ต้องการแสดงขนาดด้วยตัวแปร qsec ต้องการแสดงรูปร่างด้วยตัวแปร gear

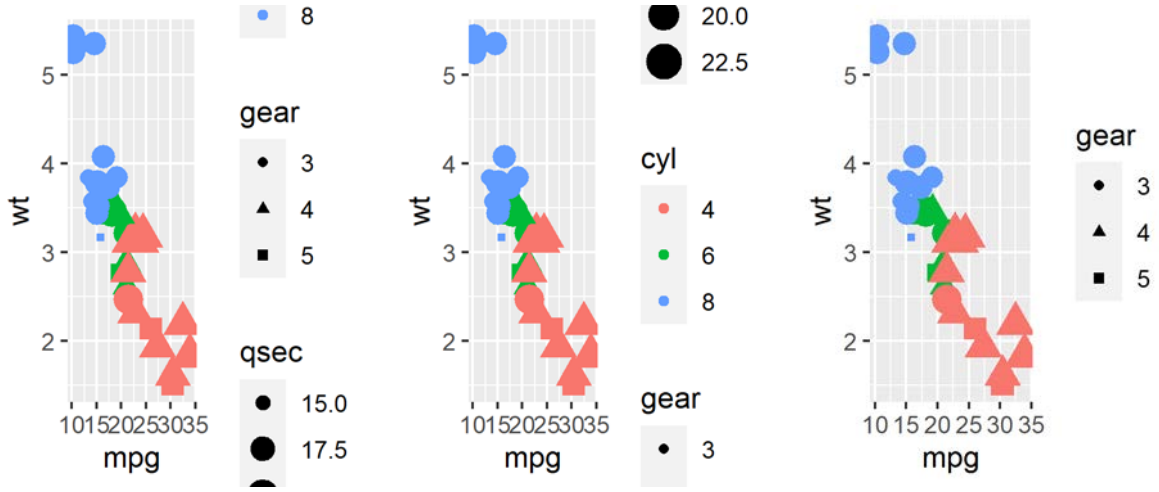
เปลี่ยนลำดับการแสดงหัวข้อสัญลักษณ์ (Legend title) ได้ด้วยคำสั่ง `guides()` โดยระบุพารามิเตอร์ `size, color` และ `shape` ให้เท่ากับ `guide_legend()` ด้วยพารามิเตอร์ `order` หรือถ้าไม่ต้องการแสดงสัญลักษณ์ (Legend title) ให้กำหนดค่า `size, color` และ `shape` ให้เท่ากับ `"none"` ดังนี้

```
# Convert cyl and gear to factor variables
R> mtcars$cyl <- as.factor(mtcars$cyl)
R> mtcars$gear <- as.factor(mtcars$gear)

# Plot with multiple aesthetics
R> p <- ggplot(mtcars, aes(x = mpg, y = wt, color = cyl, size = qsec,
                  shape = gear)) +
     geom_point()
R> p
```

```
# Change the order of guides using guide_legend()
R> p + guides(size = guide_legend(order = 1),
              color = guide_legend(order = 2),
              shape = guide_legend(order = 3))

# Remove a legend for a particular aesthetic (color and size)
R> p + guides(color = "none", size = "none")
```



รูปที่ 12-11 Scatter plots with guides

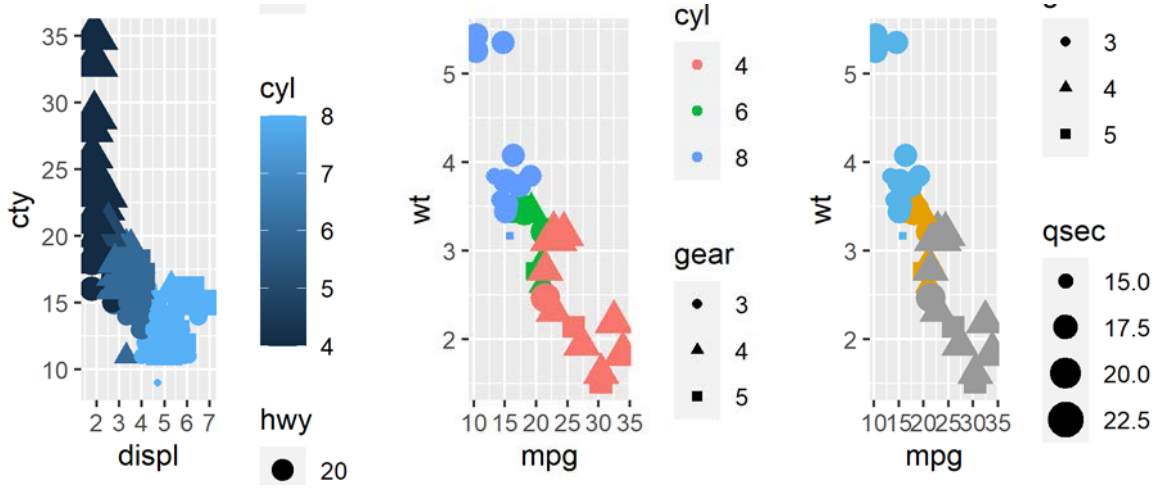
ทั้ง 3 รูปด้านบนเป็นการแสดงตำแหน่งสัญลักษณ์ (Legends) ตามลำดับที่ต้องการ หรือซ่อนการแสดงสัญลักษณ์ (Legends) ตามที่ต้องการ

แสดงสีแบบต่อเนื่อง (Continuous color) ได้ด้วยคำสั่ง **guides()** โดยระบุพารามิเตอร์ **color** ให้เท่ากับ **guide_colorbar()** ถ้าไม่ต้องการแสดงสัญลักษณ์ (Legend title) ใช้คำสั่ง **scale_size(), scale_shape()** หรือ **scale_color_manual()** โดยระบุพารามิเตอร์ **guide** ให้เท่ากับ **"none"** ดังนี้

```
# Case of continuous color: Use guide_colorbar()
R> qplot(data = mpg, x = displ, y = cty, size = hwy, color = cyl, shape = drv) +
     guides(shape = guide_legend(order = 1),
            color = guide_colorbar(order = 2),
             size = guide_legend(order = 3))

# Remove legend for size
R> p + scale_size(guide = "none")

# Remove legend for color
R> p + scale_color_manual(values = c("#999999","#E69F00","#56B4E9"),
                          guide = "none")
```



รูปที่ 12-12 Scatter plots with guides

รูปแรกเป็นการแสดงสีของสัญลักษณ์แบบต่อเนื่อง (Continuous color) รูปที่สองซ่อนการแสดงสัญลักษณ์ (Legends) บางอย่าง เช่น Size ตามที่ต้องการ รูปสุดท้ายแสดงสีแบบ Manual

12.4 สี (Color)

หัวข้อนี้จะกล่าวถึงใช้สีในกราฟ การกำหนดสีทำได้ 2 วิธี (1) การกำหนดชื่อสี เช่น `"blue", "red"` และ (2) การกำหนดโดยใช้เลขฐาน 16 เช่น `"#E69F00"`

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth และ mtcars ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

```
# Convert dose and cyl columns from numeric to factor variables
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> mtcars$cyl <- as.factor(mtcars$cyl)

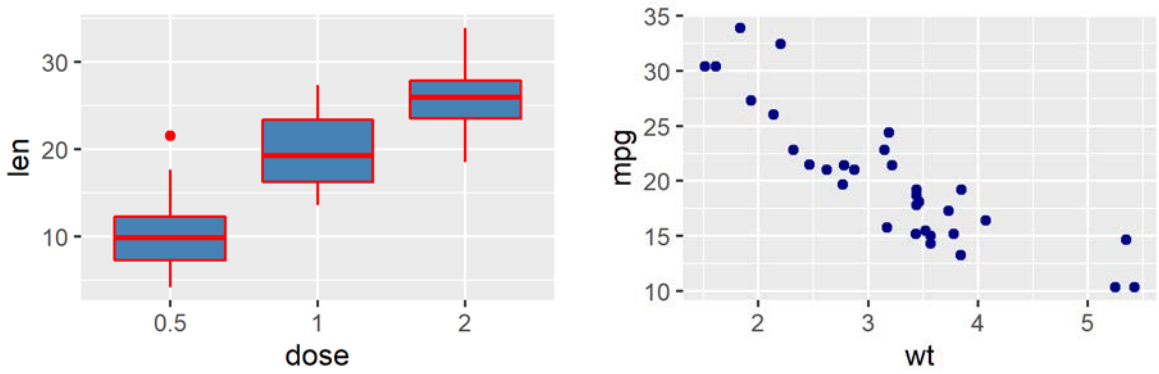
# Box plots
R> bp <- ggplot(ToothGrowth, aes(x = dose, y = len))

# Scatter plots
R> sp <- ggplot(mtcars, aes(x = wt, y = mpg))
```

การกำหนดสีใช้พารามิเตอร์ `color` และ `fill` ดังนี้

```
# Box plots
R> bp + geom_boxplot(fill = "steelblue", color = "red")

# Scatter plots
R> sp + geom_point(color = "darkblue")
```



รูปที่ 12-13 Colors with arguments fill and color

รูปแรกเป็นการแสดงสีกรอบเป็นสีแดงด้วยพารามิเตอร์ `color` และแสดงสี Box เป็นสีน้ำเงินด้วยพารามิเตอร์ `fill` รูปที่สองแสดงสีจุดเป็นสีน้ำเงินด้วยพารามิเตอร์ `color`

การกำหนดสีตามกลุ่มข้อมูล (Colors by groups) ทำได้โดยระบุตัวแปรที่ต้องการกำหนดสีด้วยฟังก์ชัน `aes()` โดยใช้พารามิเตอร์ `color` หรือ `fill` ดังนี้

```
# Box plots
R> bp <- bp + geom_boxplot(aes(fill = dose))
R> bp

# Scatter plots
R> sp <- sp + geom_point(aes(color = cyl))
R> sp
```

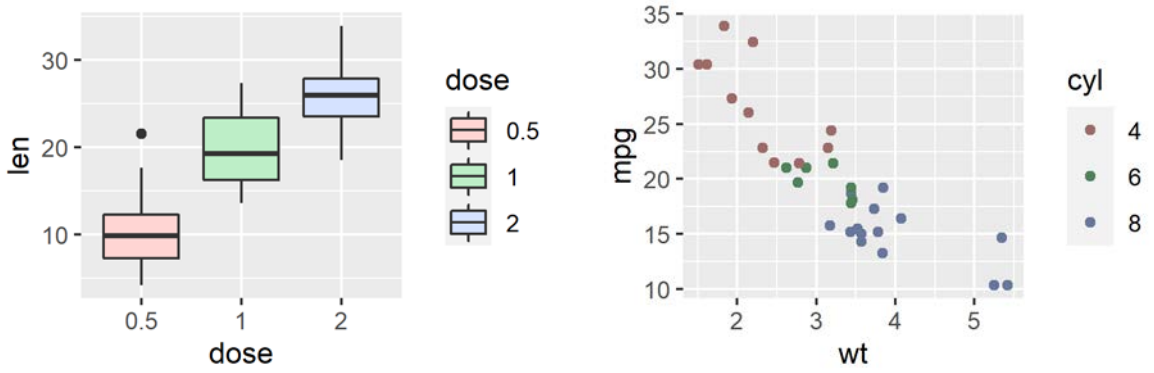
รูปที่ 12-14 Colors by groups

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl

สามารถกำหนดความสว่าง (Lightness, l) และความเข้มของสี (Intensity of color, c) ด้วยคำสั่ง `scale_fill_hue()` และ `scale_color_hue()` โดยระบุพารามิเตอร์ l และ c (ค่าโดยปริยายของ l และ c เท่ากับ 65 และ 100 ตามลำดับ) ดังนี้

```
# Box plots
R> bp + scale_fill_hue(l = 90, c = 35)
```

```
# Scatter plots
R> sp + scale_color_hue(l = 50, c = 35)
```



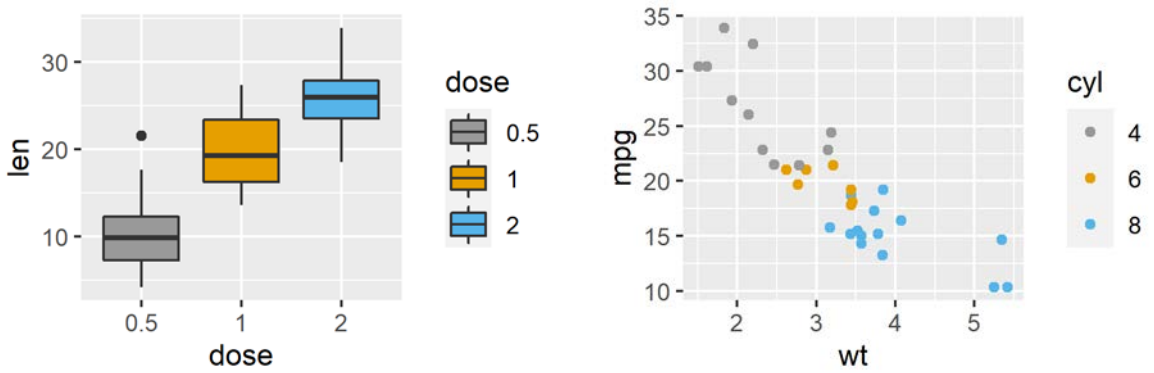
รูปที่ 12-15 Colors with lightness and intensity of color

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl และทั้ง 2 รูปมีการกำหนดความสว่าง (Lightness) และความเข้มของสี (Intensity of color) เพิ่มเติม

การกำหนดสีเองใช้คำสั่ง `scale_fill_manual()` และ `scale_color_manual()` ดังนี้

```
# Box plots
R> bp + scale_fill_manual(values = c("#999999", "#E69F00", "#56B4E9"))

# Scatter plots
R> sp + scale_color_manual(values = c("#999999", "#E69F00", "#56B4E9"))
```



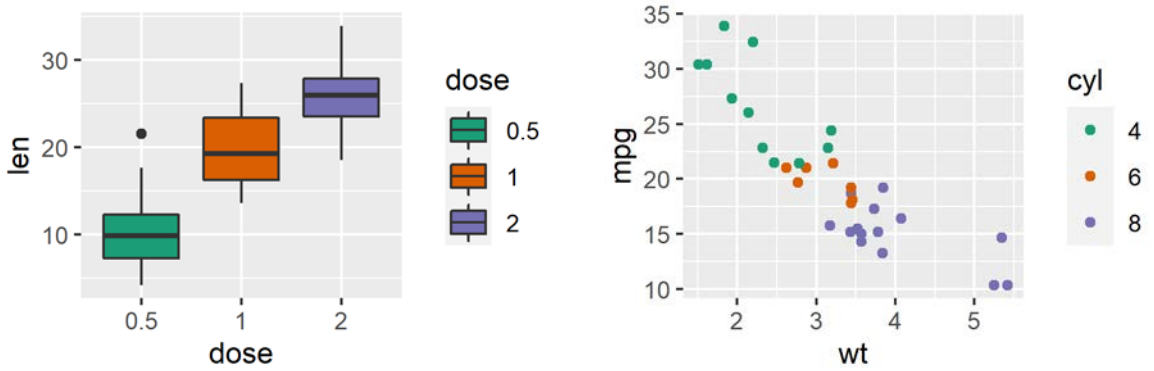
รูปที่ 12-16 Colors by manual

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl และทั้ง 2 รูปมีการกำหนดสีแบบ Manual เพิ่มเติม

สามารถกำหนดสีได้อีกหนึ่งวิธีโดยใช้จานสี (Palettes) ด้วยคำสั่ง `scale_fill_brewer()` และ `scale_color_brewer()` โดยระบุพารามิเตอร์ `palette` ให้เท่ากับชื่อสีที่ต้องการ ดังนี้

```
# Box plots
R> bp + scale_fill_brewer(palette = "Dark2")
```

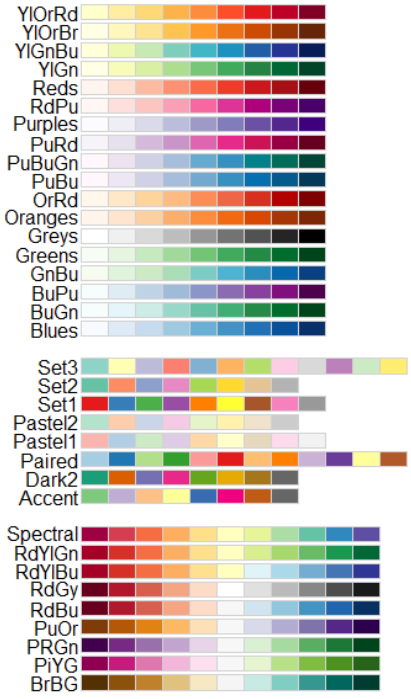
```
# Scatter plots
R> sp + scale_color_brewer(palette = "Dark2")
```



รูปที่ 12-17 Colors by palettes

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl และทั้ง 2 รูปมีการกำหนดสีโดยใช้จานสี (Palettes) เพิ่มเติม

สีโดยใช้จานสี (Palettes) จากแพ็กเกจ RColorBrewer แสดงดังรูปต่อไปนี้ (ดูรายละเอียดเพิ่มเติมจาก https://colorbrewer2.org)



รูปที่ 12-18 RColorBrewer palettes

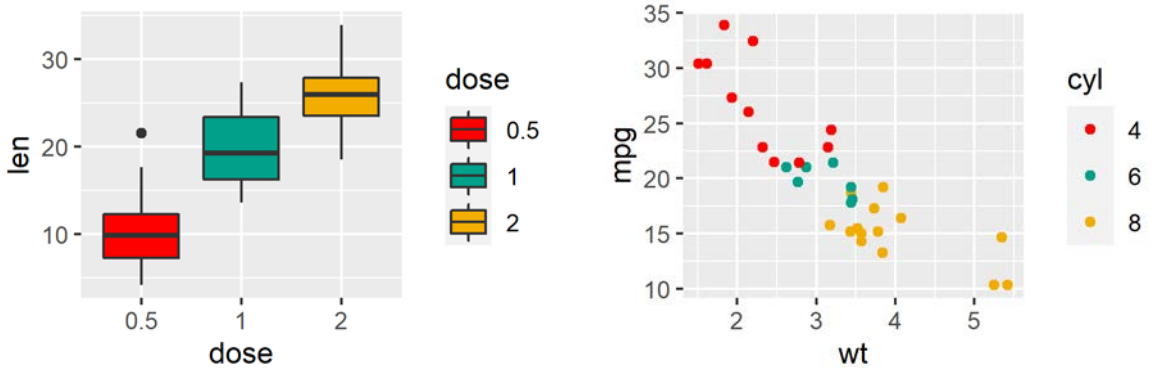
สามารถกำหนดสีโดยใช้จานสี (Palettes) จากแพ็กเกจ Wes Anderson ดังรูปต่อไปนี้

รูปที่ 12-19 Wes Anderson palettes

การกำหนดสีโดยใช้จานสี (Palettes) จากแพ็กเกจ Wes Anderson ระบุพารามิเตอร์ `values` ด้วยคำสั่ง `wes_palette()` และกำหนดชื่อสีด้วยพารามิเตอร์ `name` และจำนวนสีที่ใช้ด้วยพารามิเตอร์ `n` ดังนี้

```
R> library(wesanderson)
# Box plots
R> bp + scale_fill_manual(values = wes_palette(n = 3, name = "Darjeeling1"))

# Scatter plots
R> sp + scale_color_manual(values = wes_palette(n = 3, name = "Darjeeling1"))
```



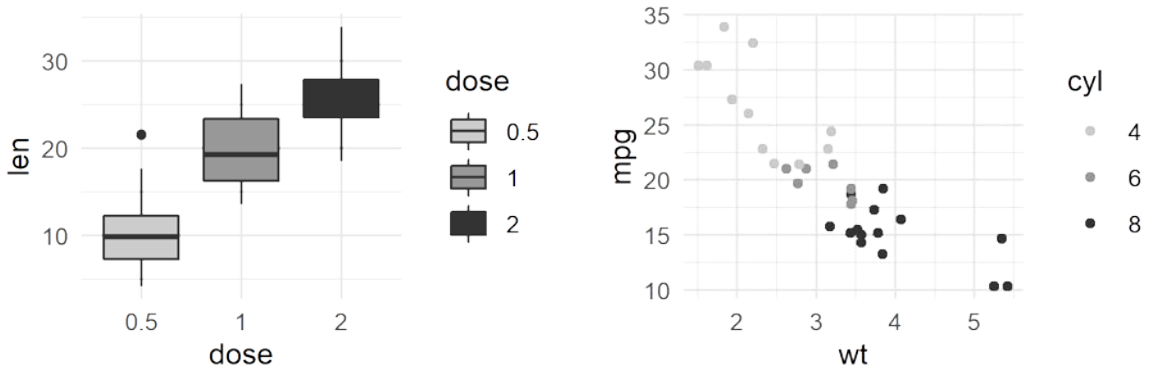
รูปที่ 12-20 Colors by Wes Anderson palettes

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl และทั้ง 2 รูปมีการกำหนดสีโดยใช้จานสี (Palettes) เพิ่มเติมจากแพ็กเกจ Wes Anderson

สามารถกำหนดสีโดยใช้จานสีเทา (Grey color palettes) ด้วยคำสั่ง `scale_fill_grey()` และ `scale_color_grey()` โดยระบุพารามิเตอร์ `start` เท่ากับค่าความเข้มต่ำสุดและ `end` เท่ากับค่าความเข้มสูงสุด ดังนี้

```
# Box plots
R> bp + scale_fill_grey(start = 0.8, end = 0.2) +
     theme_minimal()

# Scatter plots
R> sp + scale_color_grey(start = 0.8, end = 0.2) +
     theme_minimal()
```



รูปที่ 12-21 Colors by grey color palettes

รูปแรกเป็นการแสดงสีตามตัวแปร dose รูปที่สองเป็นการแสดงสีตามตัวแปร cyl และทั้ง 2 รูปมีการกำหนดสีโดยใช้จานสีเทา (Grey color palettes)

สามารถกำหนดสีโดยใช้สีต่อเนื่อง (Gradient or continuous colors) ด้วยคำสั่งดังนี้

- `scale_color_gradient(), scale_fill_gradient()` สำหรับพล็อตสีต่อเนื่องระหว่าง 2 สี
- `scale_color_gradient2(), scale_fill_gradient2()` สำหรับพล็อตสีต่อเนื่อง 2 สีแบบ Diverging gradients
- `scale_color_gradientn(), scale_fill_gradientn()` สำหรับพล็อตสีต่อเนื่อง `n` สี

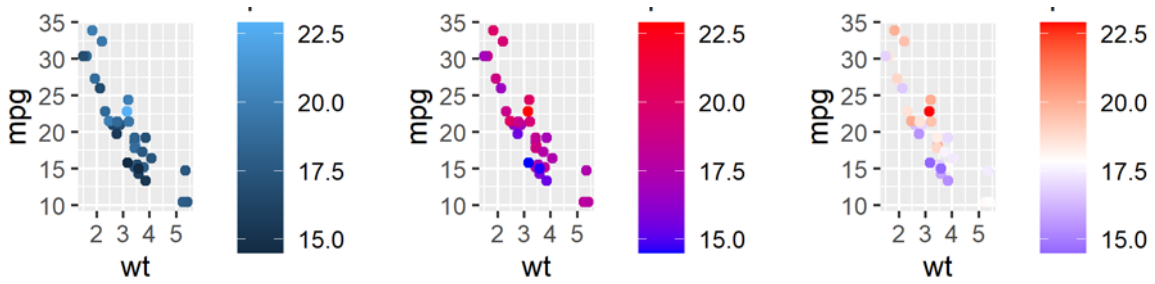
การพล็อตสีต่อเนื่อง 2 สีด้วยคำสั่ง `scale_color_gradient()` ระบุพารามิเตอร์ `low` และ `high` ด้วยสีที่ต้องการ ส่วนการใช้คำสั่ง `scale_color_gradient2()` ต้องระบุพารามิเตอร์เพิ่มเติมได้แก่ `midpoint, mid` และ `space` ดังนี้

```
# Color by qsec values
```

```
R> sp2 <- ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(aes(color = qsec))
R> sp2

# Change the low and high colors
# Sequential color scheme
R> sp2 + scale_color_gradient(low = "blue", high = "red")

# Diverging color scheme
R> mid <- mean(mtcars$qsec)
R> sp2 + scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                               high = "red", space = "Lab" )
```



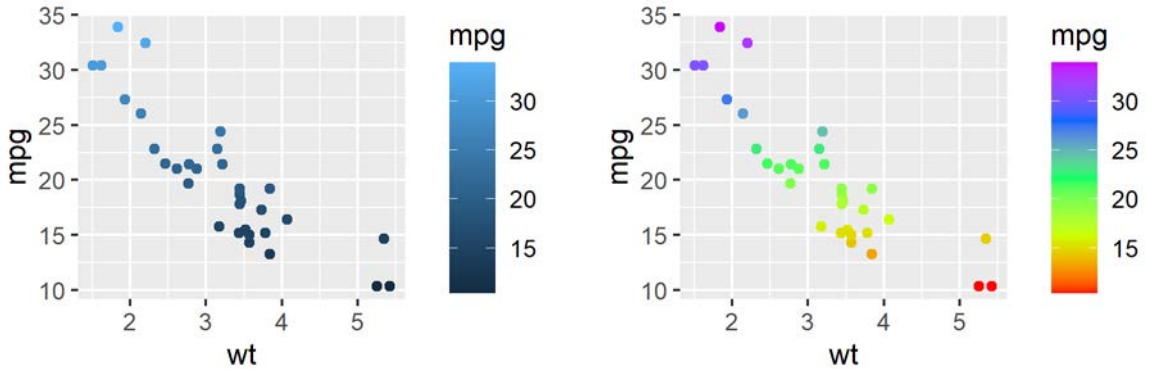
รูปที่ 12-22 Colors by gradients between 2 colors

ทั้ง 3 รูปด้านบนเป็นการกำหนดสีแบบต่อเนื่อง (Gradient or continuous colors) รูปแรกใช้สีน้ำเงินที่ความเข้มแตกต่างกัน รูปที่สองและสามเป็นการพล็อตสีต่อเนื่อง 2 สี

การพล็อตสีต่อเนื่อง n สีด้วยคำสั่ง `scale_color_gradientn()` ต้องระบุพารามิเตอร์ `color` ให้เท่ากับสีที่ต้องการแสดง ดังนี้

```
# Scatter plots
# Color points by the mpg variable
R> sp3 <- ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(aes(color = mpg))
R> sp3

# Gradient between n colors
R> sp3 + scale_color_gradientn(colors = rainbow(5))
```

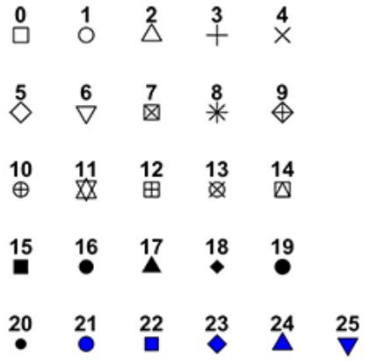


รูปที่ 12-23 Colors by gradients between n colors

ทั้ง 2 รูปด้านบนเป็นการกำหนดสีแบบต่อเนื่อง (Gradient or continuous colors) รูปแรกใช้สีน้ำเงินที่ความเข้มแตกต่างกัน รูปที่สองเป็นการพล็อตสีต่อเนื่อง 5 สี

12.5 รูปร่างจุด สี และขนาด (Point Shapes, Colors and Size)

หัวข้อนี้กล่าวถึงรูปร่างจุด สี และขนาด (Point Shapes, Colors and Size) รูปร่างของจุดแสดงได้ดังรูปต่อไปนี้



รูปที่ 12-24 Common point shapes

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล mtcars ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

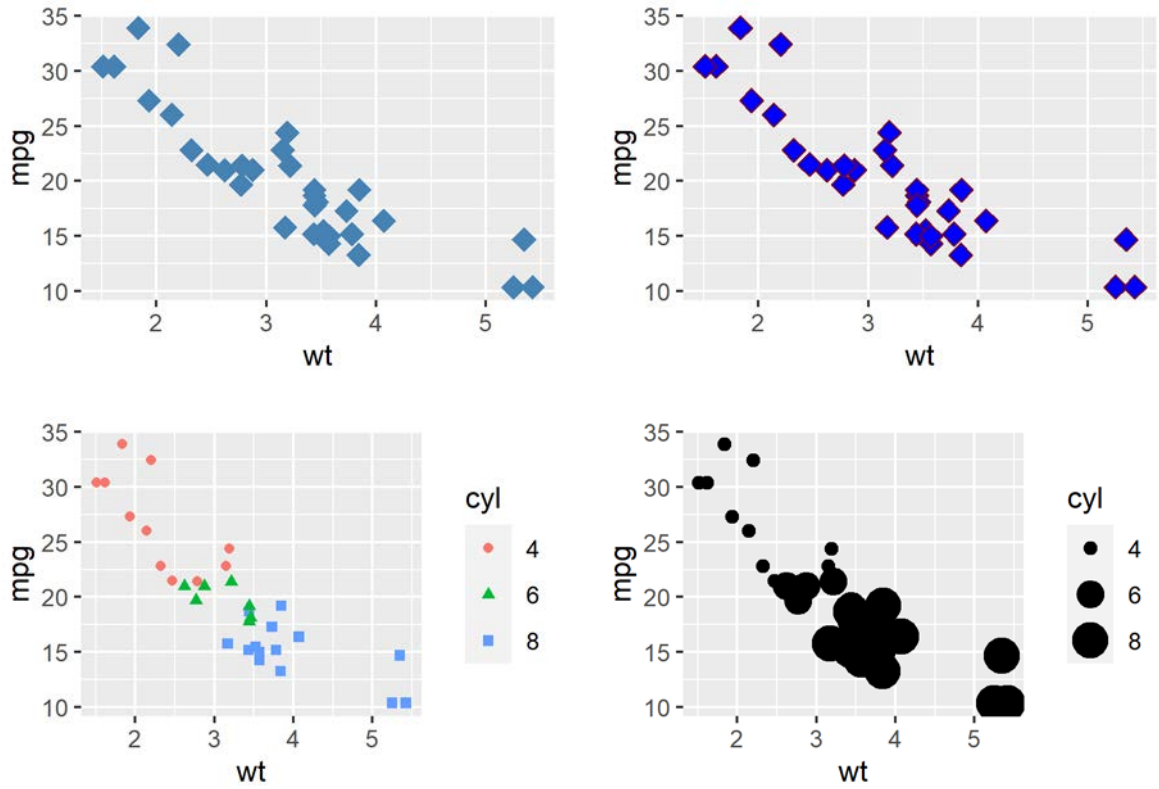
```
# Convert cyl columns from numeric to factor variables
R> mtcars$cyl <- as.factor(mtcars$cyl)
```

การสร้าง Scatter plots สามารถกำหนดรูปร่างจุด สี และขนาด ด้วยพารามิเตอร์ **shape, color** และ **size** ตามลำดับ ส่วนพารามิเตอร์ **fill** ใช้กับรูปร่างจุด (Shape) เท่ากับ 21 ถึง 25 ดังนี้

```
# Change shape, color and size
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(shape = 18, color = "steelblue", size = 4)

# change shape, color, fill, size
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(shape = 23, fill = "blue", color = "darkred", size = 3)

# Change point shapes and colors by groups
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(aes(shape = cyl, color = cyl))
# Control the size by groups
R> ggplot(mtcars, aes(x = wt, y = mpg)) +
     geom_point(aes(size = cyl))
```



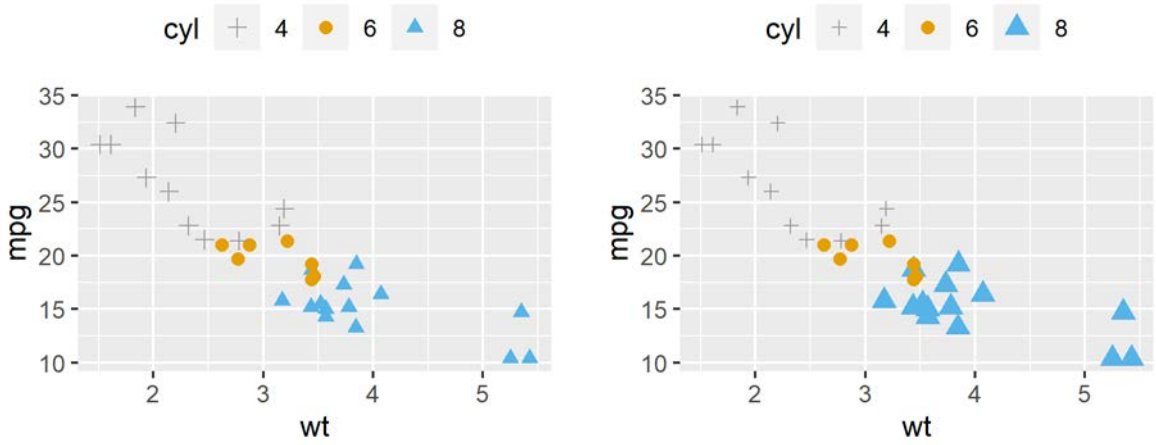
รูปที่ 12-25 Scatter plots by point shapes, colors and size

ทั้ง 4 รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันที่ใช้ (mpg) รูปบนทั้งซ้ายและขวาแสดงข้อมูลแต่ละจุดด้วยเครื่องหมาย (◆) สีเส้นและสีของเครื่องหมาย รูปซ้ายล่างแสดงสีและสัญลักษณ์ตามตัวแปร cyl รูปขวาล่างแสดงขนาดสัญลักษณ์ตามตัวแปร cyl

สามารถกำหนดรูปร่างจุด สี และขนาด (Point shapes, colors and size) ได้เองด้วยคำสั่ง `scale_shape_manual()`, `scale_color_manual()` และ `scale_size_manual()` ต้องระบุพารามิเตอร์ `values` ให้เท่ากับเวกเตอร์ที่ต้องการแสดงผล ดังนี้

```
# Change colors and shapes manually
R> ggplot(mtcars, aes(x = wt, y = mpg, group = cyl)) +
     geom_point(aes(shape = cyl, color = cyl), size = 2) +
     scale_shape_manual(values = c(3, 16, 17)) +
     scale_color_manual(values = c("#999999","#E69F00", "#56B4E9")) +
     theme(legend.position = "top")

# Change the point size manually
R> ggplot(mtcars, aes(x = wt, y = mpg, group = cyl)) +
     geom_point(aes(shape = cyl, color = cyl, size = cyl)) +
     scale_shape_manual(values = c(3, 16, 17)) +
     scale_color_manual(values = c("#999999","#E69F00", "#56B4E9")) +
     scale_size_manual(values = c(1.5,2,3)) +
     theme(legend.position = "top")
```

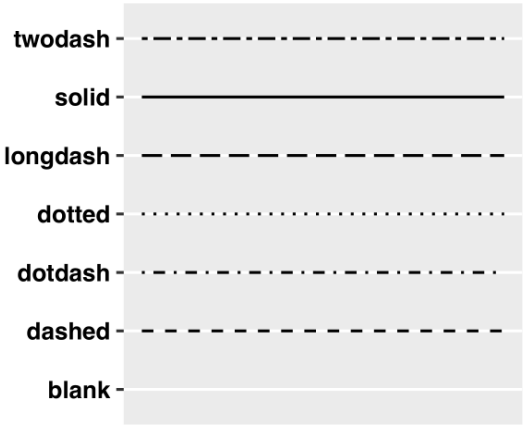


รูปที่ 12-26 Manual shapes, colors and size

ทั้ง 2 รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างน้ำหนัก (wt) และระยะทางต่อน้ำมันที่ใช้ (mpg) แสดงสีและสัญลักษณ์ตามตัวแปร cyl นอกจากนี้รูปขวายังเพิ่มขนาดสัญลักษณ์ตามตัวแปร cyl

12.6 ชนิดเส้น (Line Types)

หัวข้อนี้กล่าวถึงเส้นประเภทต่าง ๆ (Line Types) ได้แก่ “twodash”, “solid”, “longdash”, “dotted”, “dotdash”, “dashed”, “blank” ดังรูปต่อไปนี้



รูปที่ 12-27 Line types

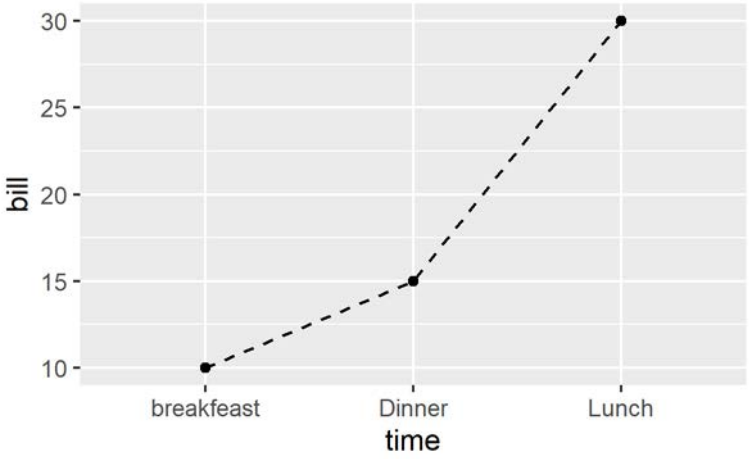
ข้อมูลที่จะใช้ในการพล็อตเป็นเดต้าเฟรม (data.frame) ดังนี้

```
# Create data.frame
R> df <- data.frame(time = c("breakfeast", "Lunch", "Dinner"),
                    bill = c(10, 30, 15))
```

การกำหนดชนิดเส้นด้วยพารามิเตอร์ `linetype` ให้เท่ากับชื่อเส้นที่ต้องการ ดังนี้

```
# Basic line plots with points
# Change the line type
```

```
R> ggplot(data = df, aes(x = time, y = bill, group = 1)) +
     geom_line(linetype = "dashed") +
     geom_point()
```



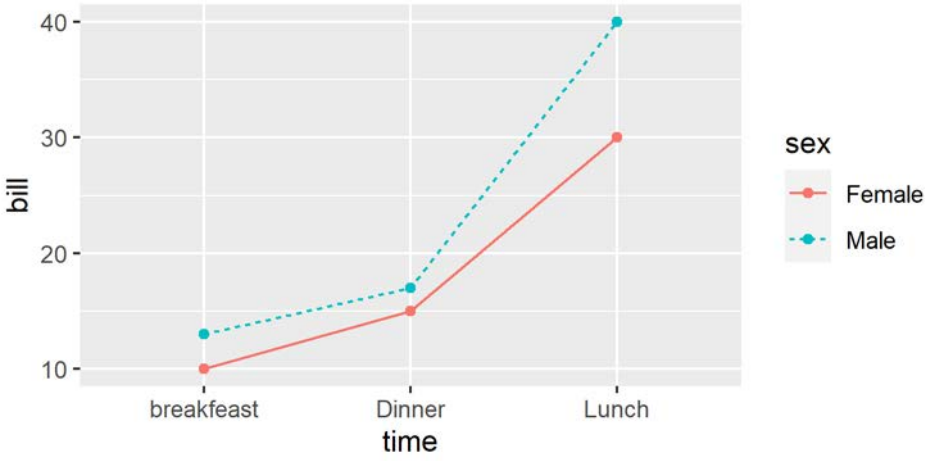
รูปที่ 12-28 Line plots with line type: dashed

รูปด้านบนแสดงความสัมพันธ์ระหว่างอาหารแต่ละมื้อ (time) และค่าใช้จ่าย (bill) โดยแสดงเส้นประ (Dashed line) เชื่อมระหว่างจุดของข้อมูลดังกล่าว

การกำหนดชนิดเส้นและสีสำหรับข้อมูลหลายกลุ่ม (Multiple groups) ทำได้ด้วยคำสั่ง **aes()** ระบุพารามิเตอร์ **linetype** และ **color** ให้เท่ากับกลุ่มข้อมูลที่ต้องการกำหนดชนิดเส้นและสี ดังนี้

```
# Create data
R> df2 <- data.frame(sex = rep(c("Female", "Male"), each = 3),
                     time = c("breakfeast", "Lunch", "Dinner"),
                      bill = c(10, 30, 15, 13, 40, 17) )

# Line plots with multiple groups
# Change line types and colors by groups (sex)
R> ggplot(df2, aes(x = time, y = bill, group = sex)) +
     geom_line(aes(linetype = sex, color = sex)) +
     geom_point(aes(color = sex))
```

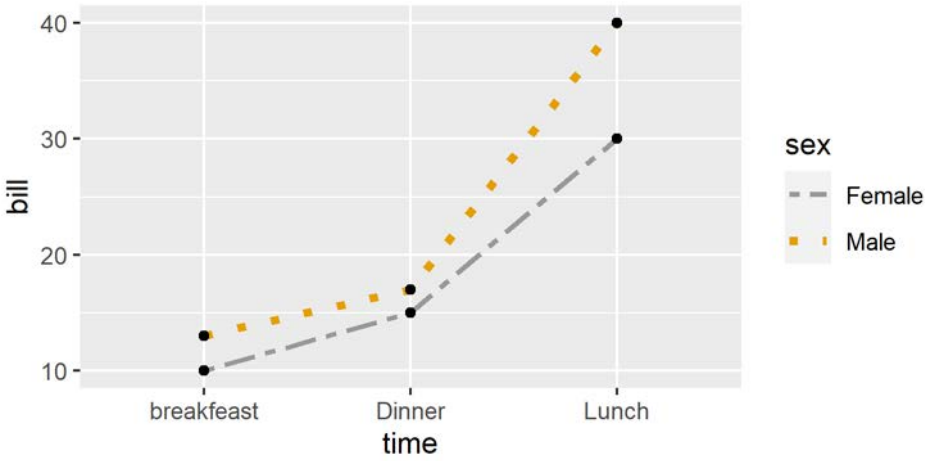


รูปที่ 12-29 Line plots with line type: dashed

รูปด้านบนแสดงความสัมพันธ์ระหว่างอาหารแต่ละมื้อ (time) และค่าใช้จ่าย (bill) โดยแสดงเส้นเชื่อมระหว่างจุดของข้อมูลดังกล่าว และใช้ประเภทเส้นและสีที่แตกต่างกันตามเพศ (sex) โดยเพศชาย (Male) ใช้เส้นประสีฟ้า เพศหญิง (Female) ใช้เส้นทึบสีชมพู

สามารถกำหนดชนิดเส้น (Line types) สี (Colors) และขนาด (Size) ได้เองด้วยคำสั่งต่าง ๆ ดังนี้ `scale_linetype_manual()`, `scale_color_manual()` และ `scale_size_manual()` และระบุพารามิเตอร์ `values` ให้เท่ากับเวกเตอร์ที่ต้องการแสดงผล ดังนี้

```
# Change line types, colors and sizes
R> ggplot(df2, aes(x = time, y = bill, group = sex)) +
     geom_line(aes(linetype = sex, color = sex, size = sex)) +
     geom_point() +
     scale_linetype_manual(values = c("twodash", "dotted")) +
     scale_color_manual(values = c("#999999","#E69F00")) +
     scale_size_manual(values = c(1, 1.5))
```



รูปที่ 12-30 Line plots manual line type and color

รูปด้านบนแสดงความสัมพันธ์ระหว่างอาหารแต่ละมื้อ (time) และค่าใช้จ่าย (bill) โดยแสดงเส้นเชื่อมระหว่างจุดของข้อมูลดังกล่าว และใช้ประเภทเส้นและสีที่แตกต่างกันตามเพศ (sex) โดยเพศชาย (Male) ใช้เส้นประสีเหลือง เพศหญิง (Female) ใช้เส้นประแบบ twodash สีเทา

12.7 การกำหนดขอบเขตแกน (Axis Limits)

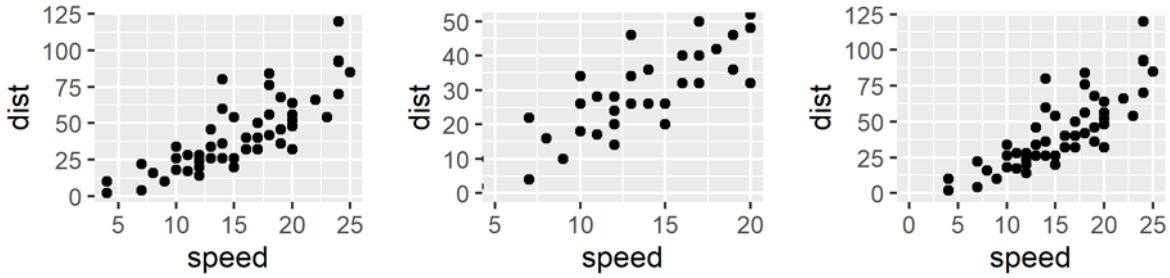
การกำหนดขอบเขตแกน (Axis limits) Cartesian coordinates ใช้คำสั่ง **coord_cartesian()** โดยการระบุพารามิเตอร์ **xlim** และ **ylim** ให้เท่ากับเวกเตอร์ขอบเขตแกน c(ค่าต่ำสุด, ค่าสูงสุด) ถ้าต้องการกำหนดให้จุดเริ่มต้น (Intercept X, Y) ที่มุมล่างซ้าย เป็นค่า coordinate x, y ตามต้องการ ให้ใช้คำสั่ง **expand_limits()** และระบุพารามิเตอร์เป็นค่าเริ่มต้นให้กับ **x** และ **y** ดังนี้

```
R> data(cars)
R> p <- ggplot(cars, aes(x = speed, y = dist)) +
     geom_point()

# Default plots
R> print(p)

# Change axis limits using coord_cartesian()
R> p + coord_cartesian(xlim = c(5, 20), ylim = c(0, 50))

# set the intercept of x and y axis at (0,0)
R> p + expand_limits(x = 0, y = 0)
```

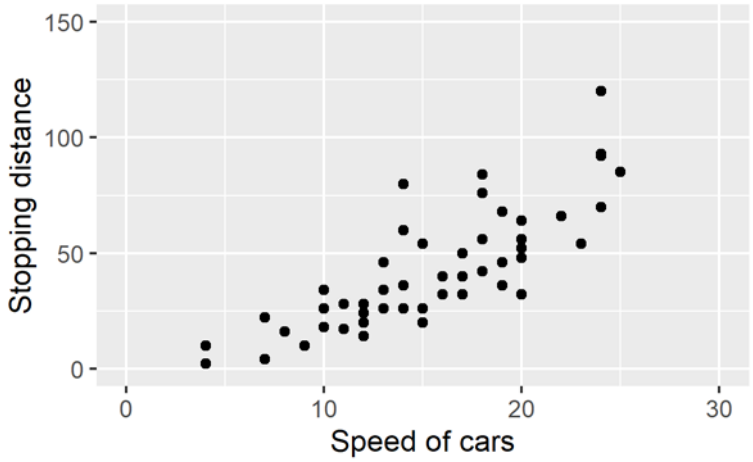


รูปที่ 12-31 Axis limits with Cartesian coordinates

ทั้ง 3 รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างความเร็ว (speed) และระยะหยุด (dist) รูปแรกแสดงข้อมูลทั้งหมด รูปที่สองแสดงแกน X ที่มีค่าระหว่าง 5 – 20 และแกน Y มีค่าระหว่าง 0 – 50 เท่านั้น รูปสุดท้ายแสดงข้อมูลทั้งหมดแต่ให้แสดงจุดเริ่มต้น (มุมล่างซ้าย) ด้วยจุดเริ่มต้น x = 0 และ y = 0

สามารถกำหนดชื่อแกน (Axis labels) และขอบเขตแกน (Axis limits) อีกวิธีหนึ่งได้ด้วยคำสั่ง **scale_x_continuous()** และ **scale_y_continuous()** โดยระบุพารามิเตอร์ **name** ให้เท่ากับชื่อที่ต้องการแสดงผล และพารามิเตอร์ **limits** ให้เท่ากับเวกเตอร์ที่ต้องการแสดงผล ดังนี้

```
# Change x and y axis labels, and limits
p + scale_x_continuous(name = "Speed of cars", limits = c(0, 30)) +
  scale_y_continuous(name = "Stopping distance", limits = c(0, 150))
```



รูปที่ 12-32 Axis labels, and limits

รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างความเร็ว (speed) และระยะหยุด (dist) โดยกำหนดชื่อแกน X และแกน Y และขอบเขตแกนตามที่กำหนด

12.8 การแปลงแกนด้วย log และ sqrt (Axis Transformations: log and sqrt)

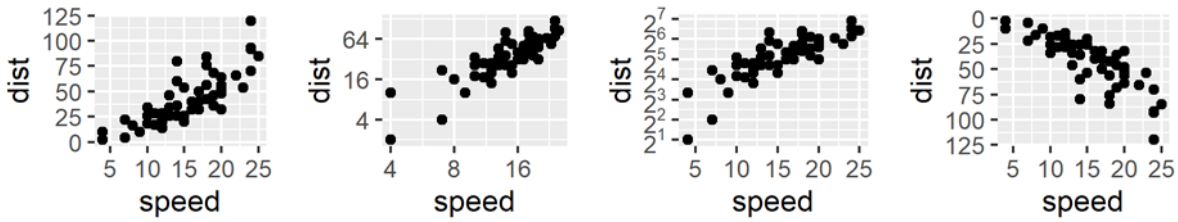
ค่าในแกน X และ Y สามารถแสดงผลเป็นค่า log หรือ sqrt ได้ด้วยคำสั่ง `scale_x_continuous()` และ `scale_y_continuous()` โดยระบุพารามิเตอร์ `trans` ให้เท่ากับชื่อฟังก์ชันที่ต้องการแสดงผล เช่น `"log2", "log10", "sqrt"` และพารามิเตอร์ `breaks` และ `labels` สำหรับกำหนดค่าในแกน และแสดงผลค่าแกนที่ต้องการ ดังนี้

```
# Default scatter plots
R> print(p)

# Log transformation using scale_xx()
# possible values for trans : "log2", "log10","sqrt"
R> p + scale_x_continuous(trans = "log2") +
     scale_y_continuous(trans = "log2")

# Format axis tick mark labels to show exponents
R> require(scales)
R> p + scale_y_continuous(trans = log2_trans(),
                          breaks = trans_breaks("log2", function(x) 2^x),
                          labels = trans_format("log2", math_format(2^.x)))

# Reverse coordinates
R> p + scale_y_reverse()
```



รูปที่ 12-33 Axis transformations

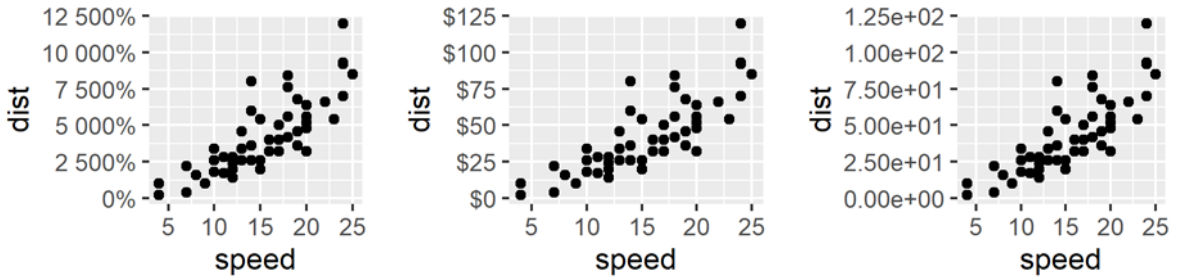
ทั้ง 4 รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างความเร็ว (speed) และระยะหยุด (dist) รูปแรกเป็นการพล็อตด้วยค่าโดยปริยายซึ่งไม่มีการแปลงค่าแกนใด ๆ รูปที่สองแปลงค่าแกน X และแกน Y ด้วย Log ฐานสอง รูปที่สามแปลงค่าเฉพาะแกน Y ด้วย Log ฐานสอง รูปสุดท้ายทำการเรียงค่า (Sort) ในแกน Y จากมากไปน้อย (จากเดิมน้อยไปมาก)

แพ็กเกจ scales ใช้สำหรับการกำหนดรูปแบบตัวเลขที่ต้องการแสดงผล เช่น แสดงเป็นเปอร์เซ็นต์ (%) เครื่องหมายดอลลาร์ ($) และค่าระบบตัวเลขทางวิทยาศาสตร์ (Scientific notation) โดยการระบุพารามิเตอร์ `labels` ให้เท่ากับ `percent, dollar` และ `scientific` ตามลำดับ ดังนี้

```
R> require(scales)
# Percent
R> p + scale_y_continuous(labels = percent)

# Dollar
R> p + scale_y_continuous(labels = dollar)

# Scientific
R> p + scale_y_continuous(labels = scientific)
```



รูปที่ 12-34 Axis labels with scales

ทั้ง 3 รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างความเร็ว (speed) และระยะหยุด (dist) รูปแรกแสดงค่าแกน Y ด้วยเปอร์เซนต์ (%) รูปที่สองแสดงค่าแกน Y ด้วยเครื่องหมายดอลลาร์ ($) รูปสุดท้ายแสดงค่าแกน Y ด้วยระบบตัวเลขทางวิทยาศาสตร์ (Scientific notation)

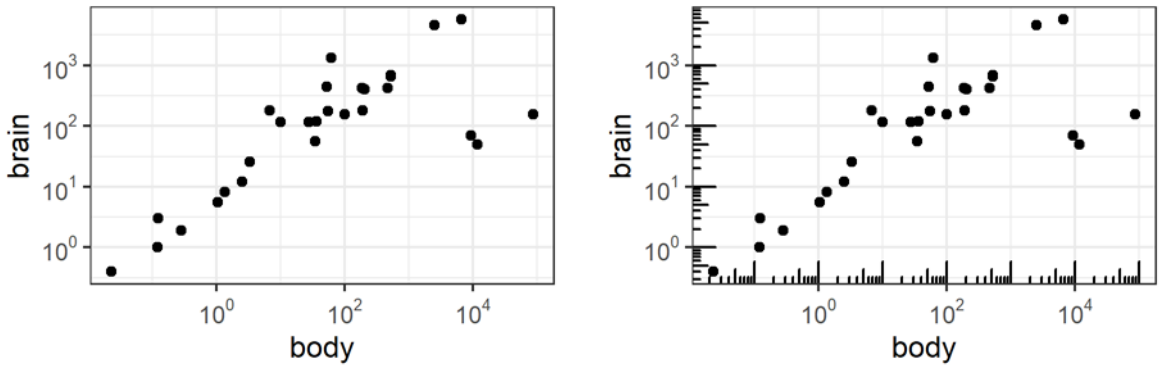
สามารถแสดงขีด Log tick marks บนแกนด้วยคำสั่ง `annotation_logticks()` ดังนี้

```
R> require(MASS) # to access Animals data sets
R> require(scales) # to access break formatting functions
```

```
R> data(Animals) # Load data

# x and y axis are transformed and formatted
R> p2 <- ggplot(Animals, aes(x = body, y = brain)) +
     geom_point() +
     scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x))) +
     scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x))) +
     theme_bw()
# log-log plots without log tick marks
R> p2

# Show log tick marks
R> p2 + annotation_logticks()
```



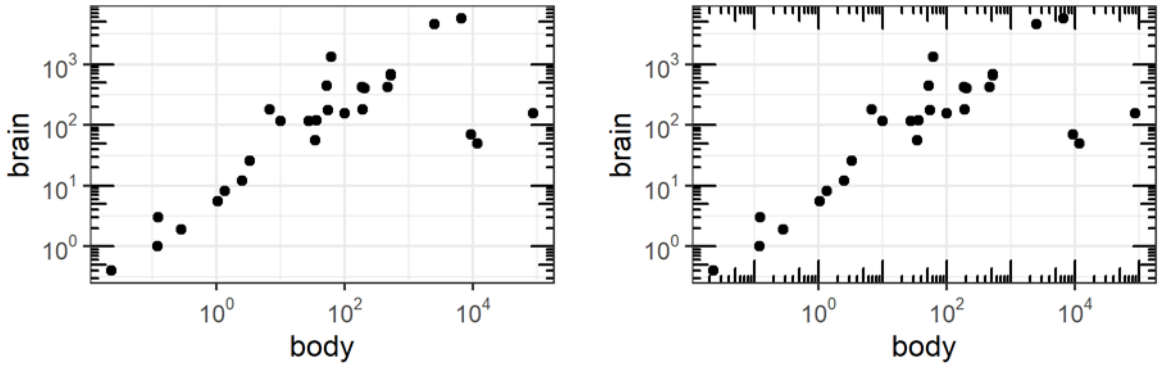
รูปที่ 12-35 Log tick marks

รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างน้ำหนักร่างกาย (body) และน้ำหนักสมอง (brain) ทั้งแกน X และแกน Y มีหน่วยเป็น log ฐาน 10 (Log-log scales) รูปขวาแสดงขีด Log tick marks เพิ่มเติม

สามารถแสดงขีด log tick marks ด้านซ้าย ขวา บน หรือล่างของกราฟด้วยพารามิเตอร์ **sides** เท่ากับอักษร **"l"** = ซ้าย (Left), **"r"** = ขวา (Right), **"t"** = บน (Top), **"b"** = ล่าง (Bottom) โดยการรวมอักษรดังกล่าวเข้าด้วยกัน เช่น **"lr"** หมายถึงแสดงขีด log tick marks ด้านซ้าย และขวา ตามลำดับ ดังนี้

```
# Log ticks on left and right
R> p2 + annotation_logticks(sides = "lr")

# All sides
R> p2 + annotation_logticks(sides = "lrtb")
```



รูปที่ 12-36 Log tick mark positions

รูปด้านบนเป็น Scatter plots แสดงความสัมพันธ์ระหว่างน้ำหนักร่างกาย (body) และน้ำหนักสมอง (brain) ทั้งแกน X และแกน Y มีหน่วยเป็น log ฐาน 10 (Log-log scales) รูปแรกแสดงขีด Log tick marks ที่ด้านข้างทั้งซ้ายและขวา รูปขวาแสดงขีด Log tick marks ทุกด้านทั้งซ้าย ขวา บน และล่าง

12.9 การกำหนดแกนในหน่วยวัน (Date Axis)

การแสดงแกนในหน่วยวัน (Date axis) ใช้คำสั่ง `scale_x_date()` และ `scale_y_date()`

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูลราคารายวันจำนวน 50 แถว ประกอบด้วยวัน และราคา ดังนี้

```
R> set.seed(456)
R> df <- data.frame(
     date = seq(Sys.Date(), length.out = 100, by = "1 day")[sample(100, 50)],
     price = runif(50)
     )
R> df <- df[order(df$date), ]
```

มีการใช้แพ็กเกจ scales สำหรับกำหนดรูปแบบการแสดงผลบนแกน X และ Y การแสดงชื่อแกนใช้คำสั่ง `scale_x_date()` หรือ `scale_y_date()` ระบุพารามิเตอร์ `labels` เป็น `date_format()` ดังนี้

```
# Default plots
R> p <- ggplot(df, aes(x = date, y = price)) +
     geom_line()
R> p

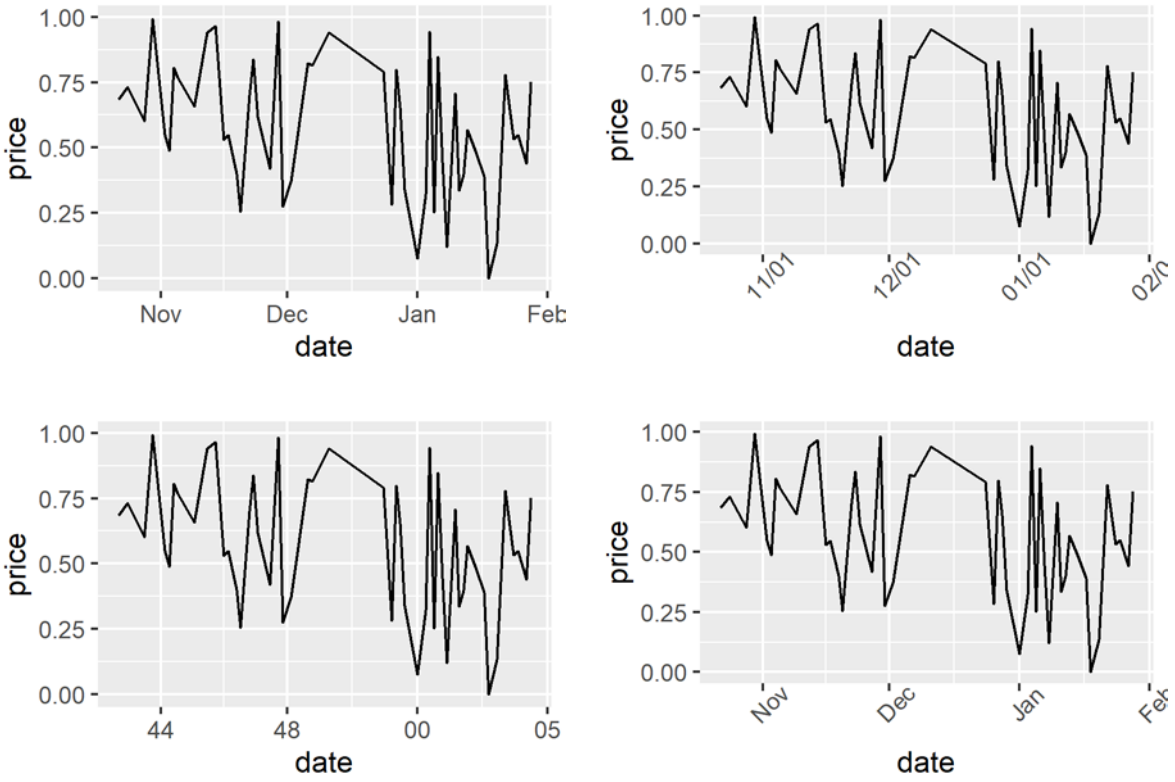
# Format axis tick mark labels
# Format : month/day
R> require(scales)
R> p + scale_x_date(labels = date_format("%m/%d")) +
     theme(axis.text.x = element_text(angle = 45))

# Format : Week
R> p + scale_x_date(labels = date_format("%W"))

# Months only
```

```
R> p + scale_x_date(breaks = date_breaks("months"),
                    labels = date_format("%b")) +
     theme(axis.text.x = element_text(angle = 45))
```

คำสั่งแรกเป็นการพล็อตโดยใช้ **geom_line()** จะเห็นว่าแกน X แสดงเป็นชื่อเดือนโดยอัตโนมัติ คำสั่งที่สองเป็นการกำหนดให้แกน X แสดงเป็นเดือน/วันโดยใช้คำสั่ง **date_format()** และระบุพารามิเตอร์เป็น **"%m/%d"** หมายถึง แสดงลำดับของเดือนและวัน ตามลำดับ (รูปแบบการแสดงผลเวลา วัน เดือน และปี ศึกษาเพิ่มเติมได้จาก **?strptime**) คำสั่งถัดมาเป็นการกำหนดให้แกน X แสดงเป็นสัปดาห์ในรอบปี คำสั่งสุดท้ายเป็นการกำหนดให้แกน X แสดงเป็นเดือนแบบย่อ (คำสั่ง **element_text()** จะกล่าวถึงในหัวข้อ 12.10)



รูปที่ 12-37 Date axis

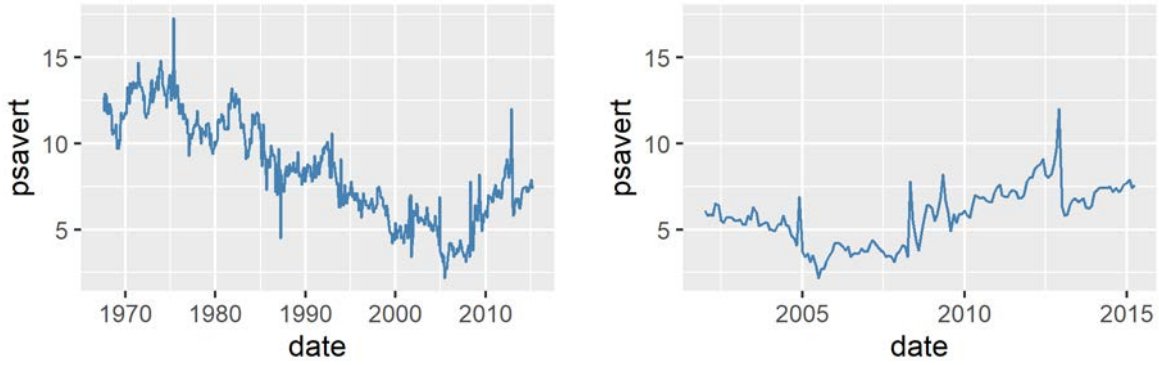
ทั้ง 4 รูปด้านบนแสดงกราฟอนุกรมเวลา (Time series) แกน X เป็นเวลา แกน Y เป็นราคา รูปซ้ายบนแสดงข้อความบนแกน (Axis Ticks) X ด้วยเดือนแบบย่อ รูปขวาบนแสดงข้อความบนแกน X ด้วยเดือน/วัน เป็นตัวเลขและหมุนทวนเข็มนาฬิกา 45 องศา รูปซ้ายล่างแสดงข้อความบนแกน X ด้วยตัวเลขสัปดาห์ในปี รูปขวาล่างแสดงข้อความบนแกน X ด้วยเดือนแบบย่อและหมุนทวนเข็มนาฬิกา 45 องศา

การกำหนดขอบเขตของวันที่ใช้พารามิเตอร์ **limits** เท่ากับเวกเตอร์ของวันเริ่มต้นและวันสุดท้ายที่จะแสดงผล ดังนี้

```
# Plot with dates
R> data("economics")
```

```
R> p <- ggplot(data = economics, aes(x = date, y = psavert)) +
     geom_line(color = "steelblue")
R> p

# Axis limits c(min, max)
R> min <- as.Date("2002-1-1")
R> max <- max(economics$date)
R> p + scale_x_date(limits = c(min, max))
```



รูปที่ 12-38 Date axis with date limits

ทั้ง 2 รูปด้านบนแสดงกราฟอนุกรมเวลา (Time series) แกน X เป็นเวลา แกน Y เป็น Personal savings rate (psavert) รูปขวากำหนดขอบเขตเวลาที่แสดงผลเริ่มตั้งแต่ปี 2002 เป็นต้นไป

12.10 การกำหนดรูปแบบการแสดงผลบนแกน (Axis Ticks : Customize Tick Marks and Labels, Reorder and Select items)

หัวข้อนี้จะกล่าวถึงการกำหนดรูปแบบการแสดงผลข้อความบนแกน (Axis Ticks) ด้วยคำสั่ง `element_text()` และ `element_blank()`

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

```
R> data("ToothGrowth")
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
     geom_boxplot()
```

การกำหนดรูปแบบการแสดงผลข้อความบนแกน (Axis Ticks) ใช้คำสั่ง **`theme()`** ระบุพารามิเตอร์ที่ต้องการแสดงผล เช่น **`axis.text.x`** สำหรับข้อความบนแกน X, **`axis.text.y`** สำหรับข้อความบนแกน Y และ **`axis.ticks`** สำหรับขีดที่อยู่บนแกน X และ Y ให้มีค่าเท่ากับฟังก์ชัน **`element_text()`** และพารามิเตอร์ที่ระบุประกอบด้วย **`face`** สำหรับกำหนดประเภทฟอนต์ (ได้แก่ **`"plain"`**, **`"italic"`**, **`"bold"`** และ **`"bold.italic"`**) **`color`** สำหรับกำหนดสี, **`size`** สำหรับกำหนดขนาดฟอนต์, และ **`angle`** สำหรับกำหมดมุมเอียงของข้อความที่จะแสดงผล ส่วนฟังก์ชัน **`element_blank()`** ใช้สำหรับซ่อนข้อความที่ไม่ต้องการแสดงผล ส่วนการกำหนดเส้นแกนใช้พารามิเตอร์ **`axis.line`** เท่ากับฟังก์ชัน **`element_line()`**

```
# Change the style of axis tick labels
# face can be "plain", "italic", "bold" or "bold.italic"
R> p + theme(axis.text.x = element_text(face = "bold", color="#993333",
                                        size = 12, angle = 45),
             axis.text.y = element_text(face = "bold", color = "blue",
                                        size = 12, angle = 45))

# Remove axis ticks and tick mark labels
R> p + theme(
     axis.text.x = element_blank(), # Remove x axis tick labels
     axis.text.y = element_blank(), # Remove y axis tick labels
     axis.ticks = element_blank()) # Remove ticks

# Change the line type and color of axis lines
R> p + theme(axis.line = element_line(color = "darkblue",
                                      size = 1, linetype = "solid"))
```

รูปที่ 12-39 Axis ticks

ทั้ง 3 รูปด้านบนเป็น Box plots แสดงตัวแปร dose ในแกน X ตัวแปร len ในแกน Y และแสดงผลข้อความบนแกน (Axis Ticks) X และ Y ด้วยประเภทฟอนต์ สี ขนาดต่าง ๆ ดังรูป รูปที่สองเป็นการซ่อนการแสดงผลข้อความบนแกน รูปสุดท้ายแสดงแกน X และแกน Y ให้มีสีน้ำเงินและหนาขึ้น

พารามิเตอร์ที่ระบุในคำสั่ง `theme()` มีมากมายเกินกว่าที่จะกล่าวถึงได้หมดในเอกสารฉบับนี้ ผู้ที่สนใจศึกษาได้ จาก Help ใน **R** หรือพิมพ์คำสั่ง `?theme`

ตัวแปรที่กำหนดให้กับแกน X และ Y เป็นได้ทั้งตัวแปรต่อเนื่อง (Continuous variable) หรือตัวแปรแบ่งกลุ่ม/ไม่ต่อเนื่อง (Discrete variable) ในการพล็อตแกนที่เป็นตัวแปรต่อเนื่อง (Continuous variable) มีคำสั่งประกอบด้วย

- `scale_x_continuous()` สำหรับแกน X
- `scale_y_continuous()` สำหรับแกน Y

ส่วนการพล็อตแกนที่เป็นตัวแปรแบ่งกลุ่ม/ไม่ต่อเนื่อง (Discrete variable) มีคำสั่งประกอบด้วย

- `scale_x_discrete()` สำหรับแกน X
- `scale_y_discrete()` สำหรับแกน Y

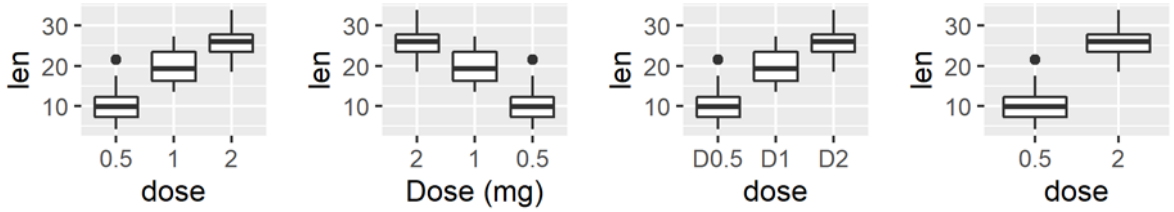
โดยมีการระบุพารามิเตอร์ได้แก่ **name** สำหรับกำหนดชื่อแกน X และแกน Y, **breaks** เป็นเวกเตอร์เพื่อ **break** การแสดงผล, **labels** สำหรับกำหนดขีด (Tick marks) ของแกน, และ **limits** เป็นเวกเตอร์เพื่อกำหนดขอบเขตการแสดงผลสำหรับตัวแปรต่อเนื่อง แต่ถ้าเป็นการแสดงผลสำหรับตัวแปรแบ่งกลุ่ม/ไม่ต่อเนื่องจะหมายถึง ลำดับของข้อมูลที่ต้องการแสดงผล ดังนี้

```
# Default plots
R> print(p)

# Change x axis label and the order of items
R> p + scale_x_discrete(name = "Dose (mg)",
                        limits = c("2","1","0.5"))

# Change tick mark labels
R> p + scale_x_discrete(breaks = c("0.5","1","2"),
                        labels = c("D0.5", "D1", "D2"))

# Choose which items to display
R> p + scale_x_discrete(limits = c("0.5", "2"))
```



รูปที่ 12-40 Axis with scale_x_discrete()

ทั้ง 4 รูปด้านบนเป็น Box plots แสดงตัวแปร dose ในแกน X ตัวแปร len ในแกน Y รูปแรกแสดงชื่อแกน (Axis title) X และ Y ด้วยตัวแปร dose และ len ตามลำดับ รูปที่สองกำหนดชื่อแกน X และลำดับการแสดงข้อมูล dose เองเริ่มจาก 2, 1 และ 0.5 ตามลำดับ รูปที่สามกำหนดข้อความบนแกน (Axis Ticks) X เอง รูปสุดท้ายแสดงข้อมูล dose เฉพาะ 0.5 และ 2 ตามลำดับ

รูปต่อไปเป็นการพล็อตด้วย **geom_point()** รูปที่สองเป็นการกำหนดชื่อให้กับแกน X และแกน Y พร้อมกับขอบเขตของแกนดังกล่าว รูปที่สามเป็นการกำหนดขีด (Tick marks) ให้แกน Y แสดงทุก ๆ ค่า 50 รูปถัดมาเป็นการกำหนดขีด (Tick marks) ให้แกน Y เท่ากับ 0, 50, 65, 75 และ 150 ตามลำดับ รูปถัดมาเป็นการซ่อนการแสดงขีด (Tick marks) ของแกน Y รูปสุดท้ายเป็นการแสดงค่าของแกน Y เป็นเปอร์เซ็นต์ (%) ดังนี้

```
R> sp <- ggplot(cars, aes(x = speed, y = dist)) +
     geom_point()
R> sp

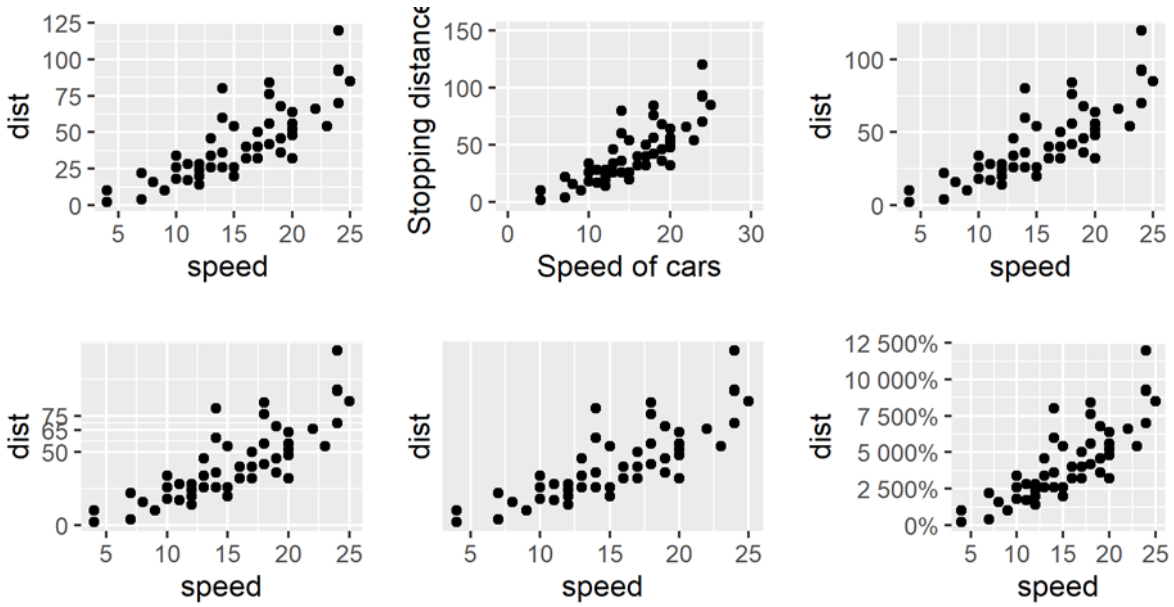
# Change x and y axis labels, and limits
R> sp + scale_x_continuous(name = "Speed of cars", limits = c(0, 30)) +
     scale_y_continuous(name = "Stopping distance", limits = c(0, 150))
```

```
# Set tick marks on y axis: a tick mark is shown on every 50
R> sp + scale_y_continuous(breaks = seq(0, 150, 50))

# Tick marks can be spaced randomly
R> sp + scale_y_continuous(breaks = c(0, 50, 65, 75, 150))

# Remove tick mark labels and gridlines
R> sp + scale_y_continuous(breaks = NULL)

# Format the labels
R> require(scales)
# Possible values for labels = percent, scientific, ..
R> sp + scale_y_continuous(labels = percent) # labels as percents
```



รูปที่ 12-41 Axis tick formats

12.11 ธีมและสีพื้น (Themes and Background Colors)

หัวข้อนี้จะกล่าวถึงใช้ธีม ซึ่งประกอบไปด้วยสีพื้น (Background colors) สีพื้นของ panel (Panel background colors) และเส้นกริด (Grid lines)

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** ดังนี้

```
# Convert dose and cyl columns from numeric to factor variables
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
     geom_boxplot()
```

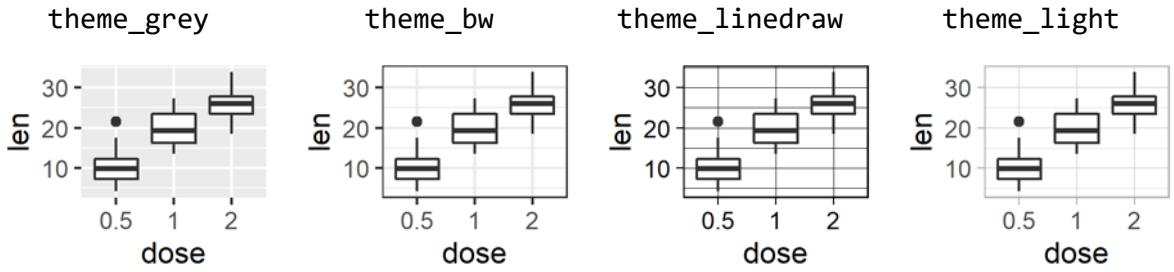
คำสั่งใน **ggplot2** ที่ช่วยเปลี่ยนธีมได้อย่างรวดเร็ว ได้แก่

- `theme_grey()` ธีมพื้นสีเทาและเส้น Grid สีขาว
- `theme_bw()` ธีมพื้นสีขาวและเส้น Grid สีเทา
- `theme_linedraw()` ธีมพื้นสีขาวและเส้นกรอบสีดำ

- `theme_light()` ธีมพื้นสีขาว เส้น Grid และแกนสีเทาอ่อน
- `theme_minimal()` ธีมที่ไม่มีคำอธิบายประกอบ (No background annotations)
- `theme_classic()` ธีมที่แสดงแกน X และแกน Y แต่ไม่แสดงเส้น Grid
- `theme_void()` ไม่แสดงธีมใด ๆ เลย
- `theme_dark()` ธีมพื้นสีเทาเข้ม

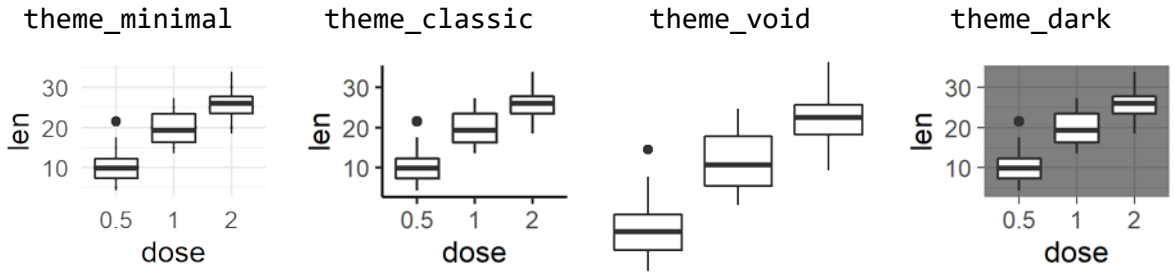
ตัวอย่างการใช้ธีมข้างต้น แสดงได้ดังนี้

```
R> p + theme_grey()
R> p + theme_bw()
R> p + theme_linedraw()
R> p + theme_light()
```



รูปที่ 12-42 Themes: grey, bw, linedraw, and light

```
R> p + theme_minimal()
R> p + theme_classic()
R> p + theme_void()
R> p + theme_dark()
```

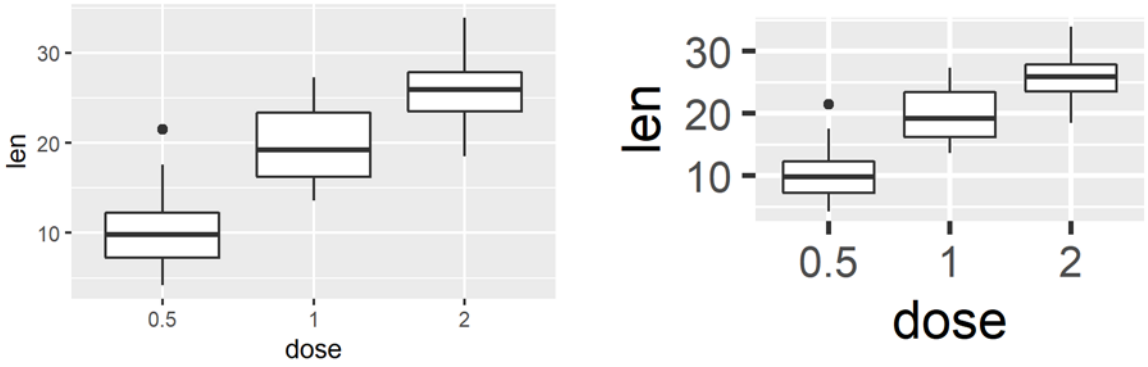


รูปที่ 12-43 Themes: minimal, classic, void, and dark

การใช้ฟังก์ชัน `theme_xxx()` สามารถกำหนดขนาดฟอนต์ด้วยพารามิเตอร์ `base_size` ให้เท่ากับขนาดที่ต้องการหน่วยเป็น pts ดังนี้

```
# base_size = 10
R> p + theme_grey(base_size = 10)

# base_size = 20
R> p + theme_grey(base_size = 20)
```



รูปที่ 12-44 Themes with base_size

รูปแรกใช้ `base_size` เท่ากับ 10 pts รูปที่สองใช้ `base_size` เท่ากับ 20 pts จะเห็นว่าขนาดฟอนต์ใหญ่กว่ารูปแรกมาก

สามารถใช้คำสั่ง `theme_set()` ในการเปลี่ยนธีมทุก ๆ Session ใน **R** ตัวอย่างเช่น

```
theme_set(theme_gray(base_size = 15))
```

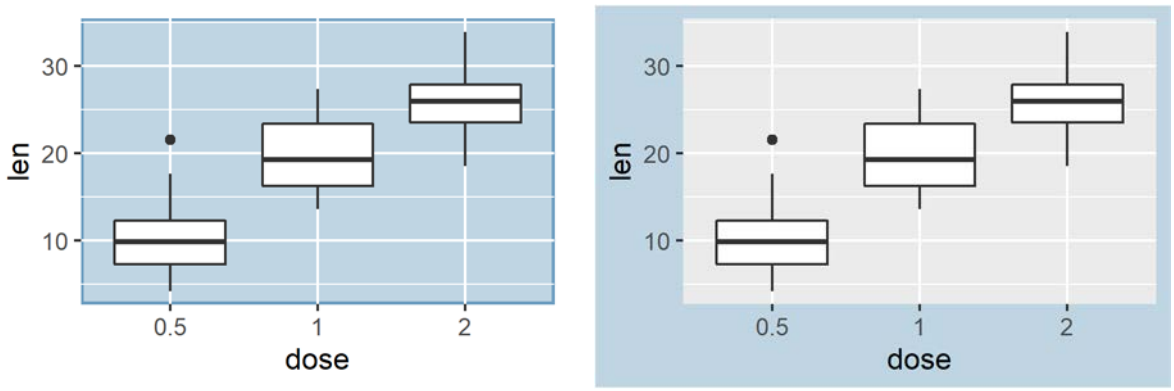
คำสั่ง `theme()` ยังใช้ในการควบคุมการแสดงผลองค์ประกอบต่าง ๆ ในกราฟ ด้วยคำสั่งดังนี้

- `element_line()` สำหรับการแสดงผลเส้น ได้แก่ axis lines, minor and major grid lines, plot panel border, axis ticks background color เป็นต้น
- `element_text()` สำหรับการแสดงผลข้อความ ได้แก่ plot title, axis titles, legend title and text, axis tick mark labels เป็นต้น
- `element_rect()` สำหรับการแสดงผลกรอบสี่เหลี่ยม ได้แก่ plot background, panel background, legend background เป็นต้น

รูปต่อไปเป็นการแสดง Panel background และ Plot background ด้วยคำสั่ง `element_rect()` สามารถระบุพารามิเตอร์ได้แก่ `fill, color, size` และ `linetype` และแสดงเส้น Grid ด้วยคำสั่ง `element_line()` ด้วยการระบุพารามิเตอร์ได้แก่ `color, size` และ `linetype` ดังนี้

```
# Change the colors of plot panel background to lightblue
# and the color of major/grid lines to white
R> p + theme(
     panel.background = element_rect(fill = "#BFD5E3", color = "#6D9EC1",
                                     size = 2, linetype = "solid"),
     panel.grid.major = element_line(size = 0.5, linetype = "solid",
                                     color = "white"),
     panel.grid.minor = element_line(size = 0.25, linetype = "solid",
                                     color = "white")
     )

# Change the plot background color
R> p + theme(plot.background = element_rect(fill = "#BFD5E3")
```

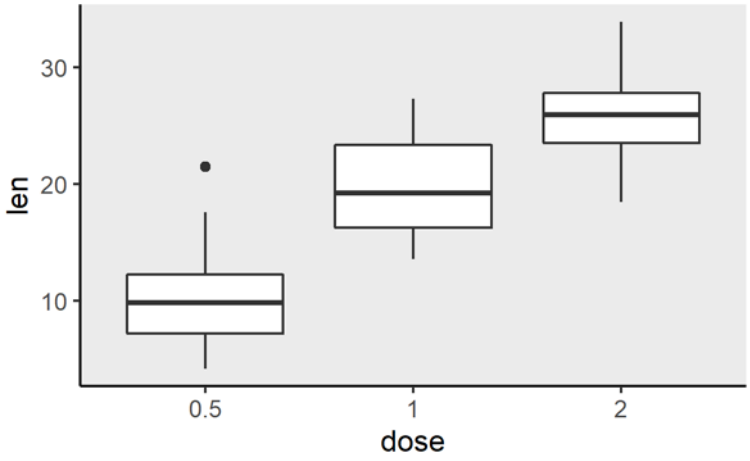


รูปที่ 12-45 Themes with element controls

รูปแรกแสดง Panel background สีฟ้า เส้น Major grid ขนาด 0.5 pts สีขาว เส้น Minor grid ขนาด 0.25 pts สีขาว รูปที่สองแสดง Plot background สีฟ้า Panel background สีเทา

รูปต่อไปนี้ซ่อนการแสดงผล Panel border และเส้น Grid ด้วยคำสั่ง `element_blank()` ดังนี้

```
# Hide panel borders and grid lines
# But change axis line
R> p + theme(panel.border = element_blank(),
             panel.grid.major = element_blank(),
             panel.grid.minor = element_blank(),
             axis.line = element_line(size = 0.5, linetype = "solid",
                                      color = "black"))
```



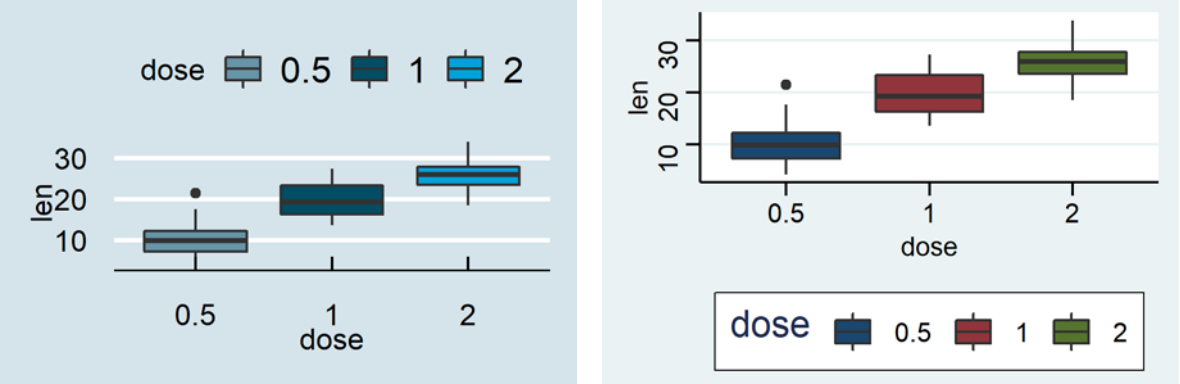
รูปที่ 12-46 Themes with element_blank()

นอกจากนี้ยังมีแพ็กเกจอีกหลายแพ็กเกจ[15] ได้สร้างธีมที่น่าสนใจไว้ เช่น แพ็กเกจ `ggthemes` ดังนี้

[15] ผู้ที่สนใจค้นคว้าเพิ่มเติมได้ที่ https://yutannihilation.github.io/allYourFigureAreBelongToUs/ggthemes/

```
library(ggthemes) # Load
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
     geom_boxplot(aes(fill = dose))
# theme_economist
R> p + theme_economist() +
     scale_fill_economist()

# theme_stata
R> p + theme_stata() +
     scale_fill_stata()
```



รูปที่ 12-47 ggthemes: theme_economist() and theme_stata()

รูปแรกใช้ธีม Economist ส่วนรูปที่สองใช้ธีม Stata

12.12 ข้อความ (Text Annotations)

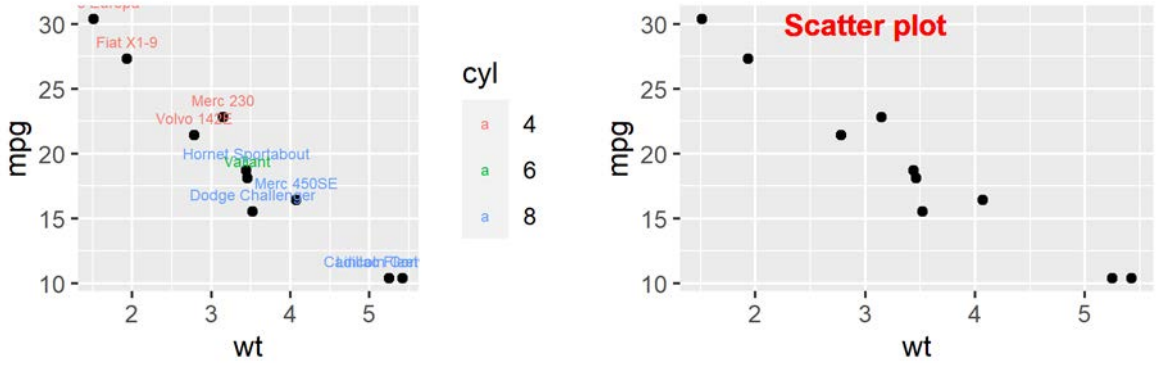
การแสดงข้อความใช้คำสั่ง `geom_text()` และกำหนดฟังก์ชั่น `aes()` โดยการระบุพารามิเตอร์ `label` ให้เท่ากับข้อความที่ต้องการแสดงผล และพารามิเตอร์อื่น ๆ ได้แก่ `x` สำหรับกำหนด coordinate X, `y` สำหรับกำหนด coordinate Y, `size` สำหรับกำหนดขนาด, `hjust` สำหรับกำหนดตำแหน่งในแนวนอน, `vjust` สำหรับกำหนดตำแหน่งในแนวตั้ง, `color` สำหรับกำหนดสี, `fontface` สำหรับกำหนดประเภทฟอนต์ และ `angle` สำหรับกำหนดมุมในการหมุนข้อความ ดังนี้

```
R> set.seed(1234)
R> df <- mtcars[sample(1:nrow(mtcars), 10), ]
R> df$cyl <- as.factor(df$cyl)

# Simple scatter plots
R> sp <- ggplot(df, aes(wt, mpg, label = rownames(df))) +
     geom_point()

# Add text, change colors by groups
# Change vertical justification
R> sp + geom_text(aes(label = rownames(df), color = cyl),
                  size = 3, vjust = -1)

# Add text at a particular coordinate
R> sp + geom_text(x = 3, y = 30, label = "Scatter plot",
                  color = "red", fontface = "bold")
```



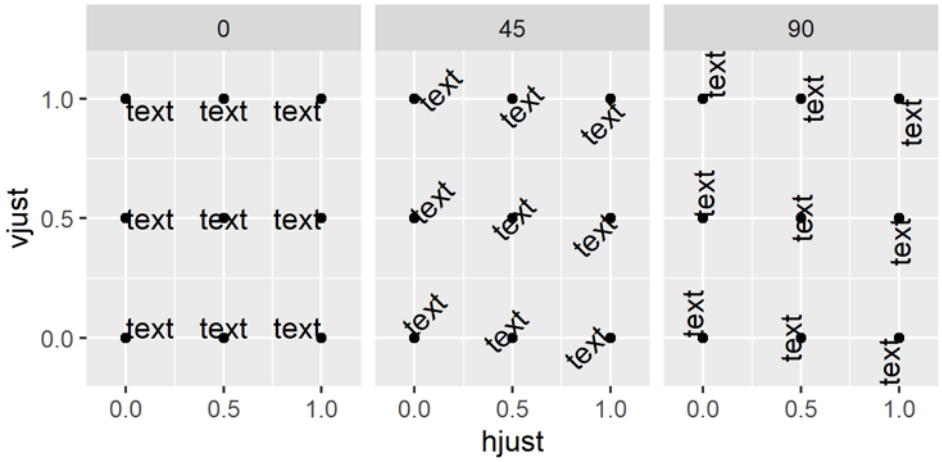
รูปที่ 12-48 Text annotations

รูปแรกแสดงข้อความด้านบนของจุดโดยการแยกสีตามตัวแปร cyl แสดงด้วยสีแดง สีเขียว และสีฟ้า เมื่อ cyl เท่ากับ 4, 6 และ 8 ตามลำดับ รูปที่สองแสดงข้อความ (Label) ด้วยตัวหนา (Bold) สีแดง

การใช้ **hjust, vjust** และ **angle** แสดงดังรูปต่อไป **hjust** เท่ากับ 0 หมายถึงข้อความวางชิดซ้าย (Left justification) **hjust** เท่ากับ 0.5 หมายถึงข้อความวางกึ่งกลาง (Center justification) **hjust** เท่ากับ 1 หมายถึงข้อความวางชิดขวา (Right justification) ถ้า **hjust** ติดลบ หมายถึงข้อความวางห่างจากตำแหน่งอ้างอิงไปทางขวา **vjust** เท่ากับ 0 หมายถึงข้อความวางด้านบน **vjust** เท่ากับ 0.5 หมายถึงข้อความวางกึ่งกลาง **vjust** เท่ากับ 1 หมายถึงข้อความวางด้านล่าง ถ้า **vjust** ติดลบ หมายถึงข้อความวางด้านบนห่างจากตำแหน่งอ้างอิงขึ้นไปอีก ส่วน **angle** เท่ากับ 0 หมายถึงข้อความไม่มีการหมุน **angle** เท่ากับ 45 หมายถึงข้อความมีการหมุนทวนเข็มนาฬิกา 45 องศา **angle** เท่ากับ 90 หมายถึงข้อความมีการหมุนทวนเข็มนาฬิกา 90 องศา และถ้า **angle** มีค่าติดลบ หมายถึงข้อความมีการหมุนตามเข็มนาฬิกา ดังนี้

```
R> td <- expand.grid(
     hjust = c(0, 0.5, 1),
     vjust = c(0, 0.5, 1),
     angle = c(0, 45, 90),
     text = "text"
   )

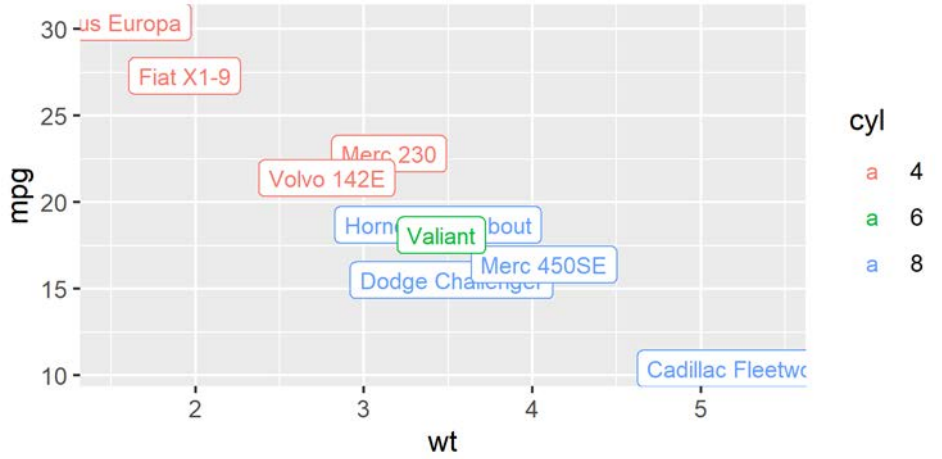
R> ggplot(td, aes(x = hjust, y = vjust)) +
     geom_point() +
     geom_text(aes(label = text, angle = angle, hjust = hjust, vjust = vjust)) +
     facet_grid(~angle) +
     scale_x_continuous(breaks = c(0, 0.5, 1), expand = c(0, 0.2)) +
     scale_y_continuous(breaks = c(0, 0.5, 1), expand = c(0, 0.2))
```



รูปที่ 12-49 hjust vjust and angle

คำสั่ง `geom_label()` คล้ายกับคำสั่ง `geom_text()` แต่มีการสร้างกรอบสี่เหลี่ยมล้อมรอบข้อความที่ต้องการแสดงผล ดังนี้

```
# geom_label()
R> sp + geom_label(aes(label = rownames(df), color = cyl), size = 3)
```



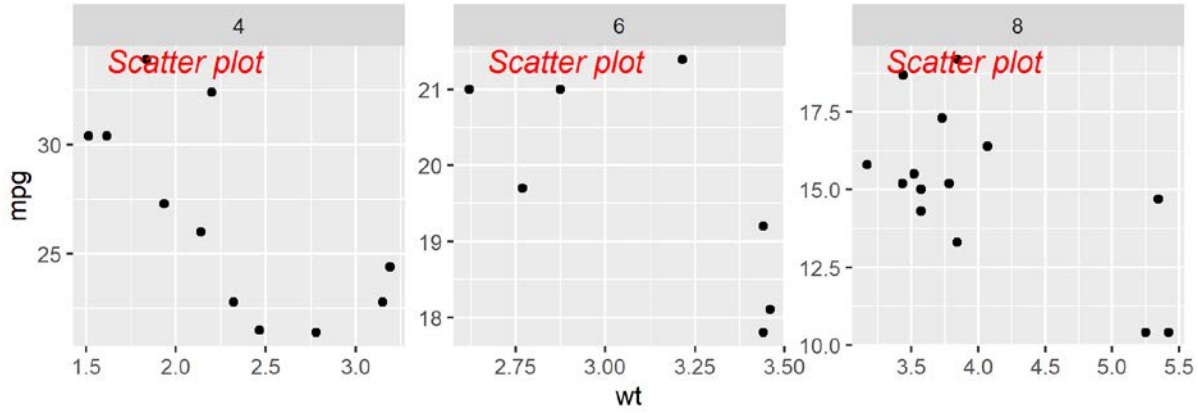
รูปที่ 12-50 Text annotations with geom_label()

สามารถสร้าง Static text annotation ซึ่งจะทำการพล็อตข้อความในทุก ๆ Panel โดยไม่ขึ้นอยู่กับสเกลของแกน X และแกน Y ได้ด้วยคำสั่ง `annotation_custom()` และ `textGrob()` ดังนี้

```
# Create a scatter plot
R> sp2 <- ggplot(mtcars, aes(x = wt, y = mpg, label = rownames(mtcars))) +
     geom_point()

# Create a static text annotation
R> library(grid)
R> grob <- grobTree(textGrob("Scatter plot", x = 0.1, y = 0.95, hjust = 0,
                             gp = gpar(col = "red", fontsize = 13,
                                       fontface = "italic")))
```

```
# Plots
R> sp2 + annotation_custom(grob) +
     facet_wrap(~cyl, scales = "free")
```



รูปที่ 12-51 Text annotations with annotation_custom()

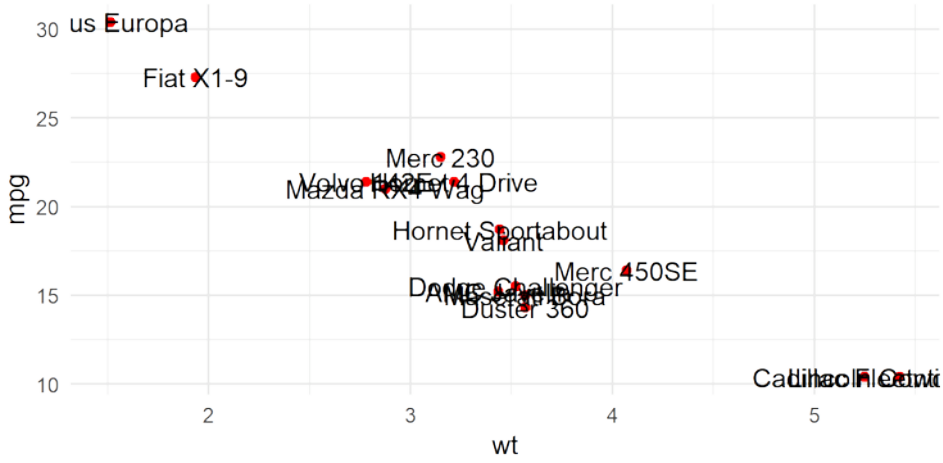
ทั้ง 3 รูปด้านบนแสดงข้อความในลักษณะ Static text annotation ซึ่งแสดงข้อความในทุก ๆ Panel โดยไม่ขึ้นอยู่กับสเกลของแกน X และแกน Y

การพล็อตด้วยคำสั่ง `geom_text()` หรือ `geom_label()` โดยปกติข้อความจะแสดงซ้อนทับกัน ทำให้อ่านได้ยาก ดังนี้

```
# Take a subset of 15 random points
R> set.seed(1234)
R> data_sample <- sample(1:32, 15)
R> df <- mtcars[data_sample, ]

R> p <- ggplot(df, aes(wt, mpg)) +
     geom_point(color = 'red') +
     theme_minimal(base_size = 10)

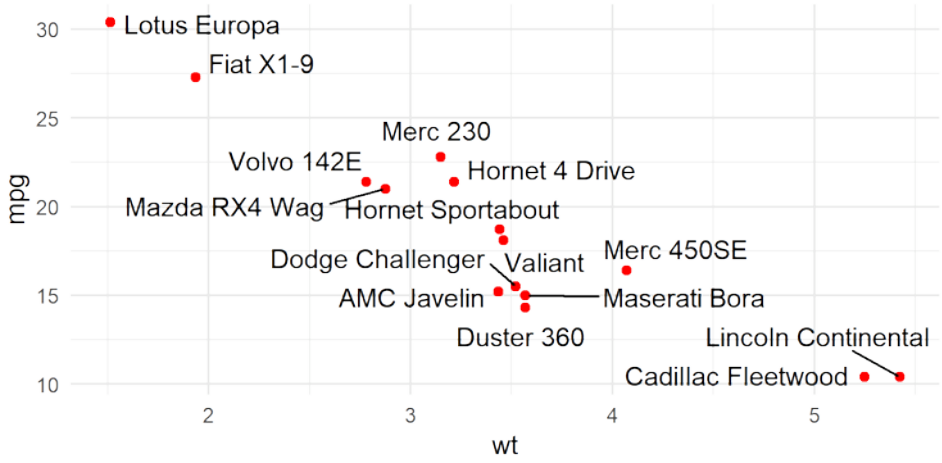
# Add text annotations using geom_text
R> p + geom_text(aes(label = rownames(df)), size = 3.5)
```



รูปที่ 12-52 Text annotations with geom_text()

สามารถพล็อตให้ข้อความเหลื่อมกันเพื่อให้อ่านข้อความได้ง่ายโดยใช้แพ็กเกจ **ggrepel** ด้วยคำสั่ง **geom_text_repel()** และ **geom_label_repel()** โดยระบุพารามิเตอร์ต่าง ๆ เหมือนกับการใช้คำสั่ง **geom_text()** และ **geom_label()** ตามลำดับ ดังนี้

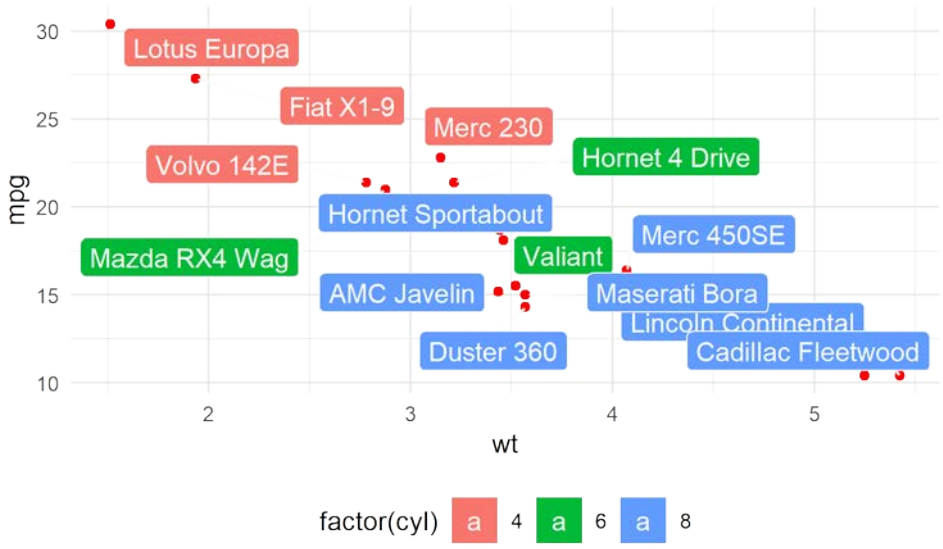
```
# Use ggrepel::geom_text_repel
R> require("ggrepel")
R> set.seed(42)
R> p + geom_text_repel(aes(label = rownames(df)), size = 3.5)
```



รูปที่ 12-53 Text annotations with geom_text_repel()

รูปด้านบนแสดงข้อความเหลื่อมกันและมีเส้นชี้ไปยังข้อความที่กำกับอยู่ ส่วนรูปต่อไปแสดงกรอบสี่เหลี่ยมล้อมรอบข้อความ และแยกสีกรอบสี่เหลี่ยมรอบข้อความตามตัวแปร cyl

```
# Use ggrepel::geom_label_repel and
# Change color by groups
R> set.seed(42)
R> p + geom_label_repel(aes(label = rownames(df),
                            fill = factor(cyl)), color = 'white', size = 3.5) +
     theme(legend.position = "bottom")
```



รูปที่ 12-54 Text annotations with geom_label_repel()

12.13 การเพิ่มเส้นตรงในกราฟ (Add Straight Lines to a Plot: Horizontal, Vertical and Regression Lines)

หัวข้อนี้จะกล่าวถึงการเพิ่มเส้นตรงในกราฟได้แก่ เส้นนอน เส้นตั้ง เส้น Regression และ บางส่วนของเส้นตรง (Segment) Geometry ประเภทต่าง ๆ ที่สามารถพล็อตได้ประกอบด้วย

- `geom_hline()` สำหรับพล็อตเส้นนอน
- `geom_vline()` สำหรับพล็อตเส้นตั้ง
- `geom_abline()` สำหรับพล็อตเส้น Regression
- `geom_segment()` สำหรับพล็อตบางส่วนของเส้นตรง (Segment)

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล mtcars ที่ติดตั้งมาพร้อมกับ base R ดังนี้

```
# Simple scatter plots
R> sp <- ggplot(data = mtcars, aes(x = wt, y = mpg)) +
     geom_point()
```

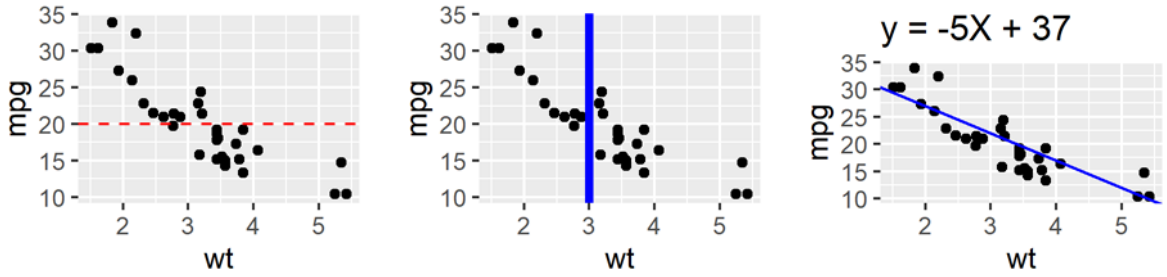
คำสั่ง `geom_hline()` ต้องระบุพารามิเตอร์ `yintercept` สำหรับกำหนดตำแหน่งจุดตัดแกน Y, คำสั่ง `geom_vline()` ต้องระบุพารามิเตอร์ `xintercept` สำหรับกำหนดตำแหน่งจุดตัดแกน X, คำสั่ง `geom_abline()` ต้องระบุพารามิเตอร์ `intercept` และ `slope` สำหรับกำหนดตำแหน่งจุดตัดแกน Y และค่าความชัน ตามลำดับ ส่วนคำสั่ง `geom_segment()` ต้องระบุพารามิเตอร์ `x, y, xend` และ `yend` สำหรับค่า coordinate X เริ่มต้น, ค่า coordinate Y เริ่มต้น, ค่า coordinate X สิ้นสุด, ค่า coordinate Y สิ้นสุด ตามลำดับ นอกจากนั้นยังสามารถควบคุมการแสดงผลของเส้นด้วยพารามิเตอร์ `linetype, color` และ `size`

การพล็อตเส้นตรงด้วยคำสั่งดังกล่าว แสดงตัวอย่างได้ดังนี้

```
# Add horizontal line at y = 20; change line type and color
R> sp + geom_hline(yintercept = 20, linetype = "dashed", color = "red")

# Add vertical line at x = 3; change line type, color and size
R> sp + geom_vline(xintercept = 3, color = "blue", size = 1.5)

# Add regression line
R> sp + geom_abline(intercept = 37, slope = -5, color = "blue")+
     ggtitle("y = -5X + 37")
```



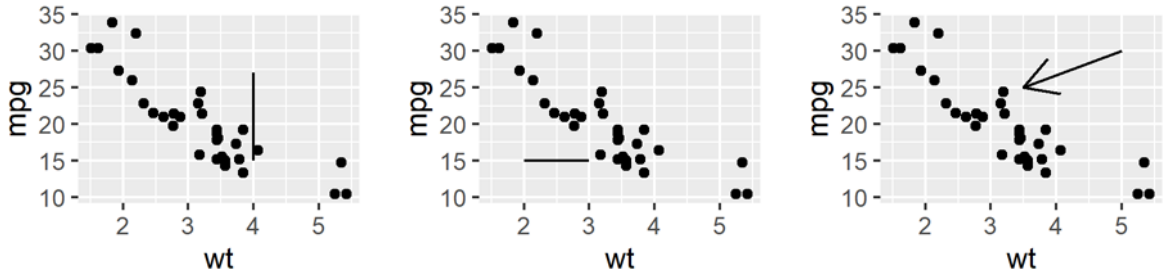
รูปที่ 12-55 Add straight lines: geom_hline(), geom_vline() and geom_abline()

รูปแรกแสดงเส้นประแนวนอนสีแดง รูปที่สองแสดงเส้นหนาแนวตั้งสีน้ำเงิน รูปสุดท้ายเส้นสีน้ำเงินแสดง Regression line และชื่อเรื่อง (Main title)

```
# Add a vertical line segment
R> sp + geom_segment(aes(x = 4, y = 15, xend = 4, yend = 27))

# Add horizontal line segment
R> sp + geom_segment(aes(x = 2, y = 15, xend = 3, yend = 15))

# Add arrow at the end of the segment
R> require(grid)
R> sp + geom_segment(aes(x = 5, y = 30, xend = 3.5, yend = 25),
                     arrow = arrow(length = unit(0.5, "cm")))
```



รูปที่ 12-56 Add straight lines: geom_segment()

รูปแรกแสดงเส้นแนวตั้ง รูปที่สองแสดงเส้นแนวนอน รูปสุดท้ายแสดงเส้นพร้อมหัวลูกศร

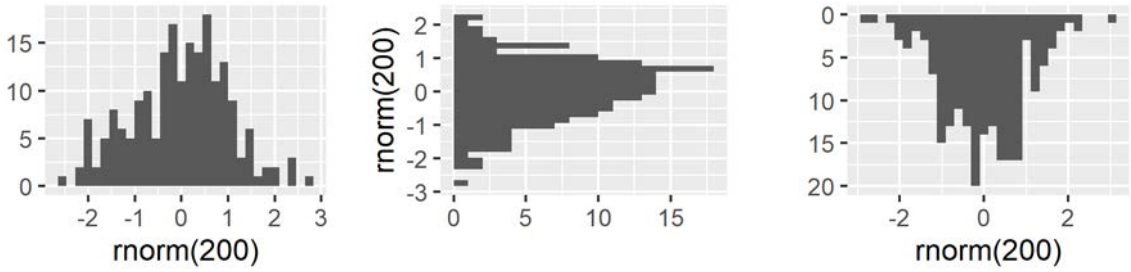
12.14 การหมุนกราฟ (Rotate a Plot: Flip and Reverse)

หัวข้อนี้จะกล่าวถึงการหมุนกราฟ (Rotate a plot) ด้วยคำสั่ง **coord_flip()** และการสลับค่าค่าสูงสุด/ค่าต่ำสุดในแกน X และแกน Y ด้วยคำสั่ง **scale_x_reverse(), scale_y_reverse()** ดังนี้

```
R> set.seed(123)
# Basic histogram
R> hp <- qplot(x = rnorm(200), geom = "histogram")
R> hp

# Horizontal histogram
R> hp + coord_flip()

# Y axis reversed
R> hp + scale_y_reverse()
```



รูปที่ 12-57 Rotate a Plot: Flip and Reverse

รูปแรกเป็นฮิสโตแกรมโดยทั่วไปแกน X แสดงข้อมูล แกน Y แสดงจำนวนข้อมูล (Count) รูปที่สองเป็นการสลับแกน X และแกน Y รูปสุดท้ายเป็นการสลับค่าค่าสูงสุด/ค่าต่ำสุดในแกน Y

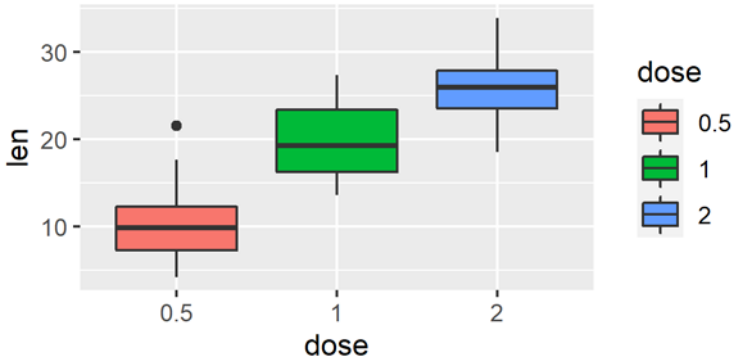
12.15 การพล็อตกราฟย่อย (Facets)

Facets เป็นการพล็อตกราฟย่อย (Subplots) จากข้อมูลตัวแปรแบ่งกลุ่มหรือไม่ต่อเนื่อง (Discrete variables) ตั้งแต่ 1 ตัวขึ้นไป

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล ToothGrowth ที่ติดตั้งมาพร้อมกับ base **R** การพล็อตปกติด้วยคำสั่ง **geom_boxplot()** แสดงได้ดังนี้

```
# Load data and convert dose to a factor variable
R> data("ToothGrowth")
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)

# Box plots
R> p <- ggplot(ToothGrowth, aes(x = dose, y = len, group = dose)) +
     geom_boxplot(aes(fill = dose))
R> p
```

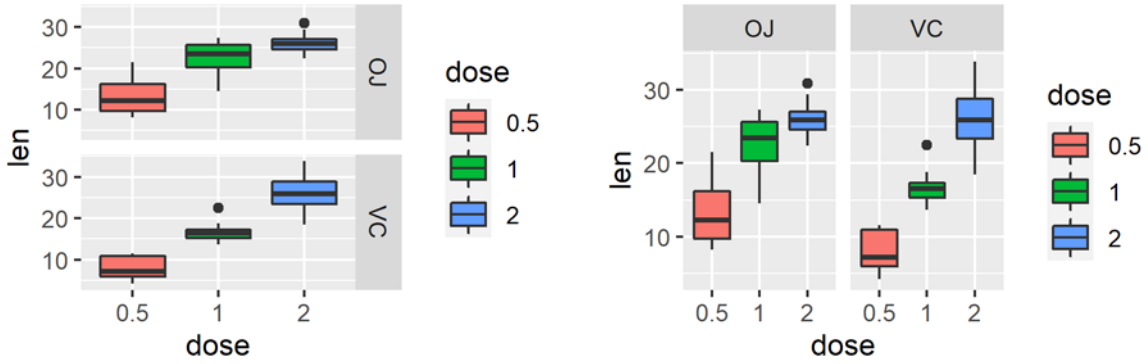


รูปที่ 12-58 Plots without facets

การพล็อตกราฟย่อย (Subplots) ด้วยคำสั่ง `facet_grid()` แสดงได้ดังนี้

```
# Split in vertical direction
R> p + facet_grid(supp ~ .)

# Split in horizontal direction
R> p + facet_grid(. ~ supp)
```

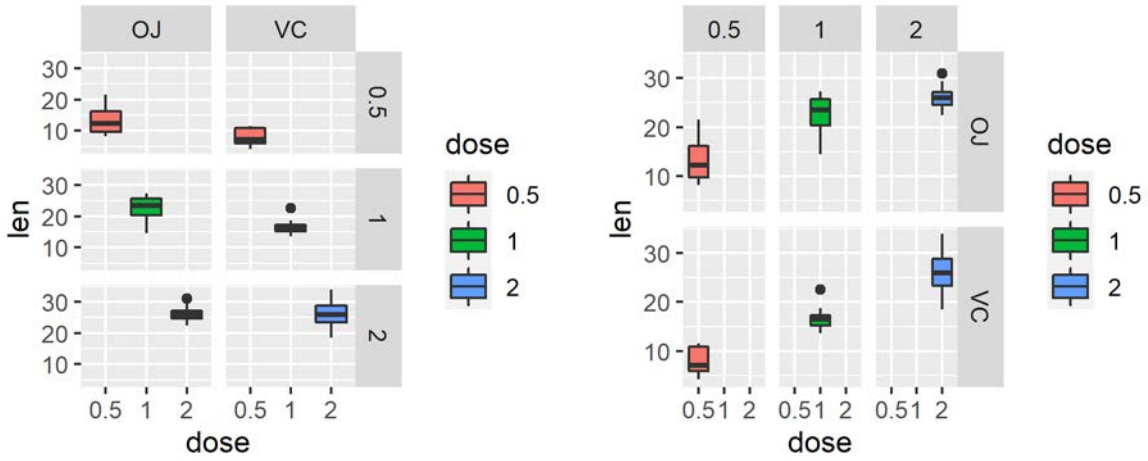


รูปที่ 12-59 Plots with facet_grid(): one discrete variable

รูปแรกเป็นการพล็อตกราฟย่อย (Subplots) ในแนวตั้งจากข้อมูลตัวแปร supp ซึ่งประกอบด้วย OJ และ VC รูปที่สองเป็นการพล็อตกราฟย่อย (Subplots) ในแนวนอนจากข้อมูลตัวแปร supp

```
# Facet by two variables: dose and supp.
# Rows are dose and columns are supp
R> p + facet_grid(dose ~ supp)

# Facet by two variables: reverse the order of the 2 variables
# Rows are supp and columns are dose
R> p + facet_grid(supp ~ dose)
```

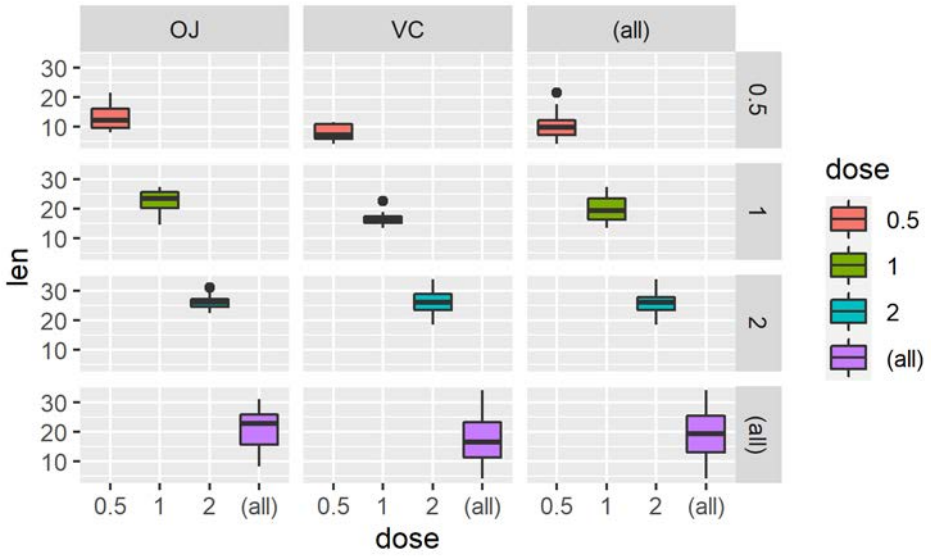


รูปที่ 12-60 Plots with facet_grid(): two discrete variables

รูปด้านบนเป็นการพล็อตกราฟย่อย (Subplots) ด้วยตัวแปรแบ่งกลุ่ม 2 ตัว ได้แก่ dose และ supp รูปแรกเป็นการพล็อตในแนวนอนจากข้อมูลตัวแปร dose ซึ่งประกอบด้วย 0.5, 1 และ 2 และพล็อตในแนวตั้งจากข้อมูลตัวแปร supp ซึ่งประกอบด้วย OJ และ VC รูปที่สองเป็นการพล็อตในแนวนอนจากข้อมูลตัวแปร supp ซึ่งประกอบด้วย OJ และ VC และการพล็อตในแนวตั้งจากข้อมูลตัวแปร dose ซึ่งประกอบด้วย 0.5, 1 และ 2

สามารถเพิ่มกราฟย่อยสำหรับแสดงข้อมูลทั้งหมด โดยระบุพารามิเตอร์ `margins = TRUE` ดังนี้

```
R> p + facet_grid(dose ~ supp, margins = TRUE)
```

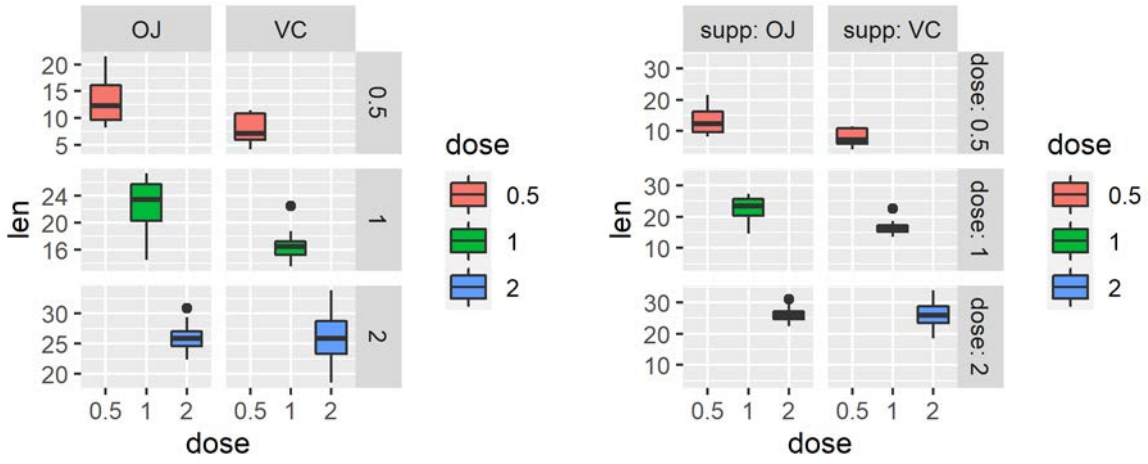


รูปที่ 12-61 facet_grid() with margins

รูปด้านบนจะเห็นว่าเป็นการแสดงกราฟย่อยสำหรับแสดงข้อมูลทั้งหมด ทั้งตัวแปร dose, supp และ len

ค่าโดยปริยายกำหนดให้ทุกกราฟย่อยใช้สเกลเดียวกัน นั้นคือพารามิเตอร์ `scales = "fixed"` ผู้ใช้สามารถกำหนดให้สเกลแกน X และแกน Y เป็นอิสระต่อกันได้ด้วยพารามิเตอร์ `scales` เท่ากับ `free`, `free_x` หรือ `free_y` และสามารถระบุพารามิเตอร์ `labeller` ในการแสดงข้อความแต่ละแกน ได้ดังนี้

```
R> p + facet_grid(dose ~ supp, scales = 'free')
R> p + facet_grid(dose ~ supp, labeller = label_both)
```



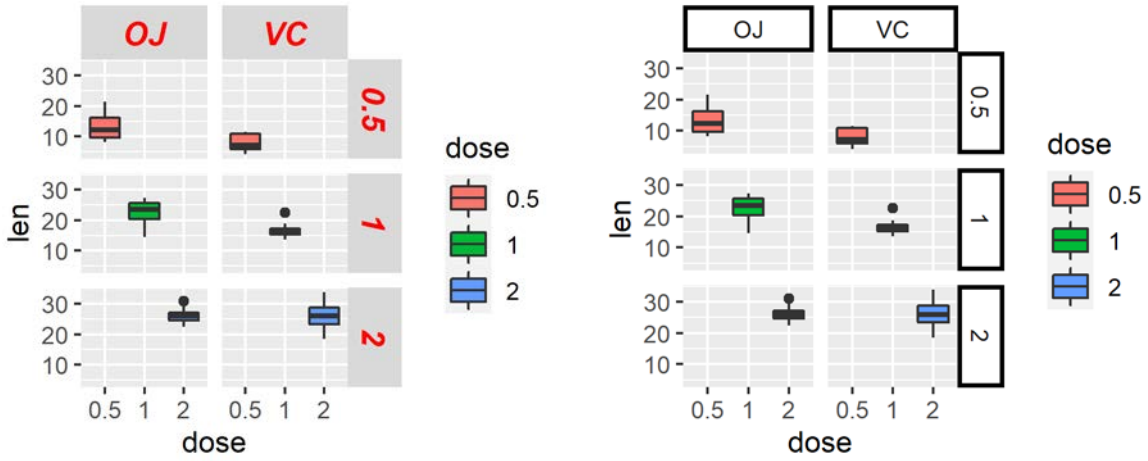
รูปที่ 12-62 facet_grid() with scales and labeller

รูปแรกแกน Y แสดงสเกลที่เป็นอิสระต่อกัน ส่วนรูปขวาแกน Y ใช้สเกลเดียวกันสำหรับตัวแปร dose ค่าต่าง ๆ ได้แก่ 0.5, 1 และ 2

สามารถปรับเปลี่ยนการแสดงข้อความแต่ละแกนได้ด้วยคำสั่ง `theme()` โดยระบุพารามิเตอร์ `element_text()` หรือ `element_rect()` ให้กับตัวแปรที่ต้องการแสดงผลด้วยพารามิเตอร์ `size`, `color`, `face`, `fill` หรือ `linetype` (การใช้คำสั่ง `element_text()` และ `element_rect()` กล่าวถึงในหัวข้อ 12.11) ได้ดังนี้

```
# Change facet text font. Possible values for the font style:
# 'plain', 'italic', 'bold', 'bold.italic'.
R> p + facet_grid(dose ~ supp) +
     theme(strip.text.x = element_text(size = 12, color = "red",
                                       face = "bold.italic"),
           strip.text.y = element_text(size = 12, color = "red",
                                       face = "bold.italic"))

# Change the apperance of the rectangle around facet label
R> p + facet_grid(dose ~ supp) +
     theme(strip.background = element_rect(colour = "black", fill = "white",
                                           size = 1.5, linetype = "solid"))
```

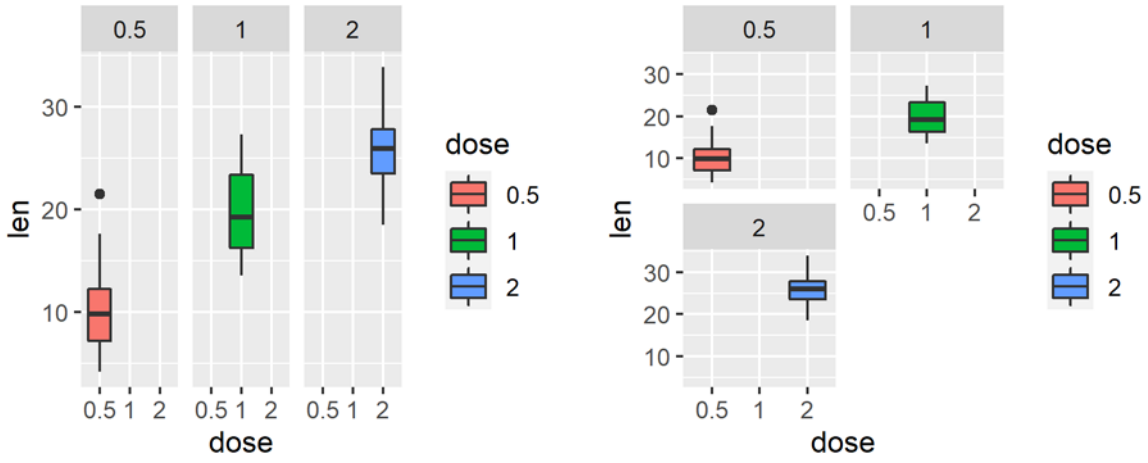


รูปที่ 12-63 facet_grid() with custom themes

รูปแรกแสดงข้อความหัวเรื่องสำหรับแกน X และแกน Y ด้วยฟอนต์สีแดงตัวหนาและเอียงขนาด 12 pts รูปที่สองแสดงเส้นกรอบสี่เหลี่ยมรอบข้อความหัวเรื่องสำหรับแกน X และแกน Y ด้วยกรอบสีดำขนาด 1.5 pts พื้นสีขาว อักษรสีดำ

คำสั่ง `facet_wrap()` เป็นการพล็อตกราฟย่อยสำหรับตัวแปรแบ่งกลุ่ม ซึ่งปกติจะแสดงผลเป็น 1 มิติ (อาจจะแสดงเป็น 1 แถวหรือ 1 คอลัมน์) ให้แสดงผลเป็น 2 มิติ (ทั้งแถวและคอลัมน์) โดยระบุพารามิเตอร์ `ncol` เท่ากับจำนวนคอลัมน์ที่ต้องการ, `nrow` เท่ากับจำนวนแถวที่ต้องการ รวมทั้งควบคุมการแสดงผลอื่น ๆ เช่น `scales, labeller` ดังที่ได้กล่าวถึงในหัวข้อก่อนหน้านี้ ดังนี้

```
R> bp + facet_wrap(~ dose)
R> bp + facet_wrap(~ dose, ncol = 2)
```



รูปที่ 12-64 Plots with facet_wrap()

รูปแรกแสดงกราฟย่อย 1 แถวแต่ละกราฟย่อยแสดงข้อมูลตัวแปร dose รูปที่สองเป็นข้อมูลกราฟย่อยเดียวกันแต่แสดงเป็น 2 แถว 2 คอลัมน์เรียงจากซ้ายไปขวาและบนลงล่าง ตามลำดับ

12.16 การจัดวางตำแหน่งองค์ประกอบในกราฟ (Position Adjustments)

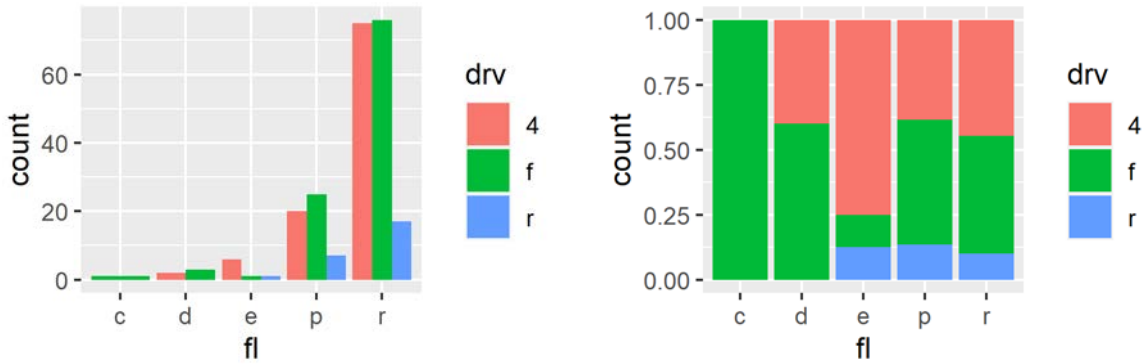
หัวข้อนี้จะกล่าวถึงการจัดวางตำแหน่งองค์ประกอบต่าง ๆ ในกราฟ (Position adjustments) เช่น แท่งกราฟ จุด เป็นต้น โดยใช้พารามิเตอร์ `position` ซึ่งประกอบด้วย `"dodge"` หมายถึงจัดวางแต่ละแท่งกราฟอยู่ข้างกัน (Side by side), `"fill"` หมายถึงจัดวางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง โดยปรับความสูงของแท่งกราฟทั้งหมดให้มีความสูงเท่ากัน, `"stack"` หมายถึงจัดวางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง, `"jitter"` หมายถึงจัดวางจุดแต่ละจุดให้เหลี่ยมกัน

ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล mpg ที่ติดตั้งมาพร้อมกับ `ggplot2` การใช้พารามิเตอร์ `position` ดังกล่าว แสดงได้ดังนี้

```
R> p <- ggplot(mpg, aes(fl, fill = drv))

# Arrange elements side by side
R> p + geom_bar(position = "dodge")

# Stack objects on top of one another,
# and normalize to have equal height
R> p + geom_bar(position = "fill")
```

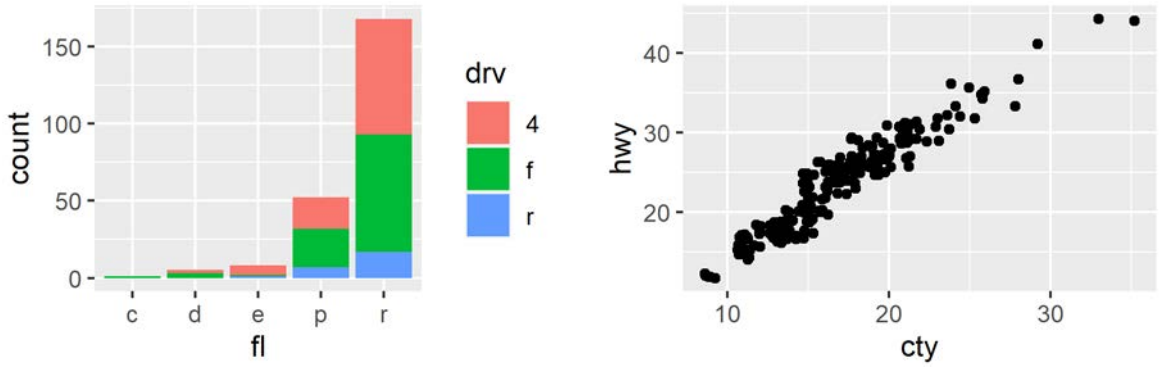


รูปที่ 12-65 Plots with positions: dodge and fill

รูปแรกแสดงกราฟแท่งแสดงสีตามตัวแปร drv เรียงอยู่ข้างกัน (Side by side) รูปที่สองเป็นข้อมูลเดียวกันแต่วางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง โดยปรับความสูงของแท่งกราฟทั้งหมดให้มีความสูงเท่ากัน

```
# Stack elements on top of one another
R> p + geom_bar(position = "stack")

# Add random noise to X and Y position
# of each element to avoid overplotting
R> ggplot(mpg, aes(cty, hwy)) +
     geom_point(position = "jitter")
```

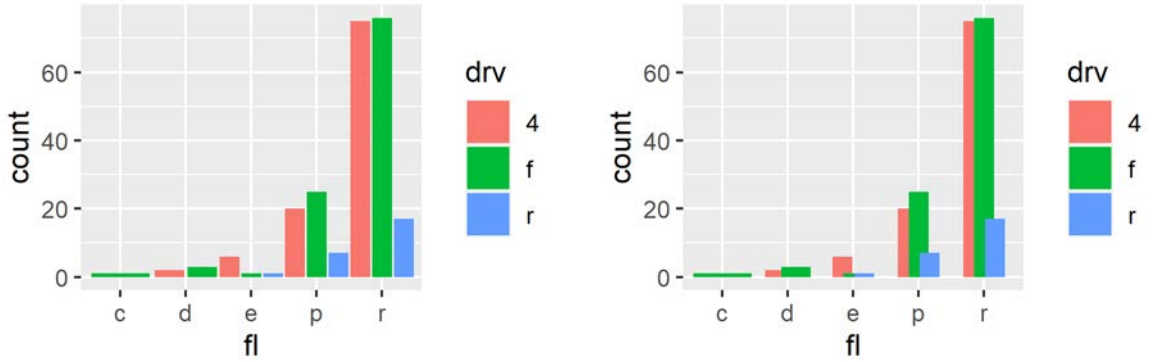


รูปที่ 12-66 Plots with positions: stack and jitter

รูปแรกแสดงกราฟแท่งแสดงสีตามตัวแปร drv จัดวางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง (Stack) รูปที่สองเป็น Scatter plots โดยให้จุดแต่ละจุดเหลื่อมกันออกมาในแนวตั้งและแนวนอน

ผู้ใช้สามารถควบคุมการแสดงผลองค์ประกอบต่าง ๆ ในกราฟด้วยคำสั่ง `position_dodge()`, `position_fill()`, `position_stack()` และ `position_jitter()` เช่นระบุพารามิเตอร์ `width` สำหรับคำสั่ง `position_dodge()` เพื่อกำหนดระยะห่างระหว่างแท่งกราฟ ดังนี้

```
R> p + geom_bar(position = position_dodge(width = 1))
R> p + geom_bar(position = position_dodge(width = 0.5))
```



รูปที่ 12-67 Bar charts with position_dodge()

รูปแรกแสดงกราฟแท่งแสดงสีตามตัวแปร drv จัดวางแต่ละแท่งกราฟอยู่ข้างกัน (Side by side) ด้วยคำสั่ง `position_dodge()` โดยระบุอากิวเมนต์ `width` เท่ากับ 1 รูปที่สองเป็นข้อมูลชุดเดียวกัน แต่ระบุอากิวเมนต์ `width` เท่ากับ 0.5 จะเห็นว่ากราฟแต่ละแท่งมีการวางซ้อนกันด้านข้างเล็กน้อย

12.17 Coordinate Systems

Coordinate systems ในการพล็อตประกอบด้วยคำสั่งต่าง ๆ ดังนี้

- `coord_cartesian()` Cartesian coordinate system เป็นค่าโดยปริยายสำหรับการพล็อตโดยทั่วไป
- `coord_fixed()` เป็น Cartesian coordinate system ที่แสดงความสัมพันธ์ระหว่างสเกลแกน X และสเกลแกน Y ค่าโดยปริยายกำหนดให้ ratio = 1
- `coord_flip()` เป็นการสลับแกน X และแกน Y (Flipped cartesian coordinates)
- `coord_polar()` เป็น Polar coordinate system ซึ่งเหมาะกับการพล็อตกราฟวงกลม (Pie charts) ลักษณะการพล็อตเหมือนกับกราฟแท่งแตกต่างกันที่แท่งกราฟเรียงตัวอยู่ใน Polar coordinates
- `coord_trans()` เป็นการแปลง Cartesian coordinate system
- `coord_map()` เป็นการสร้าง Map projections สำหรับการพล็อตแผนที่

พารามิเตอร์ต่าง ๆ สำหรับ `coord_cartesian(), coord_fixed()` และ `coord_flip()` ประกอบด้วย

- `xlim, ylim` เป็นการกำหนดค่าในแกน X และแกน Y ตามลำดับ
- `ratio` เป็นอัตราส่วนระหว่างสเกลแกน Y/X
- `...` พารามิเตอร์อื่น ๆ ที่ผ่านเข้าไปในคำสั่ง `coord_cartesian()`

พารามิเตอร์ต่าง ๆ สำหรับ `coord_polar()` ประกอบด้วย

- `theta` เป็นการกำหนดตัวแปรที่ต้องการแสดงผล ค่าโดยปริยาย `theta = "x"`
- `start` เป็นค่า offset จากจุดเริ่มต้นที่ 12 นาฬิกา (Radians)
- `direction` 1 หมายถึงตามเข็มนาฬิกา, -1 หมายถึงทวนเข็มนาฬิกา

พารามิเตอร์ต่าง ๆ สำหรับ `coord_trans()` ประกอบด้วย

- `x, y` เป็นการกำหนดค่า (ฟังก์ชัน) ที่ต้องการแปลงสำหรับแกน X และแกน Y ตามลำดับ เช่น `"log10", "sqrt"` เป็นต้น
- `xlim, ylim` เป็นการกำหนดค่าต่ำสุดและค่าสูงสุดให้กับแกน X และแกน Y ตามลำดับ

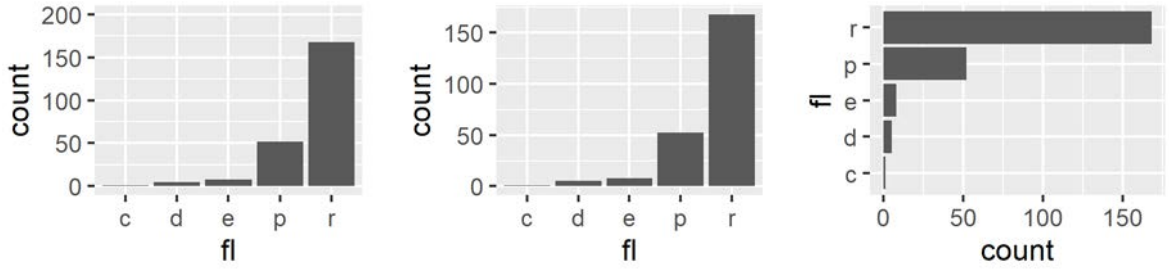
ข้อมูลที่จะใช้ในการพล็อตเป็นข้อมูล mpg ที่ติดตั้งมาพร้อมกับ `ggplot2` การพล็อตด้วย Coordinate systems ดังกล่าวข้างต้น แสดงตัวอย่างได้ดังนี้

```
R> p <- ggplot(mpg, aes(fl)) +
     geom_bar()

# change y limits
R> p + coord_cartesian(ylim = c(0, 200))

# change ratio
R> p + coord_fixed(ratio = 1/50)
```

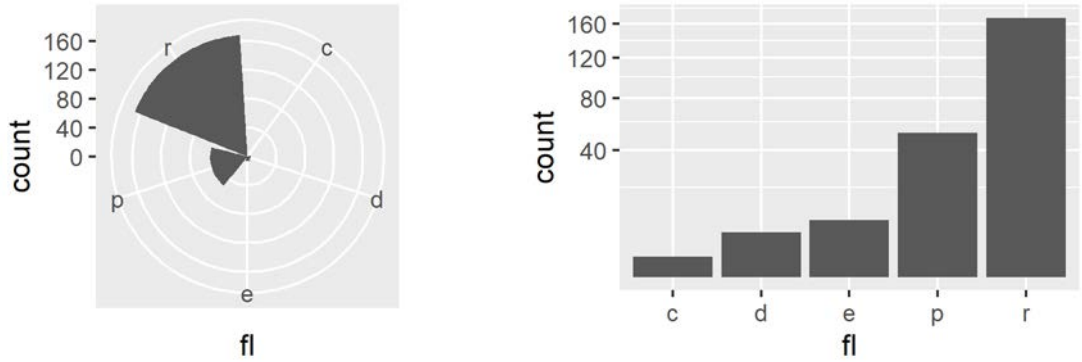
```
# flip the plots
R> p + coord_flip()
```



รูปที่ 12-68 Plots with coordinate systems: cartesian, fixed and flip

ทั้ง 3 รูปด้านบนแสดงน้ำมันเชื้อเพลิงที่ใช้แต่ละประเภท รูปแรกแสดงเป็นกราฟแท่งโดยกำหนดให้แกน Y เริ่มที่ 0 สิ้นสุดที่ 200 รูปที่สองเป็นการพล็อตโดยกำหนด aspect ration เท่ากับ 1/50 รูปสุดท้ายเป็นการสลับแกน X และแกน Y

```
R> p + coord_polar(theta = "x", direction = 1)
R> p + coord_trans(y = "sqrt") +
     scale_y_continuous(breaks = c(0, 40, 80, 120, 160))
```



รูปที่ 12-69 Plots with coordinate systems: polar and trans

รูปแรกเป็นกราฟวงกลมที่พล็อตโดยการ Map มุมที่จะพล็อต (Polar coordinate system) ด้วยแกน x (Cartesian coordinate system) และแสดงค่าตัวแปร fl ได้แก่ c, d, e, p และ r ตามเข็มนาฬิกาตามลำดับ รูปที่สองเป็นการแปลง Cartesian coordinate system ในแกน Y ด้วยค่ารากที่ 2 (Square root)

12.18 สรุปท้ายบท

บทนี้กล่าวถึงกราฟิกพื้นฐาน (Graphical primatives) ในการพล็อต ได้แก่ รูปหลายเหลี่ยม (Polygon), เส้นต่อเนื่อง (Path), กราฟ Ribbon, บางส่วนของเส้น (Segment), เส้นโค้ง, รูปสี่เหลี่ยม รวมทั้งพารามิเตอร์ที่ใช้แสดงค่าต่าง ๆ ในกราฟ ได้แก่ ชื่อเรื่อง (Main title) ชื่อแกน (Axis labels) สัญลักษณ์ (Legend) ตำแหน่งและรูปแบบต่าง ๆ ของสัญลักษณ์ (Legend position and appearance) สี (Color) รูปร่างจุด สี และขนาด (Point shapes, colors and size) ชนิดเส้น (Line types) การกำหนดขอบเขตแกน

(Axis limits) การแปลงแกนด้วย log และ sqrt (Axis transformations: log and sqrt) การกำหนดแกนในหน่วยวัน (Date Axis) การกำหนดรูปแบบการแสดงผลบนแกน (Axis ticks : customize tick marks and labels, reorder and select items) ธีมและสีพื้น (Themes and background colors) ข้อความ (Text annotations) การเพิ่มเส้นตรงในกราฟ (Add straight lines to a plot: horizontal, vertical and regression lines) การหมุนกราฟ (Rotate a plot: flip and reverse) การพล็อตกราฟย่อย (Facets) การจัดวางตำแหน่งองค์ประกอบในกราฟ (Position adjustments) และ Coordinate systems

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	**คำอธิบาย**
`ggplot()`	คำสั่งในการพล็อต
`geom_polygon()`	Geometry สำหรับพล็อตรูปหลายเหลี่ยม (Polygon)
`geom_path()`	Geometry สำหรับพล็อตเส้นต่อเนื่อง (Path)
`geom_ribbon()`	Geometry สำหรับพล็อตกราฟ Ribbon
`geom_segment()`	Geometry สำหรับพล็อตบางส่วนของเส้น (Segment)
`geom_curve()`	Geometry สำหรับพล็อตเส้นโค้ง
`geom_rect()`	Geometry สำหรับพล็อตรูปสี่เหลี่ยม
`geom_line()`	Geometry สำหรับพล็อตกราฟเส้น
`geom_point()`	Geometry สำหรับพล็อต Scatter plots
`geom_hline()`	Geometry สำหรับพล็อตเส้นนอน
`geom_vline()`	Geometry สำหรับพล็อตเส้นตั้ง
`geom_abline()`	Geometry สำหรับพล็อตเส้น Regression
`geom_label()`	Geometry สำหรับพล็อตข้อความพร้อมกรอบสี่เหลี่ยม
`geom_text()`	Geometry สำหรับพล็อตข้อความ
`geom_text_repel()`	Geometry สำหรับพล็อตข้อความเหลื่อมกัน
`geom_label_repel()`	Geometry สำหรับพล็อตข้อความเหลื่อมกันพร้อมกรอบสี่เหลี่ยม
`ggtitle()`	คำสั่งสำหรับพล็อตชื่อเรื่อง (Main title)
`xlab()`	คำสั่งสำหรับพล็อตชื่อแกน X
`ylab()`	คำสั่งสำหรับพล็อตชื่อแกน Y
`labs()`	คำสั่งสำหรับพล็อตชื่อเรื่อง (Main title) ชื่อแกน X และชื่อแกน Y
`scale_x_discrete()`	คำสั่งในการกำหนดลำดับในการพล็อตตามแกน X
`scale_fill_discrete()`	คำสั่งในการกำหนดชื่อหัวข้อสัญลักษณ์ (Legend title) และชื่อสัญลักษณ์ (Legend labels)
`stat_summary()`	คำสั่งในการแสดงค่าเฉลี่ยทางคณิตศาสตร์ เช่น Mean, Median

คำสั่ง	คำอธิบาย
guides()	คำสั่งในการกำหนดสเกล เช่น ลำดับการแสดงหัวข้อสัญลักษณ์ (Legend title) และกำหนดสี ขนาด และรูปร่างสัญลักษณ์
scale_size()	คำสั่งในการกำหนดขนาดของสเกล
scale_shape()	คำสั่งในการกำหนดรูปร่างของสเกล
scale_color_hue()	คำสั่งในการกำหนดความสว่าง (Lightness)
scale_fill_hue()	คำสั่งในการกำหนดความเข้มของสี (Intensity of color)
scale_color_gradient()	คำสั่งในการกำหนดสีต่อเนื่องระหว่าง 2 สี
scale_color_gradient2()	คำสั่งในการกำหนดสีต่อเนื่องระหว่าง 2 สีแบบ diverging gradients
scale_color_gradientn()	คำสั่งในการกำหนดสีต่อเนื่องระหว่าง n สี
scale_fill_gradient()	คำสั่งในการเติมสีต่อเนื่องระหว่าง 2 สี
scale_fill_gradient2()	คำสั่งในการเติมสีต่อเนื่องระหว่าง 2 สีแบบ diverging gradients
scale_fill_gradientn()	คำสั่งในการเติมสีต่อเนื่องระหว่าง n สี
scale_color_manual()	คำสั่งในการกำหนดสีแบบ Manual
scale_color_brewer()	คำสั่งในการกำหนดสีด้วยจานสี (Palettes)
scale_color_grey()	คำสั่งในการกำหนดสีด้วยจานสีเทา
scale_fill_manual()	คำสั่งในการเติมสีแบบ Manual
scale_fill_brewer()	คำสั่งในการเติมสีด้วยจานสี (Palettes)
scale_fill_grey()	คำสั่งในการเติมสีด้วยจานสีเทา
scale_shape_manual()	คำสั่งในการกำหนดรูปร่างแบบ Manual
scale_size_manual()	คำสั่งในการกำหนดขนาดแบบ Manual
scale_linetype_manual()	คำสั่งในการกำหนดชนิดเส้นแบบ Manual
scale_x_continuous()	คำสั่งในการกำหนดรายละเอียดต่าง ๆ ในแกน X สำหรับตัวแปรต่อเนื่อง เช่น ชื่อแกน (Axis labels) ขอบเขต (Axis limits) การแปลงแกน (Axis transformations) การกำหนดรูปแบบตัวเลขที่ต้องการแสดงผล เป็นต้น
scale_y_continuous()	คำสั่งในการกำหนดรายละเอียดต่าง ๆ ในแกน Y สำหรับตัวแปรต่อเนื่อง เช่น ชื่อแกน (Axis labels) ขอบเขต (Axis limits) การแปลงแกน (Axis transformations) การกำหนดรูปแบบตัวเลขที่ต้องการแสดงผล เป็นต้น
scale_x_discrete()	คำสั่งในการกำหนดรายละเอียดต่าง ๆ ในแกน X สำหรับตัวแปรไม่ต่อเนื่อง เช่น ชื่อแกน (Axis labels) ขอบเขต (Axis limits) การแปลงแกน (Axis transformations) การกำหนดรูปแบบตัวเลขที่ต้องการแสดงผล เป็นต้น
scale_y_discrete()	คำสั่งในการกำหนดรายละเอียดต่าง ๆ ในแกน Y สำหรับตัวแปรไม่ต่อเนื่อง เช่น ชื่อแกน (Axis labels) ขอบเขต (Axis limits) การแปลงแกน (Axis transformations) การกำหนดรูปแบบตัวเลขที่ต้องการแสดงผล เป็นต้น

คำสั่ง	คำอธิบาย
coord_cartesian()	คำสั่งในการกำหนดขอบเขตแกน (Axis limits) สำหรับ Cartesian coordinates
expand_limits()	คำสั่งในการกำหนดจุดเริ่มต้น (มุมล่างซ้าย) ของกราฟ
scale_x_reverse()	คำสั่งในการเรียงค่า (Sort) ในแกน X จากมากไปน้อย (จากเดิมน้อยไปมาก)
scale_y_reverse()	คำสั่งในการเรียงค่า (Sort) ในแกน Y จากมากไปน้อย (จากเดิมน้อยไปมาก)
scale_x_log10()	คำสั่งในการกำหนดสเกลของแกน X ให้เป็นค่า log ฐาน 10
scale_y_log10()	คำสั่งในการกำหนดสเกลของแกน Y ให้เป็นค่า log ฐาน 10
annotation_logticks()	คำสั่งในการแสดงขีดสเกล log tick marks
scale_x_date()	คำสั่งในการแสดงแกน X ในหน่วยวัน/เวลา (Date axis)
scale_y_date()	คำสั่งในการแสดงแกน Y ในหน่วยวัน/เวลา (Date axis)
element_text()	คำสั่งในการกำหนดรูปแบบการแสดงผลข้อความ ใช้ร่วมกับคำสั่ง theme()
element_line()	คำสั่งในการกำหนดรูปแบบการแสดงผลเส้น ใช้ร่วมกับคำสั่ง theme()
element_rect()	คำสั่งในการกำหนดรูปแบบการขอบและพื้น ใช้ร่วมกับคำสั่ง theme()
element_blank()	คำสั่งในการซ่อนการแสดงผล ใช้ร่วมกับคำสั่ง theme()
print()	คำสั่งในการสั่งพิมพ์
theme()	คำสั่งในการกำหนดธีม
theme_grey()	ธีมพื้นสีเทาและเส้น Grid สีขาว
theme_bw()	ธีมพื้นสีขาวและเส้น Grid สีเทา
theme_linedraw()	ธีมพื้นสีขาวและเส้นกรอบสีดำ
theme_light()	ธีมพื้นสีขาว เส้น Grid และแกนสีเทาอ่อน
theme_minimal()	ธีมที่ไม่มีคำอธิบายประกอบ (No background annotations)
theme_classic()	ธีมที่แสดงแกน X และ Y แต่ไม่แสดงเส้น Grid
theme_void()	ไม่แสดงธีมใด ๆ เลย
theme_dark()	ธีมพื้นสีเทาเข้ม
theme_economist()	ธีม Economist
theme_stata()	ธีม Stata
theme_set()	คำสั่งในการเปลี่ยนธีมทุก ๆ Session ใน R
expand.grid()	คำสั่งในการสร้าง combination ของตัวแปรที่กำหนดให้เป็นเดต้าเฟรม
annotation_custom()	คำสั่งในการสร้าง Static annotations
textGrob()	คำสั่งในการสร้างข้อความ
grobTree()	คำสั่งในการสร้าง Grid graphical objects (Grobs)

คำสั่ง	คำอธิบาย
facet_grid()	คำสั่งในการพล็อตกราฟย่อย (Subplots) เป็นตาราง Grids
facet_wrap()	คำสั่งในการพล็อตกราฟย่อย (Subplots) ให้แสดงผลเป็น 2 มิติ (ทั้งแถวและคอลัมน์)
position_dodge()	คำสั่งในการจัดวางแต่ละแท่งกราฟอยู่ข้างกัน (Side by side)
position_fill()	คำสั่งในการจัดวางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง โดยปรับความสูงของแท่งกราฟทั้งหมดให้มีความสูงเท่ากัน (Normalize)
position_stack()	คำสั่งในการจัดวางแท่งกราฟอยู่ด้านบนต่อจากกราฟแต่ละแท่ง
position_jitter()	คำสั่งในการจัดวางจุดแต่ละจุดให้เหลี่ยมกัน
coord_cartesian()	Cartesian coordinate system เป็นค่าโดยปริยายสำหรับการพล็อต
coord_fixed()	Cartesian coordinate system ที่แสดงความสัมพันธ์ระหว่างสเกลแกน X และสเกลแกน Y ค่าโดยปริยายกำหนดให้ ratio = 1
coord_flip()	คำสั่งในการสลับแกน X และแกน Y
coord_polar()	Polar coordinate system ซึ่งเหมาะกับการพล็อตกราฟวงกลม (Pie charts) ลักษณะการพล็อตเหมือนกับกราฟแท่งแตกต่างกันที่แท่งกราฟเรียงตัวอยู่ใน Polar coordinates
coord_trans()	คำสั่งในการแปลง Cartesian coordinate system
coord_map()	คำสั่งในการสร้าง Map projections สำหรับการพล็อตแผนที่
runif()	คำสั่งในการสร้างเลขสุ่มจาก Uniform Distribution

12.19 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า airquality สร้างกราฟเส้น (Line Plot) เพื่อแสดงความสัมพันธ์ระหว่าง temp และ wind

   1.1 เพิ่มชื่อกราฟโดยใช้คำสั่ง `ggtitle()`

   1.2 เพิ่มชื่อแกน X และ Y โดยใช้คำสั่ง `xlab()` และ `ylab()`

   1.3 ปรับธีมของกราฟให้เป็น `theme_minimal()`

   1.4 ใช้คำสั่ง `theme()` เพื่อปรับขนาดตัวอักษรของชื่อแกนเป็นขนาดเท่ากับ 14

2. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ palmerpenguins ชื่อว่า penguins สร้างกราฟ Scatter Plot เพื่อแสดงความสัมพันธ์ระหว่าง flipper_length_mm และ bill_length_mm

   2.1 กำหนดสีของจุดตามค่า species โดยใช้ `scale_color_manual()`

   2.2 กำหนดขนาดของจุดให้เพิ่มตามค่า body_mass_g โดยใช้ `scale_size()`

   2.3 เพิ่มคำอธิบายสีและขนาดใน Legend ด้วย `labs()`

   2.4 ปรับธีมของกราฟให้เป็น `theme_classic()`

   2.5 เพิ่มเส้นแนวนอนและแนวตั้งที่ค่าเฉลี่ยโดยใช้ `geom_hline()` และ `geom_vline()`

   2.6 เพิ่มข้อความอธิบายที่จุดสำคัญในกราฟโดยใช้ `geom_text()`

   2.7 ปรับขนาดและสีของข้อความใน `geom_text()`

   2.8 ใช้ `scale_color_gradient()` เพื่อปรับสีจุดตามค่า body_mass_g

   2.9 เปลี่ยน `scale_color_gradient()` เป็น `scale_color_gradient2()` เพื่อเพิ่มความแตกต่างในช่วงสี

   2.10 ปรับธีมกราฟให้เป็น `theme_linedraw()`

3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ palmerpenguins ชื่อว่า penguins สร้าง bar plot โดยใช้คำสั่ง `geom_bar()` เพื่อแสดงจำนวนตัวอย่างในแต่ละ species

   3.1 ปรับให้แกน X เป็นแบบกลับด้าน (reverse) โดยใช้คำสั่ง `scale_x_reverse()`

   3.2 เปลี่ยนการจัดวางแกนเป็นแนวนอนโดยใช้ `coord_flip()`

   3.3 ปรับธีมของกราฟให้เป็น `theme_dark()`

4. สร้างข้อมูลเดต้าเฟรมจากแพ็กเกจ maps โดยใช้คำสั่ง `map_data("state")` เพื่อสร้างแผนที่สำหรับรัฐต่าง ๆ ในสหรัฐอเมริกา โดยใช้คำสั่ง `geom_polygon()` และใช้คำสั่ง `geom_path()` เพื่อวาดเส้นขอบรอบพื้นที่ของรัฐ

5. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ ggplots2 ชื่อว่า economics เพื่อสร้างกราฟ Ribbon ระหว่าง date และค่าช่วง psavert ที่เป็นค่าบวกลบ 5% ของค่าเฉลี่ย โดยใช้คำสั่ง `geom_ribbon()`

   5.1 เพิ่มคำสั่ง geom_line เพื่อสร้างกราฟเส้นและให้เส้นกราฟมีลักษณะเป็น “twodash”

   5.2 ปรับสีของ Ribbon ให้เป็นสีฟ้าโปร่งแสง

   5.3 ปรับแกน X และ Y ให้แสดงค่าชัดเจนขึ้นโดยใช้ `scale_x_date()` และ `scale_y_continuous()`

6. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ ggplots2 ชื่อว่า diamonds สร้าง Scatter Plot เพื่อแสดงความสัมพันธ์ระหว่าง carat และ price

   6.1 แยกกราฟตามค่า cut โดยใช้คำสั่ง `facet_wrap()`

   6.2 เปลี่ยนเป็นแยกกราฟตามค่า color โดยใช้ `facet_grid()`

7. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ ggplots2 ชื่อว่า diamonds

   7.1 สร้าง Stacked Bar Plot โดยใช้คำสั่ง `geom_bar()` เพื่อแสดงจำนวนตัวอย่างในแต่ละประเภท cut และแยกสีตาม color

   7.2 เปลี่ยน Bar Plot เป็นแบบ dodge โดยใช้ `position_dodge()`

   7.3 เปลี่ยน Bar Plot เป็นแบบ fill โดยใช้ `position_fill()`

8. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ gapminder ชื่อว่า gapminder โดยเลือกตัวอย่างมา 12 ประเทศ ได้แก่ “Thailand”, “Singapore”, “Japan”, “United States”, “Argentina”, “Mexico”, “Finland”, “Germany”, “United Kingdom”, “Egypt”, “Morocco” และ “South Africa” สร้าง Line Plot ของปี (year) และข้อมูลอายุขัย (lifeExp) โดยแยกสีเป็นของแต่ละประเทศและเพิ่มจุดบนเส้น และแยกกราฟตามทวีป (continent) ด้วยคำสั่ง `facet_wrap()`

บทที่ 13

ส่วนขยายของ ggplot2 (Extensions of ggplot2)

บทนี้จะกล่าวถึงส่วนขยายต่าง ๆ ของแพ็กเกจ **ggplot2** ที่สามารถทำให้การพล็อตมีความหลากหลาย ทำงานได้สะดวก และคล่องตัวมากขึ้น ดังรายละเอียดที่จะกล่าวต่อไปนี้

13.1 การพล็อตกราฟหลายรูปในหน้าเดียวกัน (Arrange Multiple Graphs on the Same Page)

ข้อมูลที่จะใช้ในการพล็อตประกอบด้วยข้อมูล ToothGrowth, economics และ diamonds ที่ติดตั้งมาพร้อมกับ base **R** และ **ggplot2** ดังนี้

```
# Convert the variable dose from a numeric to a factor variable
R> data("ToothGrowth")
R> ToothGrowth$dose <- as.factor(ToothGrowth$dose)

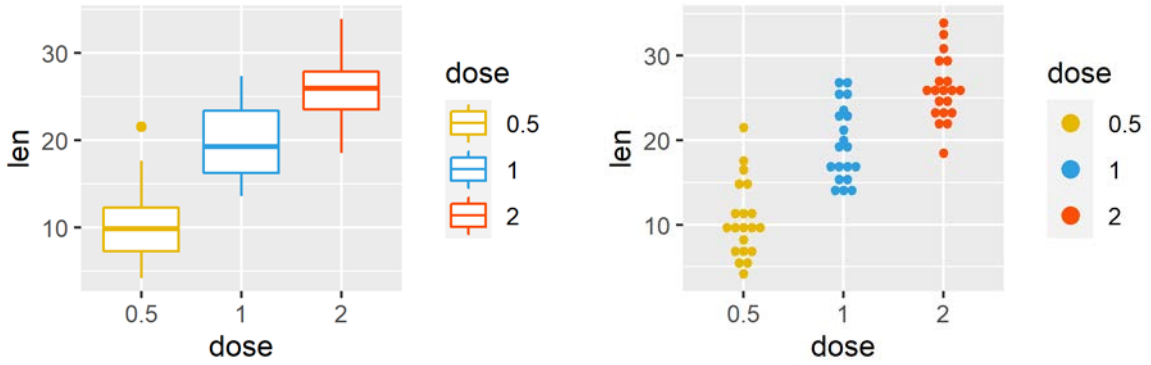
R> data("economics") # Load economics
R> data("diamonds") # Load diamonds
```

การพล็อตโดยทั่วไป ใช้คำสั่ง Geometry ประเภทต่าง ๆ ในการพล็อต ดังนี้

```
# A set of 3 colors
R> my3colors <- c("#E7B800", "#2E9FDF", "#FC4E07")

R> p <- ggplot(ToothGrowth, aes(x = dose, y = len))
# Box plots (bxp)
R> bxp <- p + geom_boxplot(aes(color = dose)) +
     scale_color_manual(values = my3colors)
R> bxp

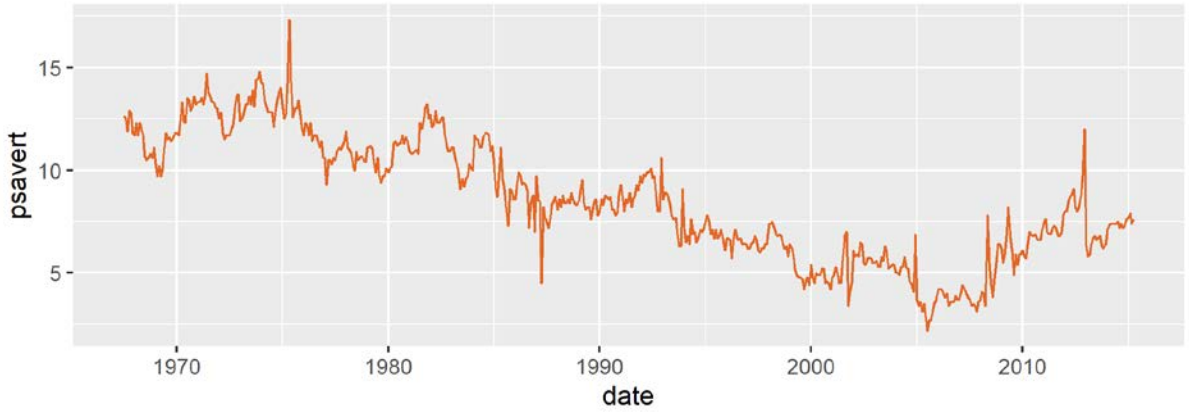
# Dot plots (dp)
R> dp <- p + geom_dotplot(aes(color = dose, fill = dose),
                          binaxis = 'y', stackdir = 'center') +
     scale_color_manual(values = my3colors) +
     scale_fill_manual(values = my3colors)
R> dp
```



รูปที่ 13-1 General plots

รูปแรกแสดง Box plots แกน X แสดงตัวแปร dose เท่ากับ 0.5, 1 และ 2 ด้วยสีเหลือง ฟ้า และแดง ตามลำดับ แกน Y แสดงตัวแปร len ส่วนรูปที่สองเป็นข้อมูลเดียวกันแต่แสดงด้วย Dot plots

```
R> lp <- ggplot(economics, aes(x = date, y = psavert)) +
     geom_line(color = "#E46726")
R> lp
```



รูปที่ 13-2 Time series plots

รูปด้านบนแสดงกราฟอนุกรมเวลา (Time series) แกน X เป็นเวลา แกน Y เป็น Personal savings rate (psavert)

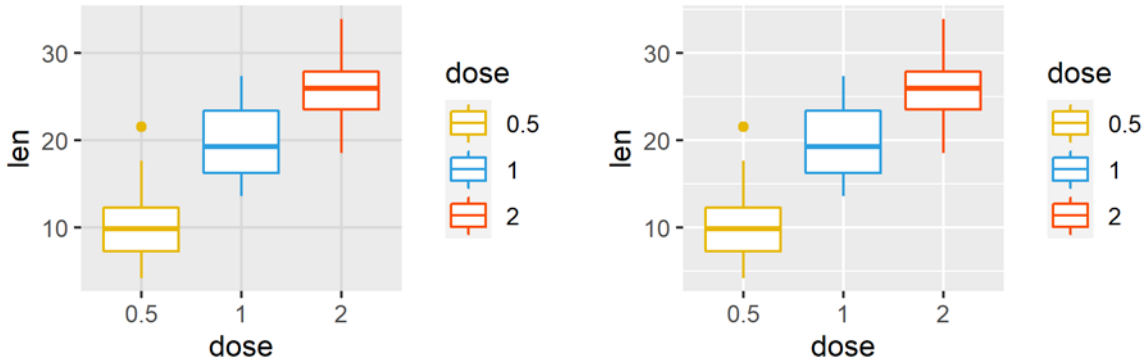
13.1.1 การใช้แพ็กเกจ cowplot

แพ็กเกจ **cowplot** ทำให้การพล็อตสามารถรวมกราฟย่อย ๆ เข้ามาอยู่ในรูปเดียวกันได้ **cowplot** จะกำหนดสีพื้น (Background color) เป็นสีขาว และไม่มี Grid คล้ายกับคำสั่ง **theme_classic()** สามารถเพิ่ม Grid ด้วยคำสั่ง **background_grid()** ดังนี้

```
R> require(cowplot)

# Add gridlines
R> bxp + background_grid(major = "xy", minor = "none")
```

```
# Use theme_gray()
R> bxp + theme_gray()
```



รูปที่ 13-3 Plots with background_grid()

ทั้ง 2 รูปด้านบนแสดง Box plots แกน X แสดงตัวแปร dose เท่ากับ 0.5, 1 และ 2 และด้วยสีเหลือง ฟ้า และแดง ตามลำดับ แกน Y แสดงตัวแปร len รูปแรกเป็นการกำหนดการแสดงพื้น (Background) ด้วยคำสั่ง **background_grid()** จากแพ็กเกจ **cowplot** แสดงเฉพาะ Major grids ไม่แสดง Minor grids รูปที่สองแสดงพื้นด้วยธีมสีเทา

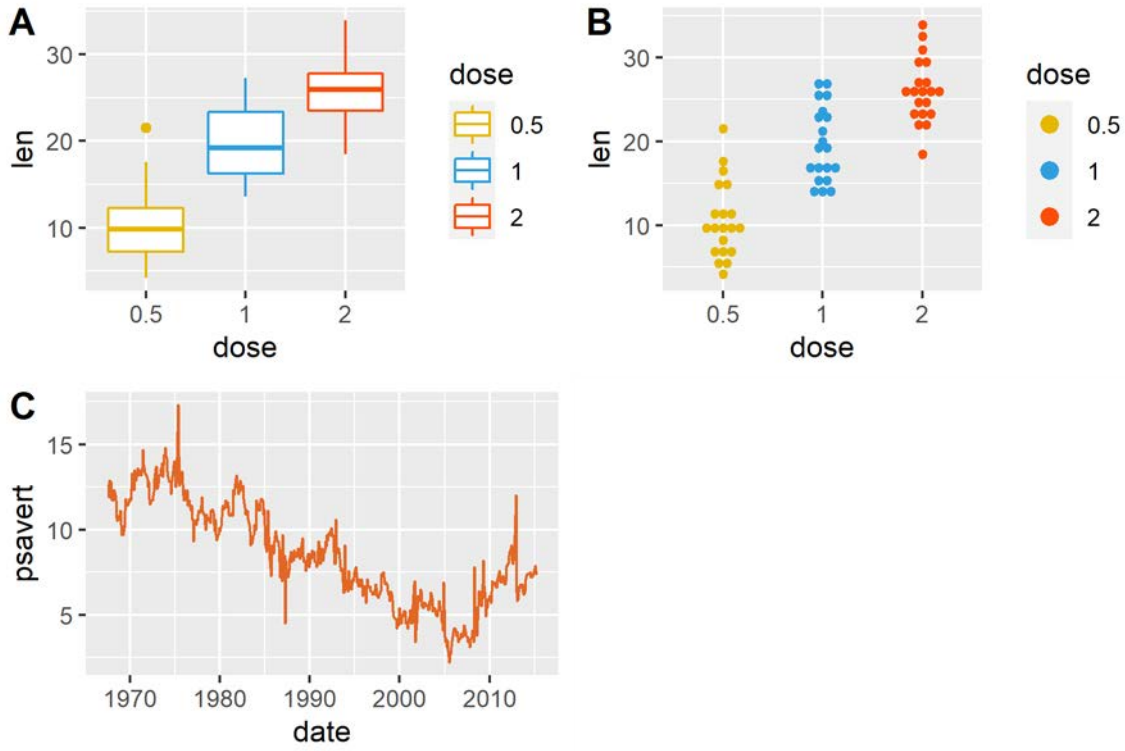
การใช้คำสั่งพล็อตในแพ็กเกจ **cowplot** มี 2 ลักษณะคือ

1. ใช้คำสั่ง **plot_grid()** ในการรวมกราฟย่อย ๆ เข้ามาอยู่ในรูปเดียวกันภายในคำสั่งดังกล่าว
2. คำสั่งแบบลูกโซ่ (Chain) โดยใช้คำสั่งต่อเนื่องดังนี้ **ggdraw() + draw_plot() + draw_plot_label()**

คำสั่ง **ggdraw()** จะทำการสร้างพื้นที่ว่าง (Empty drawing canvas) ขึ้นมาเพื่อใส่กราฟเข้าไปในพื้นที่ดังกล่าว **draw_plot()** เป็นการพล็อตกราฟย่อย ๆ ลงในตำแหน่งและขนาดที่กำหนดในพื้นที่ Drawing canvas ส่วนคำสั่ง **draw_plot_label()** เป็นการพล็อตข้อความลงบนกราฟ

รูปต่อไปนี้แสดงการพล็อตโดยรวมกราฟ 3 รูปที่พล็อตไว้ด้านบนด้วยคำสั่ง **plot_grid()** ดังนี้

```
R> plot_grid(bxp, dp, lp, labels = c("A", "B", "C"), ncol = 2, nrow = 2)
```



รูปที่ 13-4 Plots with plot_grid()

รูปด้านบนเป็นการพล็อตโดยรวมกราฟ 3 รูปที่พล็อตไว้เป็นวัตถุ (Grid graphical objects, Grob) ก่อนหน้านี้ได้แก่ bxp, dp, lp เข้ามาอยู่ในหน้าเดียวกันในลักษณะตาราง 2 แถว 2 คอลัมน์ และกำหนดชื่อรูป A, B และ C ตามลำดับ โดยใช้คำสั่ง `plot_grid()`

รูปแบบการใช้คำสั่ง `draw_plot()` แสดงได้ดังนี้

```
R> draw_plot(plot, x = 0, y = 0, width = 1, height = 1)
```

พารามิเตอร์ที่ระบุในคำสั่งนี้ ประกอบด้วย

- `plot` คือ กราฟที่ต้องการพล็อต (`ggplot2` หรือ `gtable`)
- `x, y` คือ ตำแหน่งที่ต้องการพล็อตเริ่มจากมุมล่างซ้าย
- `width, height` คือ ความกว้างและความยาวของกราฟที่ต้องการพล็อต

รูปแบบการใช้คำสั่ง `draw_plot_label()` แสดงได้ดังนี้

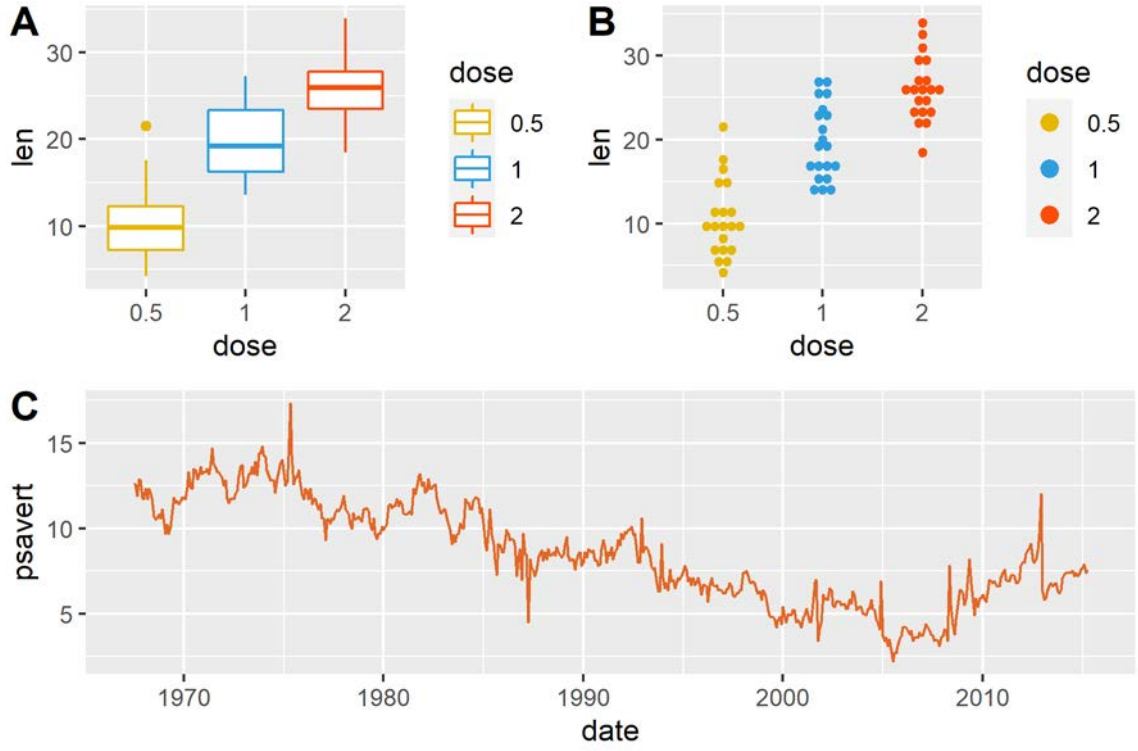
```
R> draw_plot_label(label, x = 0, y = 1, size = 16, ...)
```

พารามิเตอร์ที่ระบุในคำสั่งนี้ ประกอบด้วย

- `label` คือ เวกเตอร์ข้อความที่ต้องการพล็อต
- `x, y` คือ ตำแหน่งที่ต้องการพล็อตเริ่มจากมุมล่างซ้าย
- `size` คือ ขนาดของฟอนต์ของข้อความที่ต้องการพล็อต

การใช้คำสั่งแบบลูกโซ่ (Chain) `ggdraw() + draw_plot() + draw_plot_label()` แสดงได้ดังนี้

```
R> ggdraw() +
     draw_plot(bxp, x = 0, y = 0.5, width = 0.5, height = 0.5) +
     draw_plot(dp, x = 0.5, y = 0.5, width = 0.5, height = 0.5) +
     draw_plot(lp, x = 0, y = 0, width = 1, height = 0.5) +
     draw_plot_label(label = c("A", "B", "C"),
                     x = c(0, 0.5, 0), y = c(1, 1, 0.5), size = 15)
```



รูปที่ 13-5 Plots with ggdraw() + draw_plot() + draw_plot_label()

รูปด้านบนเป็นการพล็อตโดยรวมกราฟ 3 รูปที่พล็อตไว้เป็นวัตถุ (Grid graphical objects, Grob) ก่อนหน้านี้ได้แก่ bxp, dp, lp เข้ามาอยู่ในหน้าเดียวกัน โดยการกำหนดตำแหน่งที่ต้องการพล็อต ความกว้าง ความยาว และกำหนดชื่อรูป โดยใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) `ggdraw() + draw_plot() + draw_plot_label()`

การบันทึกผลการพล็อตสำหรับคำสั่งที่กล่าวมาด้วยแพ็กเกจ `cowplot` ใช้คำสั่ง `save_plot()` ดังนี้

```
# use save_plot() instead of ggsave() when using cowplot
R> save_plot("mpg.png", bxp,
             base_aspect_ratio = 1.3) # make room for figure legend

R> plot2by2 <- plot_grid(bxp, dp, lp, labels = c("A", "B", "C"),
                         ncol = 2, nrow = 2)
R> save_plot("plot2by2.pdf", plot2by2,
             ncol = 2, # we're saving a grid plot of 2 columns
```

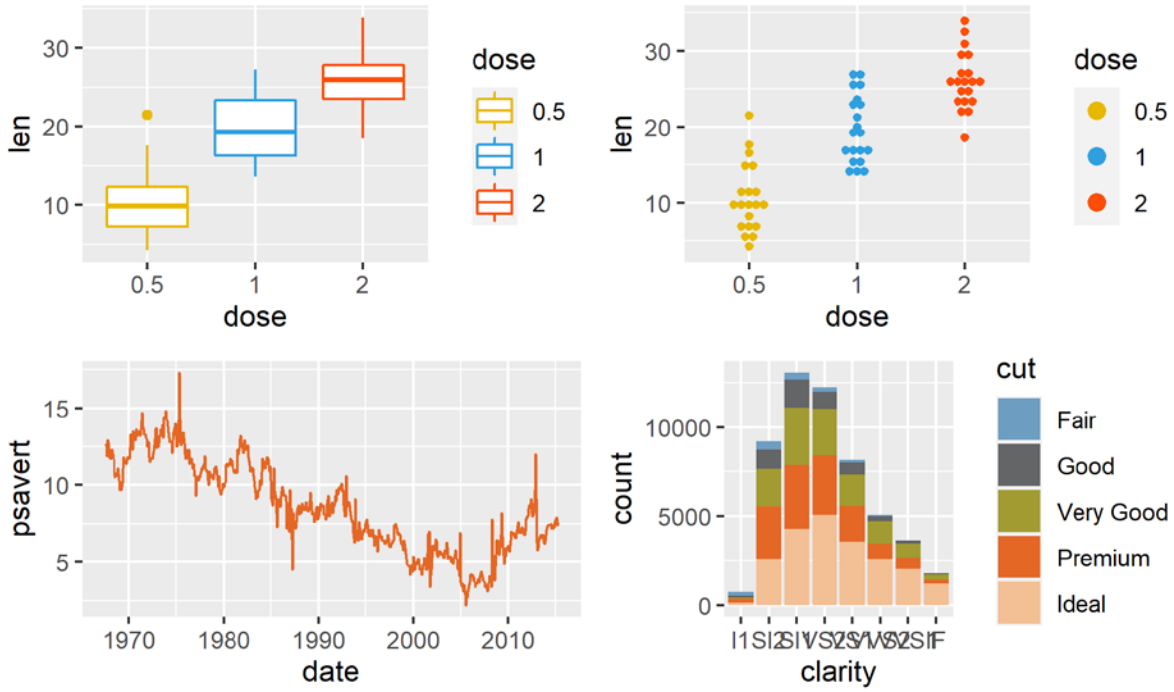
```
            nrow = 2, # and 2 rows
            # each individual subplot should have an aspect ratio of 1.3
            base_aspect_ratio = 1.3
  )
```

13.1.2 การใช้แพ็กเกจ gridExtra

การพล็อตกราฟหลายรูปในหน้าเดียวกัน ยังสามารถใช้แพ็กเกจ **gridExtra** โดยใช้คำสั่ง **grid.arrange()** โดยการรวม Box plots, dot plots, line plots และ bar charts ได้ดังนี้

```
# Define a set of 5 colors
R> my5cols <- c("#6D9EC1", "#646567", "#A29B32", "#E46726", "#F3BF94")
# Create a bar plot
R> data("diamonds")
R> brp <- ggplot(diamonds, aes(x = clarity)) +
     geom_bar(aes(fill = cut)) +
     scale_fill_manual(values = my5cols)

R> require(gridExtra)
R> grid.arrange(bxp, dp, lp, brp, ncol = 2, nrow = 2)
```

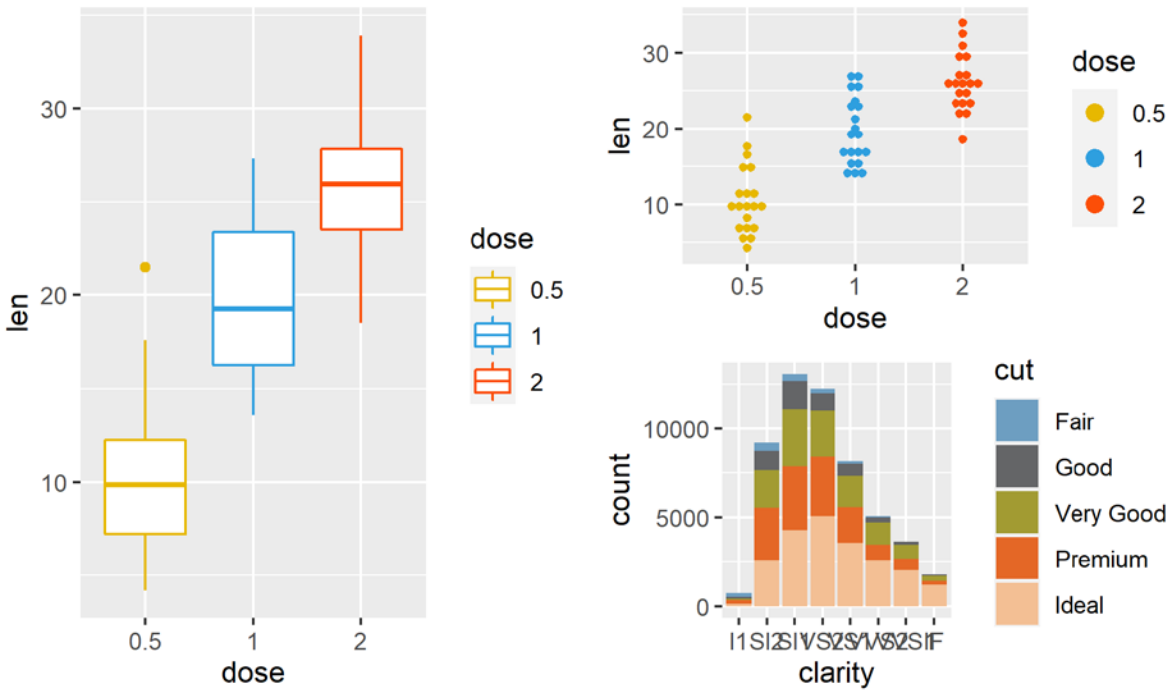


รูปที่ 13-6 Plots with grid.arrange()

รูปด้านบนเป็นการพล็อตโดยรวมกราฟ 4 รูปที่พล็อตไว้เป็นวัตถุ (Grid graphical objects, Grob) ก่อนหน้านี้ได้แก่ bxp, dp, lp และ brp เข้ามาอยู่ในหน้าเดียวกันในลักษณะตาราง 2 แถว 2 คอลัมน์ โดยใช้คำสั่ง **grid.arrange()** ในแพ็กเกจ **gridExtra**

นอกจากนี้ยังใช้คำสั่ง **arrangeGrob()** ในการรวม Grob หลาย Objects เข้าด้วยกัน ดังนี้

```
R> grid.arrange(bxp, arrangeGrob(dp, brp), ncol = 2)
```

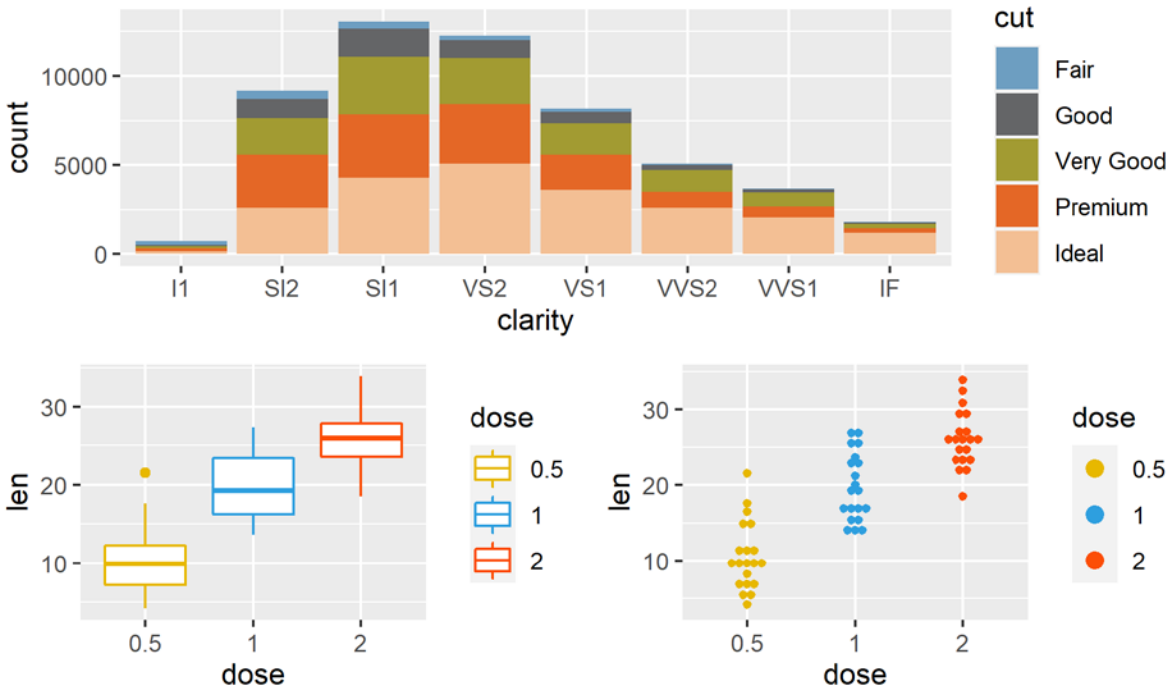


รูปที่ 13-7 Plots with grid.arrange() and arrangeGrob()

รูปด้านบนเป็นการพล็อตโดยรวมกราฟ 3 รูปที่พล็อตไว้เป็นวัตถุ (Grid graphical objects, Grob) ก่อนหน้านี้ เข้ามาอยู่ในหน้าเดียวกันเป็น 2 คอลัมน์ โดยใช้คำสั่ง `grid.arrange()` และรวม Grob ได้แก่ dp และ brp เข้าด้วยกันด้วยคำสั่ง `arrangeGrob()` ในแพ็กเกจ `gridExtra`

นอกจากนี้ยังใช้พารามิเตอร์ `layout_matrix` ในการวางตำแหน่งในการพล็อตกราฟย่อยเข้าด้วยกัน ดังนี้

```
R> plt <- grid.arrange(brp, bxp, dp, ncol = 2, nrow = 2,
                       layout_matrix = rbind(c(1,1), c(2,3)))
```



รูปที่ 13-8 Plots with grid.arrange() and layout_matrix

รูปด้านบนเป็นการพล็อตโดยรวมกราฟ 3 รูปที่พล็อตไว้เป็นวัตถุ (Grid graphical objects, Grob) ก่อนหน้านี้ เข้ามาอยู่ในหน้าเดียวกันเป็น 2 แถว 2 คอลัมน์ โดยใช้คำสั่ง `grid.arrange()` และการวางตำแหน่งในการพล็อตกราฟย่อยเข้าด้วยกันด้วยพารามิเตอร์ `layout_matrix`

13.1.3 การพล็อตโดยใช้สัญลักษณ์ร่วมกัน (Common Legend for Multiple Graphs)

การพล็อตกราฟย่อยโดยใช้สัญลักษณ์ร่วมกัน (Common legend for multiple graphs) มีขั้นตอนประกอบด้วย

1. สร้างพล็อตย่อยแต่ละรูป
2. บันทึกสัญลักษณ์ที่ต้องการพล็อต เรียกว่า External graphical element (หรือ “grob”)
3. ลบสัญลักษณ์ออกจากพล็อตทั้งหมด
4. สร้างพล็อตทั้งหมดและสัญลักษณ์ที่ต้องการพล็อต

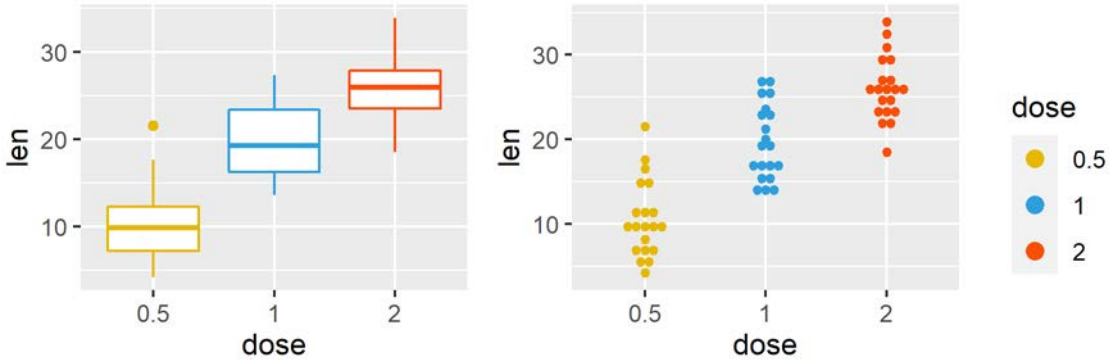
ดังตัวอย่างต่อไปนี้

```
# 1. Get plot legend
R> get_legend <- function(myggplot){
     require(gridExtra)
     tmp <- ggplot_gtable(ggplot_build(myggplot))
     leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
     legend <- tmp$grobs[[leg]]
     return(legend)
   }
```

```
# 2. Save the legend from the dot plot
R> legend <- get_legend(dp)

# 3. Remove the legend from the box plot and the dot plot
R> bxp2 <- bxp + theme(legend.position = "none")
R> dp2 <- dp + theme(legend.position = "none")

# 4. Arrange bxp2, dp and the legend with a specific width
R> grid.arrange(bxp2, dp2, legend, ncol = 3,
                widths = c(2.3, 2.3, 0.8))
```



รูปที่ 13-9 Plots with common legend with position right

รูปด้านบนเป็นการพล็อตกราฟ 2 รูปโดยใช้สัญลักษณ์ (Legend) ร่วมกัน

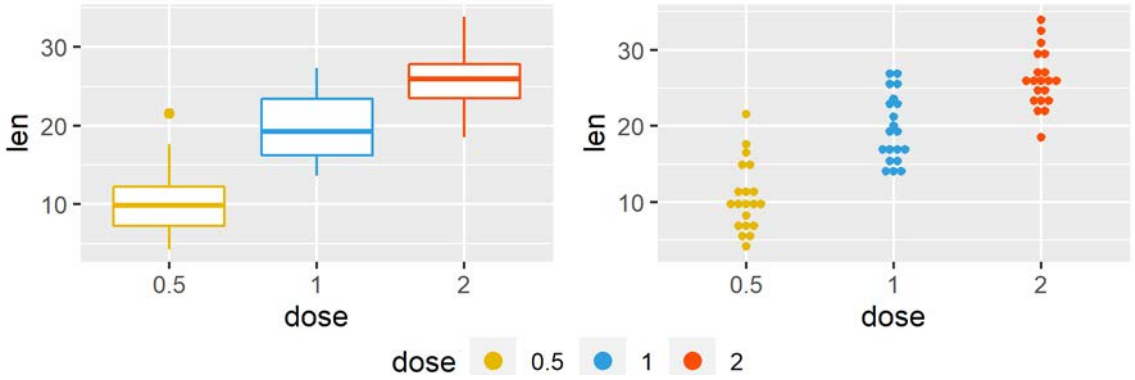
สามารถสร้างสัญลักษณ์บริเวณด้านล่างของรูปได้ ดังตัวอย่างต่อไปนี้

```
# 1. Dot plot with legend at the top
R> dp2 <- dp + theme(legend.position = "top")

# 2. Save the legend
R> legend <- get_legend(dp2)

# 3. Remove the legend from the dot plot
R> dp2 <- dp2 + theme(legend.position = "none")

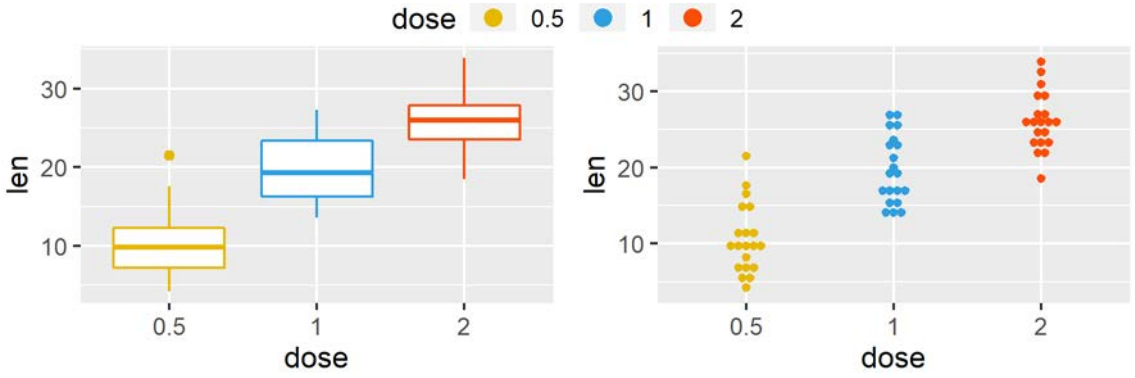
R> grid.arrange(bxp2, dp2, legend, ncol=2, nrow = 2,
                layout_matrix = rbind(c(1,2), c(3,3)),
                widths = c(2.7, 2.7), heights = c(2.5, 0.2))
```



รูปที่ 13-10 Plots with common legend with position bottom

สามารถสร้างสัญลักษณ์บริเวณด้านบนของรูปได้ ดังตัวอย่างต่อไปนี้

```
R> grid.arrange(legend, bxp2, dp2, ncol = 2, nrow = 2,
                layout_matrix = rbind(c(1,1), c(2,3)),
                widths = c(2.7, 2.7), heights = c(0.2, 2.5))
```



รูปที่ 13-11 Plots with common legend with position top

13.1.4 Scatter Plots พร้อมกับกราฟความหนาแน่น (Scatter plots with marginal density plots)

การสร้าง Scatter plots พร้อมกับกราฟความหนาแน่นที่บริเวณขอบ โดยการใช้คำสั่งในแพ็กเกจ `gridExtra` ทำได้โดยขั้นแรกเป็นการเตรียมข้อมูลสำหรับการพล็อต

```
# Another set of 3 colors
R> my3colors <- c("#6D9EC1", "#646567", "#A29B32")

R> set.seed(1234)
R> x <- c(rnorm(350, mean = -1), rnorm(350, mean = 1.5),
          rnorm(350, mean = 4))
R> y <- c(rnorm(350, mean = -0.5), rnorm(350, mean = 1.7),
          rnorm(350, mean = 2.5))

R> group <- as.factor(rep(c(1, 2, 3), each = 350))
R> df2 <- data.frame(x, y, group)
```

ขั้นตอนมาเป็นการสร้างพล็อตย่อย ได้แก่ scatterPlot, xdensity, ydensity และ blankPlot ดังนี้

```
# Scatter plot of x and y variables and color by groups
R> scatterPlot <- ggplot(df2, aes(x, y)) +
     geom_point(aes(color = group)) +
     scale_color_manual(values = my3colors) +
     theme(legend.position = c(0,1), legend.justification = c(0,1))

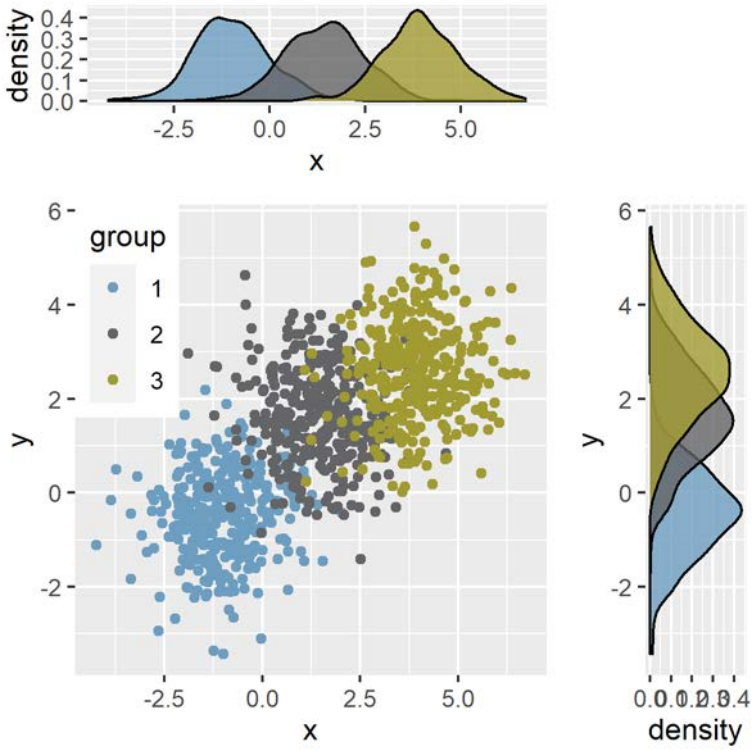
# Marginal density plot of x (top panel)
R> xdensity <- ggplot(df2, aes(x)) +
     geom_density(aes(fill = group), alpha = 0.8) +
     scale_fill_manual(values = my3colors) +
     theme(legend.position = "none")

# Marginal density plot of y (right panel)
R> ydensity <- ggplot(df2, aes(y)) +
     geom_density(aes(fill = group), alpha = 0.8) +
     scale_fill_manual(values = my3colors) +
     theme(legend.position = "none") +
     coord_flip()

R> blankPlot <- ggplot() +
     geom_blank(aes(1,1)) +
     theme_void()
```

ขั้นสุดท้ายใช้คำสั่ง `grid.arrange()` ในการพล็อตองค์ประกอบต่าง ๆ ที่เตรียมไว้รวมเข้าด้วยกัน ดังนี้

```
R> require("gridExtra")
R> grid.arrange(xdensity, blankPlot, scatterPlot, ydensity,
                ncol = 2, nrow = 2, widths = c(4, 1.4), heights = c(1.4, 4))
```



รูปที่ 13-12 Scatter plots with marginal density plots

รูปด้านบนเป็นการสร้าง Scatter plots พร้อมกับกราฟความหนาแน่นที่บริเวณขอบโดยการใช้คำสั่ง `grid.arrange()` ในแพ็กเกจ `gridExtra`

การพล็อตกราฟย่อยดังกล่าว อาจจะใช้คำสั่ง `viewport()` มีขั้นตอนต่าง ๆ ประกอบด้วย

1. สร้างพล็อตย่อยแต่ละรูป
2. สร้างพื้นที่ในการพล็อตด้วยคำสั่ง `grid.newpage()`
3. สร้างพื้นที่ในการพล็อตขนาด 2 แถว 2 คอลัมน์
4. นิยามฟังก์ชัน Grid viewport ที่ต้องการพล็อต
5. พล็อตลงในพื้นที่ Viewport ที่สร้างไว้

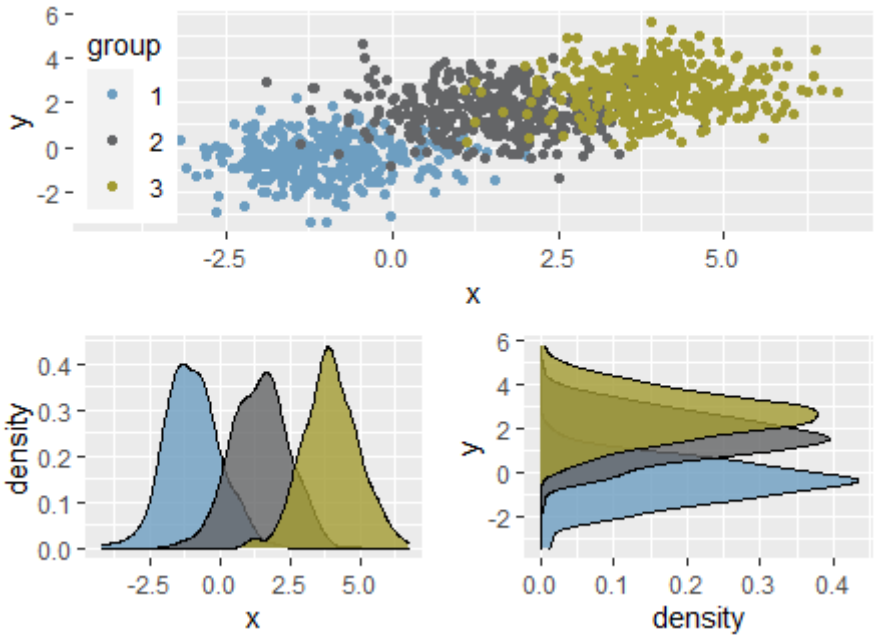
ดังตัวอย่างต่อไปนี้

```
R> require(grid)
# Move to a new page
R> grid.newpage()

# Create layout : nrow = 2, ncol = 2
R> pushViewport(viewport(layout = grid.layout(2, 2)))

# A helper function to define a region on the layout
R> define_region <- function(row, col){
     viewport(layout.pos.row = row, layout.pos.col = col)
   }

# Arrange the plots
R> print(scatterPlot, vp = define_region(1, 1:2))
R> print(xdensity, vp = define_region(2, 1))
R> print(ydensity, vp = define_region(2, 2))
```



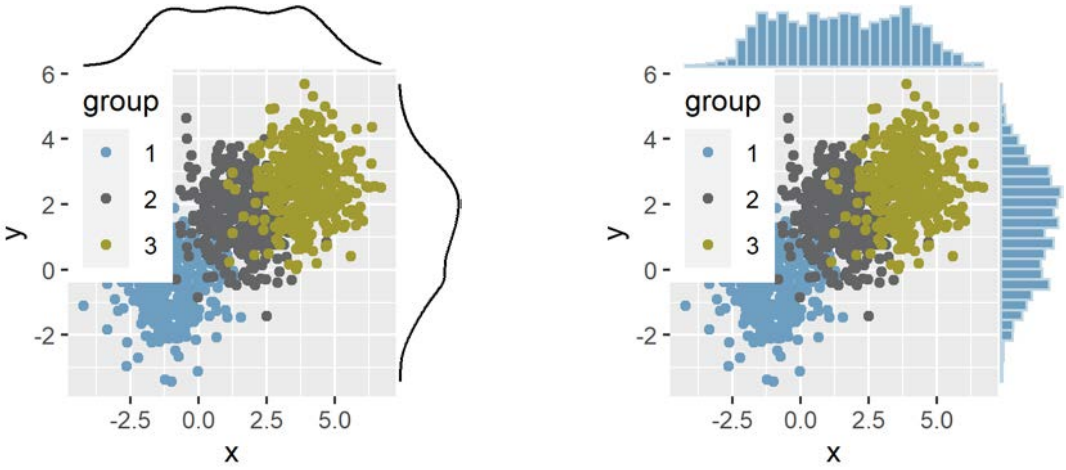
รูปที่ 13-13 Scatter plots with marginal density plots using viewport()

รูปด้านบนเป็นการพล็อตกราฟย่อยเข้าด้วยกันโดยใช้คำสั่ง **grid.newpage()** และคำสั่ง **viewport()**

นอกจากนี้ยังใช้แพ็กเกจ **ggExtra** ในการเพิ่ม Marginal histograms, box plots หรือ density plots เข้าไปใน Scatter plots ด้วยคำสั่ง **ggMarginal()** ดังนี้

```
R> library("ggExtra")
# Marginal density plot
R> ggMarginal(scatterPlot)

# Marginal histogram plot
R> ggMarginal(scatterPlot, type = "histogram",
              fill = "#6D9EC1", color = "#BFD5E3")
```



รูปที่ 13-14 Scatter plots with marginal plots using ggMarginal()

รูปแรกแสดง Scatter plots และกราฟความหนาแน่น (Density plots) รูปที่สองแสดง Scatter plots และฮีสโตแกรม

13.1.5 การเพิ่มองค์ประกอบกราฟิกภายนอกใน ggplot (External graphical elements inside a ggplot)

ฟังก์ชัน **annotation_custom()** สามารถเพิ่มตาราง กราฟ และ Grid-based elements เข้าไปใน ggplot โดยมีรูปแบบดังนี้

```
annotation_custom(grob, xmin, xmax, ymin, ymax)
```

- **grob** เป็น External graphical element ที่ต้องการแสดงผล
- **xmin, xmax** เป็นตำแหน่งค่า x ใน coordinate
- **ymin, ymax** เป็นตำแหน่งค่า y ใน coordinate

โดยขั้นตอนแรกเป็นการสร้างพล็อต เช่น Scatter plots ไว้ก่อน แล้วจึงเพิ่มกราฟที่ต้องการด้วยฟังก์ชัน **annotation_custom()** ตัวอย่างต่อไปนี้เป็นการเพิ่ม Box plots เข้าไปใน Scatter plots ซึ่งจะสร้าง Transparent background ให้กับ Box plots ก่อน ดังนี้

```
# Create a transparent theme object
R> transparent_theme <- theme(
     axis.title.x = element_blank(),
     axis.title.y = element_blank(),
     axis.text.x = element_blank(),
     axis.text.y = element_blank(),
     axis.ticks = element_blank(),
     panel.grid = element_blank(),
     axis.line = element_blank(),
     panel.background = element_rect(fill = "transparent",colour = NA),
     plot.background = element_rect(fill = "transparent",colour = NA))
```

ทำการสร้างกราฟย่อย และสร้าง External graphical elements หรือ grops ดังนี้

```
# Scatter plot of x and y variables and color by groups
R> scatterPlot <- ggplot(df2, aes(x, y)) +
     geom_point(aes(color = group)) +
     scale_color_manual(values = my3colors) +
     theme(legend.position = c(1,0), legend.justification = c(1,0))
R> p1 <- scatterPlot

# Box plot of the x variable
R> p2 <- ggplot(df2, aes(factor(1), x))+
     geom_boxplot(width = 0.3) +
     coord_flip() +
     transparent_theme

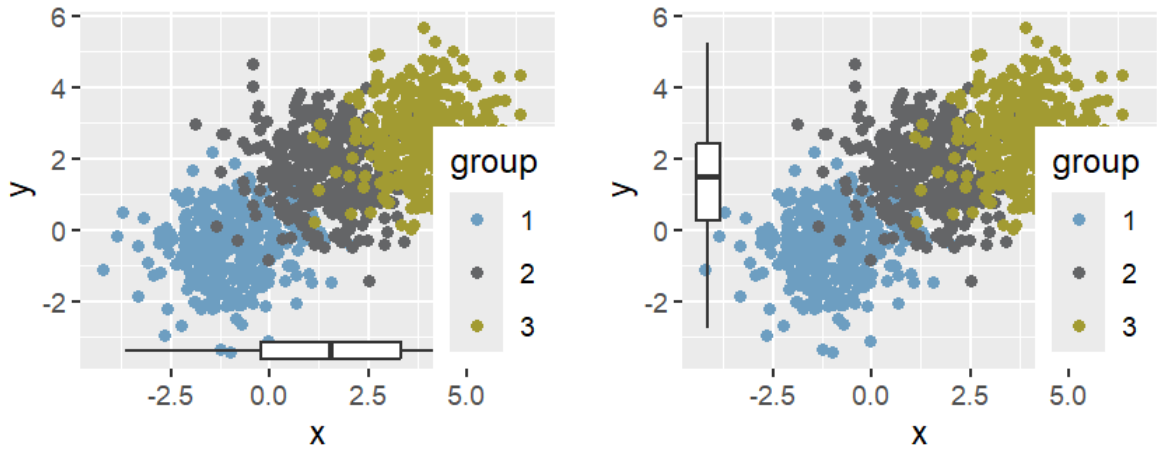
# Box plot of the y variable
R> p3 <- ggplot(df2, aes(factor(1), y)) +
     geom_boxplot(width = 0.3) +
     transparent_theme

# Create the external graphical elements, called a "grop" in Grid terminology
R> p2_grob <- ggplotGrob(p2)
R> p3_grob <- ggplotGrob(p3)
```

และทำการเพิ่มกราฟ External graphical elements เข้าไปใน Scatter plots ดังนี้

```
# Scatter plot with box plot of the x variable
R> p1 + annotation_custom(p2_grob, ymin = -5, ymax = -2)

# Scatter plot with box plot of the y variable
R> p1 + annotation_custom(p3_grob, xmax = -2.5, xmin = -6)
```



รูปที่ 13-15 Scatter plots with an external graphical element

รูปแรกแสดง Scatter plots พร้อมกับ Box plots ในแนวนอน รูปที่สองแสดง Scatter plots พร้อมกับ Box plots ในแนวตั้ง

ผู้ใช้สามารถใส่ตาราง ข้อความ เพื่อทำการพล็อตร่วมกับ ggplot ได้ด้วยฟังก์ชันดังนี้

- `tableGrob()` สำหรับพล็อตตาราง
- `splitTextGrob()` สำหรับพล็อตข้อความ

การพล็อตด้วยคำสั่งดังกล่าว แสดงได้ดังนี้

```
R> library(RGraphics)
R> library(gridExtra)
R> ToothGrowth$dose <- factor(ToothGrowth$dose)
# Table
R> p1 <- tableGrob(head(ToothGrowth, 3))

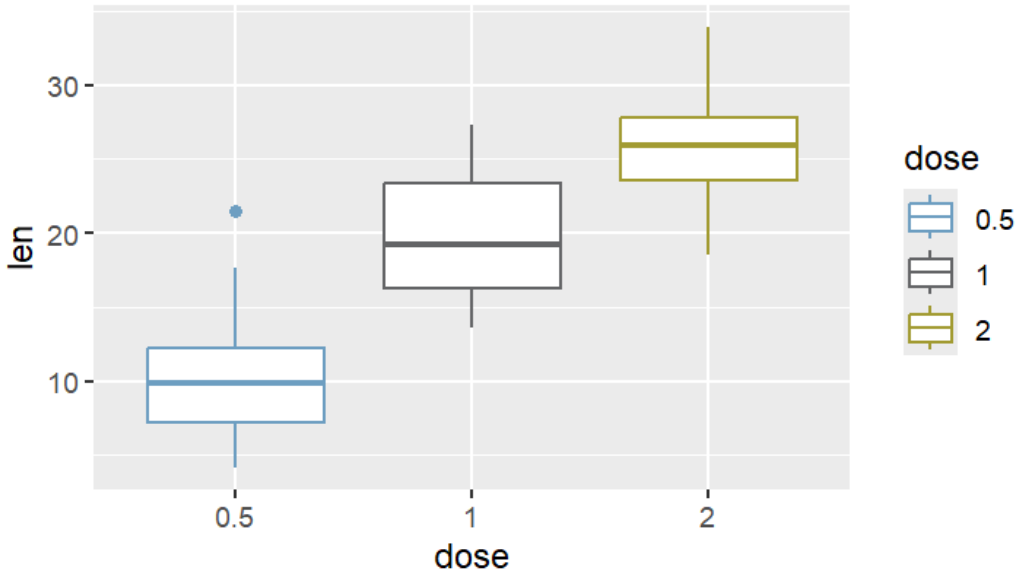
# Text
R> text <- paste0("ToothGrowth data describes the effect ",
                  "of Vitamin C on tooth growth in Guinea pigs.")
R> p2 <- splitTextGrob(text)

# Box plot
R> p3 <- ggplot(ToothGrowth, aes(x = dose, y = len)) +
     geom_boxplot(aes(color = dose)) +
     scale_color_manual(values = my3colors)

# Arrange the plots on the same page
R> grid.arrange(p1, p2, p3, ncol = 1,
                heights = c(0.25, 0.2, 0.55))
```

	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5

ToothGrowth data describes the effect of Vitamin C on tooth growth in Guinea pigs.



รูปที่ 13-16 Plots with tables and texts

รูปด้านบนเป็นการพล็อตกราฟ ตาราง และข้อความอยู่ในหน้าเดียวกัน

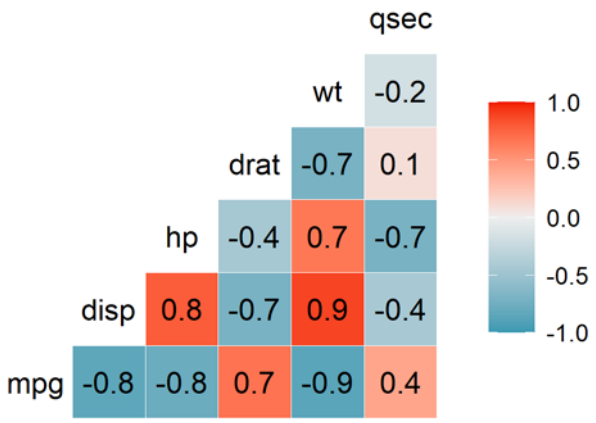
13.2 การพล็อต Correlation Matrix (Correlation Matrix Visualization)

หัวข้อนี้จะกล่าวถึงการพล็อต Correlation matrix ด้วยแพ็กเกจ `GGally` และ `ggcorrplot` ดังนี้

13.2.1 การใช้แพ็กเกจ GGally

แพ็กเกจ `GGally` มีคำสั่งในการพล็อต Correlation Matrix ได้หลายลักษณะ ดังตัวอย่างต่อไปนี้

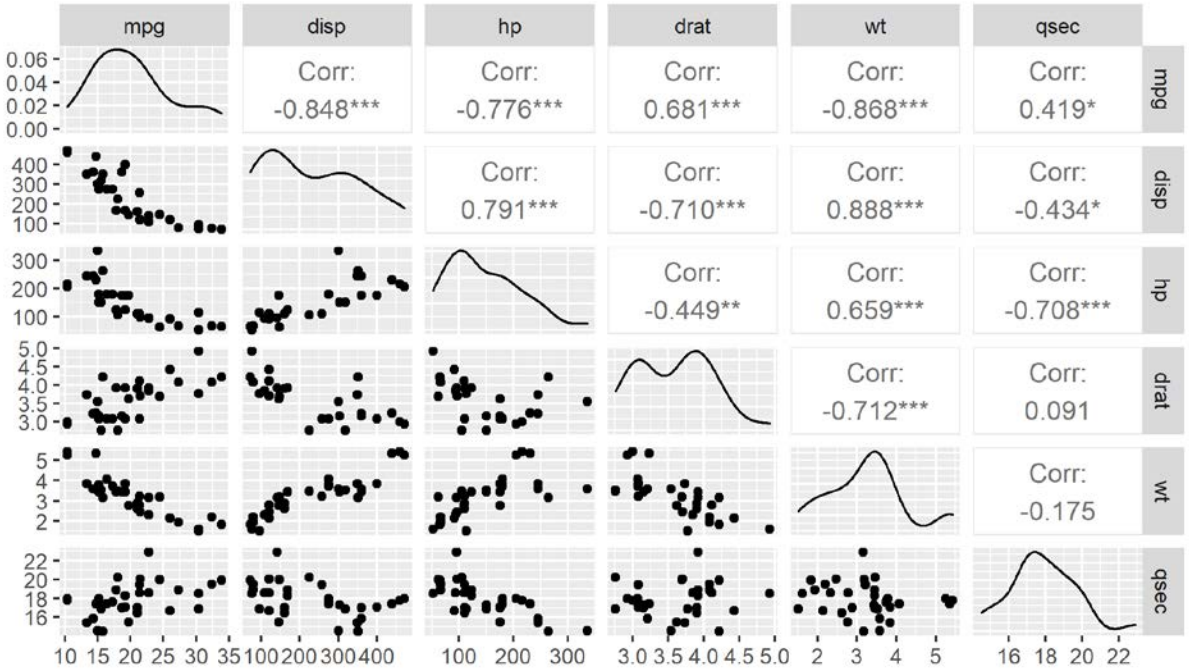
```
# Correlation matrix
R> library("GGally")
R> my_data <- mtcars[, c(1,3,4,5,6,7)]
R> ggcorr(my_data, palette = "RdBu", label = TRUE)
```



รูปที่ 13-17 Correlation Matrix with ggcorr()

รูปด้านบนเป็นการพล็อต Correlation matrix ด้วยคำสั่ง **ggcorr()**

```
# Matrix of scatter plots
R> ggpairs(my_data)
```



รูปที่ 13-18 Correlation Matrix with ggpairs()

รูปด้านบนเป็นการพล็อต Correlation matrix ด้วยคำสั่ง **ggpairs()**

13.2.2 การใช้แพ็กเกจ ggcorrplot

แพ็กเกจ **ggcorrplot** มีคำสั่ง **cor_pmat()** ในการคำนวณ *p*-values ดังต่อไปนี้

```
R> library("ggcorrplot")
R> my_data <- mtcars[, c(1,3,4,5,6,7)]
```

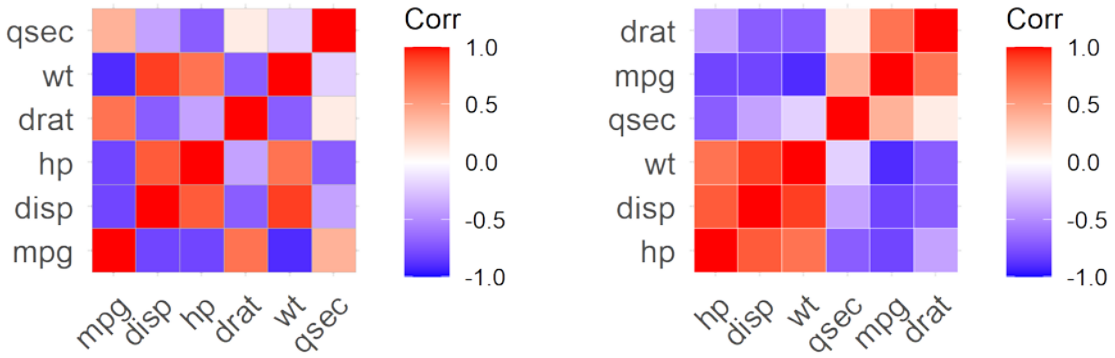
```
R> corr <- round(cor(my_data), 1)

# Compute a matrix of correlation p-values
R> p_mat <- cor_pmat(my_data)
R> head(p_mat[, 1:4], 3)
              mpg         disp           hp         drat
mpg  0.000000e+00 9.380327e-10 1.787835e-07 1.776240e-05
disp 9.380327e-10 0.000000e+00 7.142679e-08 5.282022e-06
hp   1.787835e-07 7.142679e-08 0.000000e+00 9.988772e-03
```

และมีคำสั่ง **ggcorrplot()** ในการพล็อต Correlation matrix ดังตัวอย่างต่อไปนี้

```
# Visualize the correlation matrix
# method = "square" (default)
R> ggcorrplot(corr)

# Reordering the correlation matrix
# using hierarchical clustering
R> ggcorrplot(corr, hc.order = TRUE, outline.col = "white")
```

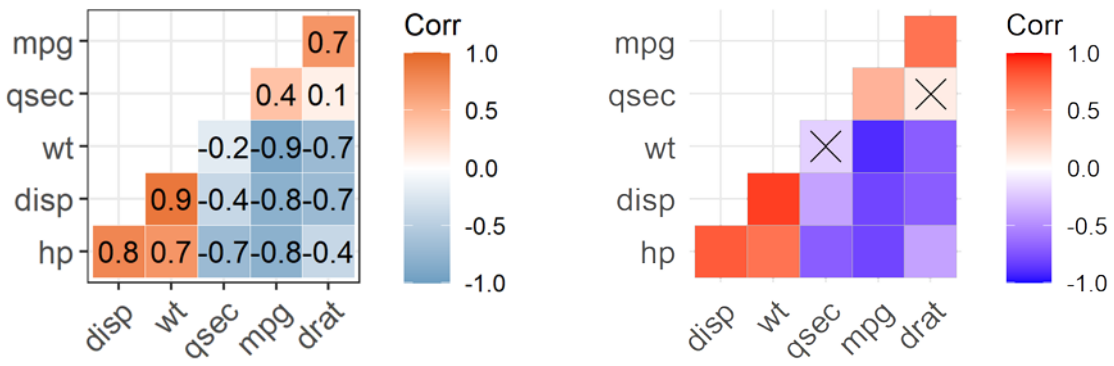


รูปที่ 13-19 Correlation Matrix with ggcorrplot()

รูปด้านบนเป็นการพล็อต Correlation matrix ด้วยคำสั่ง **ggcorrplot()**

```
# Types of correlogram layout and customization
# Add correlation coefficients
R> ggcorrplot(corr, hc.order = TRUE,
              type = "lower", # get the lower triangle
              outline.col = "white",
              ggtheme = ggplot2::theme_bw, # change theme
              colors = c("#6D9EC1", "white", "#E46726"), # change color palette
              lab = TRUE # Add correlation coefficients
   )

# Add correlation significance level
# Argument p.mat
# Barring the no significant coefficient
R> ggcorrplot(corr, hc.order = TRUE,
               type = "lower", p.mat = p_mat)
```



รูปที่ 13-20 Correlation Matrix with ggcorrplot()

รูปด้านบนเป็นการพล็อต Correlation matrix ด้วยคำสั่ง `ggcorrplot()`

13.3 การพล็อต Survival Curves

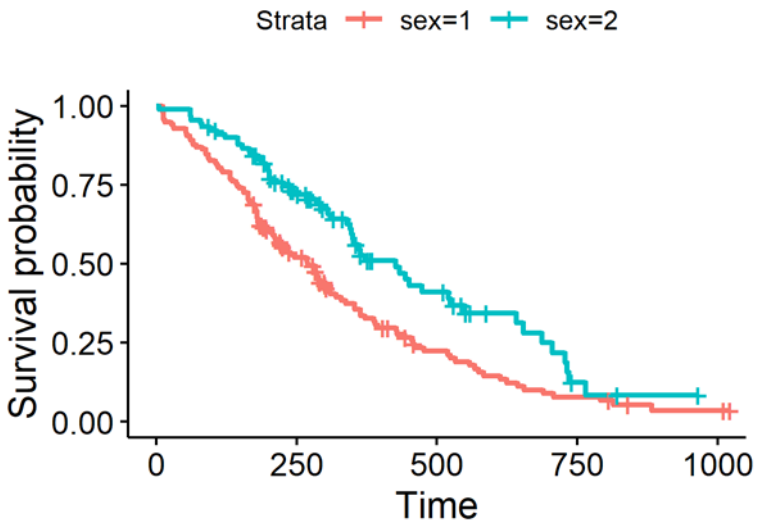
Survival analysis เป็นการวิเคราะห์ความคาดหวังของเหตุการณ์ที่สนใจที่เกิดขึ้นในช่วงเวลาหนึ่ง โดยใช้แพ็กเกจ `survival` ดังนี้

```
R> require("survival")
R> data(lung, package = "survival")

R> fit <- survfit(Surv(time, status) ~ sex, data = lung)
```

การพล็อต Survival curves ใช้คำสั่ง `ggsurvplot()` จากแพ็กเกจ `survminer` ดังนี้

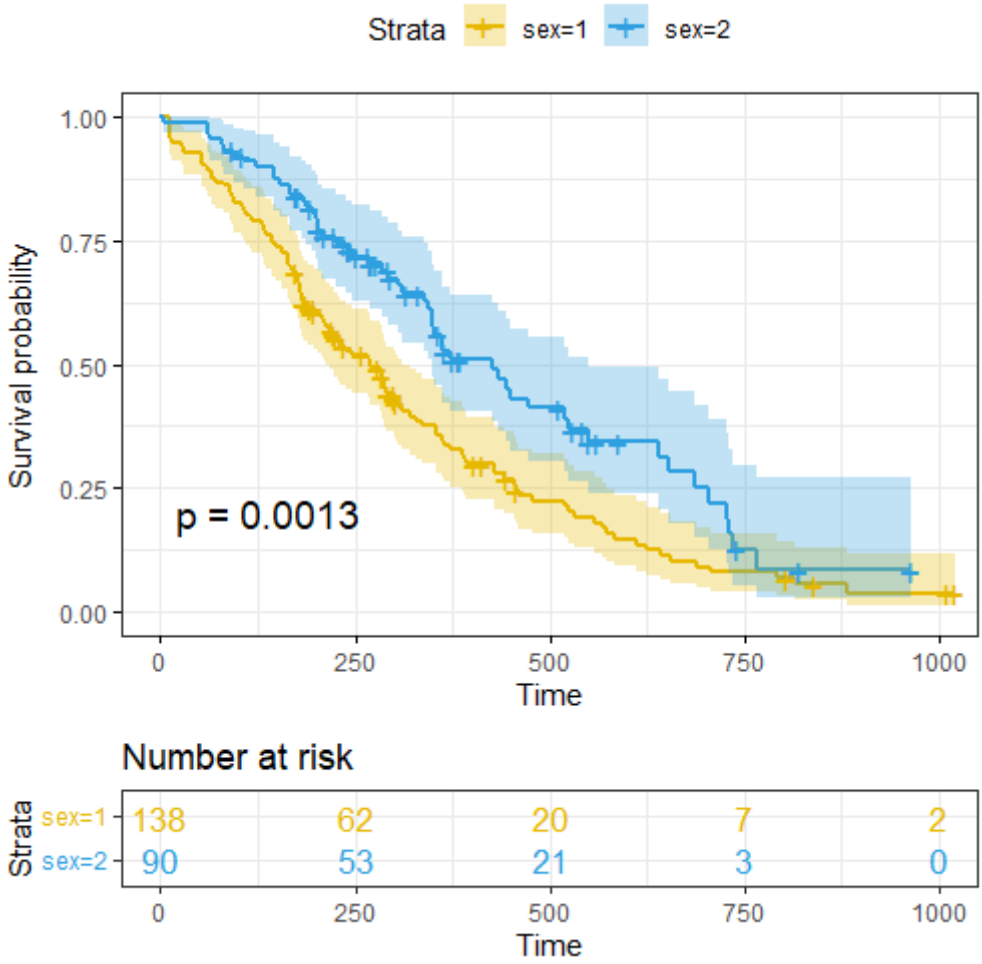
```
R> require(survminer)
# Drawing survival curves
R> ggsurvplot(fit)
```



รูปที่ 13-21 Survival curve

รูปด้านบนแสดง Survival curve แกน X แสดงช่วงเวลาที่เกิด event แกน Y แสดง Survival probability และแยกเพศด้วยสีที่แตกต่างกัน

```
R> ggsurvplot(fit, size = 1, # change line size
              palette = c("#E7B800", "#2E9FDF"), # custom color palette
              conf.int = TRUE, # Add confidence interval
              pval = TRUE, # Add p-value
              risk.table = TRUE, # Add risk table
              risk.table.col = "strata", # Risk table color by groups
              ggtheme = theme_bw() # Change ggplot2 theme
   )
```



รูปที่ 13-22 Survival curve with confidence interval, *p*-value and risk table

รูปด้านบนแสดง Survival curve พร้อมกับช่วงความเชื่อมั่น (Confidence interval), *p*-value และ Risk table

13.4 สรุปท้ายบท

บทนี้กล่าวถึงส่วนขยายต่าง ๆ ของแพ็กเกจ **ggplot2** ที่สามารถทำให้การพล็อตมีความหลากหลาย ทำงานได้สะดวก และคล่องตัวมากขึ้น ได้แก่ การพล็อตกราฟหลายรูปในหน้าเดียวกัน (Arrange multiple graphs on the same page) ด้วยการใช้แพ็กเกจ **cowplot** และ **gridExtra** การพล็อตโดยใช้สัญลักษณ์ร่วมกัน (Common legend for multiple graphs) การสร้าง Scatter plots พร้อมกับ Density plots การเพิ่มองค์ประกอบกราฟิกภายนอกใน ggplot การพล็อต Correlation Matrix ด้วยการใช้แพ็กเกจ **GGally** และ **ggcorrplot** และการพล็อต Survival Curves

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`ggplot()`	คำสั่งในการพล็อต
`geom_boxplot()`	Geometry สำหรับพล็อต Box plots
`geom_violin()`	Geometry สำหรับพล็อต Violins plots
`geom_dotplot()`	Geometry สำหรับพล็อตกราฟจุดแสดงความหนาแน่น (Dot plots)
`geom_jitter()`	Geometry สำหรับพล็อตกราฟที่ต้องการแสดงจุดที่ซ้อนทับกัน (Jitter plots)
`geom_line()`	Geometry สำหรับพล็อตกราฟเส้น
`geom_bar()`	Geometry สำหรับพล็อตกราฟแท่ง
`geom_text()`	Geometry สำหรับเพิ่มข้อความ
`background_grid()`	คำสั่งในการกำหนด Grids
`plot_grid()`	คำสั่งในการรวมกราฟย่อย ๆ เข้ามาอยู่ในรูปเดียวกัน
`ggdraw()`	คำสั่งในการสร้างพื้นที่ว่าง (Empty drawing canvas) สำหรับการพล็อต
`draw_plot()`	คำสั่งในการพล็อตกราฟย่อย ๆ ลงในตำแหน่งและขนาดที่กำหนดในพื้นที่ Drawing canvas
`draw_plot_label()`	คำสั่งในการพล็อตข้อความลงบนกราฟ
`save_plot()`	คำสั่งในการบันทึกกราฟเป็นไฟล์
`grid.arrange()`	คำสั่งในการพล็อตกราฟหลายรูปในหน้าเดียวกัน
`grid.newpage()`	คำสั่งในการสร้างพื้นที่สำหรับการพล็อต
`pushViewport()`	คำสั่งในการเพิ่ม Grid viewport ในพื้นที่สำหรับการพล็อต
`viewport()`	คำสั่งในการสร้าง Grid viewport ในการพล็อต
`ggMarginal()`	คำสั่งในการเพิ่ม Marginal histograms, box plots หรือ density plots เข้าไปใน Plots
`annotation_custom()`	คำสั่งในการสร้าง Static annotations

คำสั่ง	คำอธิบาย
tableGrob()	คำสั่งในการพล็อตตาราง
splitTextGrob()	คำสั่งในการพล็อตข้อความ
paste()	คำสั่งในการเชื่อมต่อข้อความ โดยกำหนดอักขระในการคั่นข้อความได้
paste0()	คำสั่งในการเชื่อมต่อข้อความ โดยไม่มีอักขระคั่นข้อความ
ggcorr()	คำสั่งในการพล็อต Correlation matrix
ggpairs()	คำสั่งในการพล็อต Correlation matrix
ggcorrplot()	คำสั่งในการพล็อต Correlation matrix
survfit()	คำสั่งในการแบบจำลอง Survival analysis
ggsurvplot()	คำสั่งในการพล็อต Survival curves

13.5 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า iris สร้างกราฟเพื่อเปรียบเทียบค่าของ Sepal.Length ในแต่ละ Species

   1.1 สร้างกราฟ Box Plot

   1.2 เพิ่มการแสดงผลข้อมูลแบบ Jitter ด้วยคำสั่ง **`geom_jitter()`** ซ้อนทับบน Box Plot

   1.3 ปรับแต่ง background ของกราฟด้วยคำสั่ง **`background_grid()`** เพื่อเพิ่มความชัดเจน

   1.4 บันทึกกราฟที่ได้โดยใช้คำสั่ง **`save_plot()`**

2. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า mtcars

   2.1 สร้างกราฟความหนาแน่นแบบไวโอลิน (Violin Plot) แสดงความสัมพันธ์ระหว่าง cyl และ mpg ด้วยคำสั่ง **`geom_violin()`**

   2.2 สร้างกราฟจุดแสดงความหนาแน่น (Dot Plot) แสดงจำนวนรถแต่ละประเภท (gear) ด้วย **`geom_dotplot()`**

   2.3 รวมกราฟทั้งสองเข้าด้วยกันในหน้าเดียวด้วยคำสั่ง **`plot_grid()`**

   2.4 เพิ่มข้อความกำกับบนกราฟโดยใช้คำสั่ง **`draw_plot_label()`**

   2.5 บันทึกกราฟที่ได้โดยใช้คำสั่ง **`save_plot()`**

3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ ggplots2 ชื่อว่า diamonds

   3.1 สร้างกราฟ Bar Plot เพื่อแสดงการแจกแจงของ cut

   3.2 เพิ่มคำอธิบายด้วยคำสั่ง geom_text() โดยแสดงจำนวนในแต่ละ Bar

   3.3 ใช้ annotation_custom() เพื่อเพิ่มกราฟข้อความหรือภาพลงในพื้นที่ว่างของกราฟ

4. ใช้คำสั่ง **`ggcorr()`** จากแพ็กเกจ GGally เพื่อแสดง heatmap ของความสัมพันธ์ในข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า airquality
5. ใช้คำสั่ง **`ggpairs()`** เพื่อสร้างกราฟ scatter plot matrix ของข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า PlantGrowth
6. ใช้คำสั่ง **`ggcorrplot()`** เพื่อแสดงผลความสัมพันธ์ในข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า airquality และปรับแต่งสีให้สวยงาม

ส่วนที่ 3
การทำความสะอาดข้อมูล (Data Wrangling)

บทที่ 14

โครงสร้างข้อมูลแบบ tibble และเดต้าเฟรม (data.frame)

ข้อมูลในการวิเคราะห์ส่วนมากมักเก็บในรูปแบบเดต้าเฟรม (data.frame) ซึ่งเป็นรูปแบบการเก็บข้อมูลที่มีตั้งแต่การสร้างภาษา **R** เมื่อหลายสิบปีที่ผ่านมา ปัจจุบันมีการพัฒนาข้อมูลในรูปแบบ tibble เพื่อทำงานร่วมกับแพ็กเกจ **tidyverse** ทำให้จัดการกับข้อมูลที่มีความซับซ้อนได้สะดวก และมีประสิทธิภาพมากขึ้น (H. Wickham & Grolemund, 2016) ในบทต่อ ๆ ไปจะใช้ข้อมูลในรูปแบบ tibble เป็นหลัก

14.1 การสร้างและแสดงผลข้อมูล tibble

การแปลงข้อมูลในรูปแบบเดต้าเฟรม (data.frame) เป็นข้อมูลในรูปแบบ tibble ทำได้โดยใช้คำสั่ง **as_tibble()** ดังตัวอย่างต่อไปนี้

```
R> library(tidyverse)
R> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 ...

R> tbl <- as_tibble(iris)
R> str(tbl)
tibble [150 × 5] (S3: tbl_df/tbl/data.frame)
 $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 ...
R> tbl
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# i 140 more rows
# i Use `print(n = ...)` to see more rows
```

ตัวอย่างข้างบนแสดงโครงสร้างข้อมูลของเดต้าเฟรมข้อมูล iris และข้อมูล tibble ของ tbl ที่แปลงจากข้อมูลของเดต้าเฟรมข้อมูล iris และแสดงข้อมูล tibble ของ tbl ซึ่งจะแสดงผลจำนวนหลักพอดีกับส่วนคอนโซน (Console) แถวแรกแสดงข้อมูลเป็น tibble ขนาด 150 x 5 โดยที่แถวที่ 2 แสดงชื่อตัวแปรหรือชื่อคอลัมน์ ตามด้วยข้อมูล 10 แถวแรก และแถวสุดท้ายแสดงผลว่ามีข้อมูลอีก 140 แถวที่ยังไม่ได้แสดงออกมา

ข้อมูลดังกล่าวสามารถแปลงจาก tibble ให้เป็นเดต้าเฟรม (data.frame) ด้วยคำสั่ง `as.data.frame()` ดังนี้

```
R> class(as.data.frame(tbl))
[1] "data.frame"
```

ผู้ใช้สามารถสร้างข้อมูลในรูปแบบ tibble ได้โดยใช้คำสั่ง **`tibble()`** ซึ่งคำสั่งดังกล่าวสามารถสร้างข้อมูลซ้ำในแถวอื่น ๆ เช่น `y = 1` และสร้างข้อมูลจากการอ้างถึงตัวแปรที่สร้างไว้ก่อนหน้านี้ได้ เช่น `z = x^2 + y` ดังตัวอย่างต่อไปนี้

```
R> tibble(
     x = 1:5,
     y = 1,
     z = x^2 + y
   )
# A tibble: 5 x 3
      x     y     z
  <int> <dbl> <dbl>
1     1     1     2
2     2     1     5
3     3     1    10
4     4     1    17
5     5     1    26
```

ข้อมูลในรูปแบบ tibble สามารถสร้างตัวแปรหรือชื่อคอลัมน์ แตกต่างจากชื่อในภาษา **R** ได้ เช่น ชื่อคอลัมน์ไม่จำเป็นต้องขึ้นต้นด้วยตัวอักษร โดยชื่อคอลัมน์สามารถประกอบด้วยอักขระพิเศษได้ เช่น ช่องว่าง การสร้างตัวแปรหรือชื่อคอลัมน์ ใน tibble ทำได้โดยใช้เครื่องหมาย backtick (`) อย่างไรก็ตาม backtick (`) บนแป้นพิมพ์เป็นการเปลี่ยนภาษาไทย/อังกฤษ ดังนั้นการพิมพ์เครื่องหมาย backtick (`) ให้กดปุ่ม Alt ค้างไว้แล้วตามด้วยตัวเลข 96

ตัวอย่างต่อไปนี้เป็นการสร้างตัวแปรหรือชื่อคอลัมน์ที่มีเครื่องหมายพิเศษ ตัวเลข และช่องว่าง

```
R> data_tbl <- tibble(
    `:)` = "smile",
    ` `  = "space",
    `3000` = "number",
    `name space` = "name with space"
   )
R> data_tbl
# A tibble: 1 x 4
  `:)`  ` `   `3000` `name space`
  <chr> <chr> <chr>  <chr>
1 smile space number name with space
```

ข้อมูลในรูปแบบ tibble สามารถสร้างด้วยคำสั่ง **tribble()** ซึ่งเป็นตัวย่อของคำว่า Transposed tibble คำสั่งนี้เหมาะสำหรับนำเข้าข้อมูลจำนวนไม่มาก ซึ่งง่ายต่อการอ่านข้อมูลในแต่ละแถว โดยตัวแปรหรือชื่อคอลัมน์จะเริ่มต้นด้วยเครื่องหมาย (~) สามารถสร้างเครื่องหมาย (~) โดยการกดปุ่ม Alt ค้างไว้แล้วตามด้วยตัวเลข 126 ข้อมูลแต่ละคอลัมน์คั่นด้วยเครื่องหมายจุลภาค (Comma) และสามารถใส่เครื่องหมาย # หน้าบรรทัดที่ต้องการ Comments ได้ดังตัวอย่างต่อไปนี้

```
R> tribble(
     ~x, ~y,  ~z,
    #--|---|----|
    "a",  1, 2.5,
    "b",  2, 3.7
   )
# A tibble: 2 x 3
  x         y     z
  <chr> <dbl> <dbl>
1 a         1   2.5
2 b         2   3.7
```

ข้อมูลในรูปแบบ tibble ออกแบบมาให้แสดงจำนวนคอลัมน์ พอดีกับหน้าจอแสดงผล ถ้าต้องการแสดงคอลัมน์ ทั้งหมดให้ใช้คำสั่ง `print()` และใส่อากิวเมนต์ `width = Inf` ดังนี้

```
R> nycflights13::flights
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, tailnum
   <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
# i Use `print(n = ...)` to see more rows

R> nycflights13::flights |>
     print(n = 10, width = Inf)
```

สามารถใช้เครื่องหมาย `|>` สำหรับการใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) หรือบางครั้งเรียกว่าการ Pipe (Piping)

สามารถใช้ Data viewer ของ RStudio ในการแสดงข้อมูล ซึ่งทำให้สามารถเลื่อนดูข้อมูลได้สะดวก โดยการเพิ่มคำสั่ง **`View()`** เป็นคำสั่งสุดท้าย ซึ่งเหมาะสำหรับการใช้คำสั่งต่อเนื่องแบบลูกโซ่ ซึ่งประกอบด้วยคำสั่งต่อเนื่องกันหลายคำสั่ง ดังนี้

```
R> nycflights13::flights |>
     # ... Multiple lines
     # ... Multiple lines
     # ... Multiple lines
     View()
```

14.2 สับเซต (Subset)

การดึงตัวแปรบางตัวจากข้อมูลในรูปแบบ tibble ทำได้โดยใช้เครื่องหมาย **`$`** และ **`[[ ]]`** ดังนี้

```
R> tbl <- tibble(
    x = runif(5),
    y = rnorm(5)
   )
R> tbl$x
[1] 0.1494248 0.8175165 0.7517270 0.4708122 0.9448250

R> tbl[["x"]]
[1] 0.1494248 0.8175165 0.7517270 0.4708122 0.9448250

R> tbl[[1]]
[1] 0.1494248 0.8175165 0.7517270 0.4708122 0.9448250
```

คำสั่งแรกเป็นการสร้าง tibble คำสั่งที่ 2 และ 3 เป็นการดึงข้อมูลโดยใช้ชื่อตัวแปร และคำสั่งสุดท้ายเป็นการดึงข้อมูลตำแหน่งแรกออกมา

การใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) เพื่อดึงตัวแปรออกมาต้องใช้เครื่องหมายสัญประกาศหรือขีดเส้นใต้ (**`_`**) ดังตัวอย่างต่อไปนี้

```
R> tbl |> _$x
[1] 0.1494248 0.8175165 0.7517270 0.4708122 0.9448250

R> tbl |> _[["x"]]
[1] 0.1494248 0.8175165 0.7517270 0.4708122 0.9448250
```

14.3 สรุปท้ายบท

บทนี้เป็นการแนะนำโครงสร้างข้อมูลแบบ tibble นอกเหนือการโครงสร้างข้อมูลแบบเดต้าเฟรม (data.frame) ข้อมูลในรูปแบบ tibble มักจะใช้ทำงานกับแพ็กเกจ **`tidyverse`** ทำให้จัดการกับข้อมูลที่มีความซับซ้อนได้สะดวก และมีประสิทธิภาพมากขึ้น กล่าวถึงการสร้างและแสดงผลข้อมูล tibble การดึงตัวแปรบางตัวจากข้อมูลในรูปแบบ tibble (Subset)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`as_tibble()`	คำสั่งในการแปลงเดต้าเฟรม (data.frame) ให้เป็น tibble
`as.data.frame()`	คำสั่งในการแปลงข้อมูลให้เป็นเดต้าเฟรม (data.frame)
`class()`	คำสั่งตรวจสอบ class ของข้อมูล
`tibble()`	คำสั่งในการสร้าง tibble
`tribble()`	คำสั่งในการสร้าง tibble โดยใช้รูปแบบข้อมูลที่มีการ Transpose
`runif()`	คำสั่งในการสร้างเลขสุ่มจาก Uniform Distribution
`rnorm()`	คำสั่งในการสร้างเลขสุ่มจาก Normal Distribution
`|>`	การใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) หรือบางครั้งเรียกว่าการ Pipe
`print()`	คำสั่งในการพิมพ์ข้อความ/ข้อมูล
`View()`	คำสั่งในการแสดงข้อมูลใน Data viewer ของ RStudio
`$`	คำสั่งในการเข้าถึงสมาชิกใน tibble
`[[ ]]`	คำสั่งในการเข้าถึงสมาชิกใน tibble

14.4 แบบฝึกหัดท้ายบท

1. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้ และเก็บข้อมูลไว้ในตัวแปรชื่อ student_tbl

ข้อมูลคะแนนสอบรายบุคคล

Student_ID	Name	Math	Science	English
S001	Alice	85	80	88
S002	Bob	90	85	76
S003	Charlie	78	89	93
S004	David	88	84	87
S005	Eva	92	91	90
S006	Peter	68	75	95
S007	Micle	71	84	82
S008	John	85	69	75
S009	James	74	90	89
S010	Robert	76	97	87

   1.1 ให้ดึงข้อมูลชื่อจาก student_tbl เก็บไว้ในตัวแปรชื่อว่า student_name

   1.2 ให้ดึงข้อมูลคะแนนวิชาคณิตศาสตร์จาก student_tbl เก็บไว้ในตัวแปรชื่อว่า math_score

   1.3 ให้ดึงข้อมูลคะแนนวิชาวิทยาศาสตร์จาก student_tbl เก็บไว้ในตัวแปรชื่อว่า science_score

   1.4 ให้ดึงข้อมูลคะแนนวิชาภาษาอังกฤษจาก student_tbl เก็บไว้ในตัวแปรชื่อว่า english_score

2. สร้าง tibble ที่ประกอบด้วยคอลัมน์ x เก็บข้อมูลเลขสุ่มที่มีการกระจายตัวแบบ Normal Distribution จำนวน 10 ค่าและคอลัมน์ y เก็บข้อมูลเลขสุ่มที่มีการกระจายตัวแบบ Uniform Distribution และคอลัมน์ z เก็บข้อมูลของ x + y จากนั้นดึงข้อมูล z จาก tibble มาแสดงผล
3. แปลงข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า swiss ให้อยู่ในโครงสร้าง tibble และแสดงผล เปรียบเทียบความแตกต่างของการแสดงผล

บทที่ 15
การจัดการข้อมูลด้วยแพ็กเกจ tidyr

แพ็กเกจใน **R** ที่สำคัญหลายแพ็กเกจได้ถูกพัฒนาขึ้นมารองรับการทำงานที่ซับซ้อนมากขึ้น แพ็กเกจที่มีผู้นิยมใช้มากที่สุดแพ็กเกจหนึ่งคือ **tidyverse** ซึ่งรวมแพ็กเกจย่อยหลาย ๆ แพ็กเกจเข้าด้วยกัน แพ็กเกจดังกล่าวถูกพัฒนาโดย Hadley Wickham (H. Wickham & Grolemund, 2016) ซึ่งเป็นผู้พัฒนา RStudio และ **R** แพ็กเกจหลายตัวที่ใช้กันอย่างแพร่หลาย ได้แก่ **ggplot2, tibble, tidyr, dplyr, readr, stringr, forcats** และ **purrr**

ในบทนี้และบทต่อ ๆ ไปจะกล่าวถึงการใช้งานแพ็กเกจย่อย ๆ ที่อยู่ใน **tidyverse** ซึ่งจำเป็นอย่างมากในการจัดเตรียมข้อมูล การทำความสะอาดข้อมูล และวิเคราะห์ข้อมูลที่ซับซ้อนได้ โดยทั่วไปการทำงานในขั้นตอนการทำความสะอาดข้อมูลใช้เวลาถึงร้อยละ 80 ของเวลาทั้งหมดในการทำงาน (Andrews, 2021) การใช้แพ็กเกจ **tidyverse** จะช่วยให้การทำงานมีประสิทธิภาพมากขึ้น ช่วยลดเวลาการทำงานในขั้นตอนดังกล่าวได้มาก

15.1 การแปลงข้อมูลด้วยแพ็กเกจ tidyr

ข้อมูลโดยทั่วไปมักอยู่ในรูปแบบที่ไม่เหมาะที่จะนำไปประมวลผลต่อ **R** มีแพ็กเกจที่ช่วยแปลงข้อมูลให้อยู่ในรูปแบบที่ต้องการได้ชื่อว่า **tidyr**

RStudio ได้สร้างไฟล์สรุปการใช้งาน **tidyr** ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมาก (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว)

ข้อมูลโดยทั่วไปมักบันทึกไว้ 2 รูปแบบคือ (1) แบบตามแนวกว้าง (Wide format) และ (2) แบบตามแนวยาว (Long format) ตัวอย่างเช่น ข้อมูลคะแนนสอบในรายวิชาวิทยาศาสตร์ (Science) คณิตศาสตร์ (Mathematics) และวรรณคดี (Literature) ของนักศึกษา 4 คน บันทึกในรูปแบบตามแนวกว้าง (Wide format) ดังนี้

```
R> tbl1 <- tibble(student = c("Anne", "Jim", "Joe", "Mary"),
                  science = c(90, 83, 75, 88),
                   math = c(85, 93, 79, 82),
                  literature = c(94, 76, 91, 84))
# Wide format
R> tbl1
# A tibble: 4 x 4
  student science  math literature
  <chr>     <dbl> <dbl>      <dbl>
1 Anne         90    85         94
2 Jim          83    93         76
3 Joe          75    79         91
4 Mary         88    82         84
```

แพ็กเกจ **tidyr** มีคำสั่ง **gather()** ในการแปลงข้อมูลรูปแบบตามแนวกว้างให้เป็นข้อมูลรูปแบบตามแนวยาว โดยการกำหนดอากิวเมนต์ **key** และ **value** และคอลัมน์ (Columns) ที่ต้องการแปลง ดังนี้

```
R> tbl2 <- tbl1 |> gather(key = subject, value = score, science:literature)
# equivalent to gather(tbl1, subject, score, science:literature)
# equivalent to tbl1 |> gather(subject, score, science:literature)
# equivalent to tbl1 |> gather(subject, score, -student)

# Long format
R> tbl2
# A tibble: 12 x 3
   student subject    score
   <chr>   <chr>      <dbl>
 1 Anne    science       90
 2 Jim     science       83
 3 Joe     science       75
 4 Mary    science       88
 5 Anne    math          85
 6 Jim     math          93
 7 Joe     math          79
 8 Mary    math          82
 9 Anne    literature    94
10 Jim     literature    76
11 Joe     literature    91
12 Mary    literature    84
```

คำสั่ง **gather()** จะแปลงตัวแปรคอลัมน์ ได้แก่ วิทยาศาสตร์ (Science) คณิตศาสตร์ (Mathematics) และวรรณคดี (Literature) (**science:literature**) ไปเป็นข้อมูลในคอลัมน์ใหม่ (Column) ชื่อว่า **key** ชื่อคอลัมน์ใหม่ถูกกำหนดด้วยอากิวเมนต์ **key** ในตัวอย่างนี้คือ **key = subject** ส่วนข้อมูลในแต่ละ Cell จะถูกแปลงไปเป็นข้อมูลในคอลัมน์ใหม่ชื่อว่า **value** ชื่อคอลัมน์ใหม่กำหนดด้วยอากิวเมนต์ **value** ในตัวอย่างนี้กำหนดให้ **value = score**

ข้อมูลรูปแบบตามแนวกว้าง (Wide format) นั้นเหมือนกับข้อมูลที่ผ่านการ Pivot table ใน Spread sheet เช่น Excel ส่วนข้อมูลรูปแบบตามแนวยาว (Long format) เป็นข้อมูลที่มักจัดเก็บในฐานข้อมูลทั่วไป โปรแกรมสำเร็จรูปทางด้านสถิติส่วนมากมักใช้ข้อมูลลักษณะนี้ในการประมวลผล

เครื่องหมาย **|>** เรียกว่า piping เป็นคำสั่งต่อเนื่องแบบลูกโซ่ (Chain) มีประโยชน์มากในการเขียนคำสั่งที่ซับซ้อนต่อเนื่องกันไป การใช้เครื่องหมาย **|>** หมายความว่า เอาต์พุต (Output) ของคำสั่งด้านซ้ายเครื่องหมาย **|>** จะเป็นอินพุต (Input) ของคำสั่งด้านขวาของเครื่องหมายดังกล่าว

นอกจากคำสั่ง **gather()** แพ็กเกจ **tidyr** ยังมีคำสั่ง **pivot_longer()** ที่สามารถแปลงข้อมูลรูปแบบตามแนวกว้างให้เป็นข้อมูลรูปแบบตามแนวยาว โดยการกำหนดอากิวเมนต์คอลัมน์ที่ต้องการแปลง **(cols)** ชื่อคอลัมน์ใหม่ที่เก็บค่าชื่อของคอลัมน์ที่แปลง **(names_to)** และชื่อคอลัมน์ใหม่ที่เก็บค่าของคอลัมน์ที่ถูกแปลง **(values_to)** ดังตัวอย่างต่อไปนี้

```
R> tbl1 |>
     pivot_longer(cols = c(science:literature),
                  names_to = "subject", values_to = "score")
# A tibble: 12 × 3
   student subject    score
   <chr>   <chr>      <dbl>
 1 Anne    science       90
 2 Anne    math          85
 3 Anne    literature    94
 4 Jim     science       83
 5 Jim     math          93
 6 Jim     literature    76
 7 Joe     science       75
 8 Joe     math          79
 9 Joe     literature    91
10 Mary    science       88
11 Mary    math          82
12 Mary    literature    84
```

ในทางกลับกันแพ็กเกจ **tidyr** มีคำสั่ง **spread()** ในการแปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง โดยการกำหนดอากิวเมนต์ **key** และ **value** ดังนี้

```
R> tbl2 |> spread(key = subject, value = score)
# equivalent to spread(tbl2, subject, score)
# equivalent to tbl2 |> spread(subject, score)
# wide format
# A tibble: 4 x 4
  student literature  math science
  <chr>        <dbl> <dbl>   <dbl>
1 Anne            94    85      90
2 Jim             76    93      83
3 Joe             91    79      75
4 Mary            84    82      88
```

คำสั่ง **spread()** จะแปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง โดยแปลงข้อมูลใน **key** ได้แก่ literature, math, science ให้เป็นคอลัมน์ที่มีชื่อตรงกับข้อมูลที่กำหนด ส่วนค่าของข้อมูลในคอลัมน์ที่สร้างขึ้นใหม่ได้จากข้อมูลใน **value**

นอกจากคำสั่ง **spread()** แพ็กเกจ **tidyr** ยังมีคำสั่ง **pivot_wider()** ที่สามารถแปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง โดยการกำหนดอากิวเมนต์คอลัมน์ที่ต้องการแปลง (**names_from**) และชื่อคอลัมน์ที่เก็บค่า (**values_from**) ดังตัวอย่างต่อไปนี้

```
R> tbl2 |>
     pivot_wider(names_from = subject, values_from = score)
# A tibble: 4 × 4
  student science  math literature
  <chr>     <dbl> <dbl>      <dbl>
1 Anne         90    85         94
2 Jim          83    93         76
3 Joe          75    79         91
4 Mary         88    82         84
```

15.2 การแยกข้อมูลหนึ่งคอลัมน์เป็นหลายคอลัมน์ด้วยคำสั่ง separate()

แพ็กเกจ **tidyr** มีคำสั่งในการแยกข้อมูลหนึ่งคอลัมน์เป็นหลายคอลัมน์ โดยกำหนดชื่อคอลัมน์ใหม่ด้วยอากิวเมนต์ **into** กำหนดตัวคั่นด้วยอากิวเมนต์ **sep** ถ้าต้องการเก็บคอลัมน์เดิมไว้ให้กำหนดอากิวเมนต์ **remove = FALSE** ตัวอย่างเช่น ต้องการแยกชื่อเล่น ชื่อจริง นามสกุล คำนำหน้าชื่อ และวัน เดือน ปีเกิด ออกจากกัน โดยสร้างเป็นคอลัมน์ใหม่ และเก็บคอลัมน์ birth เดิมไว้ ดังตัวอย่างต่อไปนี้

```
R> tbl3 <- tibble(student = c("Anne-Anna-Smith-Ms", "Jim-Jimmy-Tomsom-Mr",
                              "Joe-Joey-Moore-Mr", "Mary-Mary-Anne-Ms"),
                  birth = as.Date(c("1999-04-24", "2001-05-16",
                                    "2002-08-19", "1997-12-30")))
R> tbl3
# A tibble: 4 x 2
  student             birth
  <chr>               <date>
1 Anne-Anna-Smith-Ms  1999-04-24
2 Jim-Jimmy-Tomsom-Mr 2001-05-16
3 Joe-Joey-Moore-Mr   2002-08-19
4 Mary-Mary-Anne-Ms   1997-12-30

R> tbl4 <- tbl3 |>
     separate(student, into = c("nick_n","name","surname","title"),
              sep = "-") |>
     separate(birth, into = c("year","month","day"), sep = "-", remove = FALSE)
R> tbl4
# A tibble: 4 x 8
  nick_n name  surname title birth      year  month day
  <chr>  <chr> <chr>   <chr> <date>     <chr> <chr> <chr>
1 Anne   Anna  Smith   Ms    1999-04-24 1999  04    24
2 Jim    Jimmy Tomsom  Mr    2001-05-16 2001  05    16
3 Joe    Joey  Moore   Mr    2002-08-19 2002  08    19
4 Mary   Mary  Anne    Ms    1997-12-30 1997  12    30
```

ตัวอย่างข้างบนมีการใช้เครื่องหมาย **|>** เป็นคำสั่งต่อเนื่องแบบลูกโซ่จาก tbl3 ต่อด้วยคำสั่ง **separate(student, …)** และต่อเนื่องไปที่คำสั่ง **separate(birth, …)** ตามลำดับ

นอกจากนี้ **separate()** ยังสามารถแยกตัวอักษรตามลำดับตำแหน่งได้ โดยกำหนดเลขบวกแสดงลำดับตัวอักษรที่ต้องการแยกจากซ้ายไปขวา และกำหนดเลขติดลบแสดงลำดับตัวอักษรที่ต้องการแยกจากขวาไปซ้าย (จากตัวสุดท้าย) เช่น ต้องการแยกระดับชั้นของนักศึกษา (D = doctor, M = master, B = bachelor) ปีที่เข้ารับการศึกษา ลำดับนักศึกษา เพศ (M = male, F = female) และน้ำหนัก ออกเป็นคอลัมน์ใหม่ ดังนี้

```
R> tbl5 <- tibble(id = c("B561780M", "M570087F", "B574822F",
                         "D559086M", "M568643M", "B584562F"),
                  weightKg = c("102kg", "68kg", "57kg", "73kg",
                               "110kg", "48kg"))
R> tbl5
# A tibble: 6 x 2
  id       weightKg
  <chr>    <chr>
```

```
1 B561780M 102kg
2 M570087F 68kg
3 B574822F 57kg
4 D559086M 73kg
5 M568643M 110kg
6 B584562F 48kg
```

```
R> tbl6 <- tbl5 |>
     separate(id, into = c("degree","year","id_student","gender"),
              sep = c(1,3,7)) |>
     separate(weightKg, into = c("weight","unit"), sep = -2, remove = FALSE)
R> tbl6
# A tibble: 6 x 7
  degree year  id_student gender weightKg weight unit
  <chr>  <chr> <chr>      <chr>  <chr>    <chr>  <chr>
1 B      56    1780       M      102kg    102    kg
2 M      57    0087       F      68kg     68     kg
3 B      57    4822       F      57kg     57     kg
4 D      55    9086       M      73kg     73     kg
5 M      56    8643       M      110kg    110    kg
6 B      58    4562       F      48kg     48     kg
```

15.3 การรวมข้อมูลหลายคอลัมน์เป็นหนึ่งคอลัมน์ด้วยคำสั่ง unite()

แพ็กเกจ **tidyr** มีคำสั่งในการรวมข้อมูลหลายคอลัมน์ให้เป็นหนึ่งคอลัมน์ โดยกำหนดชื่อคอลัมน์ใหม่ให้พารามิเตอร์ **col** ตามด้วยคอลัมน์ที่ต้องการรวมข้อมูล กำหนดตัวคั่นด้วยอากิวเมนต์ **sep** ค่าโดยปริยาย (default) กำหนดให้ **sep = "_"** ถ้าต้องการเก็บคอลัมน์เดิมไว้ให้กำหนดอากิวเมนต์ **remove = FALSE** ตัวอย่างเช่น ต้องการสร้างรหัสนักศึกษาจากระดับชั้นของนักศึกษา (D = doctor, M = master, B = bachelor) ปีที่เข้ารับการศึกษา ลำดับนักศึกษา โดยสร้างเป็นคอลัมน์ใหม่ และเก็บคอลัมน์เดิมไว้ ดังนี้

```
R> tbl7 <- tbl6 |>
     unite(col = id, c(degree, year, id_student), remove = FALSE, sep = "")
R> tbl7
# A tibble: 6 x 8
  id      degree year  id_student gender weightKg weight unit
  <chr>   <chr>  <chr> <chr>      <chr>  <chr>    <chr>  <chr>
1 B561780 B      56    1780       M      102kg    102    kg
2 M570087 M      57    0087       F      68kg     68     kg
3 B574822 B      57    4822       F      57kg     57     kg
4 D559086 D      55    9086       M      73kg     73     kg
5 M568643 M      56    8643       M      110kg    110    kg
6 B584562 B      58    4562       F      48kg     48     kg
```

15.4 การจัดการข้อมูลที่ขาดหายไป (Missing Values)

ข้อมูลที่ขาดหายไป (Missing Values) มี 2 ลักษณะได้แก่ (1) ข้อมูลที่หายไปที่แสดงด้วยเครื่องหมาย **NA** (Explicit) (2) ข้อมูลที่หายไปที่ไม่ปรากฏอยู่ในตัวข้อมูล (Implicit) ดังตัวอย่างต่อไปนี้

```
R> stocks <- tibble(
     year = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
     qtr   = c(1, 2, 3, 4, 2, 3, 4),
     return = c(1.88, 0.59, 0.35, NA, 0.92, 0.17, 2.66)
    )

R> stocks
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4  NA
5  2016     2   0.92
6  2016     3   0.17
7  2016     4   2.66
```

ข้อมูลข้างต้นเป็นข้อมูลหุ้นชนิดหนึ่ง (Stocks) ประกอบด้วยข้อมูลผลตอบแทน (Return) ในแต่ละไตรมาส (Quarter) ตั้งแต่ปี 2015 ถึงปี 2016 ข้อมูลที่หายไปมี 2 ตัวคือ (1) ข้อมูลไตรมาสที่ 4 ปี 2015 และ (2) ข้อมูลไตรมาสที่ 1 ปี 2016

สามารถแสดงข้อมูลที่หายไปที่ไม่ปรากฏอยู่ในตัวข้อมูล (Implicit) ได้ด้วยการแยกข้อมูลหนึ่งคอลัมน์เป็นหลายคอลัมน์ด้วยคำสั่ง `spread()` หรือ `pivot_wider()` ดังนี้

```
# equivalent to stocks |> pivot_wider(names_from = year, values_from = return)
R> stocks |>
     spread(year, return)

# A tibble: 4 x 3
    qtr `2015` `2016`
  <dbl>  <dbl>  <dbl>
1     1   1.88  NA
2     2   0.59   0.92
3     3   0.35   0.17
4     4   NA     2.66
```

ถ้าไม่มีความจำเป็นในการแสดงข้อมูลที่หายไป สามารถใช้คำสั่ง `gather()` และระบุอากิวเมนต์ `na.rm = TRUE` ได้ดังนี้

```
R> stocks |>
     spread(year, return) |>
     gather(year, return, `2015`:`2016`, na.rm = TRUE)

# A tibble: 6 x 3
    qtr year  return
  <dbl> <chr>  <dbl>
1     1 2015    1.88
2     2 2015    0.59
3     3 2015    0.35
4     2 2016    0.92
5     3 2016    0.17
```

```
6     4 2016    2.66
```

ถ้าต้องการแสดงข้อมูลที่หายไป สามารถใช้คำสั่ง `complete()` ได้ดังนี้

```
R> stocks |>
     complete(year, qtr)

# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4   NA
5  2016     1   NA
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

คำสั่ง `complete()` เป็นการ combinations ตัวแปร/คอลัมน์ที่กำหนดทั้งหมด และเติมข้อมูลที่หายไปด้วยเครื่องหมาย `NA`

สร้างข้อมูล treatment ซึ่งมีข้อมูล `NA` ในตัวแปร person ในแถวที่ 2 และ 3 ดังนี้

```
R> treatment <- tribble(
     ~ person, ~ treatment, ~response,
     "Derrick Whitmore", 1, 7,
     NA, 2, 10,
     NA, 3, 9,
     "Katherine Burke", 1, 4
   )

R> treatment
# A tibble: 4 x 3
  person           treatment response
  <chr>                <dbl>    <dbl>
1 Derrick Whitmore         1        7
2 NA                       2       10
3 NA                       3        9
4 Katherine Burke          1        4
```

สามารถเติมข้อมูลที่หายไป ด้วยคำสั่ง `fill()` ซึ่งจะทำการคัดลอกข้อมูลในคอลัมน์ที่หายไปจากข้อมูลคอลัมน์ที่มี ดังนี้

```
R> treatment |>
     fill(person)

# A tibble: 4 x 3
  person           treatment response
  <chr>                <dbl>    <dbl>
1 Derrick Whitmore         1        7
2 Derrick Whitmore         2       10
3 Derrick Whitmore         3        9
4 Katherine Burke          1        4
```

จากผลลัพธ์ที่ได้จะเห็นว่าข้อมูล **NA** ในแถวที่ 2 และ 3 ที่เคยหายไปถูกแทนที่ด้วย “Derrick Whitmore”

15.5 สรุปท้ายบท

บทนี้เป็นจัดการข้อมูลด้วยแพ็กเกจ **tidyr** ได้แก่ การแปลงข้อมูลรูปแบบตามแนวกว้าง (Wide format) ให้เป็นข้อมูลรูปแบบตามแนวยาว (Long format) การแปลงข้อมูลรูปแบบตามแนวยาว (Long format) ให้เป็นข้อมูลรูปแบบตามแนวกว้าง (Wide format) การแยกข้อมูลหนึ่งคอลัมน์เป็นหลายคอลัมน์ การรวมข้อมูลหลายคอลัมน์เป็นหนึ่งคอลัมน์ และการจัดการข้อมูลที่ขาดหายไป (Missing Values)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`as_tibble()`	คำสั่งในการแปลงเดต้าเฟรม (data.frame) ให้เป็น tibble
`tibble()`	คำสั่งในการสร้าง tibble
`tribble()`	คำสั่งในการสร้าง tibble โดยใช้รูปแบบข้อมูลที่มีการ Transpose
`|>`	การใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) หรือบางครั้งเรียกว่าการ Pipe
`gather()`	คำสั่งในการแปลงข้อมูลรูปแบบตามแนวกว้างให้เป็นข้อมูลรูปแบบตามแนวยาว
`pivot_longer()`	คำสั่งในการแปลงข้อมูลรูปแบบตามแนวกว้างให้เป็นข้อมูลรูปแบบตามแนวยาว
`spread()`	คำสั่งในในการแปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง
`pivot_wider()`	คำสั่งในในการแปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง
`separate()`	คำสั่งในการแยกข้อมูลหนึ่งคอลัมน์เป็นหลายคอลัมน์
`unite()`	คำสั่งในการรวมข้อมูลหลายคอลัมน์ให้เป็นหนึ่งคอลัมน์
`complete()`	คำสั่งในการ combinations ตัวแปร/คอลัมน์ที่กำหนดทั้งหมด และเติมข้อมูลที่หายไปด้วยเครื่องหมาย NA
`fill()`	คำสั่งในการเติมข้อมูลที่ขาดหายไป (NA)

15.6 แบบฝึกหัดท้ายบท

1. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้

ข้อมูลการเดินทางรายบุคคล

Trip ID	Income	Mode	Cost
1	30,000	Car	150
2	30,000	Car	125
3	40,000	Car	125
4	50,000	Car	225
5	25,000	Bus	100
6	32,000	Car	100
7	28,000	Bus	75
8	50,000	Car	150
9	20,000	Bus	200
10	34,000	Bus	150

1.1 แปลงข้อมูลรูปแบบตามแนวยาวให้เป็นข้อมูลรูปแบบตามแนวกว้าง ใช้คำสั่ง `spread()` โดยให้และ `pivot_wider()` สร้างคอลัมน์ใหม่จากคอลัมน์ Mode และในแต่ละคอลัมน์เก็บค่าของคอลัมน์ Cost

1.2 แปลงข้อมูลให้เป็นรูปแบบตามแนวกว้างอีกครั้ง โดยใช้คำสั่ง `gather()` และ `pivot_longer()`

2. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้

ข้อมูลลูกค้าแต่ละรายในฐานข้อมูล

Name-Customer ID	Birth	Age	Gender	Income
Olivia-ID15101	1999-05-22	48	FEMALE	17,546.0
Sophia-IDB15102	2000-08-01	50	MALE	30,085.1
Emily-ID15103	2002-12-18	51	FEMALE	16,575.4
Amelia-ID15104	1997-01-30	23	FEMALE	20,375.4

2.1 ต้องการแยก Name ออกจาก Customer ID จากคอลัมน์ Name-Customer ID และแยกวัน เดือน ปีออกเป็นคอลัมน์ใหม่จากคอลัมน์ Birth โดยยังเก็บคอลัมน์เดิมไว้

2.2 ต้องการสร้างคอลัมน์ใหม่ โดยแสดงตัวย่อของ Gender (FEMALE=F, MALE=M) รวมกับ Age เช่น Gender=FEMALE, Age=48 แสดงเป็น 48-F

บทที่ 16
การจัดการข้อมูลด้วยแพ็กเกจ dplyr

แพ็กเกจชื่อว่า **dylyr** ซึ่งเป็นส่วนหนึ่งของ แพ็กเกจ **tidyverse** ช่วยจัดการข้อมูลให้อยู่ในรูปแบบที่ต้องการ เช่น เลือกแถวข้อมูล เลือกตัวแปรที่ต้องการ สร้างตัวแปรใหม่ สรุปข้อมูล และเรียงลำดับข้อมูล คำสั่งในแพ็กเกจนี้ประกอบด้วยคำสั่งที่สำคัญ ดังต่อไปนี้

- `filter()` สำหรับเลือกแถวข้อมูลที่ต้องการ
- `select()` สำหรับเลือกตัวแปรที่ต้องการตามที่กำหนด
- `mutate()` สำหรับสร้างตัวแปรใหม่จากตัวแปรเดิมที่มีอยู่
- `summarise()` สำหรับสรุปข้อมูลหลายตัวแปร
- `arrange()` สำหรับเรียงลำดับข้อมูลในแถว

RStudio ได้สร้างไฟล์สรุปการใช้งาน **dylyr** ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมากสำหรับผู้ใช้ **R** (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว)

ในการยกตัวอย่างจะใช้ข้อมูลที่ติดตั้งมาพร้อมกับ **ggplot2** ชื่อว่า mpg เป็นข้อมูลการใช้น้ำมันของรถยนต์ที่นิยมใช้ในช่วงปี ค.ศ. 1999 ถึง 2008 ซึ่งถูกรวบรวมโดย US Environmental Protection Agency (http://fueleconomy.gov) ดังนี้

```
R> mpg
# A tibble: 234 x 11
   man   model  displ  year   cyl trans drv     cty   hwy fl    class
   <chr> <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
 1 audi  a4      1.80  1999     4 auto~ f        18    29 p     comp~
 2 audi  a4      1.80  1999     4 manu~ f        21    29 p     comp~
 3 audi  a4      2.00  2008     4 manu~ f        20    31 p     comp~
 4 audi  a4      2.00  2008     4 auto~ f        21    30 p     comp~
 5 audi  a4      2.80  1999     6 auto~ f        16    26 p     comp~
 6 audi  a4      2.80  1999     6 manu~ f        18    26 p     comp~
 7 audi  a4      3.10  2008     6 auto~ f        18    27 p     comp~
 8 audi  a4 qu~  1.80  1999     4 manu~ 4        18    26 p     comp~
 9 audi  a4 qu~  1.80  1999     4 auto~ 4        16    25 p     comp~
10 audi  a4 qu~  2.00  2008     4 manu~ 4        20    28 p     comp~
# ... with 224 more rows
```

ข้อมูล mpg ประกอบด้วย

- manufacturer บริษัทผู้ผลิตรถยนต์
- model รุ่นของรถยนต์ที่ผลิต
- displ ความจุกระบอกสูบ (Engine displacement) หน่วยเป็นลิตร
- year ปีที่ผลิต

- cyl จำนวนกระบอกสูบ
- trans ระบบเกียร์แบบ auto หรือ manual
- drv ระบบขับเคลื่อน: ขับเคลื่อนล้อหน้า (f), ขับเคลื่อนล้อหลัง (r), ขับเคลื่อน 4 ล้อ (4)
- cty และ hwy คือ จำนวนระยะทางต่อน้ำมันเชื้อเพลิง (miles per gallon) สำหรับเขตเมือง (City) และบนทางหลวง (Highway) ตามลำดับ
- fl ประเภทน้ำมันที่ใช้
- class ประเภทรถ: รถสองที่นั่ง (Two seater), SUV, compact เป็นต้น

16.1 การกรองแถวข้อมูลด้วยคำสั่ง filter()

ในแพ็กเกจ **dplyr** มีคำสั่ง **filter()** ในการเลือกข้อมูลแถวตามเงื่อนไขที่ต้องการ การเขียนคำสั่งด้วย **dplyr** นิยมใช้เครื่องหมาย **|>** ในการส่งต่อเอาต์พุต (Output) ของคำสั่งหน้าเครื่องหมาย **|>** ไปเป็นอินพุต (Input) ของคำสั่งหลังเครื่องหมายดังกล่าว เช่น

```
R> mpg |> filter(year >= 2000, class == "compact", cty >= 25 | hwy > 30)
# A tibble: 5 x 11
  manufacturer model   displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>        <chr>   <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 audi         a4        2    2008     4 manua~ f        20    31 p     comp~
2 toyota       camry ~   2.4  2008     4 manua~ f        21    31 r     comp~
3 toyota       camry ~   2.4  2008     4 auto(~ f        22    31 r     comp~
4 toyota       corolla   1.8  2008     4 manua~ f        28    37 r     comp~
5 toyota       corolla   1.8  2008     4 auto(~ f        26    35 r     comp~
```

พารามิเตอร์แต่ละตัวใน **filter()** คั่นด้วยเครื่องหมายจุลภาค (**,**) มีค่าเท่ากับตัวดำเนินการ **&** (และ) นอกจากนี้ยังเปรียบเทียบค่าต่าง ๆ โดยใช้ตัวดำเนินการเปรียบเทียบ ดังนี้

ตารางที่ 16-1 ตัวดำเนินการเปรียบเทียบ (Comparison Operators)

ตัวดำเนินการ	คำอธิบาย
==	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าเท่ากับ
<=	น้อยกว่าเท่ากับ
%in%	เป็นหนึ่งในเวกเตอร์ทางขวามือ

การเปรียบเทียบเงื่อนไขที่ซับซ้อน (มากกว่า 1 เงื่อนไข) จะใช้ตัวดำเนินการตรรกะช่วย ดังนี้

ตารางที่ 16-2 ตัวดำเนินการตรรกะ (Logical Operators)

ตัวดำเนินการ	คำอธิบาย	ผลลัพธ์
&	และ (AND)	TRUE & TRUE ได้ TRUE TRUE & FALSE ได้ FALSE FALSE & TRUE ได้ FALSE FALSE & FALSE ได้ FALSE
|	หรือ (OR)	TRUE | TRUE ได้ TRUE TRUE | FALSE ได้ TRUE FALSE | TRUE ได้ TRUE FALSE | FALSE ได้ FALSE
!	ไม่ (NOT)	!TRUE ได้ FALSE !FALSE ได้ TRUE

16.2 การเลือกแถวข้อมูลด้วยคำสั่ง slice()

ในแพ็กเกจ **dplyr** มีคำสั่ง **slice()** ในการเลือกข้อมูลแถวที่ต้องการ เช่น ต้องการข้อมูลแถวที่ 3 ถึง 6

```
R> mpg |> slice(3:6)
# A tibble: 4 x 11
  manuf~ model displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>  <chr> <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 audi   a4     2.00  2008     4 manua~ f        20    31 p     comp~
2 audi   a4     2.00  2008     4 auto(~ f        21    30 p     comp~
3 audi   a4     2.80  1999     6 auto(~ f        16    26 p     comp~
4 audi   a4     2.80  1999     6 manua~ f        18    26 p     comp~
```

ต้องการข้อมูลแถวที่ 10, 20 และ 90

```
R> mpg |> slice(c(10, 20, 90))
# A tibble: 3 x 11
  manuf~ model  displ  year   cyl trans drv     cty   hwy fl    class
  <chr>  <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
1 audi   a4 qu~  2.00  2008     4 manu~ 4        20    28 p     comp~
2 chevr~ c1500~  5.30  2008     8 auto~ r        11    15 e     suv
3 ford   f150 ~  5.40  2008     8 auto~ 4        13    17 r     pick~
```

16.3 การเลือกแถวข้อมูลที่ไม่ซ้ำซ้อนด้วยคำสั่ง distinct()

คำสั่ง **distinct()** จะตัดแถวข้อมูลที่ซ้ำซ้อนออกไปและแสดงเฉพาะข้อมูลที่ไม่ซ้ำกันตามตัวแปรที่กำหนด เช่น ต้องการแสดงเฉพาะผู้ผลิต (Manufacturer)

```
R> mpg |> distinct(manufacturer)
```

```
# A tibble: 15 x 1
   manufacturer
   <chr>
 1 audi
 2 chevrolet
 3 dodge
 ...
14 toyota
15 volkswagen
```

ต้องการแสดงเฉพาะผู้ผลิต (Manufacturer) และรุ่น (Model)

```
R> mpg |> distinct(manufacturer, model)
# A tibble: 38 x 2
   manufacturer model
   <chr>        <chr>
 1 audi         a4
 2 audi         a4 quattro
 3 audi         a6 quattro
 4 chevrolet    c1500 suburban 2wd
 5 chevrolet    corvette
 ...
 9 dodge        dakota pickup 4wd
10 dodge        durango 4wd
# ... with 28 more rows
```

16.4 การสุ่มข้อมูลด้วยคำสั่ง slice_sample()

คำสั่ง `slice_sample()` จะสุ่มข้อมูลตามจำนวนที่ต้องการ โดยกำหนดพารามิเตอร์จำนวนแถวที่ต้องการด้วยพารามิเตอร์ `n` ถ้าต้องการสุ่มแบบใส่คืน (With replacement) ให้กำหนดพารามิเตอร์ `replace = TRUE` การสุ่มแต่ละครั้งจะให้ผลลัพธ์แตกต่างกันออกไป (ใช้คำสั่ง `set.seed()` เพื่อกำหนดให้การสุ่มแต่ละครั้งให้ผลลัพธ์เหมือนหรือแตกต่างกันได้)

```
R> mpg |> slice_sample(n = 5)
# A tibble: 5 x 11
  manufacturer model    displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>        <chr>    <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 nissan       altima     2.4  1999     4 auto(~ f        19    27 r     comp~
2 chevrolet    malibu     2.4  2008     4 auto(~ f        22    30 r     mids~
3 dodge        ram 150~   4.7  2008     8 auto(~ 4         9    12 e     pick~
4 volkswagen   new bee~   1.9  1999     4 manua~ f        35    44 d     subc~
5 hyundai      tiburon    2    1999     4 auto(~ f        19    26 r     subc~
```

การสุ่มจำนวนแถวตามสัดส่วน (Propotion) หรือความน่าจะเป็น (Probability) ที่ต้องการ ระบุด้วยพารามิเตอร์ `prob` ดังนี้

```
R> mpg |> slice_sample(prop = 0.1)
# A tibble: 23 x 11
   manufacturer model   displ  year   cyl trans  drv     cty   hwy fl    class
   <chr>        <chr>   <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
 1 dodge        ram 15~   5.7  2008     8 auto(~ 4        13    17 r     pick~
 2 dodge        ram 15~   4.7  2008     8 auto(~ 4         9    12 e     pick~
```

```
 3 chevrolet    malibu    2.4  1999     4 auto(~ f        19    27 r     mids~
 4 lincoln      naviga~   5.4  1999     8 auto(~ r        11    17 r     suv
 5 nissan       maxima    3    1999     6 auto(~ f        18    26 r     mids~
 6 dodge        durang~   4.7  2008     8 auto(~ 4         9    12 e     suv
 7 subaru       imprez~   2.5  2008     4 manua~ 4        20    27 r     comp~
 8 subaru       imprez~   2.5  2008     4 manua~ 4        19    25 p     comp~
 9 subaru       forest~   2.5  2008     4 manua~ 4        19    25 p     suv
10 ford         mustang   4.6  1999     8 auto(~ r        15    21 r     subc~
# ... with 13 more rows ...
```

16.5 การเลือกแถวข้อมูลด้วยคำสั่ง slice_min() และ slice_max()

คำสั่ง **slice_min()** และ **slice_max()** จะเลือกแถวข้อมูลที่มีค่าต่ำสุดและค่าสูงสุด **n** แถวตามลำดับ โดยเรียงตามตัวแปรที่กำหนดด้วยพารามิเตอร์ **order_by** เช่น ข้อมูลที่มีจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) สำหรับเขตเมือง (City) ต่ำสุด 5 อันดับแรก

```
R> mpg |> slice_min(n = 5, order_by = cty)
# A tibble: 5 x 11
  manufacturer model    displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>        <chr>    <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 dodge        dakota ~   4.7  2008     8 auto(~ 4         9    12 e     pick~
2 dodge        durango~   4.7  2008     8 auto(~ 4         9    12 e     suv
3 dodge        ram 150~   4.7  2008     8 auto(~ 4         9    12 e     pick~
4 dodge        ram 150~   4.7  2008     8 manua~ 4         9    12 e     pick~
5 jeep         grand c~   4.7  2008     8 auto(~ 4         9    12 e     suv
```

ข้อมูลที่มีจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) สำหรับเขตเมือง (City) สูงสุด 5 อันดับแรก

```
R> mpg |> slice_max(n = 5, order_by = cty)
# A tibble: 5 x 11
  manufacturer model  displ  year   cyl trans  drv     cty   hwy fl    class
  <chr>        <chr>  <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
1 volkswagen   new b~   1.9  1999     4 manua~ f        35    44 d     subco~
2 volkswagen   jetta    1.9  1999     4 manua~ f        33    44 d     compa~
3 volkswagen   new b~   1.9  1999     4 auto(~ f        29    41 d     subco~
4 honda        civic    1.6  1999     4 manua~ f        28    33 r     subco~
5 toyota       corol~   1.8  2008     4 manua~ f        28    37 r     compa~
```

16.6 การสร้างตัวแปร/คอลัมน์ใหม่ด้วยคำสั่ง mutate() และ transmute()

คำสั่ง **mutate()** เป็นการสร้างคอลัมน์ใหม่ขึ้นมา โดยการใช้ข้อมูลจากคอลัมน์เดิม เช่น ต้องการหาค่าเฉลี่ยของคะแนนวิชาสถิติ (Statistics) และวิชาคณิตศาสตร์ (Mathematics)

```
R> tbl8 <- tibble(student = c("Anne", "Jim", "Joe", "Mary", "Mike"),
                  stat = c(90, 83, 75, 88, 70),
                  math = c(85, 93, 79, 82, 72))
R> tbl8 |> mutate(average = (stat+math)/2)
# A tibble: 5 x 4
  student  stat  math average
  <chr>   <dbl> <dbl>   <dbl>
```

```
1 Anne       90    85    87.5
2 Jim        83    93    88
3 Joe        75    79    77
4 Mary       88    82    85
5 Mike       70    72    71
```

ส่วนคำสั่ง `transmute()` เป็นการสร้างคอลัมน์ (Column) ใหม่ขึ้นมา โดยลบคอลัมน์เดิมทิ้งไป

```
R> tbl8 |> transmute(average = (stat+math)/2)
# A tibble: 5 x 1
  average
    <dbl>
1    87.5
2    88
3    77
4    85
5    71
```

16.7 การเลือกตัวแปร/คอลัมน์ที่ต้องการด้วยคำสั่ง select()

คำสั่ง `select()` เป็นการเลือกตัวแปร/คอลัมน์ โดยการกำหนดเงื่อนไขที่ต้องการ ก่อนอื่นสร้าง tibble ที่จะใช้แสดงการใช้งานคำสั่งต่อไป ดังนี้

```
R> tbl <- tibble(student = c("Anne", "Jim", "Joe", "Mary", "Mike"),
                 science1 = c(90, 83, 75, 88, 70),
                 science2 = c(95, 79, 72, 83, 77),
                 science3 = c(92, 75, 81, 77, 72),
                 math1 = c(85, 93, 79, 82, 72),
                 math2 = c(79, 88, 72, 90, 81),
                 math3 = c(74, 83, 77, 92, 85),
                 literature1 = c(76, 88, 75, 80, 83),
                 literature2 = c(82, 78, 84, 85, 79),
                 literature3 = c(77, 84, 72, 75, 73))
R> tbl9 <- tbl |>
     mutate(science_total = science1 + science2 + science3,
            math_total = math1 + math2 + math3,
            literature_total = literature1 + literature2 + literature3)
```

คำสั่ง `select()` เป็นการเลือกตัวแปรตามเงื่อนไขที่กำหนด เช่น ต้องการเลือกบางคอลัมน์

```
R> tbl9 |> select(student, math_total, science_total)
# A tibble: 5 x 3
  student math_total science_total
  <chr>        <dbl>         <dbl>
1 Anne           238           277
2 Jim            264           237
3 Joe            228           228
4 Mary           264           248
5 Mike           238           219
```

ต้องการเลือกคอลัมน์ที่ขึ้นต้นด้วย “lit” ใช้ฟังก์ชัน `starts_with()`

```
R> tbl9 |> select(starts_with("lit"))
# A tibble: 5 x 4
```

```
  literature1 literature2 literature3 literature_total
        <dbl>       <dbl>       <dbl>            <dbl>
1          76          82          77              235
2          88          78          84              250
3          75          84          72              231
4          80          85          75              240
5          83          79          73              235
```

ต้องการเลือกคอลัมน์ที่ลงท้ายด้วย “total” ใช้ฟังก์ชัน `ends_with()`

```
R> tbl9 |> select(ends_with("total"))
# A tibble: 5 x 3
  science_total math_total literature_total
          <dbl>      <dbl>            <dbl>
1           277        238              235
2           237        264              250
3           228        228              231
4           248        264              240
5           219        238              235
```

ต้องการเลือกคอลัมน์ที่อย่างน้อยประกอบไปด้วยตัวอักขระ “_” ใช้ฟังก์ชัน `contains()`

```
R> tbl9 |> select(contains("_"))
# A tibble: 5 x 3
  science_total math_total literature_total
          <dbl>      <dbl>            <dbl>
1           277        238              235
2           237        264              250
3           228        228              231
4           248        264              240
5           219        238              235
```

ถ้าต้องการเลือกคอลัมน์ที่ขึ้นต้นด้วย “science” และตามด้วยตัวเลข 2 ถึง 3 ใช้ฟังก์ชัน `num_range()`

```
R> tbl9 |> select(num_range("science", 2:3))
# A tibble: 5 x 2
  science2 science3
     <dbl>    <dbl>
1       95       92
2       79       75
3       72       81
4       83       77
5       77       72
```

ถ้าไม่ต้องการเลือกบางคอลัมน์ ใช้เครื่องหมายลบหน้าคอลัมน์ที่ไม่ต้องการ เช่น ไม่ต้องการแสดงคอลัมน์ science_total, math_total, literature1, literature2 และ literature3

```
R> tbl9 |> select(-science_total, -math_total,
                  -literature1, -literature2, -literature3)
# equivalent to tbl9 |> select(-c(science_total, math_total,
#                                  literature1, literature2, literature3))
# A tibble: 5 x 8
  student science1 science2 science3 math1 math2 math3 literature_total
  <chr>      <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>            <dbl>
```

```
1 Anne          90       95       92       85       79       74              235
2 Jim           83       79       75       93       88       83              250
3 Joe           75       72       81       79       72       77              231
4 Mary          88       83       77       82       90       92              240
5 Mike          70       77       72       72       81       85              235
```

การเรียงลำดับใหม่ (Reorder column) โดยเริ่มจากบางคอลัมน์ที่ต้องการเป็นลำดับแรก และต้องการแสดงคอลัมน์ที่เหลือทุกคอลัมน์ใช้ฟังก์ชัน `everything()` ดังนี้

```
> tbl9 |> select(math1, science1, student, everything())
# A tibble: 5 x 13
  math1 science1 student science2 science3 math2 math3 literature1 literature2
  <dbl>    <dbl> <chr>      <dbl>    <dbl> <dbl> <dbl>       <dbl>       <dbl>
1    85       90 Anne          95       92    79    74          76          82
2    93       83 Jim           79       75    88    83          88          78
3    79       75 Joe           72       81    72    77          75          84
4    82       88 Mary          83       77    90    92          80          85
5    72       70 Mike          77       72    81    85          83          79
# ... with 4 more variables: literature3 <dbl>, science_total <dbl>,
#   math_total <dbl>, literature_total <dbl>
```

ถ้าต้องการเลือกคอลัมน์ด้วยเงื่อนไขที่ซับซ้อน ใช้ฟังก์ชัน `matches()` ดูรายละเอียดเพิ่มเติมใน regular expression (พิมพ์ ?regex ที่ **R** console) และหัวข้อ 18.6 เช่น ต้องการเลือกคอลัมน์ที่ขั้นต้นด้วย “sci” หรือ“math”

```
R> tbl9 |> select(matches("sci|math"))
# A tibble: 5 x 8
  science1 science2 science3 math1 math2 math3 science_total math_total
     <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>         <dbl>      <dbl>
1       90       95       92    85    79    74           277        238
2       83       79       75    93    88    83           237        264
3       75       72       81    79    72    77           228        228
4       88       83       77    82    90    92           248        264
5       70       77       72    72    81    85           219        238
```

16.8 การสรุปข้อมูลแยกตามกลุ่มด้วยคำสั่ง group_by() และ summarize()

คำสั่ง `group_by()` เป็นการจัดข้อมูลแยกตามกลุ่ม ส่วนคำสั่ง `summarize()` เป็นการรายงานสรุปข้อมูล มักใช้คู่กับคำสั่ง `group_by()` เช่น ต้องการรายงานสรุปค่าเฉลี่ยของจำนวนระยะทางต่อน้ำมันเชื้อเพลิงบนทางหลวง (hwy) แยกตามระบบขับเคลื่อน (drv): ขับเคลื่อน 4 ล้อ (4), ขับเคลื่อนล้อหน้า (f), และขับเคลื่อนล้อหลัง (r)

```
R> mpg$drv <- factor(mpg$drv, labels = c("4-Wheel drive", "Front-wheel drive",
                                         "Rear-wheel drive"))
R> mpg |>
     group_by(drv)
# A tibble: 234 × 11
# Groups:   drv [3]
   manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr> <fct> <int> <int> <chr> <chr>
```

```
 1 audi         a4           1.8  1999     4 auto… Fron…    18    29 p     comp…
 2 audi         a4           1.8  1999     4 manu… Fron…    21    29 p     comp…
 3 audi         a4           2    2008     4 manu… Fron…    20    31 p     comp…
 4 audi         a4           2    2008     4 auto… Fron…    21    30 p     comp…
 5 audi         a4           2.8  1999     6 auto… Fron…    16    26 p     comp…
 6 audi         a4           2.8  1999     6 manu… Fron…    18    26 p     comp…
 7 audi         a4           3.1  2008     6 auto… Fron…    18    27 p     comp…
 8 audi         a4 quattro   1.8  1999     4 manu… 4-Wh…    18    26 p     comp…
 9 audi         a4 quattro   1.8  1999     4 auto… 4-Wh…    16    25 p     comp…
10 audi         a4 quattro   2    2008     4 manu… 4-Wh…    20    28 p     comp…
# i 224 more rows
# i Use `print(n = ...)` to see more rows

R> mpg |>
     group_by(drv) |>
     summarize(mean(hwy))
# A tibble: 3 x 2
  drv             `mean(hwy)`
  <fct>                 <dbl>
1 4-Wheel drive          19.2
2 Front-wheel drive      28.2
3 Rear-wheel drive       21
```

คำสั่ง summarize() สามารถรายงานผลลัพธ์ได้หลายลักษณะ เช่น ค่าเฉลี่ยหรือค่าส่วนเบี่ยงเบนมาตรฐานสำหรับตัวแปรอื่น พร้อมกับตั้งชื่อตัวแปรที่ต้องการได้

```
R> mpg |>
     group_by(drv) |>
     summarize(mean_hwy = mean(hwy), mean_cty = mean(cty), sd_hwy = sd(hwy),
               sd_cty = sd(cty))
# A tibble: 3 x 5
  drv             mean_hwy mean_cty sd_hwy sd_cty
  <fct>              <dbl>    <dbl>  <dbl>  <dbl>
1 4-Wheel drive       19.2     14.3   4.08   2.87
2 Front-wheel drive   28.2     20.0   4.21   3.63
3 Rear-wheel drive    21       14.1   3.66   2.22
```

คำสั่ง group_by() ยังสามารถจัดข้อมูลแยกตามกลุ่มได้มากกว่า 1 กลุ่ม เช่น สรุปผลแยกตามระบบขับเคลื่อน (drv) และจำนวนกระบอกสูบ (cyl)

```
R> mpg |>
     group_by(drv, cyl) |>
     summarize(mean_hwy = mean(hwy), mean_cty = mean(cty), sd_hwy = sd(hwy),
               sd_cty = sd(cty))
# A tibble: 9 x 6
# Groups:   drv [3]
  drv               cyl mean_hwy mean_cty sd_hwy sd_cty
  <fct>           <int>    <dbl>    <dbl>  <dbl>  <dbl>
1 4-Wheel drive       4     24.6     18.3  2.54   1.75
2 4-Wheel drive       6     19.5     14.8  2.90   1.16
3 4-Wheel drive       8     16.4     12.1  2.22   1.59
4 Front-wheel drive   4     30.5     22.1  4.03   3.46
5 Front-wheel drive   5     28.8     20.5  0.5    0.577
6 Front-wheel drive   6     25.1     17.2  2.20   1.47
```

```
7 Front-wheel drive     8     25       16   NA     NA
8 Rear-wheel drive      6     25.2     17.2  0.957  0.957
9 Rear-wheel drive      8     20.2     13.5  3.41   1.83
```

นอกจากนี้ยังมีคำสั่ง **summarize_all()** ในการรายงานผลสำหรับทุกตัวแปร เช่น ต้องการรายงานค่าเฉลี่ย และค่าส่วนเบี่ยงเบนมาตรฐานสำหรับทุกตัวแปร

```
R> mpg |>
     group_by(drv) |>
     summarize_all(funs(mean, sd))
# A tibble: 3 x 21
  drv     manufacturer_me~ model_mean displ_mean year_mean cyl_mean trans_mean
  <fct>              <dbl>      <dbl>      <dbl>     <dbl>    <dbl>      <dbl>
1 4-Whee~               NA         NA       4.00     2004.     6.49         NA
2 Front-~               NA         NA       2.56     2003.     4.89         NA
3 Rear-w~               NA         NA       5.18     2004.     7.68         NA
# ... with 14 more variables: cty_mean <dbl>, hwy_mean <dbl>, fl_mean <dbl>,
#   class_mean <dbl>, manufacturer_sd <dbl>, model_sd <dbl>, displ_sd <dbl>,
#   year_sd <dbl>, cyl_sd <dbl>, trans_sd <dbl>, cty_sd <dbl>, hwy_sd <dbl>,
#   fl_sd <dbl>, class_sd <dbl>
There were 30 warnings (use warnings() to see them)
```

คำสั่ง **summarize_at()** ในการรายงานผลตัวแปรและฟังก์ชันที่ต้องการ เช่น ต้องการรายงานค่าเฉลี่ย ค่าส่วนเบี่ยงเบนมาตรฐาน และค่าสูงสุด สำหรับตัวแปร hwy และ cty

```
R> mpg |>
     group_by(drv) |>
     summarize_at(vars(hwy, cty), funs(mean, sd, max))
# A tibble: 3 x 7
  drv               hwy_mean cty_mean hwy_sd cty_sd hwy_max cty_max
  <fctr>               <dbl>    <dbl>  <dbl>  <dbl>   <dbl>   <dbl>
1 4-Wheel drive         19.2     14.3   4.08   2.87    28.0    21.0
2 Front-wheel drive     28.2     20.0   4.21   3.63    44.0    35.0
3 Rear-wheel drive      21.0     14.1   3.66   2.22    26.0    18.0
```

16.9 การนับจำนวนข้อมูลด้วยคำสั่ง count()

คำสั่ง **count()** จะนับจำนวนข้อมูลแยกตามตัวแปรที่กำหนด ตัวอย่างเช่น ต้องการนับจำนวนข้อมูลตามผู้ผลิต (Manufacturer) และรุ่นของรถยนต์ที่ผลิต (Model)

```
R> mpg |>
     count(manufacturer, model)
# A tibble: 38 x 3
   manufacturer model                  n
   <chr>        <chr>              <int>
 1 audi         a4                     7
 2 audi         a4 quattro             8
 3 audi         a6 quattro             3
 4 chevrolet    c1500 suburban 2wd     5
 5 chevrolet    corvette               5
 6 chevrolet    k1500 tahoe 4wd        4
 7 chevrolet    malibu                 5
```

```
 8 dodge        caravan 2wd           11
 9 dodge        dakota pickup 4wd      9
10 dodge        durango 4wd            7
# i 28 more rows
# i Use `print(n = ...)` to see more rows
```

คำสั่งข้างต้นเทียบเท่ากับคำสั่ง `group_by()` และคำสั่ง `summarize()` ดังนี้

```
R> mpg |>
     group_by(manufacturer, model) |>
     summarize(n = n(), .groups = "drop")
# A tibble: 38 × 3
   manufacturer model                 n
   <chr>        <chr>             <int>
 1 audi         a4                    7
 2 audi         a4 quattro            8
 3 audi         a6 quattro            3
 4 chevrolet    c1500 suburban 2wd    5
 5 chevrolet    corvette              5
 6 chevrolet    k1500 tahoe 4wd       4
 7 chevrolet    malibu                5
 8 dodge        caravan 2wd          11
 9 dodge        dakota pickup 4wd     9
10 dodge        durango 4wd           7
# i 28 more rows
# i Use `print(n = ...)` to see more rows
```

16.10 การเรียงลำดับข้อมูลในแถวด้วยคำสั่ง arrange()

คำสั่ง `arrange()` เป็นการเรียงลำดับข้อมูลจากน้อยไปมาก (หรือมากไปน้อย) ด้วยตัวแปรที่กำหนดตั้งแต่ 1 ตัวแปรขึ้นไป เช่น ต้องการเรียงข้อมูลตามความจุกระบอกสูบ (displ) แล้วค่อยเรียงตามจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (miles per gallon) สำหรับเขตเมือง (cty) ตามลำดับจากน้อยไปมาก

```
R> mpg |>
     arrange(displ, cty)
# A tibble: 234 x 11
   manuf~ model  displ  year   cyl trans drv     cty   hwy fl    class
   <chr>  <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
 1 honda  civic   1.60  1999     4 manu~ f        23    29 p     subc~
 2 honda  civic   1.60  1999     4 auto~ f        24    32 r     subc~
 3 honda  civic   1.60  1999     4 auto~ f        24    32 r     subc~
 4 honda  civic   1.60  1999     4 manu~ f        25    32 r     subc~
 5 honda  civic   1.60  1999     4 manu~ f        28    33 r     subc~
 6 audi   a4 qu~  1.80  1999     4 auto~ 4        16    25 p     comp~
 7 audi   a4      1.80  1999     4 auto~ f        18    29 p     comp~
 8 audi   a4 qu~  1.80  1999     4 manu~ 4        18    26 p     comp~
 9 volks~ passat  1.80  1999     4 auto~ f        18    29 p     mids~
10 audi   a4      1.80  1999     4 manu~ f        21    29 p     comp~
# i 224 more rows
# i Use `print(n = ...)` to see more rows
```

ถ้าต้องการเรียงข้อมูลตามความจุกระบอกสูบ (displ) ตามลำดับจากน้อยไปมาก แล้วค่อยเรียงตามจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (miles per gallon) สำหรับเขตเมือง (cty) ตามลำดับจากมากไปน้อย ต้องใช้ฟังก์ชัน **desc()** ช่วย ดังนี้

```
R> mpg |>
     arrange(displ, desc(cty))
# A tibble: 234 x 11
   manuf~ model  displ  year   cyl trans drv     cty   hwy fl    class
   <chr>  <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
 1 honda  civic   1.60  1999     4 manu~ f        28    33 r     subc~
 2 honda  civic   1.60  1999     4 manu~ f        25    32 r     subc~
 3 honda  civic   1.60  1999     4 auto~ f        24    32 r     subc~
 4 honda  civic   1.60  1999     4 auto~ f        24    32 r     subc~
 5 honda  civic   1.60  1999     4 manu~ f        23    29 p     subc~
 6 toyota corol~  1.80  2008     4 manu~ f        28    37 r     comp~
 7 honda  civic   1.80  2008     4 manu~ f        26    34 r     subc~
 8 toyota corol~  1.80  1999     4 manu~ f        26    35 r     comp~
 9 toyota corol~  1.80  2008     4 auto~ f        26    35 r     comp~
10 honda  civic   1.80  2008     4 auto~ f        25    36 r     subc~
# i 224 more rows
# i Use `print(n = ...)` to see more rows
```

16.11 การประยุกต์ใช้คำสั่งต่าง ๆ ในการหาผลรวมสะสม (Cummulative Summation)

ในหัวข้อนี้จะประยุกต์ใช้คำสั่งต่าง ๆ ในแพ็กเกจ **dplyr** และ **tidyr** ในการหาผลรวมสะสม (Cummulative summation) ในขั้นแรกจะเป็นการสร้าง tibble สำหรับใช้ในการวิเคราะห์ต่อไป โดยสร้างข้อมูลยอดขายรายเดือนจำนวน 2 ปี ดังนี้

```
R> tbl10 <- tibble(month = factor(rep(month.abb, 2), levels = month.abb),
                   year = as.factor(rep(c(2017, 2018), each = 12)),
                   sales = c(seq(from = 10, by = 5, length.out = 12),
                             seq(from = 20, by = 7, length.out = 12)))
```

ค่าคงที่ **month.abb** เป็นเวกเตอร์แสดงรายชื่อเดือนแบบย่อทั้งหมด 12 เดือน

```
R> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

คำสั่ง **factor()** เป็นการสร้างเวกเตอร์ที่เป็นแฟกเตอร์ ค่าคงที่ **month.abb** ใช้สร้างตัวแปรเดือนทั้ง 12 เดือน รวมทั้งกำหนดพารามิเตอร์ **levels = month.abb** เพื่อให้เดือนเรียงตามลำดับที่กำหนด (มิฉะนั้นเดือนจะเรียงตามตัวอักษรแทน) ผลลัพธ์จะได้ tibble ยอดขาย 24 เดือน ดังนี้

```
R> tbl10
# A tibble: 24 x 3
   month year  sales
   <fct> <fct> <dbl>
 1 Jan   2017     10
 2 Feb   2017     15
 3 Mar   2017     20
 4 Apr   2017     25
```

```
 5 May   2017     30
 6 Jun   2017     35
 7 Jul   2017     40
 8 Aug   2017     45
 9 Sep   2017     50
10 Oct   2017     55
# ... with 14 more rows
```

เราสามารถหาผลรวมสะสม (Cummulative summation) ได้โดยการเรียงข้อมูลตามปีและเดือนด้วยคำสั่ง **arrange()** หลังจากนั้นใช้คำสั่ง **mutate()** และ **cumsum()** เพื่อหาผลรวมสะสมโดยเก็บค่าไว้ในตัวแปรชื่อ cumTotal ดังนี้

```
R> tbl10 |>
     arrange(year, month) |>
     mutate(cum_total = cumsum(sales))
# A tibble: 24 x 4
   month year  sales cum_total
   <fct> <fct> <dbl>     <dbl>
 1 Jan   2017     10        10
 2 Feb   2017     15        25
 3 Mar   2017     20        45
 4 Apr   2017     25        70
 5 May   2017     30       100
 6 Jun   2017     35       135
 7 Jul   2017     40       175
 8 Aug   2017     45       220
 9 Sep   2017     50       270
10 Oct   2017     55       325
# ... with 14 more rows
```

ค่าผลรวมสะสมจะถูกคำนวณต่อเนื่องกันไปทุกเดือนจนครบ 24 เดือน ถ้าต้องการให้คำนวณผลรวมสะสมแยกรายปีจะต้องใช้คำสั่ง **group_by()** เพื่อจัดกลุ่มตามปี ดังนี้

```
R> tbl10 |>
     arrange(year, month) |>
     group_by(year) |>
     mutate(cum_total = cumsum(sales))
# A tibble: 24 × 4
# Groups:   year [2]
   month year sales cum_total
   <fct> <fct> <dbl>    <dbl>
1   Jan 2017    10        10
2   Feb 2017    15        25
3   Mar 2017    20        45
...
11  Nov 2017    60       385
12  Dec 2017    65       450
13  Jan 2018    20        20
14  Feb 2018    27        47
15  Mar 2018    34        81
...
23  Nov 2018    90       605
24  Dec 2018    97       702
```

จากรายงานสรุปข้างต้นจะเห็นว่าค่าผลรวมสะสมจะนับตั้งแต่เดือนที่ 1 ไปจนเดือนที่ 12 ของปี 2017 แล้วจากนั้นจึงเริ่มคำนวณผลรวมสะสมในปี 2018 ใหม่

อีกตัวอย่างหนึ่งเป็นยอดขายรายเดือนของร้านค้า 3 สาขาในระยะเวลา 2 ปีตั้งแต่ปี 2016 สร้างเป็น tibble ดังนี้

```
R> set.seed(1234)
R> tbl11 <- tibble(date = seq(from = as.Date("2016-1-1"),
                              by = "month", length.out=24),
                   branch1 = as.integer(runif(24, min=50, max=80)),
                   branch2 = as.integer(runif(24, min=30, max=60)),
                   branch3 = as.integer(runif(24, min=20, max=50)))
```

คำสั่งแรกเป็นการกำหนดค่า seed เพื่อให้การทำซ้ำได้ตัวเลขสุ่มเหมือนกันทุกครั้ง คำสั่งต่อมาเป็นการสร้าง tibble ประกอบด้วยวันที่ 1 ของทุก ๆ เดือน เริ่มตั้งแต่เดือนมกราคม ปี 2016 และข้อมูลยอดขายทั้ง 3 สาขาโดยใช้การสุ่มแบบ Uniform

เพื่อให้ง่ายในการวิเคราะห์จึงทำการแปลงข้อมูลรูปแบบตามแนวกว้าง (Wide format) ให้เป็นข้อมูลรูปแบบตามแนวยาว (Long format) โดยใช้คำสั่ง `gather()`

```
R> tbl12 <- tbl11 |>
     gather(key = branch, value = sales, branch1:branch3)
```

สรุปรายงานยอดขายสะสมรายสาขา ด้วยการเรียงข้อมูลสาขาและวันที่จากน้อยไปมากด้วยคำสั่ง `arrange()` แล้วใช้คำสั่ง `group_by()` เพื่อจัดกลุ่มตามสาขา หลังจากนั้นจึงใช้คำสั่ง `mutate()` และ `cumsum()` เพื่อหาผลรวมสะสมโดยเก็บค่าไว้ในตัวแปรชื่อ cumTotal

```
R> tbl12 |>
     arrange(branch, date) |>
     group_by(branch) |>
     mutate(cum_total = cumsum(sales))
# A tibble: 72 × 4
# Groups:   branch [3]
   date       branch  sales cum_total
   <date>     <chr>   <int>     <int>
 1 2016-01-01 branch1    53        53
 2 2016-02-01 branch1    68       121
 3 2016-03-01 branch1    68       189
 4 2016-04-01 branch1    68       257
 5 2016-05-01 branch1    75       332
 6 2016-06-01 branch1    69       401
 7 2016-07-01 branch1    50       451
 8 2016-08-01 branch1    56       507
 9 2016-09-01 branch1    69       576
10 2016-10-01 branch1    65       641
# i 62 more rows
# i Use `print(n = ...)` to see more rows
```

สรุปรายงานยอดขายสะสมรายสาขาและปี ด้วยการเรียงข้อมูลสาขาและเดือนจากน้อยไปมากด้วยคำสั่ง **arrange()** แล้วใช้คำสั่ง **group_by()** เพื่อจัดกลุ่มตามสาขาและปี โดยมีการใช้ฟังก์ชัน **format()** แปลงข้อมูลประเภท **date** ให้เหลือแค่ข้อมูลปีที่เป็นประเภท **chr** โดยใช้พารามิเตอร์ **%Y** ซึ่งหมายถึงข้อมูลปีที่แสดงด้วยเลข 4 ตัว (ข้อมูลเพิ่มเติมในหัวข้อ 17.1.4) ในการดึงข้อมูลปีออกมาจากข้อมูลวันที่ หลังจากนั้นใช้คำสั่ง **mutate()** และ **cumsum()** เพื่อหาผลรวมสะสมโดยเก็บค่าไว้ในตัวแปรชื่อ cum_total เช่นเดียวกับชุดคำสั่งก่อนหน้านี้

```
R> tbl12 |>
     arrange(branch, date) |>
     group_by(branch, year = format(date, "%Y")) |>
     mutate(cum_total = cumsum(sales))
# A tibble: 72 × 5
# Groups:   branch, year [6]
   date       branch  sales year  cum_total
   <date>     <chr>   <int> <chr>     <int>
 1 2016-01-01 branch1    53 2016         53
 2 2016-02-01 branch1    68 2016        121
 3 2016-03-01 branch1    68 2016        189
 4 2016-04-01 branch1    68 2016        257
 5 2016-05-01 branch1    75 2016        332
 6 2016-06-01 branch1    69 2016        401
 7 2016-07-01 branch1    50 2016        451
 8 2016-08-01 branch1    56 2016        507
 9 2016-09-01 branch1    69 2016        576
10 2016-10-01 branch1    65 2016        641
# i 62 more rows
# i Use `print(n = ...)` to see more rows
```

16.12 การจัดการข้อมูลที่มีความสัมพันธ์กัน (Relational Data)

ฐานข้อมูลโดยทั่วไปมักจะเก็บเป็นตารางหลาย ๆ ตารางที่มีความสัมพันธ์กันเรียกว่า Relational database ในหัวข้อนี้จะแสดงตัวอย่างโดยใช้ข้อมูลชื่อ nycflights13 ซึ่งอยู่ในแพ็กเกจ **dbplyr** ข้อมูลดังกล่าวประกอบด้วยตาราง 5 ตารางที่มีความสัมพันธ์กัน ดังต่อไปนี้

- **airlines** เป็นข้อมูลสายการบินประกอบด้วยชื่อย่อ (carrier) และชื่อเต็ม (name) ดังนี้

```
R> library(nycflights13)
R> airlines
# A tibble: 16 x 2
   carrier name
   <chr>   <chr>
 1 9E      Endeavor Air Inc.
 2 AA      American Airlines Inc.
 3 AS      Alaska Airlines Inc.
 4 B6      JetBlue Airways
 5 DL      Delta Air Lines Inc.
```

- **airports** เป็นสนามบินประกอบด้วยรหัส (faa) ชื่อเต็ม (name) ละติจูด (lat) ลองจิจูด (lon) ความสูง (alt) Timezone (tz) Daylight saving time zone (dst) และ IANA time zone (tzone) ดังนี้

```
R> airports
# A tibble: 1,458 x 8
   faa   name                  lat    lon   alt    tz dst   tzone
   <chr> <chr>               <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
 1 04G   Lansdowne Airport    41.1  -80.6  1044    -5 A     America/New~
 2 06A   Moton Field Municip~ 32.5  -85.7   264    -6 A     America/Chi~
 3 06C   Schaumburg Regional  42.0  -88.1   801    -6 A     America/Chi~
 4 06N   Randall Airport      41.4  -74.4   523    -5 A     America/New~
 5 09J   Jekyll Island Airpo~ 31.1  -81.4    11    -5 A     America/New~
 6 0A9   Elizabethton Munici~ 36.4  -82.2  1593    -5 A     America/New~
 7 0G6   Williams County Air~ 41.5  -84.5   730    -5 A     America/New~
 8 0G7   Finger Lakes Region~ 42.9  -76.8   492    -5 A     America/New~
 9 0P2   Shoestring Aviation~ 39.8  -76.6  1000    -5 U     America/New~
10 0S9   Jefferson County In~ 48.1 -123.    108    -8 A     America/Los~
# ... with 1,448 more rows
```

- **planes** เป็นข้อมูลเครื่องบินประกอบด้วยรหัส (tailnum) ปี (year) ประเภท (type) ผู้ผลิต (manufacturer) แบบ (model) จำนวนเครื่องยนต์ (engines) จำนวนที่นั่ง (seats) ความเร็ว (speed) และเครื่องยนต์ (engine) ดังนี้

```
R> planes
# A tibble: 3,322 x 9
   tailnum  year type      manufacturer  model engines seats speed engine
   <chr>   <int> <chr>     <chr>         <chr>   <int> <int> <int> <chr>
 1 N10156   2004 Fixed wi~ EMBRAER       EMB-~       2    55    NA Turbo~
 2 N102UW   1998 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 3 N103US   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 4 N104UW   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 5 N10575   2002 Fixed wi~ EMBRAER       EMB-~       2    55    NA Turbo~
 6 N105UW   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 7 N107US   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 8 N108UW   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
 9 N109UW   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
10 N110UW   1999 Fixed wi~ AIRBUS INDUS~ A320~       2   182    NA Turbo~
# ... with 3,312 more rows
```

- **weather** เป็นข้อมูลสภาพอากาศของสนามบินประกอบด้วยสถานีวัด (origin) ปี (year) เดือน (month) วัน (day) ชั่วโมง (hour) อุณหภูมิ (temp) Dewpoint (dewp) ความชื้นสัมพัทธ์ (humid) ทิศทางลม (wind_dir) ความเร็วลม (wind_speed) เป็นต้น ดังนี้

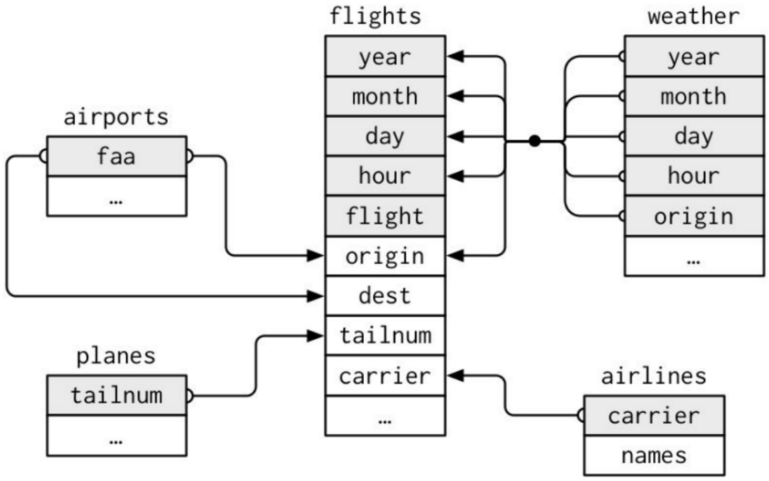
```
R> weather
# A tibble: 26,115 x 15
   origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
   <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
 1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
 2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
 3 EWR     2013     1     1     3  39.0  28.0  64.4      240      11.5
```

```
 4 EWR     2013     1     1     4  39.9  28.0  62.2      250      12.7
 5 EWR     2013     1     1     5  39.0  28.0  64.4      260      12.7
 6 EWR     2013     1     1     6  37.9  28.0  67.2      240      11.5
 7 EWR     2013     1     1     7  39.0  28.0  64.4      240      15.0
 8 EWR     2013     1     1     8  39.9  28.0  62.2      250      10.4
 9 EWR     2013     1     1     9  39.9  28.0  62.2      260      15.0
10 EWR     2013     1     1    10  41    28.0  59.6      260      13.8
# ... with 26,105 more rows, and 5 more variables: wind_gust <dbl>,
#   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>
```

- `flights` เป็นข้อมูลการบินที่ออกจากสนามบิน NYC ประกอบด้วยปี (year) เดือน (month) วัน (day) เวลาออก (dep_time) เวลาออกตามตาราง (sched_dep_time) เวลาออกล่าช้า (dep_delay) เวลาถึง (arr_time) เป็นต้น ดังนี้

```
R> flights
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

แสดงความสัมพันธ์ของตารางข้างต้นได้ดังรูปต่อไปนี้



รูปที่ 16-1 ความสัมพันธ์ของข้อมูลการบิน nycflights13

- ข้อมูล flights สัมพันธ์กับข้อมูล planes ด้วยตัวแปร talinum
- ข้อมูล flights สัมพันธ์กับข้อมูล airlines ด้วยตัวแปร carrier
- ข้อมูล flights สัมพันธ์กับข้อมูล airports ด้วยตัวแปร origin และ dest
- ข้อมูล flights สัมพันธ์กับข้อมูล weather ด้วยตัวแปร origin, year, month, day และ hour

การเชื่อมโยงความสัมพันธ์ระหว่างตารางต่าง ๆ ดังกล่าวจะต้องกำหนดคีย์ (Keys) โดยคีย์แบ่งเป็น 2 ประเภทหลัก ๆ ได้แก่

1. คีย์หลัก (Primary key) เป็นตัวแปรหรือกลุ่มตัวแปรที่มีข้อมูลไม่ซ้ำกันและมีค่าเสมอ (ไม่เป็น **NULL**) เป็นตัวแปรที่อยู่ในตารางนั้น เช่น planes$tailnum เป็นคีย์หลักเพราะเป็นตัวแปรที่มีข้อมูลไม่ซ้ำกัน (Unique) ในตาราง planes
2. คีย์นอก (Foreign key) เป็นคีย์ที่ใช้เชื่อมตารางที่เกี่ยวข้องเข้าด้วยกัน เช่น flights$tailnum เป็นคีย์นอก (Foreign key) เนื่องจากคีย์ดังกล่าวอยู่ในตาราง flights ซึ่งเชื่อมโยงข้อมูลแต่ละเที่ยวบิน (flight) กับตาราง plane

คีย์เป็นตัวแปรเดียว เช่น การนิยามเครื่องบินแต่ละลำใช้ตัวแปร talinum ในตาราง planes หรือคีย์อาจเป็นได้หลายตัวแปร เช่น การนิยามสภาพอากาศแต่ละสนามบินใช้ตัวแปร year, month, day, hour, และ origin ในตาราง weather

สามารถตรวจสอบว่าตัวแปรดังกล่าวเป็นคีย์หลักได้ด้วยการนับจำนวนข้อมูลในตัวแปรดังกล่าว ดังนี้

```
R> planes |>
     count(tailnum) |>
     filter(n > 1)
# A tibble: 0 x 2
# ... with 2 variables: tailnum <chr>, n <int>
```

16.12.1 การสร้างตัวแปรจากตารางอื่น (Mutating Joins)

Mutating joins เป็นการสร้างตัวแปรใหม่จากตัวแปรในตารางอื่น โดยการ Match คีย์เข้าด้วยกันและคัดลอกตัวแปรจากอีกตารางหนึ่งมายังตารางที่ต้องการ ดังนี้

เพื่อให้เห็นรายละเอียดที่ต้องการ จะทำการลดตัวแปรให้เหลือเฉพาะที่จำเป็นในการอธิบาย ดังนี้

```
R> flights2 <- flights |>
     select(year:day, hour, origin, dest, tailnum, carrier)

R> flights2
# A tibble: 336,776 x 8
    year month   day  hour origin dest  tailnum carrier
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
 1  2013     1     1     5 EWR    IAH   N14228  UA
 2  2013     1     1     5 LGA    IAH   N24211  UA
 3  2013     1     1     5 JFK    MIA   N619AA  AA
 4  2013     1     1     5 JFK    BQN   N804JB  B6
```

```
 5  2013     1     1     6 LGA    ATL    N668DN  DL
 6  2013     1     1     5 EWR    ORD    N39463  UA
 7  2013     1     1     6 EWR    FLL    N516JB  B6
 8  2013     1     1     6 LGA    IAD    N829AS  EV
 9  2013     1     1     6 JFK    MCO    N593JB  B6
10  2013     1     1     6 LGA    ORD    N3ALAA  AA
# ... with 336,766 more rows
```

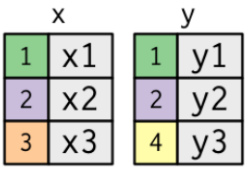
สมมติว่าต้องการเพิ่มชื่อสายการบินในข้อมูลแต่ละ flight ข้างต้น ทำได้โดยการรวมตารางด้วยคำสั่ง `left_join()` โดยใช้คีย์ carrier ดังนี้

```
R> flights2 |>
     select(-origin, -dest) |>
     left_join(airlines, by = "carrier")
# A tibble: 336,776 x 7
    year month   day  hour tailnum carrier name
   <int> <int> <int> <dbl> <chr>   <chr>   <chr>
 1  2013     1     1     5 N14228  UA      United Air Lines Inc.
 2  2013     1     1     5 N24211  UA      United Air Lines Inc.
 3  2013     1     1     5 N619AA  AA      American Airlines Inc.
 4  2013     1     1     5 N804JB  B6      JetBlue Airways
 5  2013     1     1     6 N668DN  DL      Delta Air Lines Inc.
 6  2013     1     1     5 N39463  UA      United Air Lines Inc.
 7  2013     1     1     6 N516JB  B6      JetBlue Airways
 8  2013     1     1     6 N829AS  EV      ExpressJet Airlines Inc.
 9  2013     1     1     6 N593JB  B6      JetBlue Airways
10  2013     1     1     6 N3ALAA  AA      American Airlines Inc.
# ... with 336,766 more rows
```

จะเห็นมีตัวแปร name เพิ่มเข้ามาใน tibble ชื่อ flights2

16.12.2 การเชื่อมตาราง (Join Tables)

การเชื่อมตาราง (Join Tables) เข้าด้วยกัน มีหลายลักษณะ เพื่อให้เข้าใจได้ง่ายจะใช้รูปภาพประกอบคำอธิบาย ดังนี้



รูปที่ 16-2 ตาราง x และ y

สร้างตาราง x ได้ดังนี้

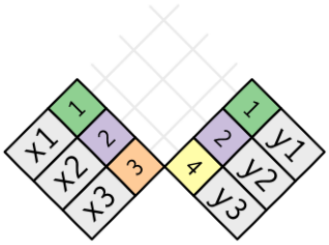
```
R> x <- tribble(
     ~key, ~val_x,
     1, "x1",
     2, "x2",
     3, "x3"
   )
```

และสร้างตาราง y ได้ดังนี้

```
R> y <- tribble(
     ~key, ~val_y,
     1, "y1",
     2, "y2",
     4, "y3"
   )
```

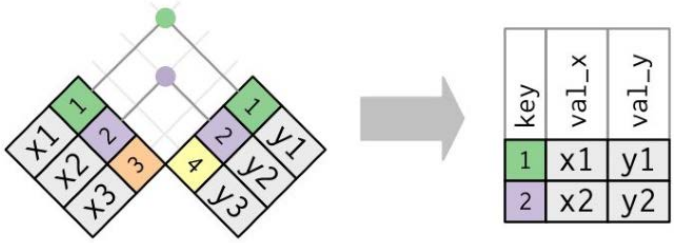
คอลัมน์ที่แสดงด้วยสีเป็นข้อมูลตัวแปร key ซึ่งจะใช้ในการ Match แถวในตาราง ส่วนคอลัมน์ที่แสดงด้วยสีเทาเป็นข้อมูลตัวแปร value

รูปด้านล่างแสดงความเป็นไปได้ในการ Match แต่ละแถวเข้าด้วยกัน ซึ่งเป็นไปได้ทั้งไม่ Match กันเลย หรือ Match 1, 2 ตัวหรือมากกว่า



รูปที่ 16-3 ตาราง x และ y

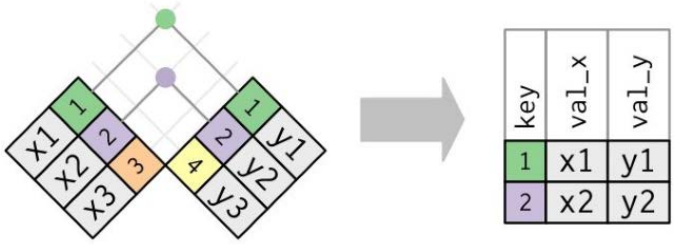
การ Match จะแสดงด้วยจุด (Dots) โดยจำนวนจุดจะเท่ากับจำนวนการ Match ที่ได้ ซึ่งจะได้เท่ากับจำนวนแถวของผลลัพธ์ (Outputs) ดังนี้



รูปที่ 16-4 แสดงการเชื่อมตารางและผลลัพธ์

16.12.3 Inner Joins

การเชื่อมตาราง (Join tables) แบบ inner join จะทำการ Match คีย์ที่เหมือนกันทั้ง 2 ตาราง ดังนี้



รูปที่ 16-5 Inner join

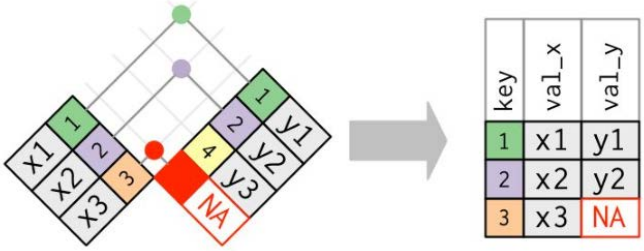
คำสั่งในการเชื่อมตาราง (Join tables) แบบ inner join ใช้คำสั่ง `inner_join()` แสดงได้ดังนี้

```
R> x |>
     inner_join(y, by = "key")
# A tibble: 2 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
```

16.12.4 Outer Joins

การเชื่อมตาราง (Join tables) แบบ Outer join จะทำการ Match คีย์ที่ปรากฏอย่างน้อย 1 ตาราง ประกอบด้วย

1. Left join เป็นการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏในตาราง x ดังนี้

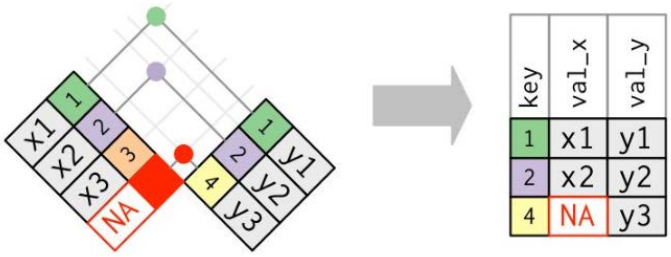


รูปที่ 16-6 Left joins

คำสั่งในการเชื่อมตาราง (Join tables) แบบ left join แสดงได้ดังนี้

```
R> x |>
     left_join(y, by = "key")
# A tibble: 3 × 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
```

2. Right join เป็นการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏในตาราง y ดังนี้

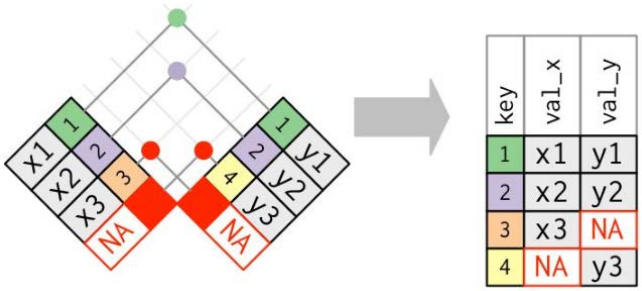


รูปที่ 16-7 Right joins

คำสั่งในการเชื่อมตาราง (Join tables) แบบ right join ใช้คำสั่ง `right_joint()` แสดงได้ดังนี้

```
R> x |>
     right_join(y, by = "key")
# A tibble: 3 × 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     4 NA    y3
```

3. Full join เป็นการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏทั้งตาราง x และ y ดังนี้

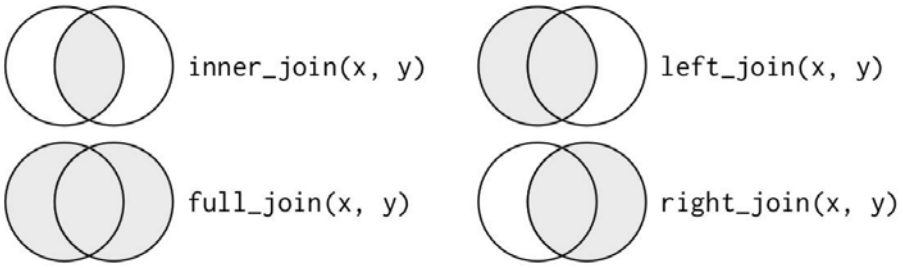


รูปที่ 16-8 Full joins

คำสั่งในการเชื่อมตาราง (Join tables) แบบ full join ใช้คำสั่ง `full_join()` แสดงได้ดังนี้

```
R> x |>
     full_join(y, by = "key")
# A tibble: 4 × 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     3 x3    NA
4     4 NA    y3
```

การเชื่อมตาราง (Join tables) แสดงอีกแบบได้ด้วย Venn diagram ดังนี้

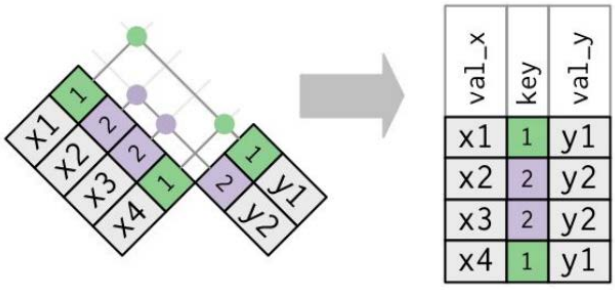


รูปที่ 16-9 Join Tables

16.12.5 คีย์ที่ซ้ำกัน (Duplicate Keys)

คีย์ที่กำหนดอาจมีข้อมูลที่ซ้ำกันได้ 2 แบบ ดังนี้

1. คีย์ที่มีข้อมูลซ้ำกัน 1 ตาราง ดังรูปต่อไปนี้



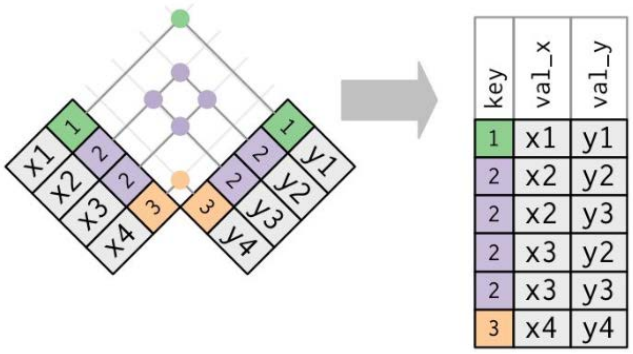
รูปที่ 16-10 Duplicate keys in one table

สร้างเป็นตาราง x และตาราง y และคำสั่ง `left_join()` ได้ดังนี้

```
R> x <- tribble(
     ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     1, "x4"
   )
R> y <- tribble(
     ~key, ~val_y,
     1, "y1",
     2, "y2"
   )
R> x |>
     left_join(y, by = "key")
# A tibble: 4 x 3
    key val_x val_y
  <dbl> <chr> <chr>
```

```
1      1 x1    y1
2      2 x2    y2
3      2 x3    y2
4      1 x4    y1
```

2. คีย์ที่มีข้อมูลซ้ำกันทั้ง 2 ตาราง ดังรูปต่อไปนี้



รูปที่ 16-11 Duplicate keys in two tables

สร้างเป็นตาราง x และตาราง y และคำสั่ง `left_join()` ได้ดังนี้

```
R> x <- tribble(
     ~key, ~val_x,
     1, "x1",
     2, "x2",
     2, "x3",
     3, "x4"
   )
R> y <- tribble(
     ~key, ~val_y,
     1, "y1",
     2, "y2",
     2, "y3",
     3, "y4"
   )
R> x |>
     left_join(y, by = "key")
# A tibble: 6 x 3
    key val_x val_y
  <dbl> <chr> <chr>
1     1 x1    y1
2     2 x2    y2
3     2 x2    y3
4     2 x3    y2
5     2 x3    y3
6     3 x4    y4
```

16.12.6 การนิยามคีย์ (Key Column Definitions)

คำสั่งในการเชื่อมตาราง (Join tables) ที่กล่าวถึงข้างต้น ใช้ตัวแปรเพียง 1 ตัวในการกำหนดคีย์โดยการระบุอากิวเมนต์ `by = "key"` ค่าโดยปริยาย (defualt) ของพารามิเตอร์ `by` เท่ากับ `NULL` ซึ่งจะใช้ตัว

แปรทั้งหมดที่ปรากฏอยู่ทั้ง 2 ตาราง ตัวอย่างการเชื่อมตาราง flights และ weather จะเป็นการ Match โดยใช้ตัวแปร year, month, day, hour และ origin ดังนี้

```
R> flights2 |>
     left_join(weather)
Joining with `by = join_by(year, month, day, hour, origin)`
# A tibble: 336,776 × 18
    year month   day  hour origin dest  tailnum carrier  temp  dewp humid
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl> <dbl>
 1  2013     1     1     5 EWR    IAH   N14228  UA       39.0  28.0  64.4
 2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8
 3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6
 4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6
 5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0  54.8
 6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0  64.4
 7  2013     1     1     6 EWR    FLL   N516JB  B6       37.9  28.0  67.2
 8  2013     1     1     6 LGA    IAD   N829AS  EV       39.9  25.0  54.8
 9  2013     1     1     6 JFK    MCO   N593JB  B6       37.9  27.0  64.3
10  2013     1     1     6 LGA    ORD   N3ALAA  AA       39.9  25.0  54.8
# i 336,766 more rows
# i 7 more variables: wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>,
#   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>
```

ตัวอย่างการเชื่อมตารางโดยการระบุอากิวเมนต์ตามด้วยเวกเตอร์ตัวอักษรที่ต้องการ Match ได้ดังนี้

```
R> flights2 |>
     left_join(planes, by = "tailnum")
# A tibble: 336,776 × 16
   year.x month   day  hour origin dest  tailnum carrier year.y type
    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>
 1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixed wing …
 2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixed wing …
 3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixed wing …
 4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixed wing …
 5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixed wing …
 6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixed wing …
 7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixed wing …
 8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixed wing …
 9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixed wing …
10   2013     1     1     6 LGA    ORD   N3ALAA  AA          NA NA
# i 336,766 more rows
# i 6 more variables: manufacturer <chr>, model <chr>, engines <int>,
#   seats <int>, speed <int>, engine <chr>
```

ถ้าชื่อตัวแปรของทั้ง 2 ตารางแตกต่างกัน ระบุด้วยเวกเตอร์ตัวอักษรที่ต้องการ Match เช่น `by = c("a" = "b")` หมายถึง Match ตัวแปร a ในตาราง x กับตัวแปร b ในตาราง y ดังตัวอย่างต่อไปนี้

```
R> flights2 |>
     left_join(airports, c("dest" = "faa"))
# A tibble: 336,776 × 15
    year month   day  hour origin dest  tailnum carrier name       lat   lon
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>    <dbl> <dbl>
 1  2013     1     1     5 EWR    IAH   N14228  UA      George …  30.0 -95.3
```

```
 2  2013     1     1     5 LGA    IAH    N24211  UA      George …  30.0 -95.3
 3  2013     1     1     5 JFK    MIA    N619AA  AA      Miami I…  25.8 -80.3
 4  2013     1     1     5 JFK    BQN    N804JB  B6      NA        NA    NA
 5  2013     1     1     6 LGA    ATL    N668DN  DL      Hartsfi…  33.6 -84.4
 6  2013     1     1     5 EWR    ORD    N39463  UA      Chicago…  42.0 -87.9
 7  2013     1     1     6 EWR    FLL    N516JB  B6      Fort La…  26.1 -80.2
 8  2013     1     1     6 LGA    IAD    N829AS  EV      Washing…  38.9 -77.5
 9  2013     1     1     6 JFK    MCO    N593JB  B6      Orlando…  28.4 -81.3
10  2013     1     1     6 LGA    ORD    N3ALAA  AA      Chicago…  42.0 -87.9
# i 336,766 more rows
# i 4 more variables: alt <dbl>, tz <dbl>, dst <chr>, tzone <chr>
```

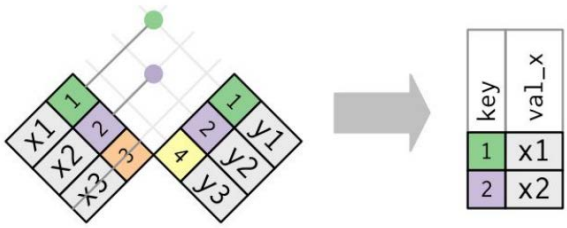
ผลลัพธ์ข้างต้นเป็นการ Match ตัวแปร dest ในตาราง flights2 กับตัวแปร faa ในตาราง airports

16.12.7 การกรองผลการเชื่อมตาราง (Filtering Joins)

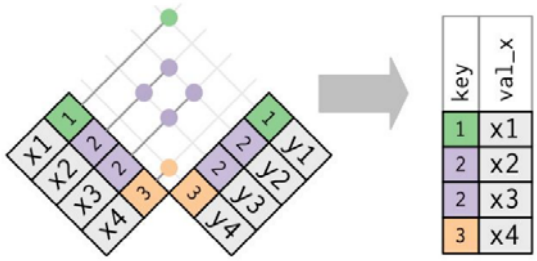
มีการเชื่อมตารางอีก 2 แบบที่สามารถแสดงข้อมูล เพื่อที่จะตรวจสอบผลการเชื่อมตารางได้ดังนี้

1. `semi_join(x, y)` เป็นการแสดงข้อมูลทั้งหมดในตาราง x ซึ่ง Match กับข้อมูลในตาราง y

คำสั่งนี้เหมาะกับการกรองผลการ Match แสดงผลการใช้คำสั่งนี้ได้ดังรูปต่อไปนี้



รูปที่ 16-12 Semi joins



รูปที่ 16-13 Semi joins: duplicate keys

ตัวอย่างเช่น ข้อมูลเที่ยวบิน (flights) 10 อันดับจุดหมายแรก ดังนี้

```
R> top_dest <- flights |>
     count(dest, sort = TRUE) |>
     head(10)
R> top_dest
# A tibble: 10 × 2
   dest      n
   <chr> <int>
 1 ORD   17283
```

```
 2 ATL   17215
 3 LAX   16174
 4 BOS   15508
 5 MCO   14082
 6 CLT   14064
 7 SFO   13331
 8 FLL   12055
 9 MIA   11728
10 DCA    9705
```

ถ้าต้องการข้อมูลแต่ละเที่ยวบินที่อยู่ในจุดปลายทาง 10 อันดับดังกล่าว สามารถเขียนคำสั่งได้ดังนี้

```
R> flights |>
     filter(dest %in% top_dest$dest)
# A tibble: 141,145 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      542            540         2      923
 2  2013     1     1      554            600        -6      812
 3  2013     1     1      554            558        -4      740
 4  2013     1     1      555            600        -5      913
 5  2013     1     1      557            600        -3      838
 6  2013     1     1      558            600        -2      753
 7  2013     1     1      558            600        -2      924
 8  2013     1     1      558            600        -2      923
 9  2013     1     1      559            559         0      702
10  2013     1     1      600            600         0      851
# i 141,135 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

แทนที่จะเขียนคำสั่งข้างต้น สามารถใช้คำสั่ง `semi_join()` ได้ดังนี้

```
R> flights |>
     semi_join(top_dest)
# A tibble: 141,145 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      542            540         2      923
 2  2013     1     1      554            600        -6      812
 3  2013     1     1      554            558        -4      740
 4  2013     1     1      555            600        -5      913
 5  2013     1     1      557            600        -3      838
 6  2013     1     1      558            600        -2      753
 7  2013     1     1      558            600        -2      924
 8  2013     1     1      558            600        -2      923
 9  2013     1     1      559            559         0      702
10  2013     1     1      600            600         0      851
# i 141,135 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

2. `anti_join(x, y)` จะแสดงผลตรงกันข้ามกับคำสั่ง `semi_join()` เป็นการเก็บข้อมูลทั้งหมดในตาราง x ซึ่งไม่ Match กับข้อมูลในตาราง y

คำสั่งนี้เหมาะกับการวิเคราะห์ผลการเชื่อมตารางที่ข้อมูลไม่ Match แสดงผลการใช้คำสั่งนี้ได้ดังรูปต่อไปนี้

รูปที่ 16-14 Anti joins

ตัวอย่างเช่น การเชื่อมข้อมูลเที่ยวบิน (flights) กับข้อมูลเครื่องบิน (planes) ถ้าต้องการทราบจำนวนเที่ยวบินที่ไม่ Match กับข้อมูลเครื่องบิน ใช้คำสั่ง `anti_join()` ได้ดังนี้

```
R> flights |>
     anti_join(planes, by = "tailnum") |>
     count(tailnum, sort = TRUE)
# A tibble: 722 × 2
   tailnum     n
   <chr>   <int>
 1 NA       2512
 2 N725MQ    575
 3 N722MQ    513
 4 N723MQ    507
 5 N713MQ    483
 6 N735MQ    396
 7 N0EGMQ    371
 8 N534MQ    364
 9 N542MQ    363
10 N531MQ    349
# ℹ 712 more rows
```

ผลลัพธ์ข้างต้นแสดงข้อมูลเที่ยวบินที่ไม่ Match กับข้อมูลเครื่องบิน จำนวน 722 เที่ยวบิน

16.13 สรุปท้ายบท

บทนี้เป็นจัดการข้อมูลด้วยแพ็กเกจ `dplyr` ได้แก่ การเลือกแถวข้อมูล การสุ่มข้อมูล การสร้างตัวแปร/คอลัมน์ใหม่ การเลือกตัวแปร/คอลัมน์ที่ต้องการ การสรุปข้อมูลแยกตามกลุ่ม การเรียงลำดับข้อมูล การประยุกต์ใช้คำสั่งดังกล่าวในการหาผลรวมสะสม และการจัดการข้อมูลที่มีความสัมพันธ์กัน ได้แก่ การเชื่อมตาราง การกำหนดคีย์ และการกรองผลการเชื่อมตาราง

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
as_tibble()	คำสั่งในการแปลงเดต้าเฟรม (data.frame) ให้เป็น tibble
tibble()	คำสั่งในการสร้าง tibble
tribble()	คำสั่งในการสร้าง tibble โดยใช้รูปแบบข้อมูลที่มีการ Transpose
|>	การใช้คำสั่งต่อเนื่องแบบลูกโซ่ (Chain) หรือบางครั้งเรียกว่าการ Pipe
filter()	คำสั่งในการเลือกข้อมูลตามเงื่อนไขที่ต้องการ
slice()	คำสั่งในการเลือกข้อมูลแถวที่ต้องการ
distinct()	คำสั่งในการตัดแถวข้อมูลที่ซ้ำซ้อนออกไปและแสดงเฉพาะข้อมูลที่กำหนด
slice_sample()	คำสั่งในการสุ่มข้อมูลตามจำนวนที่ต้องการ
slice_min()	คำสั่งในการเลือกแถวข้อมูลที่มีค่าต่ำสุด
slice_max()	คำสั่งในการเลือกแถวข้อมูลที่มีค่าสูงสุด
mutate()	คำสั่งในการสร้างคอลัมน์ (Column) ใหม่ขึ้นมา
transmute()	คำสั่งในการสร้างคอลัมน์ (Column) ใหม่ขึ้นมา โดยลบคอลัมน์เดิมทิ้งไป
select()	คำสั่งในการเลือกตัวแปร/คอลัมน์ โดยการกำหนดเงื่อนไขที่ต้องการ
group_by()	คำสั่งในการจัดข้อมูลแยกตามกลุ่ม
summarize()	คำสั่งในการรายงานสรุปข้อมูล
summarize_all()	คำสั่งในการรายงานสรุปข้อมูลทุกตัวแปร
summarize_at()	คำสั่งในการรายงานผลตัวแปรและฟังก์ชันที่ต้องการ
arrange()	การเรียงลำดับข้อมูลจากน้อยไปมาก (หรือน้อยไปมาก) ด้วยตัวแปรที่กำหนด
starts_with()	คำสั่งในการเลือกคอลัมน์ที่ขึ้นต้นด้วยสตริงที่กำหนด
ends_with()	คำสั่งในการเลือกคอลัมน์ที่ลงท้ายด้วยสตริงที่กำหนด
contains()	คำสั่งในการเลือกคอลัมน์ที่ประกอบด้วยสตริงที่กำหนด
everything()	คำสั่งในการเลือกคอลัมน์ที่เหลือทั้งหมด
num_range()	คำสั่งในการกำหนดชื่อคอลัมน์เริ่มต้น (Prefix) และตามด้วยเวกเตอร์ตัวเลขที่กำหนด
cumsum()	คำสั่งในการหาผลรวมสะสม
count()	คำสั่งในการนับจำนวนข้อมูล
inner_join()	คำสั่งในการเชื่อมตารางด้วยทำการ Match คีย์ที่เหมือนกันทั้ง 2 ตาราง
left_join()	คำสั่งในการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏในตาราง x
right_join()	คำสั่งในการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏในตาราง x
full_join()	คำสั่งในการเชื่อมตารางที่แสดงผลลัพธ์ทั้งหมดที่ปรากฏทั้งตาราง x และ y
semi_join()	คำสั่งในการเชื่อมตาราง และแสดงผลทั้งหมดในตาราง x ซึ่ง Match กับข้อมูลในตาราง y

คำสั่ง	คำอธิบาย
`anti_join()`	คำสั่งในการเชื่อมตาราง ที่ตาราง และแสดงผลทั้งหมดในตาราง x ซึ่งไม่ Match กับข้อมูลในตาราง y

16.14 แบบฝึกหัดท้ายบท

1. จากข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ mtcars ให้แสดงระยะทางรถวิ่งต่อแกลลอนน้ำมันตั้งแต่ 20 miles/gallon ขึ้นไป และแสดงจำนวนกระบอกสูบทั้งหมด ยกเว้นจำนวนกระบอกสูบเท่ากับ 4 ที่มีกำลังม้ามากกว่า 100 hp และน้ำหนักมากกว่า 3000 lbs
2. จากข้อมูลจาก datasets ที่มีอยู่ในโปรแกรม **R** ชื่อ mpg ให้แสดงผลข้อมูลที่ต้องการดังต่อไปนี้
   2.1 แสดงเฉพาะผู้ผลิต (Manufacturer) Audi, Ford, *และ* Honda ตั้งแต่ปี 2000
   2.2 จากข้อมูล mpg ให้สุ่มแถวข้อมูลจำนวน 20 แถว โดยการสุ่มแบบไม่ใส่*คืน* (Without Replacement) และกำหนด Sampling weights เป็นความจุกระบอกสูบ (Engine displacement)
   2.3 หาค่าเฉลี่ยรวมของจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) ของเขตเมือง (City) และบนทางหลวง (Highway) โดยสร้างคอลัมน์ใหม่ขึ้นมา
   2.4 ต้องการรายงานสรุปค่าเฉลี่ย ค่าส่วนเบี่ยงเบนมาตรฐาน ค่าต่ำสุด และค่าสูงสุดของจำนวนระยะทางต่อน้ำมันเชื้อเพลิง (Miles per gallon) ของเขตเมือง (City) และบนทางหลวง (Highway) แยกตามระบบเกียร์: auto และ manual
3. สร้าง tibble ข้อมูลยอดขายรายวันทั้งหมด 3 เดือน ตั้งแต่วันศุกร์ที่ 1 มิถุนายน 2561 ถึงวันอาทิตย์ที่ 30 กันยายน 2561 โดยข้อมูลยอดขายของทั้ง 4 เดือน และกำหนดค่า seed เท่ากับ 1234 เพื่อให้การทำซ้ำได้ตัวเลขสุ่มเหมือนกันทุกครั้ง ใช้การสุ่มแบบ Normal distribution ตามข้อมูลในตารางต่อไปนี้

ข้อมูลยอดขาย

เดือน	ค่าเฉลี่ย	ค่าส่วนเบี่ยงเบนมาตรฐาน
มิถุนายน	50,000	0.60
กรกฎาคม	45,000	0.80
สิงหาคม	55,000	0.50
กันยายน	48,000	0.65

   3.1 หาผลรวมสะสม (Cumulative summation) ของยอดขายแต่ละเดือน
   3.2 หาผลรวมสะสม (Cumulative summation) ของยอดขายวันจันทร์-ศุกร์ เรียงข้อมูลเดือนจากน้อยไปมาก
   3.3 หาผลรวมสะสม (Cumulative summation) ของยอดขายเฉพาะวันเสาร์-อาทิตย์ เรียงข้อมูลเดือนจากน้อยไปมาก

3.4 ต้องการรายงานสรุปค่าเฉลี่ย ค่าส่วนเบี่ยงเบนมาตรฐาน ค่าต่ำสุด และค่าสูงสุดของจำนวนยอดขาย แยกตามเดือน

3.5 ต้องการรายงานสรุปค่าเฉลี่ย ค่าส่วนเบี่ยงเบนมาตรฐาน ค่าต่ำสุด และค่าสูงสุดของจำนวนยอดขาย แยกตามวันจันทร์ถึงวันอาทิตย์

3.6 แสดงวันที่มียอดขายสูงสุด 5 อันดับแรก เรียงจากมากไปน้อยของแต่ละเดือน

บทที่ 17
การนำเข้าข้อมูลด้วยแพ็กเกจ readr และ readxl

การนำเข้าข้อมูลใช้แพ็กเกจชื่อว่า **readr** ซึ่งเป็นส่วนหนึ่งของแพ็กเกจ **tidyverse** คำสั่งใน **readr** ประกอบด้วยคำสั่งที่สำคัญ ดังนี้

- **read_csv()** เป็นคำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ ด้วยเครื่องหมายจุลภาค (Comma)
- **read_tsv()** เป็นคำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ ด้วยเครื่องหมาย Tab
- **read_delim()** เป็นคำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ ด้วยเครื่องหมายที่ผู้ใช้กำหนด
- **read_fwf()** เป็นคำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ ด้วยตำแหน่งของตัวอักษรในไฟล์
- **read_table()** เป็นคำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ ด้วยเครื่องหมาย White space ซึ่งอาจจะเป็นช่องว่าง หรือ Tab ก็ได้

ตัวอย่างต่อไปนี้แสดงการอ่านข้อมูลที่แยกแต่ละคอลัมน์ ด้วยเครื่องหมายจุลภาค (Comma) จาก Website ด้วยคำสั่ง **read_csv()** ซึ่งค่าโดยปริยาย (Default) จะอ่านข้อมูลแถวแรกเป็นตัวแปรแต่ละคอลัมน์ ที่เหลือจะเป็นข้อมูลทั้งหมด

```
R> tbl <- read_csv("http://www.stat.ucla.edu/~vlew/datasets/zipcodedata.csv")

# Exploratory Data with glimpse
R> glimpse(tbl)
Rows: 33,120
Columns: 21
$ zip               <chr> "00601", "00602", "00603", "00606", "00610"…
$ landarea          <dbl> 166659789, 79288158, 81880442, 109580061, 9…
$ waterarea         <dbl> 799296, 4446273, 183425, 12487, 4172001, 98…
$ totalpop          <dbl> 18570, 41520, 54689, 6615, 29016, 67010, 11…
$ totalhousing      <dbl> 7744, 18073, 25653, 2877, 12618, 30992, 489…
$ int.lat           <dbl> 18.18056, 18.36227, 18.45518, 18.15835, 18.…
$ int.lon           <dbl> -66.74996, -67.17613, -67.11989, -66.93291,…
$ totalUrban        <dbl> 10679, 41520, 54646, 2697, 25640, 62391, 10…
$ totalRural        <dbl> 7891, 0, 43, 3918, 3376, 4619, 579, 436, 16…
$ WhiteNH           <dbl> 80, 216, 628, 32, 187, 439, 36, 110, 129, 3…
$ AfrAmNH           <dbl> 2, 13, 101, 3, 22, 45, 11, 17, 8, 28, 14, 9…
$ AmInd             <dbl> 1, 0, 2, 0, 0, 5, 1, 0, 0, 1, 0, 10, 0, 2, …
$ Asian             <dbl> 1, 15, 48, 3, 8, 47, 1, 15, 1, 16, 10, 8, 0…
$ PacIsl            <dbl> 0, 0, 2, 1, 0, 1, 0, 0, 2, 2, 0, 1, 4, 0, 0…
$ Other             <dbl> 0, 4, 9, 0, 5, 14, 2, 3, 0, 8, 0, 3, 0, 2, …
$ Multi             <dbl> 0, 7, 22, 1, 5, 24, 1, 2, 5, 15, 4, 2, 0, 4…
$ `Hispanic/Latino` <dbl> 18486, 41265, 53877, 6575, 28789, 66435, 10…
```

```
$ TotMale           <dbl> 9078, 20396, 26597, 3268, 14104, 31735, 531…
$ TotFemale         <dbl> 9492, 21124, 28092, 3347, 14912, 35275, 569…
$ MedianAge         <dbl> 35.9, 37.5, 38.2, 36.2, 38.1, 39.2, 38.7, 3…
$ meanINCOME        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,…
```

คำสั่ง `read_csv()` สามารถอ่านข้อมูลในลักษณะ Inline csv ได้โดยตรง ดังนี้

```
R> read_csv("a,b,c
     1,2,3
     4,5,6")
# A tibble: 2 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1     1     2     3
2     4     5     6
```

คำสั่ง `glimpse()` ใช้สำหรับดูภาพรวมข้อมูลคอลัมน์ทั้งหมดโดยแสดงในแนวแถว โดยแต่ละแถวแสดงถึงข้อมูลของแต่ละคอลัมน์

คำสั่ง `read_csv()` สามารถอ่านข้อมูลข้ามบรรทัดได้ n แรกบรรทัด ด้วยอากิวเมนต์ `skip = n` ดังนี้

```
R> read_csv("The first line is title
     The second line is some metadata
     x,y,z
     1,2,3", skip = 2)

# A tibble: 1 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
```

คำสั่ง `read_csv()` จะอ่านข้อมูลข้ามบรรทัดที่มีเครื่องหมาย comment ที่กำหนดไว้ ด้วยพารามิเตอร์ `comment` ถ้ากำหนดให้ `comment = "#"` จะข้ามบรรทัดที่ขึ้นต้นด้วยเครื่องหมาย `#` ดังนี้

```
R> read_csv("# This line is a comment
     x,y,z
     1,2,3", comment = "#")
# A tibble: 1 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
```

ถ้าข้อมูลที่จะอ่านเข้ามาแถวแรกไม่ใช่ชื่อตัวแปรแต่เป็นข้อมูลที่ต้องการอ่าน สามารถกำหนดได้โดยให้อากิวเมนต์ `col_names = FALSE` ดังนี้

```
R> read_csv("1,2,3\n4,5,6", col_names = FALSE)
# A tibble: 2 x 3
     X1    X2    X3
  <dbl> <dbl> <dbl>
1     1     2     3
2     4     5     6
```

อักขระ **"\n"** หมายถึงการขึ้นบรรทัดใหม่ และสามารถกำหนดชื่อตัวแปรได้เองด้วยการผ่านชื่อตัวแปรเข้าไปในอากิวเมนต์ **col_names** ดังนี้

```
R> read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))
# A tibble: 2 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
2     4     5     6
```

สามารถกำหนดรูปแบบข้อมูลที่เป็น **NA** ได้ โดยกำหนดอักขระที่ต้องการให้เป็น **NA** อยู่ภายในเครื่องหมาย **""** ด้วยอากิวเมนต์ **na** ดังนี้

```
R> read_csv("a,b,c\n1,2,.", na = ".")
# A tibble: 1 x 3
      a     b c
  <dbl> <dbl> <lgl>
1     1     2 NA
```

ข้อดีของคำสั่ง **read_csv()** คือ **read_csv()** จะอ่านข้อมูลได้เร็วกว่า **read.csv()** ประมาณ 10 เท่า (H. Wickham & Grolemund, 2016) และมี Progress bar แสดงสถานะการอ่านเมื่อไฟล์มีขนาดใหญ่ และข้อมูลที่ได้เป็น tibble ไม่ต้องแปลงจากเดต้าเฟรม (data.frame) ให้เป็น tibble อีก

17.1 การแปลงชนิดข้อมูลเวกเตอร์ (Parsing a Vector)

การแปลงชนิดข้อมูลในเวกเตอร์ทำได้ด้วยคำสั่งในกลุ่ม **parse_*()** ซึ่งจะทำการแปลงอักขระ (Character vector) ให้เป็นชนิดข้อมูลที่กำหนด เช่น **parse_logical()** สำหรับแปลงข้อมูลอักขระให้เป็นข้อมูลตรรกะ (Logic), **parse_integer()** สำหรับแปลงข้อมูลอักขระให้เป็นเลขจำนวนเต็ม (Interger), **parse_double()** สำหรับแปลงข้อมูลอักขระให้เป็นเลขจำนวนจริงชนิด double, **parse_factor()** สำหรับแปลงข้อมูลอักขระให้เป็นแฟกเตอร์ (Factor), **parse_datetime()**, **parse_date()** และ **parse_time()** สำหรับแปลงข้อมูลอักขระให้เป็นวัน-เวลา วัน และเวลา ตามลำดับ ดังตัวอย่างต่อไปนี้

```
R> > str(parse_logical(c("TRUE", "FALSE", "NA")))
 logi[1:3] TRUE FALSE NA

R> str(parse_integer(c("1", "2", "3")))
 int[1:3] 1 2 3

R> str(parse_date(c("2021-10-11", "1988-01-01")))
 Date[1:2], format: "2021-10-11" "1988-01-01"
```

17.1.1 ข้อมูลตัวเลข (Numbers)

การแปลงข้อมูลตัวเลขมักพบปัญหา 3 ประการด้วยกัน คือ (1) แต่ละประเทศใช้ภาษาแตกต่างกัน การแยกระหว่างเลขจำนวนเต็มกับทศนิยมแตกต่างกัน เช่น ในประเทศอังกฤษหรือสหรัฐอเมริกาใช้จุดทศนิยม

(.) ในการแยก แต่ในยุโรปหลายประเทศใช้เครื่องหมายจุลภาค (,) (2) มีตัวอักขระผสมกับตัวเลข เช่น “$2000” หรือ “15%” และ (3) มีการจัดกลุ่มข้อมูลให้อ่านตัวเลขหลายหลักได้ง่าย อาทิ ในประเทศอังกฤษหรือสหรัฐอเมริกาใช้เครื่องหมายจุลภาค (,) ในการจัดกลุ่มตัวเลขที่มากกว่าหนึ่งพัน เช่น“1,000,000”

ปัญหาดังกล่าวทำให้การแปลงตัวเลขจากไฟล์ข้อมูลที่ต้องการประมวลผลมีความยุ่งยาก แต่ **readr** สามารถแก้ไขปัญหากล่าวข้างต้นได้ ดังนี้

ปัญหาข้อแรก **readr** ใช้พารามิเตอร์ **locale** ในการแยกระหว่างเลขจำนวนเต็มกับทศนิยม ดังนี้

```
R> parse_double("1.23")
[1] 1.23

R> parse_double("1,23", locale = locale(decimal_mark = ","))
[1] 1.23
```

ค่าโดยปริยาย (Default) ของ **readr** กำหนด **locale** เป็นประเทศสหรัฐอเมริกา ดังนั้นจึงใช้จุดทศนิยมในการแยกระหว่างเลขจำนวนเต็มกับทศนิยมเช่นเดียวกันกับที่ใช้อยู่ในประเทศไทย

สำหรับปัญหาข้อที่สอง **readr** ใช้คำสั่ง **parse_number()** ในการอ่านตัวเลขที่มีอักขระผสมอยู่ ดังนี้

```
R> parse_number("$200")
[1] 200

R> parse_number("30%")
[1] 30

R> parse_number("It is $123.45")
[1] 123.45
```

สำหรับปัญหาข้อสุดท้าย **readr** ใช้คำสั่ง **parse_number()** ในการอ่านตัวเลขที่มีอักขระผสมอยู่ และกำหนด **locale** ด้วย อากิวเมนต์ **grouping_mark** ดังนี้

```
R> parse_number("$123,456,789")
[1] 123456789
R> parse_number(
    "123.456.789",
    locale = locale(grouping_mark = ".")
)
[1] 123456789
```

17.1.2 ข้อมูลตัวอักษร (Strings)

การอ่านข้อมูลตัวอักษรที่ถูกเข้ารหัสด้วยภาษาอื่น ๆ นอกเหนือจากรหัส ASCII **readr** ใช้คำสั่ง **parse_character()** โดยกำหนด **locale** ด้วย อากิวเมนต์ **encoding** ดังตัวอย่างต่อไปนี้

```
R> x1 <- "El Ni\xf1o was particularly bad this year"
R> x2 <- "\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd"
```

```
R> parse_character(x1, locale = locale(encoding = "Latin1"))
[1] "El Niño was particularly bad this year"

R> parse_character(x2, locale = locale(encoding = "Shift-JIS"))
[1] "こんにちは"
```

ปัจจุบันนิยมเข้ารหัส UTF-8 ซึ่งรองรับแทบจะทุกภาษาที่มีใช้อยู่ในปัจจุบัน และอักขระพิเศษ เช่น emoji ด้วย ซึ่ง **readr** ใช้รหัส UTF-8 เป็นค่าโดยปริยายในการทำงานทุก ๆ ฟังก์ชันอยู่แล้ว

17.1.3 ข้อมูลแฟกเตอร์ (Factors)

การอ่านข้อมูลอักขระแล้วแปลงเป็นแฟกเตอร์ (Factors) **readr** ใช้คำสั่ง `parse_factor()` โดยกำหนดอากิวเมนต์ `levels` ดังตัวอย่างต่อไปนี้

```
R> fruit <- c("apple", "banana")
R> parse_factor(c("apple", "banana", "orange"), levels = fruit)
Warning: 1 parsing failure.
row col           expected actual
  3  -- value in level set orange

[1] apple  banana <NA>
attr(,"problems")
# A tibble: 1 x 4
    row   col expected           actual
  <int> <int> <chr>              <chr>
1     3    NA value in level set orange
Levels: apple banana
```

17.1.4 ข้อมูลวัน-เวลา วัน และเวลา (Date-Times, Dates, and Times)

การเลือกใช้งานระหว่าง `parse_datetime()`, `parse_date()` และ `parse_time()` เพื่อแปลงข้อมูลอักขระให้เป็นวัน-เวลา วัน และเวลา ขึ้นอยู่กับผู้ใช้งานต้องการ รายละเอียดของข้อมูลทั้ง 3 ประเภทประกอบด้วย (1) ข้อมูลวัน-เวลา มีหน่วยเป็นวินาที ตั้งแต่เที่ยงคืน วันที่ 1 มกราคม 1970 (2) ข้อมูลวัน มีหน่วยเป็นวัน ตั้งแต่วันที่ 1 มกราคม 1970 และ (3) ข้อมูลเวลา มีหน่วยเป็นวินาที ตั้งแต่เที่ยงคืน วันที่ 1 มกราคม 1970 ตามลำดับ

เมื่อเรียกใช้คำสั่ง `parse_datetime()` วัน-เวลาจะใช้มาตรฐานตาม ISO 8601 ซึ่งข้อมูลวัน-เวลาจะประกอบด้วยปี เดือน วัน ชั่วโมง นาที และวินาที ตามลำดับ ดังตัวอย่างต่อไปนี้

```
R> parse_datetime("2020-10-01T2010")
[1] "2020-10-01 20:10:00 UTC"

R> parse_datetime("20201010")
[1] "2020-10-10 UTC"
```

คำสั่ง `parse_date()` ต้องการข้อมูลปี เดือน และวัน ตามลำดับ คั้นด้วยเครื่องหมาย `-` หรือ `/` ดังต่อไปนี้

```
R> parse_date("2020-10-01")
[1] "2020-10-01"
```

คำสั่ง **parse_time()** ต้องการข้อมูลชั่วโมง และนาที คั้นด้วยเครื่องหมาย **:** ส่วนข้อมูลวินาที จะใส่หรือไม่ใส่ก็ได้ และสามารถกำหนดอากิวเมนต์ **am** หรือ **pm** ได้ดังต่อไปนี้

```
R> parse_time("01:30 am")
01:30:00

R> parse_time("01:30 pm")
13:30:00

R> parse_time("20:10:05")
20:10:05
```

ผู้ใช้สามารถกำหนดรูปแบบการนำเข้าข้อมูลได้ ดังนี้

- ปี

  %Y (4 digits)

  %y (2 digits; 00-69 ⟶ 2000-2069, 70-99 ⟶ 1970-1999)

- เดือน

  %m (2 digits)

  %b (ตัวย่อ เช่น “Jan”)

  %B (ตัวเต็ม เช่น “January”)

- วัน

  %d (2 digits)

  %e (ตัวย่อ เช่น “Jan”)

- เวลา

  %H (0-23 ชั่วโมง)

  %I (0-12 ใช้ร่วมกับ **%p**)

  %p (แสดง am หรือ pm)

  %M (นาที)

  %S (วินาที เป็นเลขจำนวนเต็ม)

  %OS (วินาที เป็นเลขจำนวนจริง)

ดังตัวอย่างต่อไปนี้

```
R> parse_date("01/03/20", "%m/%d/%y")
[1] "2020-01-03"

R> parse_date("01/03/20", "%d/%m/%y")
[1] "2020-03-01"

R> parse_date("01/03/20", "%y/%m/%d")
[1] "2001-03-20"
```

17.2 การเขียนไฟล์ด้วยแพ็กเกจ readr

แพ็กเกจ **readr** มีคำสั่งที่ใช้เขียนไฟล์ประกอบด้วย

- **write_csv()** เป็นคำสั่งในการเขียนไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายจุลภาค (Comma)
- **write_excel_csv()** เป็นคำสั่งในการเขียนไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายจุลภาค (Comma) สำหรับไฟล์ Excel โดยเฉพาะ
- **write_tsv()** เป็นคำสั่งในการเขียนไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมาย Tab
- **write_delim()** เป็นคำสั่งในการเขียนไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายที่ผู้ใช้กำหนด

การเขียนไฟล์ด้วยคำสั่งดังกล่าวจะทำการเขียนไฟล์โดยเข้ารหัสเป็น UTF-8 ข้อมูลวัน-เวลาจะใช้มาตรฐานตาม ISO 8601 ทำให้การอ่านข้อมูลกลับเข้าไปมีความเป็นมาตรฐานลดโอกาสที่จะทำให้เกิดความผิดพลาด

รูปแบบการเขียนไฟล์ **write_*()** แสดงดังต่อไปนี้

```
write_csv(
  x,
  file,
  na = "NA",
  append = FALSE,
  col_names = TRUE,
  quote_escape = "double",
  eol = "\n",
)
```

พารามิเตอร์แต่ละตัวในคำสั่ง **write_csv()** มีความหมายและการใช้งานดังนี้

- พารามิเตอร์ตัวแรก **x** คือ เดต้าเฟรม หรือ tibble ที่ต้องการเขียนเป็นไฟล์
- พารามิเตอร์ **file** คือ ชื่อและ Path ไฟล์ที่ต้องการเขียน
- พารามิเตอร์ **na** เป็นการระบุว่าถ้าข้อมูลเป็น **na** ให้เขียนเป็นตัวอักษรอะไรภายในเครื่องหมาย ""

- พารามิเตอร์ **append** ถ้าเท่ากับ **FALSE** หมายถึงให้เขียนทับไฟล์เดิม ถ้าเท่ากับ **TRUE** หมายถึงให้เขียนไฟล์ใหม่
- พารามิเตอร์ **col_names** ถ้าเท่ากับ **FALSE** หมายถึง ไม่เขียนชื่อคอลัมน์ลงในบรรทัดแรกของไฟล์ ถ้าเท่ากับ **TRUE** หมายถึง เขียนชื่อคอลัมน์ลงในบรรทัดแรกของไฟล์
- พารามิเตอร์ **quote_escape** ค่าโดยปริยายคือ **"double"** ซึ่งเป็นรูปแบบสำหรับโปรแกรม Excel อยู่แล้ว
- พารามิเตอร์ **eol** เป็นการกำหนดรหัสสำหรับการขึ้นบรรทัดใหม่

ตัวอย่างการเขียนไฟล์ csv แสดงดังต่อไปนี้

```
R> data(mpg)
R> write_csv(mpg, file = "mpg.csv")
R> write_excel_csv(mpg, file = "mpg_excel.csv")
```

17.3 การนำเข้าข้อมูล Excel ด้วยแพ็กเกจ readxl

แพ็กเกจ **readxl** มีคำสั่งในการอ่านไฟล์ Excel ชื่อว่า **read_excel()** ซึ่งจะทำการอ่านไฟล์โดยดูจากนามสกุลไฟล์ xls หรือ xlsx แต่ถ้าต้องการระบุประเภทไฟล์เอง ก็ใช้คำสั่ง **read_xls()** หรือ **read_xlsx()** ได้โดยตรง โดยไฟล์ที่อ่านเข้ามาจะเป็น tibble

รูปแบบการอ่านไฟล์ **read_excel()** แสดงดังต่อไปนี้

```
read_excel(path,
  sheet = NULL,
  range = NULL,
  col_names = TRUE,
  col_types = NULL,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  progress = readxl_progress(),
  .name_repair = "unique"
)
```

พารามิเตอร์ที่สำคัญประกอบด้วย **path** หมายถึง ชื่อไฟล์และ Path ที่ต้องการอ่าน, **sheet** หมายถึง ชื่อ worksheet ที่ต้องการอ่าน, **range** หมายถึง ขอบเขตข้อมูลที่ต้องการอ่าน, **col_names** ถ้าเท่ากับ **TRUE** บรรทัดแรกของไฟล์เป็นชื่อคอลัมน์ (Column) ถ้าเท่ากับ **FALSE** บรรทัดแรกของไฟล์เริ่มด้วยข้อมูลที่ต้องการอ่าน, **col_types** ค่าโดยปริยาย (Defualt) กำหนดให้เท่ากับ **NULL** หมายถึง ให้ตัวโปรแกรมกำหนดประเภทให้เองโดยดูจากข้อมูลที่อ่านเข้ามา, **skip** คือ จำนวนแถวที่ไม่ทำการอ่านข้อมูล

พารามิเตอร์ **range** สามารถกำหนดขอบเขตอ้างอิงข้อมูลที่ต้องการอ่านได้หลายลักษณะ ได้แก่

1. การอ้างอิงเหมือน Excel

เป็นการอ้างอิงโดยกำหนดชื่อ Cell เหมือนการอ้างถึง Excel โดยทั่วไป ตัวอย่างเช่น

```
R> path <- readxl_example("geometry.xls")
## Rows 1 and 2 are empty (as are rows 7 and higher)
## Column 1 aka "A" is empty (as are columns 5 of "E" and higher)

R> read_excel(path)

# Specific rectangle that is subset of populated cells, possibly improper
R> read_excel(path, range = "B3:D6")
R> read_excel(path, range = "C3:D5")

# Specific rectangle that forces inclusion of unpopulated cells
R> read_excel(path, range = "A3:D5")
R> read_excel(path, range = "A4:E5")
```

2. การอ้างอิงด้วยคำสั่ง **anchored()**

เป็นการอ้างอิงโดยกำหนดตำแหน่งซ้ายบน จำนวนแถวและจำนวนคอลัมน์ของพื้นที่ที่ต้องการ โดยมีรูปแบบดังนี้

```
anchored(anchor = "A1",
  dim = c(1L, 1L),
  input = NULL,
  col_names = NULL,
  byrow = FALSE)
```

พารามิเตอร์ที่สำคัญประกอบด้วย **anchor** เป็นการกำหนดตำแหน่งอ้างอิง (ซ้ายบน), **dim** เป็นการกำหนดขนาดกว้างยาวของพื้นที่ที่ต้องการ ในรูปเวกเตอร์จำนวนเต็มที่มีสมาชิก 2 ตัว ตัวแรกแทนจำนวนแถว ตัวที่สองแทนจำนวนคอลัมน์ เช่น c(3, 2) หมายถึง 3 แถว 2 คอลัมน์, **col_names** ถ้าเท่ากับ **TRUE** บรรทัดแรกของไฟล์เป็นชื่อคอลัมน์ (Column) ถ้าเท่ากับ **FALSE** บรรทัดแรกของไฟล์เริ่มด้วยข้อมูลที่ต้องการอ่าน ตัวอย่างเช่น

```
# Anchor a rectangle of specified size at a particular cell
R> read_excel(path, range = anchored("C4", dim = c(3, 2)), col_names = FALSE)
```

3. การอ้างอิงด้วยคำสั่ง **cell_limits()**

เป็นการอ้างอิงโดยกำหนดตำแหน่งซ้ายบน และตำแหน่งขวาล่างของพื้นที่ที่ต้องการ โดยมีรูปแบบดังนี้

```
cell_limits(ul = c(NA_integer_, NA_integer_),
  lr = c(NA_integer_, NA_integer_),
  sheet = NA_character_
)
```

พารามิเตอร์ที่สำคัญประกอบด้วย **ul** เป็นการกำหนดเวกเตอร์ตำแหน่งซ้ายบนของพื้นที่ที่ต้องการ, **lr** เป็นการกำหนดเวกเตอร์ตำแหน่งขวาล่างของพื้นที่ที่ต้องการ, **sheet** หมายถึง ชื่อ worksheet ที่ต้องการอ่าน ตัวอย่างเช่น

```
# Identify cell boundary by cell_limits
R> read_excel(path, range = cell_limits(c(1, 3), c(6, 4)), col_names = FALSE)
R> read_excel(path, range = cell_limits(c(NA, 3), c(7, NA)), col_names = FALSE)
R> read_excel(path, range = cell_limits(c(NA, 3)), col_names = FALSE)
R> read_excel(path, range = cell_limits(lr = c(NA, 3)), col_names = FALSE)
R> read_excel(path, range = cell_limits(c(1, 3), c(6, 4), sheet = "Sheet2"),
              col_names = FALSE)
```

คำสั่งแรกเป็นการอ่านข้อมูลจาก Excel เริ่มจากแถวที่ 1 คอลัมน์ที่ 3 จนถึงแถวที่ 6 คอลัมน์ที่ 4 โดยไม่มีการกำหนดชื่อตัวแปร คำสั่งที่สองเป็นการอ่านข้อมูลเริ่มจากทุกแถวคอลัมน์ที่ 3 จนถึงแถวที่ 7 คำสั่งที่สามเป็นการอ่านข้อมูลเริ่มจากคอลัมน์ที่ 3 ทุกแถวที่มีข้อมูลจนไม่พบข้อมูลใน Excel คำสั่งที่สี่เป็นการอ่านข้อมูลเริ่มจากคอลัมน์แรกทุกแถวที่มีข้อมูลจนถึงคอลัมน์ที่ 3 คำสั่งสุดท้ายเหมือนคำสั่งแรกแต่อ่านจาก worksheet ชื่อ "Sheet2"

4. การอ้างอิงด้วยคำสั่ง `cell_rows()`

เป็นการอ้างอิงโดยการกำหนดแถวที่ต้องการอ่าน โดยมีรูปแบบดังนี้

```
cell_rows(x)
```

พารามิเตอร์ **x** คือ เวกเตอร์ตัวเลขแถวที่ต้องการ ถ้าเวกเตอร์ความยาวมากกว่า 2 จะเป็นการกำหนดแถวต่ำสุดและแถวสูงสุด พร้อมกับพารามิเตอร์ **NA.rm = TRUE** ตัวอย่างเช่น

```
# Identify cell boundary by cell_rows
R> read_excel(path, range = cell_rows(c(NA, 3)))
R> read_excel(path, range = cell_rows(c(4, NA)))
R> read_excel(path, range = cell_rows(4:10))
```

คำสั่งแรกเป็นการอ่านข้อมูลจาก Excel เฉพาะคอลัมน์ที่มีข้อมูลเริ่มจากแถวแรกจนถึงแถวที่ 3 คำสั่งที่สองเป็นการอ่านข้อมูลเฉพาะคอลัมน์ที่มีข้อมูลเริ่มจากแถวที่ 4 เป็นต้นไปจนไม่พบข้อมูลใน Excel คำสั่งสุดท้ายเป็นการอ่านข้อมูลเฉพาะคอลัมน์ที่มีข้อมูลเริ่มจากแถวที่ 4 จนถึงแถวที่ 10 ทั้งสามคำสั่งกำหนดให้แถวแรกเป็นชื่อตัวแปร

5. การอ้างอิงด้วยคำสั่ง `cell_cols()`

เป็นการอ้างอิงโดยการกำหนดคอลัมน์ (Column) ที่ต้องการอ่าน โดยมีรูปแบบดังนี้

```
cell_cols(x)
```

พารามิเตอร์ **x** คือ เวกเตอร์ตัวเลขคอลัมน์ที่ต้องการ ถ้าเวกเตอร์ความยาวมากกว่า 2 จะเป็นการกำหนดคอลัมน์ต่ำสุดและคอลัมน์สูงสุด พร้อมกับพารามิเตอร์ **NA.rm = TRUE** ตัวอย่างเช่น

```
# Identify cell boundary by cell_cols
R> read_excel(path, range = cell_cols(c(NA, 3)))
R> read_excel(path, range = cell_cols(c(2, NA)))
R> read_excel(path, range = cell_cols(3:6))
```

คำสั่งแรกเป็นการอ่านข้อมูลจาก Excel เฉพาะแถวที่มีข้อมูลเริ่มจากคอลัมน์แรกจนถึงคอลัมน์ที่ 3 คำสั่งที่สองเป็นการอ่านข้อมูลเฉพาะแถวที่มีข้อมูลเริ่มจากคอลัมน์ที่ 2 เป็นต้นไปจนไม่พบข้อมูลใน Excel คำสั่งสุดท้ายเป็นการอ่านข้อมูลเฉพาะแถวที่มีข้อมูลเริ่มจากคอลัมน์ที่ 3 จนถึงคอลัมน์ที่ 6 ทั้งสามคำสั่งกำหนดให้แถวแรกเป็นชื่อตัวแปร

17.4 สรุปท้ายบท

บทนี้กล่าวถึงการนำเข้าข้อมูลด้วยแพ็กเกจ **readr** การแปลงชนิดข้อมูลเวกเตอร์ (Parsing a Vector) ทั้งข้อมูลตัวเลข (Numbers) ข้อมูลตัวอักษร (Strings) ข้อมูลแฟกเตอร์ (Factors) ข้อมูลวัน-เวลา วัน และเวลา (Date-Times, Dates, and Times) การเขียนไฟล์ด้วยแพ็กเกจ **readr** การนำเข้าข้อมูล Excel ด้วยแพ็กเกจ **readxl**

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`read_csv()`	คำสั่งในการอ่านไฟล์ csv แยกแต่ละคอลัมน์ด้วยเครื่องหมายจุลภาค (Comma)
`read_tsv()`	คำสั่งในการอ่านไฟล์ tsv แยกแต่ละคอลัมน์ด้วยเครื่องหมาย Tab
`read_delim()`	คำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายที่ผู้ใช้กำหนดเอง
`read_fwf()`	คำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ด้วยตำแหน่งของตัวอักขระในไฟล์
`read_table()`	คำสั่งในการอ่านไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมาย White space ซึ่งอาจจะเป็นช่องว่าง หรือ Tab ก็ได้
`parse_logical()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นข้อมูลตรรกะ (Logic)
`parse_integer()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเลขจำนวนเต็ม (Interger)
`parse_double()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเลขจำนวนจริงชนิด double
`parse_number()`	คำสั่งในการแปลงข้อมูลตัวเลขที่มีอักขระผสมอยู่ให้เป็นตัวเลข
`parse_character()`	คำสั่งในการอ่านข้อมูลตัวอักษรที่ถูกเข้ารหัสด้วยภาษาอื่น
`parse_factor()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเป็นแฟกเตอร์ (Factor)
`parse_datetime()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเป็นวัน-เวลา
`parse_date()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเป็นวัน
`parse_time()`	คำสั่งในการแปลงข้อมูลอักขระให้เป็นเป็นเวลา
`write_csv()`	คำสั่งในการเขียนไฟล์ csv ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายจุลภาค (Comma)
`write_excel_csv()`	คำสั่งในการเขียนไฟล์ csv ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายจุลภาค (Comma) สำหรับไฟล์ Excel โดยเฉพาะ

คำสั่ง	คำอธิบาย
`write_tsv()`	คำสั่งในการเขียนไฟล์ tsv ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมาย Tab
`write_delim()`	คำสั่งในการเขียนไฟล์ที่แยกแต่ละคอลัมน์ด้วยเครื่องหมายที่ผู้ใช้กำหนดเอง
`read_excel()`	คำสั่งในการอ่านไฟล์ Excel ทั้งไฟล์นามสกุล xls และ xlsx
`read_xls()`	คำสั่งในการอ่านไฟล์ Excel นามสกุล xls
`read_xlsx()`	คำสั่งในการอ่านไฟล์ Excel นามสกุล xlsx
`anchored()`	คำสั่งในการอ้างอิงโดยกำหนดตำแหน่งซ้ายบน จำนวนแถวและจำนวนคอลัมน์ของพื้นที่ที่ต้องการ
`cell_limits()`	คำสั่งในการอ้างอิงโดยกำหนดตำแหน่งซ้ายบน และตำแหน่งขวาล่างของพื้นที่ที่ต้องการ
`cell_rows()`	คำสั่งในการอ้างอิงโดยการกำหนดแถวที่ต้องการอ่าน
`cell_cols()`	คำสั่งในการอ้างอิงโดยการกำหนดคอลัมน์ (Column) ที่ต้องการอ่าน

17.5 แบบฝึกหัดท้ายบท

1. ให้ดาวโหลดไฟล์ข้อมูลที่สนใจ จากเว็บไซต์ต่าง ๆ ที่เป็นข้อมูล Open Source เช่น Kaggle Datasets (https://www.kaggle.com/datasets) เป็นต้น ที่มีข้อมูลนามสกุล .csv จากนั้นนำเข้าข้อมูลด้วยคำสั่ง `read.csv()` และ `read_csv()` และแสดงผลข้อมูลที่นำเข้ามา พร้อมทั้งตรวจสอบโครงสร้างและอธิบายว่าเหมือนหรือต่างกันอย่างไร
2. จากข้อมูลจากแพ็กเกจ datasets ชื่อว่า swiss และทำการเพิ่มคอลัมน์ใหม่ที่ชื่อว่า Province เก็บข้อมูลชื่อจังหวัดของแต่ละแถวในข้อมูล จากนั้นให้ทำการเขียนไฟล์ต่าง ๆ ต่อไปนี้

   2.1 เขียนไฟล์ชื่อว่า swiss.csv

   2.2 เขียนไฟล์ชื่อว่า swiss.tsv

3. จากไฟล์ที่ได้ในข้อ 2 ให้ทำการนำเข้าข้อมูลต่าง ๆ เหล่านั้น และแสดงผลข้อมูลที่นำเข้ามา
4. จากไฟล์ swiss.csv ในข้อ 2 ให้ทำการเขียนไฟล์เป็นรูปแบบ Excel และทำการ save ไฟล์เป็นชื่อ swiss.xlsx จากนั้นนำเข้าข้อมูลมาทำงานใน **R** และแสดงผลข้อมูลที่นำเข้ามา
5. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้ และเก็บข้อมูลไว้ในตัวแปรชื่อ weather_tbl

ข้อมูลสภาพภูมิอากาศ

month	temperature	humidity
Jan	32	45%
Feb	27	81%
Mar	23	78%
Apr	41	38%
May	40	20%

   5.1 ให้แปลงข้อมูลในคอลัมน์ month ให้เป็นชื่อเดือนแบบเต็ม (Jan = January)

   5.2 ให้แปลงข้อมูลในคอลัมน์ humidity ให้เป็นข้อมูลตัวเลข

   5.3 ให้เขียนไฟล์ชื่อว่า weather.csv ที่เป็นข้อมูลจากการแปลงข้อมูลในข้อ 5.1 และ 5.2

บทที่ 18
การจัดการสตริง (Strings) ด้วยแพ็กเกจ stringr

ในประเภทข้อมูลของโปรแกรม R นั้น อักขระ (Characters) เป็นตัวอักษร/หรือสัญลักษณ์เพียงตัวเดียว ส่วนสตริง (Strings) เป็นการนำอักขระแต่ละตัวมาประกอบกันเป็นข้อความ หัวข้อนี้จะกล่าวถึงการจัดการข้อมูลสตริง (String) หรือข้อความด้วยแพ็กเจก **stringr** คำสั่งจัดการสตริงจะขึ้นต้นด้วย **str_**

RStudio ได้สร้างไฟล์สรุปการใช้งาน **stringr** ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมากสำหรับผู้ใช้ **R** (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว)

18.1 การสร้างสตริง

การสร้างสตริงจะอยู่ภายในเครื่องหมายอัญประกาศเดี่ยว Single quote (**' '**) หรืออัญประกาศคู่ Double quote (**" "**) แต่โดยทั่วไป **R** นิยมใช้เครื่องหมาย Double quote ยกเว้นต้องการเครื่องหมาย Double quote อยู่ภายในข้อมูลสตริงให้ใช้เครื่องหมาย Single quote ช่วยดังตัวอย่างต่อไปนี้

```
R> string1 <- "Here is a string"
R> string2 <- 'To put a "quote" inside a string, use single quotes'
```

การสร้างข้อมูลสตริงที่มีอักขระ Single quote หรือ Double quote เพียงตัวเดียว หรือมีอักขระ **** ให้ใช้เครื่องหมาย **** นำหน้าอักขระดังกล่าว ดังนี้

```
R> double_quote <- "\""     # or '"'
R> single_quote <- '\''     # or "'"
R> back_slash <- "\\"
```

18.2 การหาความยาวสตริง

การหาความยาวสตริงว่ามีความยาวกี่ตัวอักษรใช้คำสั่ง **str_length()** ดังนี้

```
R> str_length(c("abc", "Data science", NA))
[1]  3 12 NA
```

18.3 การเชื่อมสตริง

การเชื่อมสตริงกับสตริงอื่น ๆ ใช้คำสั่ง **str_c()** ดังนี้

```
R> str_c("x", "y")
[1] "xy"

R> str_c("x", "y", "z")
[1] "xyz"
```

ข้อมูลที่มีค่า **NA** อยู่จะแสดงค่า **NA** ออกมา แต่ถ้าต้องการแสดงออกเป็นสตริง “NA” ให้ใช้คำสั่ง `str_replace_na()` ดังนี้

```
R> string <- c("abc", NA)

R> str_c("|-", string, "-|")
[1] "|-abc-|" NA

R> str_c("|-", str_replace_na(string), "-|")
[1] "|-abc-|" "|-NA-|"
```

คำสั่ง `str_c()` ทำงานกับเวกเตอร์แบบ vectorization โดยจะทำซ้ำ (Recycle) เวกเตอร์ที่ความยาวน้อยกว่าให้มีความยาวเท่ากับเวกเตอร์ที่ความความยาวมากกว่า ดังตัวอย่างต่อไปนี้

```
R> str_c("prefix-", c("a", "b", "c"), "-suffix")
[1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

ถ้าต้องการรวมสตริงที่อยู่ในเวกเตอร์เข้าด้วยกันให้ใช้อากิวเมนต์ `collapse` ดังนี้

```
R> str_c(c("x", "y", "z"))    # no option collapse
[1] "x" "y" "z"

R> str_c(c("x", "y", "z"), collapse = ", ")
[1] "x, y, z"

R> str_c(c("x", "y", "z"), collapse = "")
[1] "xyz"
```

18.4 การดึงข้อมูลบางส่วนของสตริง (Subsetting Strings)

การดึงข้อมูลบางส่วนของสตริงจะใช้คำสั่ง `str_sub()` โดยกำหนดตำแหน่งเริ่มต้น และตำแหน่งสุดท้ายที่จะดึงข้อมูลบางส่วนของสตริงออกมา ดังนี้

```
R> fruit <- c("Apple", "Banana", "Orange")
R> str_sub(fruit, start = 1, end = 3)
[1] "App" "Ban" "Ora"
```

แต่ถ้าค่าเป็นลบ หมายถึง ตำแหน่งดังกล่าวนับจากด้านหลังมาด้านหน้า ดังนี้

```
R> str_sub(fruit, start = -3, end = -1)
[1] "ple" "ana" "nge"

R> str_sub(fruit, start = -3, end = 5)
[1] "ple" "an"  "ng"
```

คำสั่ง `str_sub()` จะแสดงข้อมูลเท่าที่มี ถ้าสตริงมีความยาวน้อยกว่าตำแหน่งที่ต้องการดึงข้อมูลออกมา ดังนี้

```
R> str_sub("abc", 1, 5)
[1] "abc"
```

18.5 การเรียงข้อมูลสตริง และการแปลงสตริงเป็นตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก

การเรียงข้อมูลสตริงในเวกเตอร์จะใช้คำสั่ง `str_sort()` ดังนี้

```
R> str_sort(c("apple", "orange", "banana"))
[1] "apple"  "banana" "orange"
```

ใช้คำสั่ง `str_to_lower()` ในการแปลงสตริงเป็นตัวพิมพ์เล็ก `str_to_upper()` ในการแปลงสตริงเป็นตัวพิมพ์ใหญ่ และ `str_to_title()` ให้แต่ละคำขึ้นต้นด้วยตัวพิมพ์ใหญ่ ดังนี้

```
R> fruits <- c("This is some fruits", "apple", "banana")
R> str_to_lower(fruits)
[1] "this is some fruits" "apple"               "banana"

R> str_to_upper(fruits)
[1] "THIS IS SOME FRUITS" "APPLE"               "BANANA"

R> str_to_title(fruits)
[1] "This Is Some Fruits" "Apple"               "Banana"
```

18.6 การจัดการสตริงด้วย Regular Expression

Regular Expression หรือย่อว่า Regex เป็นเครื่องมือในการหาสตริงที่ตรงกับรูปแบบที่ต้องการ Regex มีคำสั่งที่กระชับ แต่มีประสิทธิภาพในการจัดการสตริงสูงมาก ซึ่งภาษาคอมพิวเตอร์มาตรฐานจะมีคำสั่งรองรับการใช้ฟังก์ชัน Regex แทบทุกภาษารวมทั้ง **R** Regex สามารถค้นหา แทนที่ และแบ่งสตริงเป็นหลายส่วน การเรียนรู้การใช้งาน Regex จะใช้คำสั่ง `str_view()` และ `str_view_all()` รูปแบบพื้นฐานที่สุดเป็นการหาสตริงที่ตรงกับ (Match) คำที่ต้องการ ดังนี้

```
R> fruits <- c("apple", "banana", "pear")
R> str_view(fruits, "an")     # Exact match
[2] │ b<an><an>a
```

คำสั่งแรกเป็นการสร้างเวกเตอร์ fruits เก็บข้อมูลชื่อผลไม้ 3 ประเภท คำสั่งที่สองเป็นการหาสตริงที่ตรงกับ "an" ซึ่งจะพบในคำว่า banana

ต่อมาเป็นการค้นหาสตริงโดยใช้เครื่องหมาย `"."` หมายถึง หาอักขระอะไรก็ได้ยกเว้นอักขระแสดงการขึ้นบรรทัดใหม่ (\n) ดังนี้

```
R> str_view(fruits, ".a.")
[2] │ <ban>ana
[3] │ p<ear>
```

คำสั่งข้างบนเป็นการหาสตริงที่ตรงกับ `".a."` ซึ่งจะพบในคำว่า banana และ pear

การหาอักขระ `"."` จะต้องใช้เครื่องหมาย Backslash 2 ตัวนำหน้าจุด `"\\."` ดังนี้

```
R> str_view(c("abc", "a.c", "bef"), "a\\.c")
[2] │ <a.c>
```

แต่ถ้าต้องการหาเครื่องหมาย **"\"** ต้องใช้เครื่องหมาย Backslash 4 ตัว ดังนี้

```
R> x <- "a\\b"
R> writeLines(x)
a\b

R> str_view(x, "\\\\")
[1] │ a<\>b
```

โดยปกติการหารูปแบบในสตริง Regex จะทำการหาทุกส่วนของสตริง แต่ถ้าต้องการหาเฉพาะจุดเริ่มต้นสตริงจะใช้เครื่องหมาย (^) ดังนี้

```
R> str_view(fruits, "^a")
[1] │ <a>pple
```

ถ้าต้องการหาเฉพาะจุดสิ้นสุดสตริงจะใช้เครื่องหมาย (**$**) ดังนี้

```
R> str_view(fruits, "a$")
[2] │ banan<a>
```

ถ้าต้องการหาสตริงที่ตรงกับรูปแบบที่ต้องการทั้งจุดเริ่มต้นและจุดสิ้นสุดสตริง จะต้องใช้เครื่องหมาย (^) และ (**$**) ร่วมกัน ดังนี้

```
R> x <- c("apple pie", "apple", "apple cake")
R> str_view(x, "apple")
[1] │ <apple> pie
[2] │ <apple>
[3] │ <apple> cake

R> str_view(x, "^apple$")
[2] │ <apple>
```

มีรูปแบบพิเศษในการหาสตริงมากกว่าอักขระ 1 ตัว โดยใช้สัญลักษณ์ ดังต่อไปนี้

- **\d** สำหรับค้นหาตัวเลข (0-9) 1 ตัว
- **\D** สำหรับค้นหาตัวอักษรที่ไม่ใช่ตัวเลข (0-9) 1 ตัว
- **\w** สำหรับค้นหาตัวอักษร alphanumeric (a-z, A-Z, 0-9) 1 ตัว
- **\W** สำหรับค้นหาตัวอักษรที่ไม่ใช่ alphanumeric (a-z, A-Z, 0-9) 1 ตัว
- **\s** สำหรับค้นหา whitespace (ได้แก่ ช่องว่าง tab และอักขระขึ้นบรรทัดใหม่) ใด ๆ ก็ได้
- **[]** สำหรับค้นหาตัวอักษรตัวใดตัวหนึ่ง (ตัวเดียว) ในวงเล็บ เช่น **[abc]**
  - ✓ ซึ่งจะ Match กับ “a”, “b”, หรือ “c”
  - ✓ แต่ไม่ Match กับ “abc”, “ab”, “bc”, “aa”, “d”
- **[^]** สำหรับค้นหาตัวอักษรตัวใดตัวหนึ่ง (ตัวเดียว) ยกเว้นในวงเล็บ เช่น **[^abc]** หมายถึง Match กับสตริงความยาวหนึ่งตัวที่ไม่ใช่ “a”, “b”, “c”
  - ✓ ซึ่งจะ Match กับ “d”, “y”, หรือ “z”, ...
  - ✓ แต่ไม่ Match กับ “a”, “b”, “c”, “aa”, “dd”

- `|` สำหรับค้นหาอักขระก่อนหรือหลังเครื่องหมาย `|`
- `[A-Z]` สำหรับค้นหาอักษร 1 ตัวที่มีค่า Unicode ตั้งแต่ A-Z (A, B, C, ..., Z)
- `[a-z]` สำหรับค้นหาอักษร 1 ตัวที่มีค่า Unicode ตั้งแต่ a-z (a, b, c, ..., z)

ถ้าต้องการหาตัวอักษรซ้ำที่อยู่ข้างหน้าตามจำนวนที่กำหนด ทำได้โดยใช้สัญลักษณ์ ดังต่อไปนี้

- `?` สำหรับค้นหาอักขระ 0 หรือ 1 ตัว
- `+` สำหรับค้นหาอักขระ 1 ตัวขึ้นไป
- `*` สำหรับค้นหาอักขระ 0 ตัวหรือมากกว่า
- `{m}` สำหรับค้นหาอักขระ m ตัวเท่านั้น
- `{m, }` สำหรับค้นหาอักขระ m ตัวหรือมากกว่า
- `{, n}` สำหรับค้นหาอักขระไม่เกิน n ตัว
- `{m, n}` สำหรับค้นหาอักขระตั้งแต่ m ถึง n ตัว
- `()` สำหรับใช้จับกลุ่มตัวอักษรเพื่อกำหนดขอบเขตการค้นหา เช่น `(abc)+`
  - ✓ ซึ่งจะ Match กับ “abc”, “abcabc”, ...
  - ✓ แต่ไม่ Match กับ “a”, “ab”, “bc”, “aa”, “d”, ...

ตัวอย่างการใช้รูปแบบพิเศษในการหาสตริงดังกล่าวแสดงได้ดังนี้

```
R> str_view(c("grey", "gray"), "gr(e|a)y")
[1] │ <grey>
[2] │ <gray>

R> text <- "1888 is the longest year in Roman numerals: MDCCCCLXXXVIII"
R> str_view(text, "CC?")
[1] │ 1888 is the longest year in Roman numerals: MD<CC><CC>LXXXVIII

R> str_view(text, "CC+")
[1] │ 1888 is the longest year in Roman numerals: MD<CCCC>LXXXVIII

R> str_view(text, "C[LX]+")
[1] │ 1888 is the longest year in Roman numerals: MDCCC<CLXXX>VIII
```

มีเครื่องหมายพิเศษที่สามารถค้นหาเครื่องหมายต่าง ๆ หลายรูปแบบ ดังต่อไปนี้

- `[:punct:]` สำหรับค้นหาเครื่องหมายวรรคตอน
- `[:alpha:]` สำหรับค้นหาตัวอักษร
- `[:lower:]` สำหรับค้นหาอักษรตัวเล็ก
- `[:upper:]` สำหรับค้นหาอักษรตัวใหญ่
- `[:digit:]` สำหรับค้นหาตัวเลข
- `[:alnum:]` สำหรับค้นหาตัวอักษรและตัวเลข
- `[:cntrl:]` สำหรับค้นหาอักขระควบคุม เช่น อักขระขึ้นบรรทัดใหม่ “\n”

- `[:graph:]` สำหรับค้นหาตัวอักษร ตัวเลข และเครื่องหมายวรรคตอน
- `[:print:]` สำหรับค้นหาตัวอักษร ตัวเลข เครื่องหมายวรรคตอน และ whitespace (ได้แก่ ช่องว่าง tab และอักขระขึ้นบรรทัดใหม่)
- `[:blank:]` สำหรับค้นหาช่องว่าง และ tab

Regex สามารถค้นหาตัวอักษรบางคำ เช่น **colou?r** ที่เขียนในรูปแบบภาษาอังกฤษและอเมริกันได้ ตัวอย่าง Regex เพื่อตรวจจับสตริงที่ขึ้นต้นด้วย Hello หรือ hello ลงท้ายด้วย world หรือ World ตรงกลางจะเป็นอักษรอะไรก็ได้ เขียนได้ดังนี้

```
R> text <- c("Hello world", "hello.. world", "Hi.. Hello world", "Hello world...")
R> str_view(text, "^(Hello|hello).*(world|World)$")
[1] │ <Hello world>
[2] │ <hello.. world>
```

ตัวอย่าง Regex เพื่อตรวจจับ email อย่างง่าย เขียนได้ดังนี้

```
R> email <- c("student@sut.ac.th", "sales@amazon.com", "email#amazon.com")
R> str_view(email, "\\w+@\\w+\\.(ac.th|com)")
[1] │ <student@sut.ac.th>
[2] │ <sales@amazon.com>
```

18.7 การตรวจจับสตริง (Detect Matches)

การตรวจจับสตริง (Detect matches) ใช้คำสั่ง **str_detect()** และผลที่ได้จะคืนเวกเตอร์ตรรกะ (Logical vector) ที่มีจำนวนสมาชิกเท่ากับค่าที่ต้องการตรวจสอบ ดังนี้

```
R> x <- c("apple", "banana", "pear")
R> str_detect(x, "e")
[1]  TRUE FALSE  TRUE
```

ถ้ามีการใช้ฟังก์ชันการคำนวณทางคณิตศาสตร์กับเวกเตอร์ตรรกะ ค่า **TRUE** จะเท่ากับ 1 และค่า **FALSE** จะเท่ากับ 0 ซึ่งจะมีประโยชน์ในการคำนวณค่าบางอย่าง เช่น

```
# words = sample character vectors for practicing string manipulations
R> str(words)
chr [1:980] "a" "able" "about" "absolute" "accept" "account" ...

# How many common words start with t?
R> sum(str_detect(words, "^t"))
[1] 65

# What proportion of common words end with a vowel?
R> mean(str_detect(words, "[aeiou]$"))
[1] 0.2765306
```

การตรวจจับสตริงที่มีเงื่อนไขซับซ้อนด้วยการรวม (Combine) คำสั่ง **str_detect()** หลายคำสั่งมักจะง่ายกว่าการเขียน Regex ที่ซับซ้อน ตัวอย่างเช่น การหาคำที่ไม่มีสระ ด้วยคำสั่งแรกจะเขียนง่ายกว่าคำสั่งที่สอง ซึ่งทั้งสองคำสั่งให้ผลลัพธ์ออกมาเหมือนกัน ดังนี้

```
# Find all words containing at least one vowel, and negate
R> no_vowels_1 <- !str_detect(words, "[aeiou]")

# Find all words consisting only of consonants (non-vowels)
R> no_vowels_2 <- str_detect(words, "^[^aeiou]+$")

R> identical(no_vowels_1, no_vowels_2)
[1] TRUE
```

สามารถใช้คำสั่ง `str_subset()` ในการเลือกข้อความที่ตรงกับรูปแบบที่กำหนด ได้ดังนี้

```
R> str_subset(words, "x$")
[1] "box" "sex" "six" "tax"

R> words[str_detect(words, "x$")]     # Equavalent to the above command
[1] "box" "sex" "six" "tax"
```

โดยปกติข้อความที่ต้องการตรวจจับมักจะเป็นข้อมูลอยู่ในเดต้าเฟรมหรือ tibble โดยใช้คำสั่ง `filter()` และระบุฟังก์ชัน `str_detect()` อยู่ภายในคำสั่งดังกล่าว ดังนี้

```
R> tbl <- tibble(
     word = words,
     item = seq_along(word)
   )
R> tbl |>
     filter(str_detect(words, "x$"))
# A tibble: 4 x 2
  word   item
  <chr> <int>
1 box     108
2 sex     747
3 six     772
4 tax     841
```

คำสั่ง `str_count()` เป็นการนับจำนวนข้อมูลที่ตรงกับรูปแบบที่กำหนด ดังนี้

```
R> x <- c("apple", "banana", "pear")
R> str_count(x, "a")
[1] 1 3 1

# On average, how many vowels per word?
R> mean(str_count(words, "[aeiou]"))
[1] 1.991837
```

โดยปกติมักจะใช้คำสั่ง `str_count()` ร่วมกับคำสั่ง `mutate()` ดังนี้

```
R> tbl |>
     mutate(
        vowels = str_count(word, "[aeiou]"),
        consonants = str_count(word, "[^aeiou]")
     )
# A tibble: 980 × 4
   word      item vowels consonants
   <chr>    <int>  <int>      <int>
 1 a            1      1          0
```

```
 2 able          2     2          2
 3 about         3     3          2
 4 absolute      4     4          4
 5 accept        5     2          4
 6 account       6     3          4
 7 achieve       7     4          3
 8 across        8     2          4
 9 act           9     1          2
10 active       10     3          3
# i 970 more rows
```

คำสั่งในการ Match จะไม่มีการซ้อนทับกัน (overlab) กัน ตัวอย่างเช่น คำว่า "abababa" จะ Match กับ "aba" แค่ 2 ครั้งโดยไม่มีการซ้อนทับกัน ดังนี้

```
> str_count("abababa", "aba")
[1] 2

> str_view_all("abababa", "aba")
[1] │ <aba>b<aba>
```

18.8 การดึงข้อมูลจากสตริง (Extract Matches)

ข้อมูลที่จะใช้ทดสอบการดึงข้อมูลจากสตริง (Extract matches) คือ ข้อมูล sentences ที่ติดตั้งมาพร้อมกับแพ็กเกจ **stringr** ดังนี้

```
# A collection of "Harvard sentences" used for standardised testing of voice
R> length(sentences)
[1] 720

R> head(sentences)
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."
[6] "The juice of lemons makes fine punch."
```

ถ้าต้องการหาประโยคที่ประกอบด้วยสีดังต่อไปนี้ red, orange, yellow, green, blue, และ purple จะทำการสร้างตัวแปรสำหรับเก็บรูปแบบการ Match ดังนี้

```
R> colors <- c("red", "orange", "yellow", "green", "blue", "purple")
R> color_match <- str_c(colors, collapse = "|")
R> color_match
[1] "red|orange|yellow|green|blue|purple"
```

ใช้คำสั่ง **str_subset()** ในการดึงประโยคที่มีสีที่กำหนดด้วยคำสั่งในบรรทัดที่หนึ่ง และใช้คำสั่ง **str_extract()** ในการดึงสีในแต่ละประโยค ได้ดังนี้

```
R> has_color <- str_subset(sentences, color_match)
R> matches <- str_extract(has_color, color_match)
R> head(matches)
[1] "blue" "blue" "red"  "red"  "red"  "blue"
```

คำสั่ง **str_extract()** จะดึงเฉพาะ Match ที่พบเป็นลำดับแรกเท่านั้น ดังนี้

```
R> more <- sentences[str_count(sentences, color_match) > 1]
R> str_view_all(more, color_match)
[1] │ It is hard to erase <blue> or <red> ink.
[2] │ The <green> light in the brown box flicke<red>.
[3] │ The sky in the west is tinged with <orange> <red>.

R> str_extract(more, color_match)
[1] "blue"   "green"  "orange"
```

ส่วนคำสั่ง **str_extract_all()** จะทำการ Match ทุกตัวที่พบ และคืนค่ากลับมาเป็นลิสต์ ดังนี้

```
R> str_extract_all(more, color_match)
[[1]]
[1] "blue" "red"

[[2]]
[1] "green" "red"

[[3]]
[1] "orange" "red"
```

คำสั่ง **str_extract_all()** ที่ระบุอากิวเมนต์ **simplify = TRUE** จะทำการ Match ทุกตัวที่พบ และคืนค่ากลับมาเป็นเมทริกซ์ ดังนี้

```
R> str_extract_all(more, color_match, simplify = TRUE)
     [,1]     [,2]
[1,] "blue"   "red"
[2,] "green"  "red"
[3,] "orange" "red"

R> x <- c("a", "a b", "a b c")
R> str_extract_all(x, "[a-z]", simplify = TRUE)
     [,1] [,2] [,3]
[1,] "a"  ""   ""
[2,] "a"  "b"  ""
[3,] "a"  "b"  "c"
```

18.9 การดึงกลุ่มข้อมูลจากสตริง (Grouped Matches)

หัวข้อ 18.6 กล่าวถึงพื้นฐาน Regex และการใช้สัญลักษณ์พิเศษต่าง ๆ หัวข้อนี้จะเป็นการประยุกต์ใช้ Regex ในการดึงกลุ่มข้อมูลจากสตริง ตัวอย่างการดึงคำนามจากประโยคด้วยเงื่อนไข Match คำที่ขึ้นต้นด้วย a หรือ the ตามด้วยตัวอักษรที่ไม่ใช่ช่องว่างตั้งแต่ 1 ตัวขึ้นไป ดังนี้

```
R> noun <- "(a|the) ([^ ]+)"
R> has_noun <- sentences |>
          str_subset(noun) |>
          head(10)

R> has_noun |>
     str_extract(noun)
```

```
[1] "the smooth" "the sheet"  "the depth"  "a chicken"  "the parked"
[6] "the sun"    "the huge"   "the ball"   "the woman"  "a helps"
```

คำสั่ง `str_extract()` คืนค่าเป็นเวกเตอร์ข้อความที่ Match ทั้งหมด แต่คำสั่ง `str_match()` คืนค่าเป็นเมทริกซ์ คอลัมน์แรกเป็นข้อความที่ Match ทั้งหมด ส่วนคอลัมน์ต่อไปที่เหลือเป็นข้อความแต่ละคำ ดังนี้

```
R> has_noun |>
     str_match(noun)
     [,1]         [,2]  [,3]
[1,] "the smooth" "the" "smooth"
[2,] "the sheet"  "the" "sheet"
[3,] "the depth"  "the" "depth"
[4,] "a chicken"  "a"   "chicken"
[5,] "the parked" "the" "parked"
[6,] "the sun"    "the" "sun"
[7,] "the huge"   "the" "huge"
[8,] "the ball"   "the" "ball"
[9,] "the woman"  "the" "woman"
[10,] "a helps"    "a"   "helps"
```

18.10 การแทนที่สตริง (Replacing Matches)

คำสั่ง `str_replace()` เป็นการแทนที่ข้อความที่ Match ด้วยสตริงที่กำหนดเฉพาะส่วนที่ Match ตัวแรกเท่านั้น ส่วนคำสั่ง `str_replace_all()` เหมือนคำสั่งแรกแต่แทนที่ข้อความที่ Match ทั้งหมด ดังนี้

```
R> x <- c("apple", "pear", "banana")
R> str_replace(x, "[aeiou]", "-")
[1] "-pple"  "p-ar"   "b-nana"

R> str_replace_all(x, "[aeiou]", "-")
[1] "-ppl-"  "p--r"   "b-n-n-"
```

คำสั่ง `str_replace_all()` สามารถแทนที่ข้อความได้มากกว่า 1 เงื่อนไข เมื่อระบุอากิวเมนต์ด้วยเวกเตอร์ ดังนี้

```
R> x <- c("1 house", "2 cars", "3 people")
R> str_replace_all(x, c("1" = "one", "2" = "two", "3" = "three"))
[1] "one house"    "two cars"     "three people"
```

สามารถสลับตำแหน่งการแสดงข้อความได้ด้วยเครื่องหมาย `\\` ตามด้วยตัวเลขลำดับข้อความ ดังนี้

```
R> sentences |>
     str_replace("([^ ]+) ([^ ]+) ([^ ]+)", "\\1 \\3 \\2") |>
     head(5)
[1] "The canoe birch slid on the smooth planks."
[2] "Glue sheet the to the dark blue background."
[3] "It's to easy tell the depth of a well."
[4] "These a days chicken leg is a rare dish."
[5] "Rice often is served in round bowls."
```

18.11 การแยกสตริง (Splitting)

คำสั่ง `str_split()` ใช้ในการแยกสตริงออกเป็นส่วน ๆ โดยคืนค่ากลับมาเป็นลิสต์ ดังนี้

```
R> sentences |>
     head(2) |>
     str_split(" ")

[[1]]
[1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"
[8] "planks."

[[2]]
[1] "Glue"        "the"         "sheet"       "to"          "the"
[6] "dark"        "blue"        "background."
```

ถ้าต้องการข้อมูลที่ส่งกลับมาเป็นเวกเตอร์ สามารถเลือกสมาชิกที่ต้องการออกมา ได้ดังนี้

```
R> sentences |>
     str_split(" ") |>
     _[[1]]
[1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"
[8] "planks."
```

ถ้าต้องการข้อมูลที่ส่งกลับมาเป็นเมทริกซ์ ให้ระบุอากิวเมนต์ `simplify = TRUE` ดังนี้

```
R> sentences |>
     head(2) |>
     str_split(" ", simplify = TRUE)
     [,1]   [,2]    [,3]    [,4]   [,5]  [,6]   [,7]     [,8]
[1,] "The"  "birch" "canoe" "slid" "on"  "the"  "smooth" "planks."
[2,] "Glue" "the"   "sheet" "to"   "the" "dark" "blue"   "background."
```

ถ้าต้องการข้อมูลที่ส่งกลับมาตามจำนวนที่ต้องการ ให้ระบุอากิวเมนต์ `n` ดังนี้

```
R> fields <- c("Name: Petch", "Country: TH", "Age: 25")
R> fields |> str_split(": ", n = 2, simplify = TRUE)
     [,1]      [,2]
[1,] "Name"    "Petch"
[2,] "Country" "TH"
[3,] "Age"     "25"
```

สามารถแยกข้อความหรือตัวอักษรด้วยฟังก์ชัน `boundary()` โดยระบุอากิวเมนต์ `character, line, sentence` หรือ `word` ดังนี้

```
R> x <- "This is a sentence. This is another sentence."
R> str_view_all(x, boundary("word"))
[1] | <This> <is> <a> <sentence>. <This> <is> <another> <sentence>.

R> str_split(x, " ")[[1]]
[1] "This"    "is"    "a"    "sentence." "This"    "is"    "another"   "sentence."

R> str_split(x, boundary("word"))[[1]]
[1] "This"    "is"    "a"    "sentence" "This"    "is"    "another"   "sentence"
```

18.12 การหาตำแหน่งสตริง (Find Matches)

คำสั่ง `str_locate()` ใช้ในการหาตำแหน่งสตริงที่ Match โดยการคืนค่ากลับมาเป็นเมทริกซ์แสดงตำแหน่งเริ่มต้น และตำแหน่งสุดท้าย ดังนี้

```
R> x <- c("apple", "pear", "banana")
R> str_locate(x, "[aeiou]")
     start end
[1,]     1   1
[2,]     2   2
[3,]     2   2

R> str_locate(x, "na")
     start end
[1,]    NA  NA
[2,]    NA  NA
[3,]     3   4
```

คำสั่ง `str_locate_all()` ใช้ในการหาตำแหน่งสตริงที่ต้องการ Match โดยการคืนค่ากลับมาเป็นลิสต์แสดงตำแหน่งเริ่มต้น และตำแหน่งสุดท้าย ดังนี้

```
R> str_locate_all(x, "na")
[[1]]
     start end

[[2]]
     start end

[[3]]
     start end
[1,]     3   4
[2,]     5   6
```

18.13 การจัดการสตริงรูปแบบอื่น (Other Types of Pattern)

การใช้รูปแบบสตริงในการค้นหาด้วยคำสั่งต่าง ๆ ที่กล่าวมาในหัวข้อก่อนหน้านี้ R จะทำการเรียกฟังก์ชัน `regex()` โดยอัตโนมัติ ดังนี้

```
R> fruit <- c("apple", "orange", "banana")
# The regular call:
R> str_view(fruit, "nana")
[3] │ ba<nana>

# Is shorthand for
R> str_view(fruit, regex("nana"))
[3] │ ba<nana>
```

ผู้ใช้สามารถระบุอากิวเมนต์ในการควบคุมการค้นหาได้หลายแบบ ดังนี้

- `ignore_case = TRUE` สำหรับการ Match ทั้งอักษรตัวพิมพ์เล็กและอักษรตัวพิมพ์ใหญ่ ดังนี้

```
R> bananas <- c("banana", "Banana", "BANANA")
R> str_view_all(bananas, "banana")
[1] │ <banana>
[2] │ Banana
[3] │ BANANA
R> str_view_all(bananas, regex("banana", ignore_case = TRUE))
[1] │ <banana>
[2] │ <Banana>
[3] │ <BANANA>
```

- `multiline = TRUE` สำหรับการ Match ด้วย ^ และ $ สำหรับจุดเริ่มต้นและจุดสิ้นสุดแต่ละแถว (ลงท้ายด้วย \n) แทนที่จะ Match จุดเริ่มต้นและจุดสิ้นสุดของทั้งสตริง ดังนี้

```
R> x <- "Line 1\nLine 2\nLine 3"
R> str_extract_all(x, "^Line")[[1]]
[1] "Line"

R> str_extract_all(x, regex("^Line", multiline = TRUE))[[1]]
[1] "Line" "Line" "Line"
```

- `comments = TRUE` สำหรับการเพิ่ม comments ในสตริงรูปแบบการค้นหา ดังนี้

```
R> phone <- regex("
     \\(?      # optional opening parens
     (\\d{3})  # area code
     [)- ]?    # optional closing parens, dash, or space
     (\\d{3})  # another three numbers
     [ -]?     # optional space or dash
     (\\d{3})  # three more numbers
     ", comments = TRUE)
R> str_match("081-234-5678", phone)
     [,1]          [,2]  [,3]  [,4]
[1,] "081-234-567" "081" "234" "567"
R> str_match("(081)234-5678", phone)
     [,1]           [,2]  [,3]  [,4]
[1,] "(081)234-567" "081" "234" "567"
```

- `dotall = TRUE` เป็นการอนุญาติให้ใช้จุด . ในการ Match ได้ทุกอักขระรวมทั้ง \n

นอกจากนี้ยังมีคำสั่งใน base R ที่ใช้ Regex ได้แก่

- `apropos()` ใช้ในการค้นหาวัตถุทั้งหมดใน Global environment

```
R> apropos("replace")
[1] "%+replace%"                    ".rs.registerReplaceHook"
[3] ".rs.replaceBinding"            ".rs.rpc.replace_comment_header"
[5] "replace"                       "replace_na"
[7] "setReplaceMethod"              "str_replace"
[9] "str_replace_all"               "str_replace_na"
[11] "theme_replace"
```

- **dir()** ใช้ในการแสดงไฟล์ทั้งหมดใน directory ที่กำหนด ตัวอย่างการแสดงไฟล์ R Markdown โดยระบุนามสกุล .Rmd ดังนี้

```
R> head(dir(pattern = "\\.Rmd$"))
[1] "Network Preparation.Rmd"    "Matrix Preparation.Rmd"
[2] "Model Specification.Rmd"
```

18.14 สรุปท้ายบท

บทนี้เป็นการแนะนำการใช้แพ็กเกจ **stringr** ซึ่งเป็นแพ็กเกจที่นิยมใช้ในการจัดการสตริงมากที่สุดตัวหนึ่ง โดยกล่าวถึงการสร้างสตริง การหาความยาวสตริง การเชื่อมสตริง การดึงข้อมูลบางส่วนของสตริง (Subsetting strings) การเรียงข้อมูลสตริง การแปลงสตริงเป็นตัวใหญ่และตัวเล็ก การจัดการสตริงด้วย Regular expression การตรวจจับสตริง (Detect matches) การดึงข้อมูลจากสตริง (Extract matches) การดึงกลุ่มข้อมูลจากสตริง (Grouped matches) การแทนที่สตริง (Replacing matches) การแยกสตริง (Splitting) การหาตำแหน่งสตริง (Find matches) และการจัดการสตริงรูปแบบอื่น (Other types of pattern) โดยใช้ **regex()**

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
str_length()	คำสั่งในการหาความยาวสตริง
str_c()	คำสั่งในการการเชื่อมสตริง
str_replace_na()	คำสั่งในการแสดงค่า NA ด้วยเครื่องหมาย “NA”
str_sub()	คำสั่งในการดึงข้อมูลบางส่วนของสตริง
str_sort()	คำสั่งในการเรียงข้อมูลสตริง
str_to_lower()	คำสั่งในการแปลงสตริงเป็นอักษรตัวเล็ก
str_to_upper()	คำสั่งในการแปลงสตริงเป็นอักษรตัวใหญ่
str_to_title()	คำสั่งในการแปลงแต่ละคำให้ขึ้นต้นด้วยอักษรตัวใหญ่
str_view()	คำสั่งในการแสดงผลการ Match ตัวแรกที่พบ
str_view_all()	คำสั่งในการแสดงผลการ Match ทุกตัว
str_detect()	คำสั่งในการตรวจจับสตริง
str_subset()	คำสั่งในการเลือกสตริงที่ตรงกับรูปแบบที่ Match
str_count()	คำสั่งในการนับจำนวนสตริงที่ตรงกับรูปแบบที่ Match
str_extract()	คำสั่งในการดึงสตริงที่ Match ตัวแรกออกมา
str_extract_all()	คำสั่งในการดึงสตริงที่ Match ทุกตัวออกมา
str_replace()	คำสั่งในการแทนที่สตริงที่ Match ตัวแรกด้วยสตริงที่กำหนด
str_replace_all()	คำสั่งในการแทนที่สตริงที่ Match ทุกตัวด้วยสตริงที่กำหนด

คำสั่ง	คำอธิบาย
`str_split()`	คำสั่งในการแยกสตริง
`boundary()`	คำสั่งในการกำหนดขอบเขตสตริงที่ Match ด้วยอากิวเมนต์ character, line, sentence หรือ word
`str_locate()`	คำสั่งในการหาตำแหน่งสตริงที่ Match ตัวแรก
`str_locate_all()`	คำสั่งในการหาตำแหน่งสตริงที่ Match ทุกตัว
`regex()`	คำสั่งในการกำหนดรูปแบบการ Match
`apropos()`	คำสั่งในการค้นหาวัตถุทั้งหมดใน Global environment
`dir()`	คำสั่งในการแสดงไฟล์ทั้งหมดใน directory ที่กำหนด

18.15 แบบฝึกหัดท้ายบท

1. สร้างตัวแปรชื่อ text เป็นเวกเตอร์ มีค่า คือ “Learning R is fun and powerful!” ให้เขียนคำสั่งเพื่อจัดการสตริงดังต่อไปนี้

   1.1 นับจำนวนตัวอักษรใน text (รวมช่องว่าง)

   1.2 นับจำนวนตัวอักษรพิมพ์ใหญ่ทั้งหมดใน text

   1.3 นับจำนวนตัวอักษรพิมพ์เล็กทั้งหมดใน text

   1.4 ตรวจสอบว่าข้อความใน text มีคำว่า “powerful” หรือไม่

   1.5 แสดงตำแหน่งที่คำว่า “fun” ที่ปรากฏใน text

   1.6 แปลงข้อความเป็นตัวพิมพ์ใหญ่ทั้งหมด

   1.7 แปลงข้อความเป็นตัวพิมพ์เล็กทั้งหมด

2. สร้างตัวแปรชื่อ text เป็นเวกเตอร์ มีค่า คือ “apple”, “banana”, “cherry”, “apple” และ “banana” ให้เขียนคำสั่งเพื่อจัดการสตริงดังต่อไปนี้

   2.1 แทนที่คำว่า “banana” ด้วย “kiwi” ใน text

   2.2 นับจำนวนครั้งที่คำว่า “apple” ปรากฏใน text

   2.3 จัดเรียงคำใน text ตามลำดับตัวอักษร

   2.4 จัดเรียงคำใน text ตามลำดับตัวอักษรย้อนกลับ

   2.5 รวมคำทั้งหมดใน text ให้เป็นข้อความเดียว โดยใช้ “,” เป็นตัวเชื่อม และให้เก็บค่าไว้กับตัวแปร text_merge

   2.6 จากข้อ 2.5 ให้แยกข้อความใน text_merge ออกจากกันโดยใช้ “,” เป็นตัวแยก

   2.7 แปลงตัวอักษรตัวแรกในทุกค่าใน text ให้เป็นตัวพิมพ์ใหญ่

3. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้ และเก็บข้อมูลไว้ในตัวแปรชื่อ bicycle_order_tbl

ข้อมูลรายการจำหน่ายรถจักรยาน

order_date	order_id	quantity	unit_price	model
2024-07-07	1	2	6070	Jekyll Carbon 2
2024-07-07	1	1	5970	Trigger Carbon 2
2024-07-07	2	3	2770	Beast of the East 1
2024-07-07	2	1	5970	Trigger Carbon 2
2024-08-07	3	2	10660	Supersix Evo Hi-Mod Team

order_date	order_id	quantity	unit_price	model
2024-08-07	4	1	3200	Jekyll Carbon 4
2024-09-07	5	2	12790	Supersix Evo Black Inc.
2024-09-07	6	1	5330	Supersix Evo Hi-Mod Dura Ace 2
2024-09-07	7	1	1570	Synapse Disc 105
2024-09-07	7	1	4800	Synapse Carbon Disc Ultegra D12

3.1 ให้เขียนคำสั่งเพิ่มคอลัมน์ชื่อว่า supersix โดยมีค่าเป็น 1 เมื่อข้อมูลในคอลัมน์ model มีคำว่า “Supersix” และเป็น 0 เมื่อไม่มีคำดังกล่าว

3.2 จากข้อ 3.1 ให้เพิ่มคอลัมน์ชื่อว่า jekyll โดยมีค่าเป็น 1 เมื่อข้อมูลในคอลัมน์ model มีคำว่า “Jekyll” และเป็น 0 เมื่อไม่มีคำดังกล่าว

3.3 จากข้อ 3.2 ให้เพิ่มคอลัมน์ชื่อว่า synapse โดยมีค่าเป็น 1 เมื่อข้อมูลในคอลัมน์ model มีคำว่า “Synapse” และเป็น 0 เมื่อไม่มีคำดังกล่าว

3.4 จากข้อ 3.3 ให้เพิ่มคอลัมน์ชื่อว่า group_model โดยมีค่าเป็น supersix หรือ Jekyll หรือ synapse โดยพิจารณาจากข้อมูลในคอลัมน์ model ว่ามีคำดังกล่าวหรือไม่ หากไม่มีเข้าเงื่อนไขใดให้จัดเป็น other

3.5 จาก 3.4 ให้ทำการสรุปยอดขายรวมแบ่งตาม group_model และเรียงข้อมูลจากมากไปน้อย พร้อมเพิ่มคอลัมน์ ranking

3.6 จาก 3.5 ให้ทำการเขียนไฟล์ชื่อว่า summary_bike_order.csv

บทที่ 19
การจัดการแฟกเตอร์ (Factors) ด้วยแพ็กเกจ forcats

แฟกเตอร์มีประโยชน์อย่างมากในการจัดการกับตัวแปรที่ไม่ต่อเนื่อง/หรือตัวแปรที่สามารถแบ่งเป็นกลุ่มได้ (Categorical variables) ในบทนี้จะกล่าวถึงการจัดการแฟกเตอร์ (Factors) ด้วยแพ็กเกจ `forcats` ซึ่งมีรายละเอียดปลีกย่อยเพิ่มเติมนอกเหนือจากการใช้แฟกเตอร์ด้วยคำสั่งใน base **R**

RStudio ได้สร้างไฟล์สรุปการใช้งาน `forcats` ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมากสำหรับผู้ใช้ **R** (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว)

19.1 การสร้างแฟกเตอร์ (Factors)

ข้อมูลเวกเตอร์สตริงโดยทั่วไป เมื่อนำมาใช้เป็นตัวแปรแบ่งกลุ่ม (Categorical variables) จะพบปัญหา 2 ประการด้วยกันคือ (1) อาจจะเกิดการพิมพ์ผิดนอกเหนือจากข้อมูลแบ่งกลุ่มที่ควรจะเป็น เช่น ข้อมูลเดือนมีทั้งหมด 12 เดือน อาจจะป้อนข้อมูลผิด เช่น ตัวแปร x2 ต้องการพิมพ์คำว่า Jan แต่พิมพ์ผิดเป็น Jam (2) การเรียง (Sort) ไม่เป็นไปตามที่ต้องการ ดังนี้

```
R> x1 <- c("Dec", "Apr", "Jan", "Mar")
R> x2 <- c("Dec", "Apr", "Jam", "Mar")
R> sort(x1)
[1] "Apr" "Dec" "Jan" "Mar"
```

จากผลลัพธ์ข้างต้นจะเห็นว่า ข้อมูลถูกเรียงตามตัวอักษร แทนที่จะเรียงตามลำดับเดือนจาก Jan, Mar, Apr และ Dec ปัญหาดังกล่าวแก้ไขได้ด้วยการสร้างตัวแปรแฟกเตอร์ โดยระบุพารามิเตอร์ `levels` ในคำสั่ง `factor()` ดังนี้

```
R> month_levels <- c(
     "Jan", "Feb", "Mar", "Apr", "May", "Jun",
     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
   )
R> y1 <- factor(x1, levels = month_levels)
R> y1
[1] Dec Apr Jan Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

R> sort(y1)
[1] Jan Mar Apr Dec
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

จากผลลัพธ์ข้างต้นจะเห็นว่า ข้อมูลถูกเรียงตามเดือนที่ถูกต้อง ถ้ามีการป้อนข้อมูลผิด เช่น เดือน Jam แทนที่จะเป็นเดือน Jan ตัวแปรแฟกเตอร์จะทำการแปลงข้อมูลดังกล่าวให้เป็น NA โดยอัตโนมัติ ดังนี้

```
R> y2 <- factor(x2, levels = month_levels)
R> y2
[1] Dec  Apr  <NA> Mar
```

```
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

ถ้าต้องการให้แสดงข้อผิดพลาดออกมาด้วย ใช้คำสั่ง `parse_factor()` ดังนี้

```
R> y2 <- parse_factor(x2, levels = month_levels)
Warning: 1 parsing failure.
row col           expected actual
  3  -- value in level set    Jam
```

คำสั่ง `factor()` ถ้าไม่ระบุพารามิเตอร์ `levels` จะสร้าง levels ให้เองโดยเรียงลำดับตามตัวอักษร ดังนี้

```
R> factor(x1)
[1] Dec Apr Jan Mar
Levels: Apr Dec Jan Mar
```

ถ้าต้องการสร้าง Levels ให้เรียงลำดับเหมือนที่แสดงในข้อมูล ทำได้ 2 แบบด้วยคำสั่ง `unique()` และคำสั่ง `fct_inorder()` ดังนี้

```
R> f1 <- factor(x1, levels = unique(x1))
R> f1
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar

R> f2 <- x1 |> factor() |> fct_inorder()
R> f2
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar
```

ถ้าต้องการแสดง Levels ออกมาเป็นเวกเตอร์ใช้คำสั่ง `levels()` ดังนี้

```
R> levels(f2)
[1] "Dec" "Apr" "Jan" "Mar"
```

ข้อมูลที่ใช้ในการแสดงตัวอย่างต่อ ๆ ไป คือ ข้อมูลสำรวจทางด้านสังคมศาสตร์ (General social survey, GSS) ที่ติดตั้งมาพร้อมกับแพ็กเกจ `forcats` ดังนี้

```
R> gss_cat
# A tibble: 21,483 x 9
    year marital     age race  rincome   partyid    relig   denom   tvhours
   <int> <fct>     <int> <fct> <fct>     <fct>      <fct>   <fct>     <int>
 1  2000 Never ma~    26 White $8000 to~ Ind,near ~ Protes~ Southe~      12
 2  2000 Divorced     48 White $8000 to~ Not str r~ Protes~ Baptis~      NA
 ...
 9  2000 Married      44 White $25000 o~ Not str d~ Protes~ Other         0
10  2000 Married      47 White $25000 o~ Strong re~ Protes~ Southe~       3
# ... with 21,473 more rows
```

การแสดง levels ในข้อมูล tibble ทำได้หลายลักษณะ ได้แก่ (1) เข้าถึงตัวแปรที่ต้องการโดยตรง (2) ใช้คำสั่ง `count()` ช่วย (3) ตรวจสอบใน Global environment ดังตัวอย่างต่อไปนี้

```
R> head(gss_cat$race)
[1] White White White White White White
```

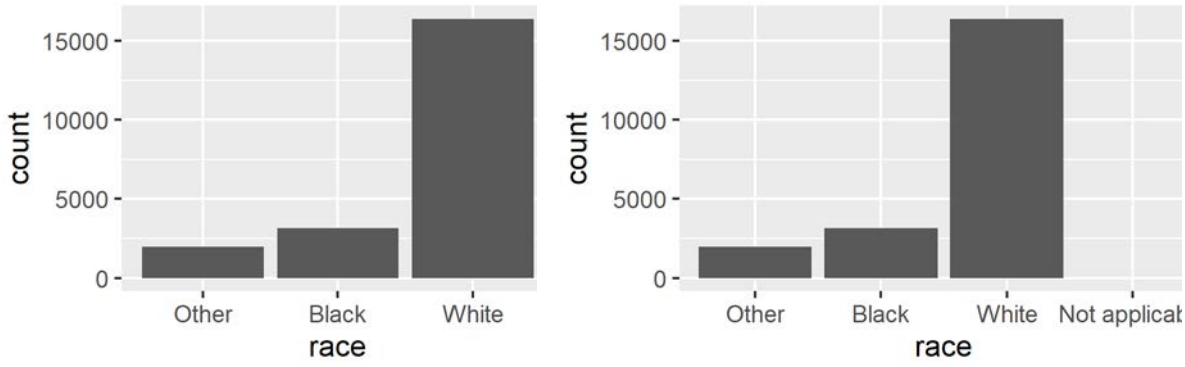
```
Levels: Other Black White Not applicable

R> gss_cat |>
     count(race)
# A tibble: 3 x 2
  race      n
  <fct> <int>
1 Other  1959
2 Black  3129
3 White 16395
```

แสดงข้อมูลในตัวแปร race เป็นกราฟแท่ง ได้ดังนี้

```
R> ggplot(gss_cat, aes(race)) +
     geom_bar()

R> ggplot(gss_cat, aes(race)) +
     geom_bar() +
     scale_x_discrete(drop = FALSE)
```



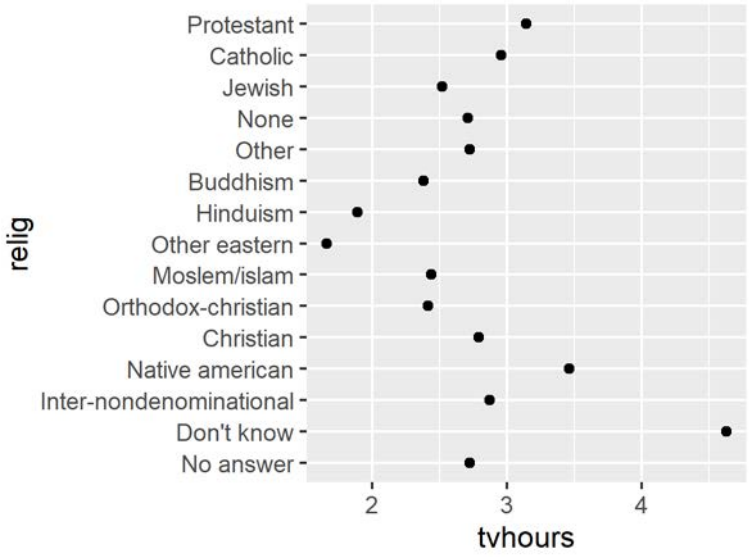
รูปที่ 19-1 Bar charts with some levels and all levels

รูปแรกเป็นกราฟแท่งแกน X แสดง levels ที่มีข้อมูล ได้แก่ Other, Black และ White รูปที่สองเป็นข้อมูลเดียวกันแต่แกน X แสดง levels ที่มีอยู่ทั้งหมดแม้ว่าจะไม่มีข้อมูลก็ตาม จะเห็นว่าแสดง levels ชื่อ Not applicable ออกมาด้วย

19.2 การเปลี่ยนลำดับ Levels (Modifying Level Order)

สมมติว่าต้องการสร้างกราฟข้อมูล GSS ข้างต้น โดยแสดงจำนวนชั่วโมงเฉลี่ยที่ใช้ในการดูโทรทัศน์แบ่งตามศาสนาที่นับถือ ดังนี้

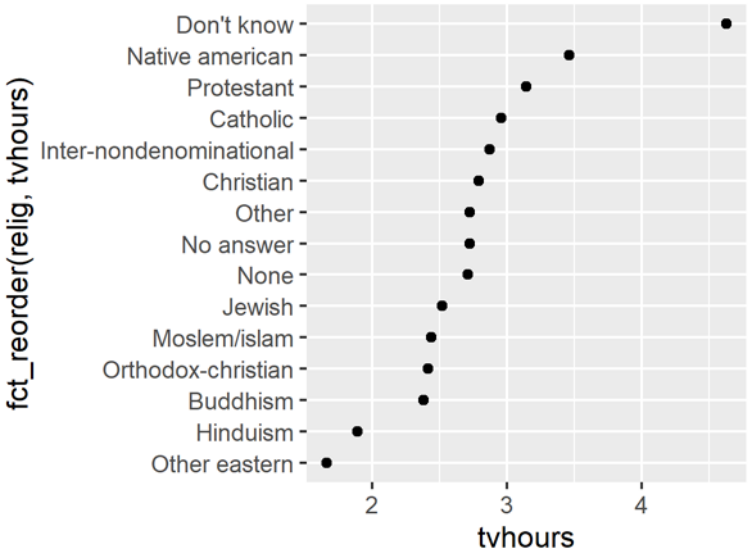
```
R> religion <- gss_cat |>
     group_by(relig) |>
     summarize(
       age = mean(age, na.rm = TRUE),
       tvhours = mean(tvhours, na.rm = TRUE),
       n = n()
     )
R> ggplot(religion, aes(tvhours, relig)) +
     geom_point()
```



รูปที่ 19-2 Scatter plots

จากรูปจะเห็นว่าการแปลผลจากกราฟทำได้ยาก เนื่องจากไม่ได้เรียงลำดับตามจำนวนชั่วโมงที่ใช้ดูโทรทัศน์ ปัญหาดังกล่าวแก้ไขด้วยการปรับลำดับของแฟกเตอร์ตามจำนวนชั่วโมงที่ใช้ดูโทรทัศน์ ด้วยฟังก์ชัน `fct_reorder()` ดังนี้

```
R> ggplot(religion, aes(tvhours, fct_reorder(relig, tvhours))) +
     geom_point()
```



รูปที่ 19-3 Scatter plots with reorder levels

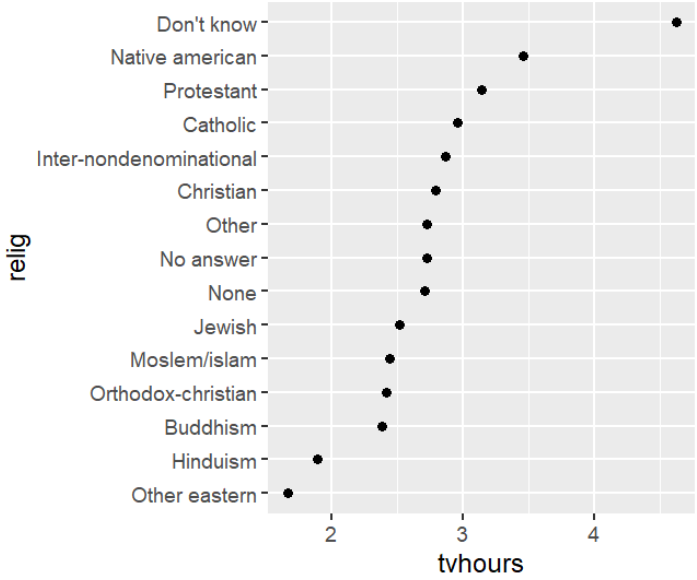
รูปแบบการใช้ฟังก์ชัน `fct_reorder()` ประกอบด้วยพารามิเตอร์ `.f` คือแฟกเตอร์ที่ต้องการเรียงลำดับ `.x` คือตัวแปรที่ใช้ในการเรียงแฟกเตอร์ `.fun` คือค่าที่ใช้ในการเรียงลำดับ ค่าโดยปริยาย

(Default) คือ Median และ **.desc** ค่าโดยปริยาย (Default) มีค่าเป็น **FALSE** หมายถึงเรียงจากน้อยไปมาก ถ้ามีค่าเป็น **TRUE** หมายถึงเรียงจากมากไปน้อย ดังนี้

```
fct_reorder(.f, .x, .fun = median, ..., .desc = FALSE)
```

การเขียนคำสั่งที่ซับซ้อน แนะนำให้เขียนด้วยการ Chain คำสั่งต่อเนื่องกัน ด้วยคำสั่งต่าง ๆ ในแพ็กเกจ **tidyverse** ซึ่งจะทำให้การเขียนสะดวก เข้าใจง่าย ตรวจสอบ และแก้ไขข้อผิดพลาดได้ง่ายขึ้น ดังนี้

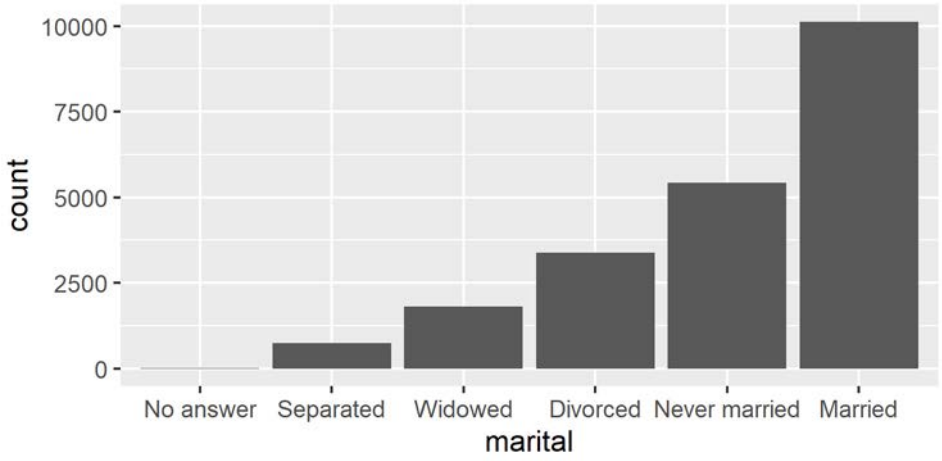
```
R> religion |>
     mutate(relig = fct_reorder(relig, tvhours)) |>
     ggplot(aes(tvhours, relig)) +
     geom_point()
```



รูปที่ 19-4 Scatter plots with reorder levels and using pipe

มีอากิวเมนต์ในการเปลี่ยนลำดับของแฟกเตอร์อีกหลายแบบ เช่น **fct_infreq()** สำหรับเรียงลำดับตามความถี่ของข้อมูล **fct_rev()** สำหรับการสลับการเรียงลำดับ ดังนี้

```
R> gss_cat |>
     mutate(marital = marital |> fct_infreq() |> fct_rev()) |>
     ggplot(aes(marital)) +
     geom_bar()
```



รูปที่ 19-5 Bar charts by number of observation for each level

นอกจากนี้ยังมีคำสั่งในการเปลี่ยนลำดับ Levels อีกหลายลักษณะ ศึกษาเพิ่มเติมได้จาก Help ของ **R** หรือพิมพ์ `?fct_` จะมี dropdown list มาให้เลือก

19.3 การปรับเปลี่ยน Levels (Modifying Levels)

การปรับเปลี่ยน Levels ทำให้การทำงานกับข้อมูลมีความยืดหยุ่นสูง ข้อมูลที่จะแสดงตัวอย่างเป็นข้อมูล GSS แสดงความนิยมในพรรคการเมือง (Political party) ดังนี้

```
R> gss_cat |>
     count(partyid) |>
     mutate(levels = as.numeric(partyid))
# A tibble: 10 × 3
   partyid               n levels
   <fct>             <int> <dbl>
 1 No answer           154     1
 2 Don't know            1     2
 3 Other party         393     3
 4 Strong republican  2314     4
 5 Not str republican 3032     5
 6 Ind,near rep       1791     6
 7 Independent        4119     7
 8 Ind,near dem       2499     8
 9 Not str democrat   3690     9
10 Strong democrat    3490    10
```

คำสั่ง `fct_relevel()` เป็นการเปลี่ยนระดับ Levels โดยการกำหนดให้ Levels ที่ต้องการปรับเปลี่ยนไปเป็น Levels ที่เท่าใดใน factor ตัวอย่างเช่น ต้องการเปลี่ยนจาก “Other party” จาก Levels ลำดับที่ 3 เป็น Levels ลำดับที่ 1 ดังนี้

```
R> gss_cat |>
     count(partyid) |>
     mutate(partyid = partyid |> fct_relevel("Other party", after = 0)) |>
     mutate(levels = as.numeric(partyid)) |>
     arrange(levels)
```

```
# A tibble: 10 × 3
   partyid               n levels
   <fct>             <int>  <dbl>
 1 Other party         393      1
 2 No answer           154      2
 3 Don't know            1      3
 4 Strong republican  2314      4
 5 Not str republican 3032      5
 6 Ind,near rep       1791      6
 7 Independent        4119      7
 8 Ind,near dem       2499      8
 9 Not str democrat   3690      9
10 Strong democrat    3490     10
```

คำสั่ง `fct_recode()` เป็นการเปลี่ยนชื่อ Levels โดยการกำหนดให้ ชื่อใหม่ = ชื่อเก่า ดังนี้

```
R> gss_cat |>
     mutate(partyid = fct_recode(partyid,
                                 "Republican, strong" = "Strong republican",
                                 "Republican, weak" = "Not str republican",
                                "Independent, near rep" = "Ind,near rep",
                                 "Independent, near dem" = "Ind,near dem",
                                "Democrat, weak" = "Not str democrat",
                                 "Democrat, strong" = "Strong democrat"
     )) |>
     count(partyid)

# A tibble: 10 x 2
   partyid                   n
   <fct>                 <int>
 1 No answer               154
 2 Don't know                1
 3 Other party             393
 4 Republican, strong     2314
 5 Republican, weak       3032
 6 Independent, near rep  1791
 7 Independent            4119
 8 Independent, near dem  2499
 9 Democrat, weak         3690
10 Democrat, strong       3490
```

คำสั่ง `fct_recode()` สามารถรวม Levels เข้าด้วยกันเป็น Levels ใหม่ได้โดยการกำหนด ชื่อใหม่ที่ใช้ร่วมกัน = ชื่อเก่า ดังนี้

```
R> gss_cat |>
     mutate(partyid = fct_recode(partyid,
                                 "Republican, strong" = "Strong republican",
                                 "Republican, weak" = "Not str republican",
                                 "Independent, near rep" = "Ind,near rep",
                                 "Independent, near dem" = "Ind,near dem",
                                "Democrat, weak" = "Not str democrat",
                                 "Democrat, strong" = "Strong democrat",
                                 "Other" = "No answer",
                                 "Other" = "Don't know",
```

```
                                  "Other" = "Other party"
     )) |>
     count(partyid)

# A tibble: 8 x 2
  partyid                   n
  <fct>                 <int>
1 Other                   548
2 Republican, strong     2314
3 Republican, weak       3032
4 Independent, near rep  1791
5 Independent            4119
6 Independent, near dem  2499
7 Democrat, weak         3690
8 Democrat, strong       3490
```

จากผลที่แสดงด้านบนจะเห็นว่ามี Levels ชื่อ Other เพิ่มเข้ามาแทน Levels เดิม ซึ่งเกิดจากการรวม Other ของ Levels เดิมซึ่งได้แก่ No answer, Don't know และ Other party ส่งผลให้มีจำนวนข้อมูล n = 548

การรวม Levels เข้าด้วยกันยังสามารถใช้คำสั่ง `fct_collapse()` โดยการกำหนด Levels ใหม่ = เวกเตอร์ของ Levels เก่า ดังนี้

```
R> gss_cat |>
     mutate(partyid = fct_collapse(partyid,
                                   other = c("No answer", "Don't know",
                                             "Other party"),
                                   rep = c("Strong republican",
                                           "Not str republican"),
                                   ind = c("Ind,near rep", "Independent",
                                           "Ind,near dem"),
                                   dem = c("Not str democrat", "Strong democrat")
     )) |>
     count(partyid)

# A tibble: 4 x 2
  partyid     n
  <fct>   <int>
1 other     548
2 rep      5346
3 ind      8409
4 dem      7180
```

มีคำสั่งในการรวม Levels เข้าด้วยกันอีกได้แก่ `fct_lump_min()` สำหรับรวม Levels ที่น้อยกว่าค่าที่กำหนดเข้าด้วยกัน, `fct_lump_prop()` สำหรับรวม Levels ที่น้อยกว่าสัดส่วนที่กำหนดเข้าด้วยกัน, `fct_lump_n()` สำหรับรวม Levels ตามจำนวน Levels ที่กำหนด, `fct_lump_lowfreq()` สำหรับรวม Levels ที่เหลือเข้าด้วยกันโดย Level ที่เหลือต้องมีความถี่ข้อมูลน้อยที่สุด ดังนี้

```
R> x <- factor(rep(LETTERS[1:9], times = c(40, 10, 5, 27, 1, 1, 1, 1, 1)))
R> x |> table()
 A  B  C  D  E  F  G  H  I
40 10  5 27  1  1  1  1  1

R> x |> fct_lump_min(5) |> table()
    A     B     C     D Other
   40    10     5    27     5

R> x |> fct_lump_prop(0.10) |> table()
    A     B     D Other
   40    10    27    10

R> x |> fct_lump_n(3) |> table()
    A     B     D Other
   40    10    27    10

R> x |> fct_lump_lowfreq() |> table()
    A     D Other
   40    27    20
```

นอกจากนี้ยังมีคำสั่งในการปรับเปลี่ยน Levels อีกหลายแบบ ศึกษาเพิ่มเติมได้จาก Help ของ **R** หรือพิมพ์ `?fct_` จะมี dropdown list มาให้เลือก

19.4 สรุปท้ายบท

บทนี้เป็นการแนะนำการใช้แพ็กเกจ `forcats` ในการจัดการแฟกเตอร์ ทำให้ผู้ใช้จัดการกับตัวแปรที่ไม่ต่อเนื่อง/หรือตัวแปรแบ่งกลุ่มได้ (Categorical variables) อย่างสะดวก และยืดหยุ่นมากขึ้น โดยกล่าวถึงการสร้างแฟกเตอร์ (Factors) การเปลี่ยนลำดับ Levels (Modifying Level Order) การปรับเปลี่ยน Levels (Modifying Levels)

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`factor()`	คำสั่งในการสร้างแฟกเตอร์
`parse_factor()`	คำสั่งในการสร้างแฟกเตอร์ พร้อมกับแสดงข้อผิดพลาดออกมาด้วย (ถ้ามี)
`levels()`	คำสั่งในการแสดง Levels
`fct_inorder()`	คำสั่งในการสร้าง Levels ให้เรียงลำดับเหมือนที่แสดงในข้อมูล
`fct_infreq()`	คำสั่งในการเปลี่ยนลำดับ Levels ให้เรียงลำดับตามความถี่ของแต่ละ Level
`fct_inseq()`	คำสั่งในการเปลี่ยนลำดับ Levels ให้เรียงลำดับตามค่าของแต่ละ Level
`fct_reorder()`	คำสั่งในการเปลี่ยนลำดับ Levels ให้เรียงลำดับตามตัวแปรที่กำหนด
`fct_rev()`	คำสั่งในการสลับการเรียงลำดับ

คำสั่ง	คำอธิบาย
`fct_relevel()`	คำสั่งในการปรับเปลี่ยน Levels
`fct_recode()`	คำสั่งในการเปลี่ยนชื่อ Levels และรวม Levels
`fct_collapse()`	คำสั่งในการรวม Levels
`fct_lump_min()`	คำสั่งในการรวม Levels ที่น้อยกว่าค่าที่กำหนดเข้าด้วยกัน
`fct_lump_prop()`	คำสั่งในการรวม Levels ที่น้อยกว่าสัดส่วนที่กำหนดเข้าด้วยกัน
`fct_lump_n()`	คำสั่งในการรวม Levels ตามจำนวน Levels ที่กำหนด
`fct_lump_lowfreq()`	คำสั่งในการรวม Levels ที่เหลือเข้าด้วยกัน โดย Level ที่เหลือต้องมีความถี่ข้อมูลน้อยที่สุด

19.5 แบบฝึกหัดท้ายบท

1. สร้างแฟกเตอร์ satisfaction ที่มีค่าคือ “low”, “medium”, “high”, “medium”, “high”, “low” และ “low”

   1.1 จัดเรียงลำดับ (Levels) ตามลำดับตัวอักษร

   1.2 จัดเรียงลำดับ (Levels) ตามความความถี่ของข้อมูลจากมากไปน้อย

   1.3 จัดเรียงลำดับ (Levels) เป็น high > medium > low

2. สร้างแฟกเตอร์ education_level ที่มีค่าคือ “Highschool”, “Bachelor”, “Master”, “Doctorate”, “Highschool”, “Bachelor” และ “Highschool”

   2.1 จัดเรียงลำดับ (Levels) ตามความความถี่ของข้อมูลจากมากไปน้อย

   2.2 ลดระดับ (Levels) ที่มีจำนวนปรากฏน้อยกว่า 2 เป็น “Other”

   2.3 เปลี่ยนระดับ “Highschool” เป็น “Low”, “Bachelor” เป็น “medium”, และ “Master” หรือ “Doctorate” เป็น "high"

3. สร้าง tibble ตามข้อมูลในตารางต่อไปนี้ และเก็บข้อมูลไว้ในตัวแปรชื่อ car_selling_tbl

ข้อมูลยอดขายรถยนต์

id	brand	model	body	color	sellingprice
001	Kia	Sorento	SUV	white	21500
002	Kia	Sorento	SUV	white	21500
003	BMW	3 Series	Sedan	gray	30000
004	Volvo	S60	Sedan	white	27750
005	BMW	6 Series Gran Coupe	Sedan	gray	67000
006	Nissan	Altima	Sedan	gray	10900
007	BMW	M5	Sedan	black	65000
008	Chevrolet	Cruze	Sedan	black	9800
009	Audi	A4	Sedan	white	32250
010	Chevrolet	Camaro	Convertible	red	17500
011	Audi	A6	Sedan	black	49750
012	Kia	Optima	Sedan	red	17700
013	Ford	Fusion	Sedan	white	12000
014	Kia	Sorento	SUV	silver	21500
015	Chevrolet	Cruze	Sedan	blue	10600
016	Nissan	Altima	Sedan	black	14100

id	brand	model	body	color	sellingprice
017	Hyundai	Sonata	Sedan	red	4200
018	Audi	Q5	SUV	white	40000
019	Chevrolet	Camaro	Coupe	black	17000
020	BMW	6 Series	Convertible	black	67200

3.1 สรุปข้อมูลยอดขายรวมตามประเภทของ brand

3.2 คอลัมน์ brand ให้เป็นแฟกเตอร์และจัดเรียงลำดับ (Levels) ตามยอดขายรวมมากไปน้อย

3.3 สร้างกราฟแท่งแสดงยอดขายรวมของแต่ละ brand โดยให้กราฟแท่งเรียงลำดับตามยอดขายรวมจากยอดขายมากไปน้อย

3.4 สรุปข้อมูลยอดขายรวมตามประเภทของ body

3.5 คอลัมน์ body ให้เป็นแฟกเตอร์และจัดเรียงลำดับ (Levels) ตามยอดขายรวมมากไปน้อย

3.6 สร้างกราฟแท่งแสดงยอดขายรวมของแต่ละ body โดยให้กราฟแท่งเรียงลำดับตามยอดขายรวมจากยอดขายมากไปน้อย

บทที่ 20
การจัดการวันและเวลาด้วยแพ็กเกจ lubridate

หัวข้อนี้จะกล่าวถึงการจัดการวันและเวลาด้วยแพ็กเกจ **lubridate** ชนิดข้อมูลที่ใช้ใน **lubridate** ประกอบด้วย (1) date ใน tibble แสดงในรูปแบบ <*date*> (2) time ใน tibble แสดงในรูปแบบ <*time*> และ (1) date-time ใน tibble แสดงในรูปแบบ <*dttm*>

การดึงข้อมูลวัน และเวลาปัจจุบัน ใช้คำสั่ง **today()** และ **now()** ตามลำดับ ดังนี้

```
R> today()
[1] "2021-10-12"

R> now()
[1] "2021-10-12 21:21:02 +07"
```

20.1 การสร้างข้อมูลวันและเวลา

การสร้างข้อมูลวันและเวลาทำได้ 3 วิธี ได้แก่ (1) สร้างจากสตริง (2) สร้างจากแต่ละองค์ประกอบของวันและเวลา (3) แปลงจากชนิดข้อมูลวัน/เวลาหรือตัวเลข

20.1.1 การสร้างข้อมูลวันและเวลาจากสตริง

ข้อมูลวัน-เดือน-ปี สร้างได้จากคำสั่ง **ymd() mdy()** และ **dmy()** โดย “y” แทนปี “m” แทนเดือน “d” แทนวัน ได้ดังนี้

```
R> library(tidyverse)
R> ymd("2021-01-31")
[1] "2021-01-31"

R> mdy("January 31st, 2021")
[1] "2021-01-31"

R> dmy("31-Jan-2021")
[1] "2021-01-31"
```

รูปแบบสตริงที่จะสร้างข้อมูลวัน-เดือน-ปี อาจจะติดกัน หรือมีตัวอักษรคั่นระหว่างวัน-เดือน-ปีได้หลายลักษณะ ดังนี้

```
R> x <- c(20210101, "2021-01-02", "2021 01 03", "2021/1/4",
          "2021-1, 5", "Created on 2021 1 6", "202101 !!! 07")
R> ymd(x)
[1] "2021-01-01" "2021-01-02" "2021-01-03" "2021-01-04"
[5] "2021-01-05" "2021-01-06" "2021-01-07"
```

การสร้างวัน-เดือน-ปีและเวลา ใช้คำสั่งก่อนหน้านี้ตามด้วยเครื่องหมาย (_) และตามด้วย “h” แทนชั่วโมง “m” แทนนาที “s” แทนวินาที ดังนี้

```
R> ymd_hms("2021-01-31 15:30:59")
[1] "2021-01-31 15:30:59 UTC"

R> dmy_hm("31/01/2021 08:05")
[1] "2021-01-31 08:05:00 UTC"
```

20.1.2 การสร้างข้อมูลวันและเวลาจากแต่ละองค์ประกอบ

บางครั้งข้อมูลวัน-เวลาถูกเก็บเป็นตัวเลขในแต่ละคอลัมน์ ดังนี้

```
R> require(nycflights13)
R> flights |>
       select(year, month, day, hour, minute)
# A tibble: 336,776 x 5
    year month   day  hour minute
   <int> <int> <int> <dbl>  <dbl>
 1  2013     1     1     5     15
 2  2013     1     1     5     29
 3  2013     1     1     5     40
 4  2013     1     1     5     45
 5  2013     1     1     6      0
 6  2013     1     1     5     58
 7  2013     1     1     6      0
 8  2013     1     1     6      0
 9  2013     1     1     6      0
10  2013     1     1     6      0
# ... with 336,766 more rows
```

สามารถสร้างข้อมูลวัน-เวลาจากแต่ละคอลัมน์ ได้ด้วยคำสั่ง **make_date()** สำหรับสร้างข้อมูลวัน และ **make_datetime()** สำหรับสร้างข้อมูลวัน-เวลา ดังนี้

```
R> flights |>
     select(year, month, day, hour, minute) |>
     mutate(departure = make_datetime(year, month, day, hour, minute))
# A tibble: 336,776 x 6
    year month   day  hour minute departure
   <int> <int> <int> <dbl>  <dbl> <dttm>
 1  2013     1     1     5     15 2013-01-01 05:15:00
 2  2013     1     1     5     29 2013-01-01 05:29:00
 3  2013     1     1     5     40 2013-01-01 05:40:00
 4  2013     1     1     5     45 2013-01-01 05:45:00
 5  2013     1     1     6      0 2013-01-01 06:00:00
 6  2013     1     1     5     58 2013-01-01 05:58:00
 7  2013     1     1     6      0 2013-01-01 06:00:00
 8  2013     1     1     6      0 2013-01-01 06:00:00
 9  2013     1     1     6      0 2013-01-01 06:00:00
10  2013     1     1     6      0 2013-01-01 06:00:00
# ... with 336,766 more rows
```

20.1.3 การสร้างข้อมูลวัน-เวลาจากการแปลงข้อมูลเวลา/วันหรือตัวเลข

การแปลงจากรูปแบบวัน-เวลา (data-time) ไปเป็นวัน (date) ใช้คำสั่ง `as_date()` ในทางกลับกัน การแปลงจากรูปแบบวัน (data) ไปเป็นวัน-เวลา (date-time) ใช้คำสั่ง `as_datetime()` ดังนี้

```
R> as_date(now())
[1] "2021-10-12"

R> as_datetime(today())
[1] "2021-10-12 UTC"
```

20.2 องค์ประกอบของวันและเวลา

`lubridate` สามารถดึงองค์ประกอบแต่ละส่วนในข้อมูลวัน-เวลาออกมาเพื่อใช้ในการประมวลผลตามที่ต้องการ ด้วยคำสั่งดังนี้

- `year()` ปี
- `month()` เดือน
- `mday()` วันที่ของเดือน
- `yday()` วันที่ของปี
- `wday()` วันในสัปดาห์
- `hour()` ชั่วโมง
- `minute()` นาที
- `second()` วินาที

```
R> datetime <- ymd_hms("2021-10-12 12:34:56")
R> year(datetime)
[1] 2021

R> month(datetime)
[1] 10

R> mday(datetime)
[1] 12

R> yday(datetime)
[1] 285

R> wday(datetime)
[1] 3
```

คำสั่ง `month()` และ `wday()` สามารถแสดงเป็นชื่อเดือนหรือวันด้วย อากิวเมนต์ `label = TRUE` แสดงชื่อเต็มด้วย อากิวเมนต์ `abbr = FALSE` และแสดงเป็นชื่อภาษาไทยด้วยอากิวเมนต์ `locale = "Thai_Thailand.utf8"` ดังนี้

```
R> month(datetime, label = TRUE)
[1] Oct
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec

R> month(datetime, label = TRUE, abbr = FALSE, locale = "Thai_Thailand.utf8")
[1] ตุลาคม
12 Levels: มกราคม < กุมภาพันธ์ < มีนาคม < เมษายน < พฤษภาคม < มิถุนายน < ... < ธันวาคม

R> wday(datetime, label = TRUE, abbr = FALSE)
[1] Tuesday
7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

20.3 การกำหนดค่าวันและเวลา

ผู้ใช้สามารถกำหนดหรือเปลี่ยนแปลงข้อมูลวันและเวลาในตัวแปรที่เก็บข้อมูลวัน-เวลา ได้ดังนี้

```
R> datetime <- ymd_hms("2011-03-05 12:34:56")
R> year(datetime) <- 2021
R> datetime
[1] "2021-03-05 12:34:56 UTC"

R> month(datetime) <- 01
R> datetime
[1] "2021-01-05 12:34:56 UTC"

R> hour(datetime) <- hour(datetime) + 1
R> datetime
[1] "2021-01-05 13:34:56 UTC"
```

ผู้ใช้สามารถปรับแก้ (update) วัน-เวลาได้ด้วยคำสั่ง **update()** ดังนี้

```
R> update(datetime, year = 2022, month = 2, mday = 2, hour = 2)
[1] "2022-02-02 02:34:56 UTC"
```

20.4 ช่วงเวลา (Time Spans)

หัวข้อนี้จะอธิบายถึงการคำนวณเชิงตัวเลข (Arithmetic) สำหรับวัน-เวลา ได้แก่ การบวก การลบ การหาร ซึ่งจำเป็นต้องเข้าใจคลาสต่าง ๆ ที่ใช้ในการแสดงช่วงเวลา ได้แก่

- Durations แสดงช่วงเวลาจริงแต่ละ time zone ในหน่วยวินาที
- Periods แสดงช่วงเวลาในหน่วยปกติที่ผู้ใช้รับรู้ เช่น วัน สัปดาห์ หรือเดือน
- Intervals แสดงเวลาเริ่มต้น และเวลาสิ้นสุด

20.4.1 การใช้คำสั่งในคลาส Duration

ใน **R** ผลต่างระหว่าง 2 ช่วงเวลาจะได้เป็นวัตถุชื่อว่า difftime ดังนี้

```
# How old are you?
R> age <- today() - ymd("2000-03-15")
R> age
Time difference of 7881 days
```

lubridate มีคลาส Duration ใช้แสดงตัวเลขจริงในหน่วยวินาที ซึ่งจะช่วยอำนวยความสะดวกในการคำนวณ การแปลงข้อมูลจากคลาส difftime เป็น duration ใช้คำสั่ง **as.duration()** ดังนี้

```
# Convert difftime to duration
R> as.duration(age)
[1] "680918400s (~21.58 years)"
```

คำสั่งในคลาส Duration จะแปลงตัวเลขให้เป็นวินาที ดังนี้

```
R> dseconds(15)
[1] "15s"

R> dminutes(10)
[1] "600s (~10 minutes)"

R> dhours(c(12, 24))
[1] "43200s (~12 hours)" "86400s (~1 days)"

R> ddays(0:5)
[1] "0s"                "86400s (~1 days)"  "172800s (~2 days)"
[4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"

R> dweeks(3)
[1] "1814400s (~3 weeks)"

R> dyears(1)
[1] "31557600s (~1 years)"
```

ค่าที่ได้จากคลาส Duration จะอยู่ในหน่วยวินาที ซึ่งสามารถทำการคำนวณ ได้ดังนี้

```
R> 3 * dyears(1)
[1] "94672800s (~3 years)"

R> dyears(1) + dweeks(10) + dhours(5)
[1] "37623600s (~1.19 years)"
```

สามารถบวก ลบ Duration จากคำสั่ง **today()** ได้ดังนี้

```
R> tomorrow <- today() + ddays(1)
R> tomorrow
[1] "2021-10-14"

R> last_year <- today() - dyears(1)
R> last_year
[1] "2020-10-12 18:00:00 UTC"
```

อย่างไรก็ตาม เนื่องจาก Duration แสดงช่วงเวลาจริงในหน่วยวินาที บางครั้งอาจจะแสดงผลที่ไม่คาดคิด ดังตัวอย่างต่อไปนี้

```
R> one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
R> one_pm
[1] "2016-03-12 13:00:00 EST"

R> one_pm + ddays(1)
[1] "2016-03-13 14:00:00 EDT"
```

ตัวอย่างข้างบนจะเห็นว่า เดิมวันที่ 12 มีนาคม 2016 เวลา 13:00 มีการบวกเพิ่ม 1 วันโดยใช้คำสั่งในคลาส Duration ซึ่งผลที่ได้ ควรจะเป็นวันที่ 13 มีนาคม 2016 เวลาเดียวกัน แต่กลับแสดงเป็นเวลา 14:00 ซึ่งสาเหตุมาจากมีการปรับเวลา Daylight saving time (DST) ในช่วงนั้นใน time zone ดังกล่าว ทำให้แสดงผลผิดจากที่คาดไว้ ในการแก้ปัญหาดังกล่าว **lubridate** ได้ใช้คลาส Period ที่จะกล่าวในหัวข้อต่อไป

20.4.2 การใช้คำสั่งในคลาส Period

lubridate มีคลาส Period แสดงช่วงเวลาในหน่วยปกติที่ผู้ใช้รับรู้ (ไม่ใช่เวลาตาม time zone) เช่น วัน สัปดาห์ หรือเดือน จากปัญหาการแสดงผลผิดจากที่คาดไว้ในหัวข้อก่อนหน้านี้ คลาส Period มีคำสั่งที่ช่วยแก้ปัญหาเหล่านี้ได้ ดังนี้

```
R> one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
R> one_pm
[1] "2016-03-12 13:00:00 EST"

R> one_pm + days(1)
[1] "2016-03-13 13:00:00 EDT"
```

คลาส Period มีคำสั่งในการสร้างช่วงเวลาหลายแบบ ได้แก่ `seconds()` `minutes()` `hours()` `days()` `months()` `weeks()` และ `years()` ดังนี้

```
R> seconds(30)
[1] "30S"

R> minutes(15)
[1] "15M 0S"

R> hours(c(12, 24))
[1] "12H 0M 0S" "24H 0M 0S"

R> days(7)
[1] "7d 0H 0M 0S"

R> months(1:6)
[1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S"
[5] "5m 0d 0H 0M 0S" "6m 0d 0H 0M 0S"

R> weeks(4)
[1] "28d 0H 0M 0S"

R> years(2)
[1] "2y 0m 0d 0H 0M 0S"
```

คลาส Period สามารถคำนวณได้ตามปกติ เช่น

```
R> 10 * (months(3) + days(1))
[1] "30m 10d 0H 0M 0S"

R> days(10) + hours(25) + minutes(3)
[1] "10d 25H 3M 0S"
```

เปรียบเทียบคลาส Period และคลาส Duration เมื่อรวมกับข้อมูลวัน ได้ดังนี้

```
# A leap year
R> ymd("2020-01-01") + dyears(1)
[1] "2020-12-31 06:00:00 UTC"

R> ymd("2020-01-01") + years(1)
[1] "2021-01-01"

# Daylight Savings Time
R> one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
R> one_pm + ddays(1)
[1] "2016-03-13 14:00:00 EDT"

R> one_pm + days(1)
[1] "2016-03-13 13:00:00 EDT"
```

20.4.3 การใช้ Interval

การคำนวณช่วงเวลาโดยการหารช่วงเวลา 1 ปีด้วยช่วงเวลา 1 วัน ด้วยคำสั่ง `year(1)/days(1)` จะได้ค่าเฉลี่ยเท่ากับ 365.25 เช่นเดียวกัน คำสั่ง `dyears(1)/ddays(365.25)` มีค่าเท่ากับ 1 ดังนี้

```
R> years(1) / days(1)
[1] 365.25

R> dyears(1) / ddays(365.25)
[1] 1
```

คลาส Interval ช่วยในการสร้างช่วงเวลาแบบ Interval คือ ระบุเวลาเริ่มต้น และเวลาสิ้นสุด ด้วยคำสั่ง `interval()` หรือใช้เครื่องหมาย `%--%` โดยมีรูปแบบคำสั่ง ดังนี้

```
interval(start = NULL, end = NULL)
start %--% end
```

ตัวอย่างการสร้าง Interval เริ่มจากวันปัจจุบันถึงปีหน้า แสดงได้ดังนี้

```
R> next_year <- today() + years(1)
R> next_year
[1] "2022-11-04"

R> interval1 <- interval(start = today(), end = next_year)
R> interval1
[1] 2021-11-04 UTC--2022-11-04 UTC

R> interval2 <- (today() %--% next_year)
```

```
R> interval2
[1] 2021-11-04 UTC--2022-11-04 UTC

R> interval1 == interval2
[1] TRUE
```

ถ้าต้องการหาจำนวนวันใน 1 ปีที่กำหนด ต้องใช้การคำนวณทางคณิตศาสตร์โดยการกำหนด interval ที่ต้องการแล้วหารด้วยช่วงเวลา 1 วัน ดังนี้

```
R> interval1 / ddays(1)
[1] 365
```

ตัวอย่างปีที่มี 366 วัน แสดงได้ดังนี้

```
R> next_year <- ymd("2020-01-01") + years(1)
R> interval3 <- interval(start = ymd("2020-01-01"), end = next_year)

R> interval3 / ddays(1)
[1] 366
```

20.5 สรุปท้ายบท

บทนี้กล่าวถึงการจัดการวันและเวลาด้วยแพ็กเกจ `lubridate` ได้แก่ การสร้างข้อมูลวันและเวลาจากสตริง การสร้างข้อมูลวันและเวลาจากแต่ละองค์ประกอบ การสร้างข้อมูลวัน-เวลาจากการแปลงข้อมูลเวลา/วันหรือตัวเลข องค์ประกอบของวันและเวลา การกำหนดค่าวันและเวลา การคำนวณเชิงตัวเลข (Arithmetic) สำหรับวัน-เวลาด้วยการใช้คำสั่งในคลาส Duration การใช้คำสั่งในคลาส Period และการใช้คลาส Interval

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`today()`	คำสั่งในการดึงข้อมูลวันปัจจุบัน
`now()`	คำสั่งในการดึงข้อมูลเวลาปัจจุบัน
`ymd()`	คำสั่งในการสร้างข้อมูลวันและเวลาจากสตริง ปี เดือน และวัน ตามลำดับ
`mdy()`	คำสั่งในการสร้างข้อมูลวันและเวลาจากสตริง เดือน วัน และปี ตามลำดับ
`dmy()`	คำสั่งในการสร้างข้อมูลวันและเวลาจากสตริง วัน เดือน และปี ตามลำดับ
`ymd_hms()`	คำสั่งในการสร้างข้อมูลวันและเวลาจากสตริง ปี เดือน วัน ชั่วโมง นาที และวินาที ตามลำดับ
`dmy_hm()`	คำสั่งในการสร้างข้อมูลวันและเวลาจากสตริง วัน เดือน ปี ชั่วโมง และนาที ตามลำดับ
`as_date()`	คำสั่งในการแปลงข้อมูลวันและเวลาจากรูปแบบวัน-เวลา (data-time) ไปเป็นวัน (date)

คำสั่ง	คำอธิบาย
as_datetime()	คำสั่งในการแปลงข้อมูลวันและเวลาจากรูปแบบวัน (date) ไปเป็นวัน-เวลา (data-time)
year()	คำสั่งในการดึงปีจากข้อมูลวัน-เวลา
month()	คำสั่งในการดึงเดือนจากข้อมูลวัน-เวลา
mday()	คำสั่งในการดึงวันที่ของเดือนจากข้อมูลวัน-เวลา
yday()	คำสั่งในการดึงวันที่ของปีจากข้อมูลวัน-เวลา
wday()	คำสั่งในการดึงวันที่ของสัปดาห์จากข้อมูลวัน-เวลา
hour()	คำสั่งในการดึงชั่วโมงจากข้อมูลวัน-เวลา
minute()	คำสั่งในการดึงนาทีจากข้อมูลวัน-เวลา
second()	คำสั่งในการดึงวินาทีจากข้อมูลวัน-เวลา
update()	คำสั่งในการปรับแก้ (update) ข้อมูลวัน-เวลา
as.duration()	คำสั่งในการแปลงข้อมูลจากคลาส difftime เป็น duration
dseconds()	คำสั่งในการแปลงตัวเลขในหน่วยวินาทีให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
dminutes()	คำสั่งในการแปลงตัวเลขในหน่วยนาทีให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
dhours()	คำสั่งในการแปลงตัวเลขในหน่วยชั่วโมงให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
ddays()	คำสั่งในการแปลงตัวเลขในหน่วยวันให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
dweeks()	คำสั่งในการแปลงตัวเลขในหน่วยสัปดาห์ให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
dyears()	คำสั่งในการแปลงตัวเลขในหน่วยปีให้เป็นวัตถุของคลาส Duration ในหน่วยวินาที
seconds()	คำสั่งในการแปลงตัวเลขในหน่วยวินาทีให้เป็นช่วงเวลาในคลาส Period
minutes()	คำสั่งในการแปลงตัวเลขในหน่วยนาทีให้เป็นช่วงเวลาในคลาส Period
hours()	คำสั่งในการแปลงตัวเลขในหน่วยชั่วโมงให้เป็นช่วงเวลาในคลาส Period
days()	คำสั่งในการแปลงตัวเลขในหน่วยวันให้เป็นช่วงเวลาในคลาส Period
weeks()	คำสั่งในการแปลงตัวเลขในหน่วยสัปดาห์ให้เป็นช่วงเวลาในคลาส Period
months()	คำสั่งในการแปลงตัวเลขในหน่วยเดือนให้เป็นช่วงเวลาในคลาส Period
years()	คำสั่งในการแปลงตัวเลขในหน่วยปีให้เป็นช่วงเวลาในคลาส Period
interval()	คำสั่งในการสร้างช่วงเวลาแบบ interval โดยการระบุอาก้วเมนต์เวลาเริ่มต้นและเวลาสิ้นสุด

คำสั่ง	คำอธิบาย
%--%	คำสั่งในการสร้างช่วงเวลาแบบ interval โดยการระบุเวลาเริ่มต้น **%--%** เวลาสิ้นสุด

20.6 แบบฝึกหัดท้ายบท

1. สร้างตัวแปรชื่อ dates เป็นข้อมูลเวกเตอร์ที่มีค่าเท่ากับ “2024-12-07”, “07/12/2024”, “12-07-24 14:30:00” และ “7 ธันวาคม 2567”

   1.1 ใช้คำสั่งเพื่อจัดการวันที่ทั้งหมดใน dates ให้อยู่ในรูปแบบ (yyyy-mm-dd HH:MM:SS)

   1.2 คำนวณจำนวนวันตั้งแต่วันที่ใน dates แต่ละตัวถึงวันที่ปัจจุบัน

   1.3 เปลี่ยนเวลาทั้งหมดใน dates ให้เป็นเวลา 15 นาฬิกา 30 นาที

2. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า airquality

   2.1 สร้างคอลัมน์ใหม่ใน airquality ชื่อ date โดยใช้ข้อมูลจากคอลัมน์ Day และ Month รวมกับปี 2019

   2.2 คำนวณช่วงเวลาจากวันที่แรกจนถึงวันที่สุดท้ายในข้อมูล

   2.3 คำนวณช่วงเวลาระหว่างวันที่แรกและวันสุดท้ายในข้อมูล ในหน่วยวินาที

   2.4 เพิ่มคอลัมน์ days_from_start เพื่อระบุจำนวนวันตั้งแต่วันที่แรกในข้อมูล

   2.5 เพิ่มคอลัมน์ seconds_from_start เพื่อระบุจำนวนวินาทีตั้งแต่วันที่แรกในข้อมูล

3. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ ggplot2 ชื่อว่า economics

   3.1 สร้างคอลัมน์ใหม่ใน economics ชื่อ day เป็นข้อมูลวันในสัปดาห์จากข้อมูลในคอลัมน์ date

   3.2 สร้างคอลัมน์ is_end_of_month ที่ระบุว่าแต่ละวันที่เป็นวันสุดท้ายของเดือนหรือไม่ ถ้าใช่ ให้เท่ากับ 1 ไม่ใช่เท่ากับ 0

   3.3 สร้างคอลัมน์ is_weekend ที่ระบุว่าแต่ละวันที่เป็นวันหยุดเสาร์-อาทิตย์หรือไม่ ถ้าใช่ ให้เท่ากับ 1 ไม่ใช่เท่ากับ 0

   3.4 คำนวณจำนวนวันที่เป็นวันหยุดสุดสัปดาห์ในข้อมูล

   3.5 หาค่าเฉลี่ยของ psavert (personal saving rate) ของแต่ละเดือน

บทที่ 21
การวนลูปด้วยแพ็กเกจ purrr

บทนี้จะกล่าวถึงการวนลูปด้วยแพ็กเกจ **purrr** ซึ่งจะทำให้การเข้าถึงสมาชิกแต่ละตัวในเวกเตอร์หรือลิสต์ทำได้คล่องตัว มีคำสั่งในการใช้งานที่กระชับ ใช้ง่าย ลดโอกาสที่ทำให้เกิดความผิดพลาดได้ (H. Wickham & Grolemund, 2016)

RStudio ได้สร้างไฟล์สรุปการใช้งาน **purrr** ไว้ใน Cheat sheet ซึ่งมีประโยชน์อย่างมากสำหรับผู้ใช้ **R** (ผู้ใช้สามารถค้นหาได้ทางอินเตอร์เน็ตโดยใช้ Keywords ดังกล่าว)

ก่อนอื่นขอกล่าวถึงการวนลูปโดยใช้ for loops ซึ่งภาษาโปรแกรมโดยทั่วไปใช้กัน รวมทั้งภาษา **R** ด้วย ตัวอย่างเช่น การหาค่า Median ถ้าไม่ใช้การวนลูป ผู้ใช้ต้องคำนวณหาค่า Median ของสมาชิกแต่ละตัวดังต่อไปนี้

```
R> library(tidyverse)
R> set.seed(123)
R> tbl <- tibble(
     a = rnorm(10),
     b = rnorm(10),
     c = rnorm(10),
     d = rnorm(10)
   )

R> median(tbl$a)
[1] -0.07983455
R> median(tbl$b)
[1] 0.3802926
R> median(tbl$c)
[1] -0.6769652
R> median(tbl$d)
[1] 0.4901909
```

การใช้ for loops แสดงได้ดังนี้

```
R> output <- vector("double", ncol(tbl))   # 1. output
R> for (i in seq_along(tbl)) {             # 2. sequence
     output[[i]] <- median(tbl[[i]])       # 3. body
   }
R> output
[1] -0.07983455  0.38029264 -0.67696525  0.49019094
```

คำสั่งแรกเป็นการสร้างเวกเตอร์สำหรับเก็บผลลัพธ์ที่ได้จากการคำนวณ ซึ่งจะสร้างตัวแปร output ไว้สำหรับกำหนดพื้นที่ในหน่วยความจำชั่วคราวในเครื่องคอมพิวเตอร์ด้วยคำสั่ง `vector()` โดยกำหนดให้มีค่าเป็น double ความยาวเท่ากับจำนวนตัวแปร/คอลัมน์ใน tibble ชื่อ tbl คำสั่งต่อมาเป็นการวนลูปโดยใช้ for loops โดยการระบุตัวแปร i เป็นสมาชิกแต่ละตัวที่อ้างถึงในเวกเตอร์ที่สร้างขึ้นจากคำสั่ง `seq_along()`

ภายใน for loops มีการคำนวณค่า Median แล้วเก็บค่าที่ได้ในตัวแปร output คำสั่งสุดท้ายเป็นการแสดงผลลัพธ์ที่ได้จากตัวแปร output

21.1 การวนลูปลักษณะต่าง ๆ

การวนลูปโดยทั่วไปแบ่งออกเป็น 4 ลักษณะ ดังนี้

1. การปรับปรุงวัตถุเดิม (Modifying an existing object) แทนที่จะการสร้างวัตถุใหม่

ตัวอย่างเช่น การปรับสเกล (Rescale) ข้อมูลทุกคอลัมน์ของ tibble ทำได้ดังนี้

```
# Modifying an existing object
R> rescale01 <- function(x) {
     rng <- range(x, na.rm = TRUE)
     (x - rng[1]) / (rng[2] - rng[1])
   }

R> for (i in seq_along(tbl)) {
     tbl[[i]] <- rescale01(tbl[[i]])
   }
R> tbl
# A tibble: 10 × 4
       a     b     c      d
   <dbl> <dbl> <dbl>  <dbl>
 1 0.236 0.850 0.210 0.633
 2 0.347 0.620 0.499 0.0669
 3 0.948 0.631 0.225 1
 4 0.448 0.553 0.326 0.987
 5 0.468 0.376 0.361 0.942
 6 1     1     0     0.838
 7 0.579 0.657 0.859 0.733
 8 0     0     0.626 0.250
 9 0.194 0.711 0.187 0.0584
10 0.275 0.398 1     0
```

คำสั่งแรกเป็นการสร้างฟังก์ชัน rescale01 เพื่อปรับสเกลค่าแต่ละคอลัมน์ให้มีค่าระหว่าง 0 – 1 คำสั่งต่อมาเป็นการวนลูปโดยใช้ for loops โดยการระบุตัวแปร i เป็นสมาชิกแต่ละตัวที่อ้างถึงในเวกเตอร์ที่สร้างขึ้นจากคำสั่ง `seq_along()` ภายใน for loops มีการเรียกใช้ฟังก์ชัน rescale01 แล้วเก็บค่าที่ได้ในตัวแปรเดิม การเข้าถึงตัวแปรแต่ละตัวในเดต้าเฟรมหรือ tibble ใช้เครื่องหมาย `[[…]]` ในการเข้าถึง เนื่องจาก tibble ก็เหมือนกับลิสต์หลาย ๆ คอลัมน์มารวมกันนั้นเอง คำสั่งสุดท้ายเป็นการแสดงผลลัพธ์ที่ได้

2. การวนลูปโดยใช้ชื่ออ้างอิง (Looping over names) แทนที่จะใช้ index

เป็นการใช้ชื่อ (Names) ในการอ้างถึงสมาชิกในแต่ละตัวในเวกเตอร์ ซึ่งก่อนอ้างอิงด้วยชื่อจะต้องกำหนดชื่อให้กับตัวแปร แล้วถึงจะทำการวนลูปโดยใช้ชื่อ ตัวอย่างเช่น

```
# Make sure to name the result of the vector
results <- vector("list", length(x))
names(results) <- names(x)
```

```
# Loop by name
for (i in seq_along(x)) {
  name <- names(x)[[i]]
  value <- x[[i]]
}
```

3. การวนลูปโดยไม่ทราบความยาวของผลลัพธ์ (Unknown output length)

การวนลูปโดยไม่ทราบความยาวของผลลัพธ์อาจทำได้โดยการสร้างเวกเตอร์และเพิ่มสมาชิกในเวกเตอร์โดยการวนลูป ตัวอย่างเช่น ต้องการสร้างเวกเตอร์เลขสุ่มตามความยาวด้วยการสุ่ม ดังนี้

```
# Unknown Output Length
R> means <- c(0, 1, 2)

R> output <- double()
R> for (i in seq_along(means)) {
     n <- sample(100, 1)
     output <- c(output, rnorm(n, means[[i]]))
   }
R> str(output)
# num [1:27] 0.345 -0.529 2.932 0.513 1.07 ...
```

คำสั่งด้านบนมีการเพิ่มสมาชิก (ผลลัพธ์) ในเวกเตอร์ด้วยการรวมผลลัพธ์ใหม่เข้ากับเวกเตอร์ output ต่อเนื่องกันไปโดยใช้ for loops แต่วิธีการดังกล่าวจะไม่มีประสิทธิภาพเมื่อจำนวนสมาชิกเพิ่มมากขึ้น วิธีการที่ดีกว่าคือ สร้างผลลัพธ์เป็นลิสต์ แล้วค่อยรวมเป็นเวกเตอร์ในขั้นตอนสุดท้าย ดังนี้

```
R> out <- vector("list", length(means))
R> for (i in seq_along(means)) {
     n <- sample(100, 1)
     out[[i]] <- rnorm(n, means[[i]])
   }
R> str(out)
# List of 3
#  $ : num [1:99] -0.9901 -0.4882 -0.7211 -0.1251 -0.0635 ...
#  $ : num [1:51] 0.247 3.022 -2.176 -0.577 0.814 ...
#  $ : num [1:11] 1.61 2.93 2.75 2.33 0.24 ...

R> str(unlist(out))
# num [1:161] -0.9901 -0.4882 -0.7211 -0.1251 -0.0635 ...
```

คำสั่งแรกเป็นการสร้างลิสต์ที่มีจำนวนสมาชิกเท่ากับค่าเวกเตอร์ means คำสั่งต่อมาเป็นการวนลูปโดยใช้ for loops ภายใน for loops มีการเก็บค่าที่ได้ไว้ในลิสต์ และคำสั่งสุดท้ายเป็นการสร้างเวกเตอร์โดยการรวมสมาชิกทุกตัวในลิสต์เข้าด้วยกัน ด้วยคำสั่ง `unlist()`

ถ้าเป็นการรวมสตริงในเวกเตอร์เข้าด้วยกันมีคำสั่ง `paste(output, collapse = "")` ในการสร้างสตริง ซึ่งทำงานได้เร็วกว่าการวนลูปโดยทั่วไป ถ้าเป็นการรวมเดต้าเฟรมขนาดใหญ่เข้าด้วยกันมีคำสั่ง `bind_rows()` ซึ่งทำงานได้เร็วกว่าการวนลูปแล้วใช้คำสั่ง `rbind()`

4. การวนลูปโดยไม่ทราบความยาวของข้อมูลนำเข้า (Unknown input length)

การวนลูปโดยไม่ทราบความยาวของข้อมูลนำเข้ามักจะใช้คำสั่ง while ดังนี้

```
# Unknown input length
while (condition) {
  # body
}
```

ตัวอย่างการใช้ while loops ในการหาจำนวนครั้งในการสุ่มแล้วออกหัวต่อเนื่อง 3 ครั้ง แสดงดังนี้

```
R> set.seed(123)
R> flip <- function() sample(c("T", "H"), 1)

R> flips <- 0
R> nheads <- 0

R> while (nheads < 3) {
     if (flip() == "H") {
        nheads <- nheads + 1
     } else {
        nheads <- 0
     }
     flips <- flips + 1
   }
R> flips
[1] 8
```

21.2 การใช้คำสั่ง map

ใน **R** นิยมการใช้ฟังก์ชันในการเข้าถึงสมาชิกในเวกเตอร์หรือลิสต์ (Vecterization) มากกว่าการเขียนคำสั่งด้วยการวนลูป หัวข้อ 6.2.3 ได้กล่าวถึงการใช้ฟังก์ชันดังกล่าวใน base **R** ในหัวข้อนี้จะกล่าวถึงการใช้คำสั่งในแพ็กเกจ `purrr` ซึ่งมีมาตรฐานคงเส้นคงวากว่าคำสั่งใน base **R** (More consistent) ทำให้ง่ายในการเรียนรู้ (H. Wickham & Grolemund, 2016) ได้แก่

- `map()` คืนค่า (Return) เป็นลิสต์ (List)
- `map_lgl()` คืนค่า (Return) เป็นเวกเตอร์ตรรกะ (Logical vector)
- `map_int()` คืนค่า (Return) เป็นเวกเตอร์เลขจำนวนเต็ม (Interger vector)
- `map_dbl()` คืนค่า (Return) เป็นเวกเตอร์เลขจำนวนจริงแบบ double (Double vector)
- `map_chr()` คืนค่า (Return) เป็นเวกเตอร์ตัวอักษร (Character vector)

คำสั่งดังกล่าวรับค่าเป็นเวกเตอร์ ทำการประมวลผลด้วยฟังก์ชันที่กำหนด แล้วส่งค่ากลับเป็นเวกเตอร์หรือลิสต์ที่มีจำนวนสมาชิกเท่ากับเวกเตอร์นำเข้า รูปแบบการใช้คำสั่ง `map()` และ `map_*()` ประกอบด้วย

```
map(.x, .f, ...)
```

โดยที่ `.x` คือ เวกเตอร์หรือลิสต์

`.f` คือ ฟังก์ชันที่ต้องการเรียกใช้

`...` คือ อากิวเมนต์เพิ่มเติมที่ใส่เข้ามาในฟังก์ชัน

ตัวอย่างการใช้คำสั่ง `map()` และ `map_*()` แสดงได้ดังนี้

```
R> set.seed(123)
# The Map Functions
R> tbl <- tibble(
     a = rnorm(10),
     b = rnorm(10),
     c = rnorm(10),
     d = rnorm(10)
   )

R> tbl |> map_dbl(mean)
          a           b           c           d
 0.07462564  0.20862196 -0.42455887  0.32204455
R> tbl |> map_dbl(median)
          a           b           c           d
-0.07983455  0.38029264 -0.67696525  0.49019094
R> tbl |> map_dbl(sd)
        a         b         c         d
0.9537841 1.0380734 0.9308092 0.5273024
```

ฟังก์ชัน maps มีความแตกต่างจะคำสั่งอื่น คือ (1) สามารถส่งอากิวเมนต์ Dot-Dot-Dot (`...`) เพิ่มเข้ามาในฟังก์ชันได้ (2) ยังคงชื่อ (Names) ของคอลัมน์/ตัวแปรนำเข้าไว้ผลลัพธ์ ดังนี้

```
R> map_dbl(tbl, mean, trim = 0.2)
         a          b          c          d
-0.0959338  0.2662805 -0.5853934  0.3556033
R> z <- list(x = 1:3, y = 4:5)
R> map_int(z, length)
x y
3 2
```

คำสั่งแรกจะเห็นว่ามีการส่งอากิวเมนต์ **`trim`** เข้ามาในฟังก์ชัน คำสั่งที่สองจะเห็นว่าผลลัพธ์ที่ได้ มีชื่อตัวแปร x และ y เช่นเดียวกับชื่อตัวแปรของลิลต์นำเข้า

ฟังก์ชัน map เขียนได้กระชับ อ่านง่าย ตัวอย่างเช่น ต้องการสร้างสมการเชิงเส้น (Simple linear regression) สำหรับข้อมูล mtcars โดยแยกตามจำนวนกระบอกสูบ (cyl) ดังนี้

```
R> models <- mtcars |>
     split(mtcars$cyl) |>
     map(function(tbl) lm(mpg ~ wt, data = tbl))

# More concise
R> models <- mtcars |>
     split(mtcars$cyl) |>
     map(~lm(mpg ~ wt, data = .))
```

คำสั่งชุดแรกมีการเรียกใช้ `map()` และฟังก์ชัน `lm()` แบบเต็มรูปแบบ ส่วนคำสั่งที่สองมีการใช้เครื่องหมาย `~` นำหน้าฟังก์ชัน `lm()` เพื่อแสดงการเรียกใช้ฟังก์ชันดังกล่าวแบบไม่ต้องตั้งชื่อ (Anonymous

functions) และมีการใช้จุด . แทนสมาชิกแต่ละตัวที่ส่งเข้ามาในคำสั่ง map() ซึ่งคล้ายกับการใช้ตัวแปร i ใน for loops นั้นเอง

การรายงานผลแบบจำลองที่สร้างขึ้น เช่น ต้องการค่า R^2 สามารถใช้คำสั่ง summary() และเรียกคำสั่ง map_dbl() โดยระบุฟังก์ชันที่ต้องการดึงค่าออกมา หรือระบุด้วยสตริงที่ต้องการ ดังนี้

```
R> models |>
     map(summary) |>
     map_dbl(~.$r.squared)

R> models |>
     map(summary) |>
     map_dbl("r.squared")
```

สามารถใช้เลขจำนวนเต็มในการอ้างถึงสมาชิกแต่ละตัว ได้ดังนี้

```
R> x <- list(list(1, 2, 3), list(4, 5, 6), list(7, 8, 9))
R> x |> map_dbl(2)
# [1] 2 5 8
```

คำสั่งข้างต้นเป็นการเข้าถึงสมาชิกตัวที่ 2 ของลิสต์ทั้ง 3 ตัว

21.3 การใช้คำสั่ง map สำหรับอากิวเมนต์หลายตัว

คำสั่ง map ในหัวข้อที่ผ่านมามีอากิวเมนต์เพียง 1 ตัว ในหัวข้อนี้จะกล่าวถึงคำสั่ง map ที่สามารถรับอากิวเมนต์ได้ตั้งแต่ 2 ตัวขึ้นไป ได้แก่ map2() และ pmap() ตัวอย่างเช่น ถ้าต้องการสร้างเลขสุ่มจากค่าเฉลี่ย (Mean) ด้วยคำสั่ง map() จากหัวข้อก่อนหน้านี้ เขียนได้ดังนี้

```
R> set.seed(123)
R> mu <- list(5, 10, -3)
R> mu |>
     map(rnorm, n = 5) |>
     str()
List of 3
 $ : num [1:5] 4.44 4.77 6.56 5.07 5.13
 $ : num [1:5] 11.72 10.46 8.73 9.31 9.55
 $ : num [1:5] -1.78 -2.64 -2.6 -2.89 -3.56
```

คำสั่งด้านบนเป็นการสร้างเลขสุ่มที่มีค่า Mean = 5, 10 และ -3 ตามลำดับ โดยมีส่วนเบี่ยงเบนมาตรฐาน (Standard deviation, SD) เท่ากับ 1 ถ้าต้องการเลขสุ่มที่มีค่า SD เท่ากับ 1, 5 และ 10 สามารถเขียนโดยใช้คำสั่ง map() ได้ดังนี้

```
R> set.seed(123)
R> sigma <- list(1, 5, 10)
R> seq_along(mu) |>
     map(~rnorm(5, mu[[.]], sigma[[.]])) |>
     str()
List of 3
 $ : num [1:5] 4.44 4.77 6.56 5.07 5.13
 $ : num [1:5] 18.58 12.3 3.67 6.57 7.77
```

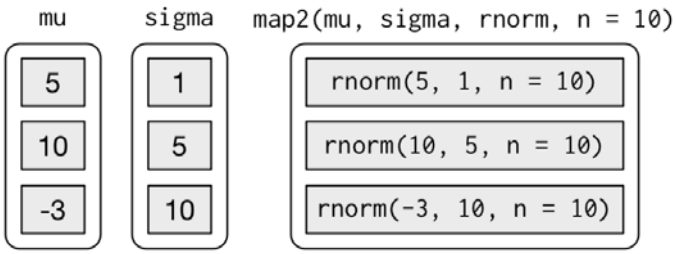
```
 $ : num [1:5] 9.241 0.598 1.008 -1.893 -8.558
```

คำสั่งข้างต้นเป็นการสร้างฟังก์ชันที่ไม่กำหนดชื่อ (Anonymous function) โดยเรียกใช้ฟังก์ชัน `rnorm()` เพื่อส่งอากิวเมนต์ทั้ง 2 ตัวได้แก่ ค่าเฉลี่ย (Mean) และส่วนเบี่ยงเบนมาตรฐาน (Standard deviation, SD) ในการหาเลขสุ่มที่ต้องการ

ส่วนคำสั่ง `map2()` สามารถรับอากิวเมนต์ได้ 2 ตัว ทำให้การเขียนโค้ดดังกล่าวกระชับ อ่าน เขียนง่ายขึ้น ดังนี้

```
R> set.seed(123)
R> map2(mu, sigma, rnorm, n = 5) |>
     str()
List of 3
 $ : num [1:5] 4.44 4.77 6.56 5.07 5.13
 $ : num [1:5] 18.58 12.3 3.67 6.57 7.77
 $ : num [1:5] 9.241 0.598 1.008 -1.893 -8.558
```

คำสั่งดังกล่าวแสดงด้วยรูปภาพดังนี้



รูปที่ 21-1 map2() function

คำสั่ง `map2_*()` สามารถส่งค่ากลับมาเป็นเวกเตอร์แบบต่าง ๆ ได้เช่นเดียวกับ คำสั่ง `map()` เช่น `map2_lgl()` `map2_int()` `map2_dbl()` และ `map2_chr()` เป็นต้น

รูปแบบการใช้คำสั่ง `map2()` และ `map2_*()` ประกอบด้วย

```
map2(.x, .y, .f, ...)
```

โดยที่ `.x` คือ เวกเตอร์ตัวที่ 1

`.y` คือ เวกเตอร์ตัวที่ 2

`.f` คือ ฟังก์ชันที่ต้องการเรียกใช้

`...` คือ อากิวเมนต์เพิ่มเติมที่ใส่เข้ามาในฟังก์ชัน

ถ้ามีอากิวเมนต์มากกว่า 2 ตัว แพ็กเกจ `purrr` มีคำสั่ง `pmap()` และ `pmap_*()` โดยมีรูปแบบการใช้ดังนี้

```
pmap(.l, .f, ...)
```

โดยที่ `.l` คือ ลิสต์ของเวกเตอร์หรือเดต้าเฟรม

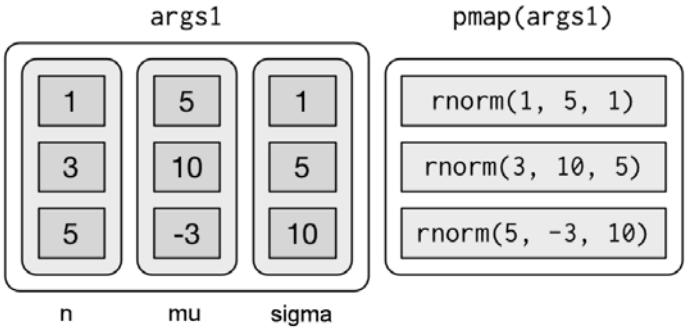
`.f` คือ ฟังก์ชันที่ต้องการเรียกใช้

`...` คือ อากิวเมนต์เพิ่มเติมที่ใส่เข้ามาในฟังก์ชัน

ตัวอย่างเช่น ต้องการเรียกใช้ฟังก์ชัน `rnorm()` ด้วยอากิวเมนต์ 3 ตัว ได้แก่ จำนวนตัวอย่าง ค่าเฉลี่ย และส่วนเบี่ยงเบนมาตรฐาน ตามลำดับ เขียนได้ดังนี้

```
R> set.seed(123)
R> n <- list(1, 3, 5)
R> args1 <- list(n, mu, sigma)
R> args1 |>
     pmap(rnorm) |>
     str()
List of 3
 $ : num 4.44
 $ : num [1:3] 8.85 17.79 10.35
 $ : num [1:5] -1.71 14.15 1.61 -15.65 -9.87
```

คำสั่งข้างต้นจะทำการเรียกใช้ฟังก์ชัน `rnorm()` โดยผ่านอากิวเมนต์เข้าไปตามลำดับ คำสั่งดังกล่าวแสดงด้วยรูปภาพดังนี้

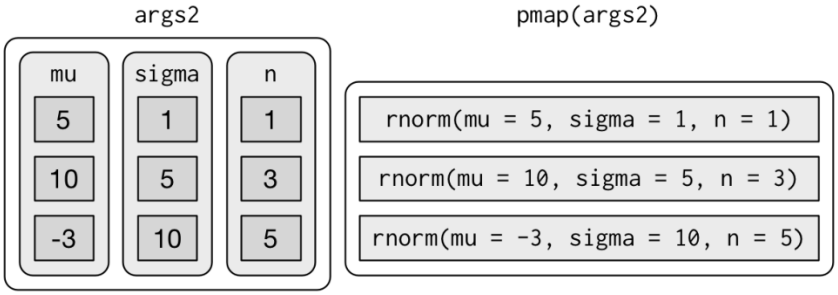


รูปที่ 21-2 pmap() function pass arguments by positions of elements

อีกวิธีหนึ่งในการเรียกใช้ `pmap()` คือ ผ่านค่าลิสต์ที่มีชื่อสมาชิก (Named list) ดังนี้

```
R> set.seed(123)
R> args2 <- list(mean = mu, sd = sigma, n = n)
R> args2 |>
     pmap(rnorm) |>
     str()
List of 3
 $ : num 4.44
 $ : num [1:3] 8.85 17.79 10.35
 $ : num [1:5] -1.71 14.15 1.61 -15.65 -9.87
```

คำสั่งดังกล่าวแสดงด้วยรูปภาพดังนี้



รูปที่ 21-3 pmap() function pass arguments by a named list

อีกวิธีหนึ่งในการเรียกใช้ `pmap()` คือ ผ่านเดต้าเฟรมที่มีชื่อคอลัมน์/ตัวแปรเดียวกับอากิวเมนต์เข้าไป ดังนี้

```
R> set.seed(123)
R> params <- tribble(
     ~mean, ~sd, ~n,
     5, 1, 1,
     10, 5, 3,
     -3, 10, 5
   )
R> params |>
     pmap(rnorm)
[[1]]
[1] 4.439524

[[2]]
[1]  8.849113 17.793542 10.352542

[[3]]
[1]  -1.707123  14.150650   1.609162 -15.650612  -9.868529
```

21.4 การใช้ invoke_map() สำหรับจัดการกับฟังก์ชันมากกว่า 1 ตัว

คำสั่งที่ผ่านมาก่อนหน้านี้เป็นการเรียกใช้ฟังก์ชัน 1 ฟังก์ชัน (มีอากิวเมนต์ 1 ตัวหรือมากกว่า) หัวข้อนี้จะใช้คำสั่ง `invoke_map()` ที่ทำงานกับหลายฟังก์ชัน มีรูปแบบการใช้งาน ดังนี้

```
invoke_map(.f, .x = list(NULL), ...)
```

โดยที่ `.f` คือ ลิสต์ของฟังก์ชัน

`.x` คือ ลิสต์ของอากิวเมนต์

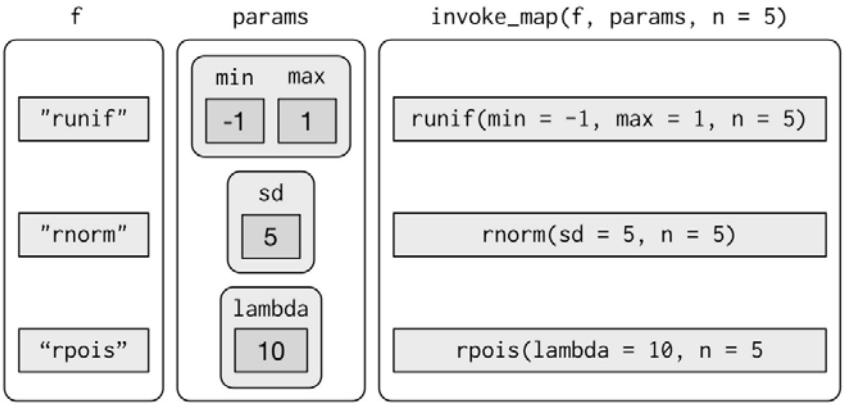
`...` คือ อากิวเมนต์เพิ่มเติมที่ใส่เข้ามาในฟังก์ชัน

ตัวอย่างการใช้ฟังก์ชันดังกล่าว แสดงด้วยการเรียกใช้ `runif(), rnorm()` และ `rpois()` โดยแต่ละฟังก์ชันมีอากิวเมนต์แตกต่างกันซึ่งเก็บอยู่ในลิสต์ ดังนี้

```
R> set.seed(123)
R> f <- c("runif", "rnorm", "rpois")
R> param <- list(
```

```
     list(min = -1, max = 1),
     list(sd = 5),
     list(lambda = 10)
   )
R> invoke_map(f, param, n = 5) |>
     str()
List of 3
 $ : num [1:5] -0.425 0.577 -0.182 0.766 0.881
 $ : num [1:5] -8.448 6.197 -0.545 -0.586 0.915
 $ : int [1:5] 14 4 13 11 11
```

คำสั่งดังกล่าวแสดงด้วยรูปภาพดังนี้



รูปที่ 21-4 invoke_map() function

อีกวิธีหนึ่งในการเรียกใช้ **invoke_map()** คือ ผ่าน tibble ที่ประกอบด้วยตัวแปรแสดงชื่อฟังก์ชันและลิสต์ที่เก็บอากิวเมนต์เข้าไป ดังนี้

```
R> set.seed(123)
R> simulation <- tribble(
     ~f, ~params,
     "runif", list(min = -1, max = 1),
     "rnorm", list(sd = 5),
     "rpois", list(lambda = 10)
   )
R> simulation |>
     mutate(sim = invoke_map(f, params, n = 10))
# A tibble: 3 × 3
  f     params           sim
  <chr> <list>           <list>
1 runif <named list [2]> <dbl [10]>
2 rnorm <named list [1]> <dbl [10]>
3 rpois <named list [1]> <int [10]>
```

21.5 คำสั่ง walk() สำหรับการแสดงผลลัพธ์

คำสั่ง **walk() walk2()** และ **pwalk()** อีกวิธีการหนึ่งในการใช้คำสั่ง **map()** คือ การเรียกใช้คำสั่งเพื่อดูผลข้างเคียง (Side effects) แทนที่จะสนใจค่าที่ส่งกลับมา โดยทั่วไปมักจะใช้เพื่อดูผลลัพธ์ทางจอภาพ (Console) หรือบันทึกลงไฟล์ ตัวอย่างดังนี้

```
R> x <- list(1, "a", 3)
R> x |>
     walk(print)
[1] 1
[1] "a"
[1] 3
```

อีกตัวอย่างหนึ่งแสดงการใช้คำสั่ง **pwalk()** สำหรับการแสดงผลลัพธ์ทาง console แทนที่จะเขียนลงไฟล์ ดังนี้

```
R> plots <- mtcars |>
     split(.$cyl) |>
     map(~ggplot(., aes(mpg, wt)) + geom_point())
R> paths <- str_c(names(plots), ".pdf")
R> pwalk(list(paths, plots), ggsave, path = tempdir())
# pmap(list(paths, plots), ggsave)
# Saving 7 x 7 in image
# Saving 7 x 7 in image
# Saving 7 x 7 in image
```

คำสั่ง **walk() walk2()** และ **pwalk()** นิยมใช้เพื่อแสดงผลภายในระหว่างการใช้คำสั่งต่อเนื่อง (Chain commands)

21.6 คำสั่งที่ใช้กับฟังก์ชันในการตรวจสอบ

แพ็กเกจ **purrr** มีคำสั่งที่ใช้กับฟังก์ชันในการตรวจสอบ ซึ่งฟังก์ชันในการตรวจสอบจะคืนค่ากลับมาเป็น **TRUE** หรือ **FALSE** ได้แก่ คำสั่ง **keep()** จะเก็บตัวแปรทั้งหมดที่มีค่าเป็น **TRUE** ดังนี้

```
R> iris |>
     keep(is.factor) |>
     str()
# 'data.frame': 150 obs. of  1 variable:
# $ Species: Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

คำสั่ง **discard()** จะเก็บตัวแปรทั้งหมดที่มีค่าเป็น **FALSE** ดังนี้

```
R> iris |>
     discard(is.factor) |>
     str()
# 'data.frame': 150 obs. of  4 variables:
# $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
# $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
# $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
# $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

คำสั่ง some() จะคืนค่า TRUE ถ้าสมาชิกตัวใดตัวหนึ่งมีค่าเป็น TRUE ดังนี้

```
R> x <- list(1:5, letters, list(10))
R> x |>
     some(is_character)
[1] TRUE
```

คำสั่ง every() จะคืนค่า TRUE ถ้าสมาชิกทุกตัวมีค่าเป็น TRUE ดังนี้

```
R> x <- list(1:5, letters, list(10))
R> x |>
     every(is_vector)
[1] TRUE
```

คำสั่ง detect() จะคืนค่าสมาชิกตัวแรกที่เป็น TRUE คำสั่ง detect_index() จะคืนค่าตำแหน่งสมาชิกตัวแรกที่เป็น TRUE ดังนี้

```
R> set.seed(123)
R> x <- sample(10)
R> x
[1]  3 10  2  8  6  9  1  7  5  4

R> x |>
     detect(~ . > 5)
[1] 10
R> x |>
     detect_index(~ . > 5)
[1] 2
```

คำสั่ง head_while() จะเก็บตัวแปรทั้งหมดจากตำแหน่งเริ่มต้นที่ตรวจสอบแล้วมีค่าเป็น TRUE คำสั่ง tail_while() จะเก็บตัวแปรทั้งหมดจากตำแหน่งสุดท้ายที่ตรวจสอบแล้วมีค่าเป็น TRUE ดังนี้

```
R> x <- c(6, 7, 8, 9, 1, 2, 3, 4, 9, 6)
R> x
[1] 6 7 8 9 1 2 3 4 9 6

R> x |>
     head_while(~ . > 5)
[1] 6 7 8 9

R> x |>
     tail_while(~ . > 5)
[1] 9 6
```

21.7 คำสั่ง reduce() และ accumulate()

คำสั่ง reduce() จะทำการลดความซับซ้อนของลิสต์ที่ประกอบด้วย tibble หลายตัวให้เหลือเพียง 1 tibble ดังนี้

```
R> tibbles <- list(
     age = tibble(name = "John", age = 30),
     sex = tibble(name = c("John", "Mary"), sex = c("M", "F")),
     trt = tibble(name = "Mary", treatment = "A")
```

```
  )
R> tibbles
$age
# A tibble: 1 × 2
  name    age
  <chr> <dbl>
1 John     30

$sex
# A tibble: 2 × 2
  name  sex
  <chr> <chr>
1 John  M
2 Mary  F

$trt
# A tibble: 1 × 2
  name  treatment
  <chr> <chr>
1 Mary  A

R> tibbles |> reduce(full_join)
Joining with `by = join_by(name)`
Joining with `by = join_by(name)`
# A tibble: 2 × 4
  name    age sex   treatment
  <chr> <dbl> <chr> <chr>
1 John     30 M     NA
2 Mary     NA F     A
```

อีกตัวอย่างหนึ่ง คือ การใช้คำสั่ง `reduce()` ในการหา intersection ของลิสต์ที่ประกอบด้วยเวกเตอร์หลายตัว ดังนี้

```
R> list_of_vectors <- list(
     c(1, 3, 5, 6, 10),
     c(1, 2, 3, 7, 8, 10),
     c(1, 2, 3, 4, 8, 9, 10)
  )
R> list_of_vectors |> reduce(intersect)
[1]  1  3 10
```

คำสั่ง `reduce()` จะทำงานกับข้อมูลนำเข้าทีละคู่จากสมาชิกตัวที่ 1 และ 2 ไปจนเหลือสมาชิก (เวกเตอร์หรือ tibble) เพียง 1 ตัว

คำสั่ง `accumulate()` จะทำงานกับข้อมูลนำเข้าทีละคู่จากสมาชิกตัวที่ 1 และ 2 ไปจนถึงสมาชิกตัวสุดท้าย แต่จะเก็บผลลัพธ์สะสมไว้ทั้งหมดทุกขั้นตอน ดังนี้

```
R> x <- c(6, 7, 8, 9, 1, 2, 3, 4, 9, 6)
R> x
[1] 6 7 8 9 1 2 3 4 9 6
R> x |> accumulate(`+`)
[1]  6 13 21 30 31 33 36 40 49 55
```

21.8 สรุปท้ายบท

บทนี้กล่าวถึงการวนลูปด้วยแพ็กเกจ `purrr` ซึ่งจะทำให้การเข้าถึงสมาชิกแต่ละตัวในเวกเตอร์หรือลิสต์ทำได้คล่องตัว มีคำสั่งในการใช้งานที่กระชับ ใช้ง่าย ลดโอกาสที่ทำให้เกิดความผิดพลาดได้ กล่าวถึงการวนลูป 4 แบบ ได้แก่ (1) การปรับปรุงวัตถุเดิม (Modifying an existing object) แทนที่จะการสร้างวัตถุใหม่ (2) การวนลูปโดยใช้ชื่ออ้างอิง (Looping over names) แทนที่จะใช้ index (3) การวนลูปโดยไม่ทราบความยาวของผลลัพธ์ (Unknown output length) (4) การวนลูปโดยไม่ทราบความยาวของข้อมูลนำเข้า (Unknown input length) การใช้คำสั่ง maps ได้แก่ `map()` `map_*()` (ตัวย่างเช่น `map_lgl()` `map_int()` `map_dbl()` และ `map_chr()`) การใช้คำสั่ง maps สำหรับอากิวเมนต์หลายตัว ได้แก่ `map2()` `map2_*()` `pmap()` และ `pmap_*()` การใช้คำสั่ง `walk()` `walk2()` และ `pwalk()` สำหรับการแสดงผลลัพธ์ คำสั่งที่ใช้กับฟังก์ชันในการตรวจสอบ และคำสั่ง `reduce()` และ `accumulate()`

คำสั่งสำคัญในบทนี้ประกอบด้วย

คำสั่ง	คำอธิบาย
`seq_along()`	คำสั่งในการสร้างเวกเตอร์ลำดับที่ 1 จนถึงจำนวนสมาชิกที่กำหนด
`unlist()`	คำสั่งในการแปลงลิสต์ให้เป็นเวกเตอร์ 1 ตัว
`paste()`	คำสั่งในการเชื่อมสตริง
`bind_rows()`	คำสั่งในการรวมเดต้าเฟรมหลายตัวเป็นเดต้าเฟรม 1 ตัวตามแนวแถว
`bind_cols()`	คำสั่งในการรวมเดต้าเฟรมหลายตัวเป็นเดต้าเฟรม 1 ตัวตามแนวคอลัมน์
`map()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 1 ตัว และคืนค่าเป็นลิสต์ (List)
`map_lgl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 1 ตัว และคืนค่าเป็นเวกเตอร์ตรรกะ (Logical vector)

คำสั่ง	คำอธิบาย
`map_int()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 1 ตัว และคืนค่าเป็นเวกเตอร์เลขจำนวนเต็ม (Interger vector)
`map_dbl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 1 ตัว และคืนค่าเป็นเวกเตอร์เลขจำนวนจริงแบบ double (Double vector)
`map_chr()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 1 ตัว และคืนค่าเป็นเวกเตอร์ตัวอักษร (Character vector)
`split()`	คำสั่งในการแบ่งเวกเตอร์เป็นกลุ่มตามตัวแปรที่กำหนด
`map2()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 2 ตัว และคืนค่าเป็นลิสต์ (List)
`map2_lgl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 2 ตัว และคืนค่าเป็นเวกเตอร์ตรรกะ (Logical vector)
`map2_int()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 2 ตัว และคืนค่าเป็นเวกเตอร์เลขจำนวนเต็ม (Interger vector)
`map2_dbl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 2 ตัว และคืนค่าเป็นเวกเตอร์เลขจำนวนจริงแบบ double (Double vector)
`map2_chr()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์ 2 ตัว และคืนค่าเป็นเวกเตอร์ตัวอักษร (Character vector)
`pmap()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์เป็นลิสต์ของเวกเตอร์ และคืนค่าเป็นลิสต์ (List)
`pmap_lgl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์เป็นลิสต์ของเวกเตอร์ และคืนค่าเป็นเวกเตอร์ตรรกะ (Logical vector)
`pmap_int()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์เป็นลิสต์ของเวกเตอร์ และคืนค่าเป็นเวกเตอร์เลขจำนวนเต็ม (Interger vector)
`pmap_dbl()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์เป็นลิสต์ของเวกเตอร์ และคืนค่าเป็นเวกเตอร์เลขจำนวนจริงแบบ double (Double vector)
`pmap_chr()`	คำสั่งในการเรียกใช้ฟังก์ชันที่ต้องการ โดยรับอากิวเมนต์เป็นลิสต์ของเวกเตอร์ และคืนค่าเป็นเวกเตอร์ตัวอักษร (Character vector)
`invoke_map()`	คำสั่งในการทำงานกับหลายฟังก์ชัน
`walk()`	คำสั่งในการแสดงผล มีอากิวเมนต์เหมือนกับคำสั่ง `map()`
`walk2()`	คำสั่งในการแสดงผล มีอากิวเมนต์เหมือนกับคำสั่ง `map2()`
`pwalk()`	คำสั่งในการแสดงผล มีอากิวเมนต์เหมือนกับคำสั่ง `pmap()`

คำสั่ง	คำอธิบาย
keep()	คำสั่งในการจะเก็บตัวแปรทั้งหมดที่ตรวจสอบแล้วมีค่าเป็น TRUE
discard()	คำสั่งในการเก็บตัวแปรทั้งหมดที่ตรวจสอบแล้วมีค่าเป็น FALSE
some()	คำสั่งในการตรวจสอบแล้วคืนค่า TRUE ถ้าสมาชิกตัวใดตัวหนึ่งมีค่าเป็น TRUE
every()	คำสั่งในการตรวจสอบแล้วคืนค่า TRUE ถ้าสมาชิกทุกตัวมีค่าเป็น TRUE
detect()	คำสั่งในการตรวจสอบแล้วคืนค่าสมาชิกตัวแรกที่เป็น TRUE
detect_index()	คำสั่งในการตรวจสอบแล้วคืนค่าตำแหน่งสมาชิกตัวแรกที่เป็น TRUE
head_while()	คำสั่งในการเก็บตัวแปรทั้งหมดจากตำแหน่งเริ่มต้นที่ตรวจสอบแล้วมีค่าเป็น TRUE
tail_while()	คำสั่งในการเก็บตัวแปรทั้งหมดจากตำแหน่งสุดท้ายที่ตรวจสอบแล้วมีค่าเป็น TRUE
reduce()	คำสั่งในการลดลิสต์ที่ประกอบด้วย tibble หลายตัวให้เหลือเพียง 1 tibble หรือ ลดลิสต์ที่ประกอบด้วยเวกเตอร์หลายตัวให้เหลือเพียงเวกเตอร์ 1 ตัว ด้วยฟังก์ชันที่กำหนด
accumulate()	คำสั่งในการทำงานกับฟังก์ชันที่กำหนด แล้วเก็บผลลัพธ์สะสมไว้ทั้งหมดทุกขั้นตอน

21.9 แบบฝึกหัดท้ายบท

1. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ dplyr ชื่อว่า starwars

   1.1 ใช้คำสั่ง `map()` แปลงค่าในแต่ละคอลัมน์ใน starwars ให้เป็นประเภท character และตรวจสอบผลลัพธ์

   1.2 ใช้คำสั่ง `map_lgl()` เพื่อตรวจสอบว่าในแต่ละคอลัมน์ใน starwars มีค่า **NA** หรือไม่

   1.3 ใช้คำสั่ง `map_int()` เพื่อหาจำนวนค่าที่เป็น **NA** ในแต่ละคอลัมน์ของ starwars และตรวจสอบผลลัพธ์ว่าคอลัมน์ใดที่มีจำนวนมากที่สุด

   1.4 ใช้คำสั่ง `map_dbl()` เพื่อคำนวณค่าเฉลี่ยของแต่ละคอลัมน์ที่เป็นตัวเลขและไม่เป็น **NA** ใน starwars แล้วนำค่าเฉลี่ยเหล่านี้มาหาผลรวม

2. ใช้คำสั่ง `map2()` ในการสร้างเลขสุ่มที่มีการแจกแจงแบบปกติ จำนวน 2 ชุด โดยที่ชุดแรกมีค่าเฉลี่ยเท่ากับ 5 ส่วนเบี่ยงเบนมาตรฐานเท่ากับ 2 และชุดที่สองมีค่าเฉลี่ยเท่ากับ 25 ส่วนเบี่ยงเบนมาตรฐานเท่ากับ 9 โดยแต่ละชุดมีจำนวนเลขสุ่มเท่ากับ 20 จำนวน
3. ใช้คำสั่ง `pmap()` ในการสร้างเลขสุ่มที่มีการแจกแจงแบบปกติ จำนวน 5 ชุด โดยแต่ละชุดมีค่าเฉลี่ยเท่ากับ 25, 20, 15, 10 และ 5 ตามลำดับ ส่วนเบี่ยงเบนมาตรฐานเท่ากับ 10, 9, 8, 7 และ 6 ตามลำดับ และแต่ละชุดมีจำนวนเลขสุ่มเท่ากับจำนวนค่าเฉลี่ยดังกล่าว
4. ใช้ข้อมูลเดต้าเฟรมจากแพ็กเกจ datasets ชื่อว่า mtcars และ iris

   4.1 ใช้คำสั่ง `keep()` เพื่อกรองคอลัมน์ใน mtcars ที่มีค่าเฉลี่ยมากกว่า 5 แล้วรวมผลรวมของค่าเฉลี่ยเหล่านี้

   4.2 ใช้คำสั่ง `discard()` เพื่อกรองคอลัมน์ใน iris ที่ไม่ใช่ตัวเลข แล้วนำคอลัมน์ที่เหลือมาคำนวณค่าเฉลี่ย

   4.3 ใช้คำสั่ง `some()` เพื่อตรวจสอบว่ามีค่าใดใน mtcars มากกว่า 30 หรือไม่ และแสดงค่าที่พบ

   4.4 ใช้คำสั่ง `every()` เพื่อตรวจสอบว่าค่าทั้งหมดในคอลัมน์ Sepal.Length ของข้อมูล iris มากกว่า 4 หรือไม่ และแสดงผลลัพธ์

   4.5 ใช้คำสั่ง `reduce()` เพื่อคำนวณผลคูณของค่าทั้งหมดในคอลัมน์ hp ของข้อมูล mtcars และแสดงผลลัพธ์

   4.6 ใช้คำสั่ง `reduce()` เพื่อรวมข้อมูลใน iris (เฉพาะคอลัมน์ตัวเลข) ให้เป็นคอลัมน์เดียวโดยการบวกค่า

   4.7 สร้างคอลัมน์ใหม่ชื่อว่า accum_mpg และใช้คำสั่ง `accumulate()` เพื่อคำนวณผลรวมสะสมของค่าในคอลัมน์ mpg ของข้อมูล mtcars แล้วตรวจสอบว่าผลรวมสะสมครั้งแรกที่มีค่ามากกว่า 50 คือค่าใด

รายการอ้างอิง

Andrews, M. (2021). *Doing Data Science in R: An Introduction for Social Scientists*: SAGE Publications.

Chambers, J. M. (1998). *Programming with Data: A Guide to the S Language*: Springer.

Chang, W. (2012). *R Graphics Cookbook: Practical Recipes for Visualizing Data*: O'Reilly Media, Inc.

Crawley, M. J. (2012). *The R book*: John Wiley & Sons.

Davies, T. M. (2016). *The Book of R: A First Course in Programming and Statistics*: No Starch Press.

Dowle, M. (2018). Rdatatable/data.table. Retrieved from https://github.com/Rdatatable/data.table/wiki

Hothorn, T., & Everitt, B. S. (2009). *A Handbook of Statistical Analyses Using R, Second Edition*: Taylor & Francis.

Koushik, R. B., & Ravindran, S. K. (2016). *R Data Science Essentials*: Packt Publishing Ltd.

Matloff, N. (2011). *The Art of R Programming: A Tour of Statistical Software Design*: No Starch Press.

Scott, D. W. (2015). *Multivariate Density Estimation: Theory, Practice, and Visualization*: Wiley.

Wickham, H. (2016). *ggplot2: elegant graphics for data analysis*: Springer.

Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*: O'Reilly Media.

Wilkinson, L., Wills, D., Rope, D., Norton, A., & Dubbs, R. (2006). *The Grammar of Graphics*: Springer New York.

Zhao, Y. (2012). *R and data mining: Examples and case studies*: Academic Press.

ดัชนี (Index)

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

ก

ข

ค

ช

ต

ธ

ฟ

ภ