R

Emmanuel Paradis, Chantima Piyapong, Julien Claude

R สำหรับผู้เริ่มต้น

Emmanuel Paradis[1] (translation: Chantima Piyapong[2], edition: Julien Claude[1])

1: *Institut des Sciences de l'Évolution*
Université Montpellier
F-34095 Montpellier cédex 05
France

E-mail: *emmanuel.paradis@ird.fr*

2: Department of Biology, Faculty of Science
Burapha University
Chonburi, 20131
Thailand

E-mail: *chantimap@buu.ac.th*

กิตติกรรมประกาศ

การแปลเอกสาร R สำหรับผู้เริ่มต้นฉบับนี้สำเร็จลุล่วงได้เนื่องจากได้รับการสนับสนุนจากโครงการวิจัยเรื่อง Analysis of Biodiversity in R ภายใต้ความร่วมมือด้านอุดมศึกษาและการวิจัยระหว่างไทย-ฝรั่งเศส ประจำปี พ.ศ. 2558–2559 (PHC SIAM Franco-Thai Analysis of Biodiversity in R: 2015–2016) จึงขอขอบพระคุณอย่างสูงสำหรับความร่วมมือ ฯ นี้ รวมทั้งขอขอบพระคุณ พันเอกรุ่งศักดิ์ จิตต์แก้ว ผศ.ดร.วรางคณา กิ้มปาน และ ผศ.ดร.สำรวม บัวประดิษฐ์ ที่ได้ให้คำแนะนำและข้อคิดเห็นต่างๆ อันเป็นประโยชน์อย่างยิ่งในการแปลเอกสารในครั้งนี้ และขอขอบคุณสมาชิกทีมผู้ช่วยในการแปลเอกสารนี้ ได้แก่ นางสาวปิยะรักษ์ ประดับเพชรรัตน์ นางสาวจิราพัชร แบนมาก นางสาวกรองทอง ตั้งสิทธิ และนายสันติ สวนลา ที่ช่วยในการตรวจแก้ไขทำให้เอกสารนี้สำเร็จได้ด้วยดี สุดท้ายนี้หากมีข้อผิดพลาดประการใดทางคณะผู้จัดทำยินดีแก้ไขและปรับปรุงให้ดียิ่งขึ้นในการจัดทำฉบับปรับปรุงถัดไป

สารบัญ

1 คำนำ

วัตถุประสงค์ของการจัดทำเอกสารนี้เพื่อเป็นจุดตั้งต้นสำหรับผู้ที่เริ่มสนใจใน R ผู้เขียนเน้นไปที่การทำเข้าใจว่า R ทำงานอย่างไร โดยกลุ่มเป้าหมายคือผู้ใช้ R เบื้องต้นมากกว่าผู้ที่มีความเชี่ยวชาญใน R โดยข้อดีของ R ที่ให้กับผู้ใช้นั้นมีอยู่มาก ดังนั้นการเข้าใจแนวคิดและหลักการจะเป็นประโยชน์ต่อผู้เริ่มใช้ R ในการที่จะใช้ R ได้ดีขึ้นโดยง่าย ผู้เขียนพยายามใช้คำอธิบายที่เข้าใจง่ายให้มากที่สุดสำหรับทุกคนขณะเดียวกันผู้เขียนได้ให้รายละเอียดที่เป็นประโยชน์และบางครั้งรวมถึงตารางด้วย

R เป็นระบบหนึ่งสำหรับการวิเคราะห์ทางสถิติและภาพกราฟิกที่คิดค้นโดย Ross Ihaka และ Robert Genteleman[1] R เป็นทั้งซอฟต์แวร์และภาษาคอมพิวเตอร์ที่ได้รับการพัฒนามาจากภาษา S ที่ถูกคิดค้นโดยห้องปฏิบัติการ AT&T Bell S สามารถนำมาใช้ได้ในฐานะที่เป็นซอฟต์แวร์ทางพาณิชย์ของ S-PLUS โดย Insightful[2] รูปแบบของ R และ S มีความแตกต่างกันอย่างมากสำหรับผู้ที่ประสงค์จะทราบข้อมูลความแตกต่างนี้สามารถอ่านได้จากเอกสารโดย Ihaka & Gentleman (1996) หรือ R-FAQ[3] ซึ่งเป็นเอกสารฉบับหนึ่งที่เผยแพร่โดย R

R ถูกออกแบบมาเพื่อให้ใช้งานฟรีภายใต้เงื่อนไขของ *GNU General Public Licence*[4] ซึ่งได้มีการพัฒนาและเผยแพร่โดยนักสถิติมากมายซึ่งรู้จักกันในนาม *R Development Core Team*

R สามารถถูกใช้งานได้ในหลายรูปแบบเนื่องจากแหล่งต้นทาง (sources) ของ R ส่วนใหญ่ถูกเขียนจากภาษา C และ บางส่วนในภาษา Fortran ซึ่งเป็นภาษาคอมพิวเตอร์ที่จำเป็นสำหรับเครื่องที่เป็นระบบ Unix และ Linux หรือจากการคอมไพล์รหัสเลขฐานสอง (pre-compiled binaries) สำหรับ Windows, Linux และ Macintosh เพื่อติดตั้ง R จำเป็นต้องมีไฟล์เหล่านี้ไม่ว่าจะมาจากจากแหล่งต้นทางหรือจากการคอมไพล์รหัสเลขฐานสอง ซึ่งถูกเผยแพร่ในแหล่งข้อมูลอินเทอร์เน็ตของ *Comprehensive R Archive Network* (CRAN)[5] ที่มีคู่มือการติดตั้งให้ด้วย จากการสร้างของ Linux (Debian,. . .) เลขฐานสองเป็นรูปแบบภาษาคอมพิวเตอร์ที่ถูกใช้งานโดยทั่วไป สำหรับเวอร์ชันที่เป็นปัจจุบันดูได้ที่ CRAN ถ้าจำเป็น

[1] Ihaka R. & Gentleman R. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5: 299–314.

[2] ดู http://www.insightful.com/products/splus/default.asp for more information

[3] http://cran.r-project.org/doc/FAQ/R-FAQ.html

[4] สำหรับข้อมูลเพิ่มเติม http://www.gnu.org/

[5] http://cran.r-project.org/

R มีฟังก์ชันมากมายสำหรับการวิเคราะห์ทางสถิติและภาพกราฟิก โดยภาพกราฟิกนี้ถูกเห็นได้ในทันทีในวินโดว์ (window) ของมันเองและสามารถถูกบันทึกได้ในหลายรูปแบบ (format) ตัวอย่างเช่น jpg, png, bmp, ps, pdf, emf, pictex, xfig ซึ่งรูปแบบที่มีอาจขึ้นอยู่กับระบบปฏิบัติการ ส่วนผลจากการวิเคราะห์ทางสถิติถูกแสดงผ่านหน้าจอโดยผลการวิเคราะห์ที่ปรากฏในทันทีบางค่า เช่น (P-values, regression coefficients, residuals,. . .) สามารถถูกบันทึก เขียนขึ้นเป็นไฟล์หรือถูกใช้เพื่อการวิเคราะห์ต่อไป

ภาษา R ให้ผู้ใช้สามารถโปรแกรมได้ ตัวอย่างเช่น โปรแกรมเป็นแบบทำซ้ำหรือแบบเป็นรอบ (loop) เพื่อวิเคราะห์ชุดของข้อมูลหลายชุดได้อย่างต่อเนื่อง รวมทั้งยังสามารถรวมฟังก์ชันทางสถิติที่แตกต่างกันหลายฟังก์ชันให้อยู่ในโปรแกรมเดียวเพื่อแสดงการวิเคราะห์ขั้นสูงมากขึ้นได้ ผู้ใช้ R อาจได้ประโยชน์จากโปรแกรมเป็นจำนวนมากที่ถูกเขียนขึ้นสำหรับ S[6] และสามารถหาได้จากในอินเทอร์เน็ตซึ่งโปรแกรมเหล่านี้ส่วนใหญ่สามารถถูกใช้ได้โดยตรงกับ R

ในระยะแรก R อาจดูเหมือนซับซ้อนเกินไปในการใช้งานสำหรับผู้ที่ไม่เชี่ยวชาญในการใช้ R ซึ่งแท้ที่จริงแล้วไม่เป็นความจริง ตามจริงแล้วคุณลักษณะที่โดดเด่นของ R คือความยืดหยุ่นของ R ในขณะที่ซอฟต์แวร์ดั้งเดิมแสดงผลของการวิเคราะห์ทันทีแต่ว่า R เก็บผลการวิเคราะห์เหล่านี้ในรูปของ “วัตถุหรืออ็อบเจ็กต์” (“object”) เพื่อที่ว่าการวิเคราะห์สามารถทำให้เรียบร้อยโดยที่ไม่ต้องแสดงผลออกมา ผู้ใช้อาจจะประหลาดใจในกรณีนี้แต่ด้วยคุณลักษณะนี้ของ R ถือได้ว่าเป็นประโยชน์อย่างมากเนื่องจากตามความเป็นจริงแล้วผู้ใช้ R สามารถคัดลอกเฉพาะส่วนที่เป็นผลจากการวิเคราะห์ที่สนใจได้เลย ตัวอย่างเช่น ถ้าผู้ใช้ R คนหนึ่งทำการวิเคระห์ชุดข้อมูลของสมการถดถอย (regression) 20 ชุด และต้องการเปรียบเทียบค่าสัมประสิทธิ์การถดถอย (regression coefficient) ที่แตกต่างกันแล้ว R สามารถแสดงผลเพียงค่าสัมประสิทธิ์ที่ถูกประมาณเท่านั้น โดยผลนี้อาจจะมีหนึ่งบรรทัดในขณะที่ซอฟต์แวร์ดั้งเดิมอาจแสดงผลผ่าน 20 หน้าของหน้าต่าง (Window) เราสามารถเห็นตัวอย่างอื่นๆ อีกที่แสดงให้เห็นถึงความยืดหยุ่นของระบบเช่นที่มีใน R เมื่อเปรียบเทียบกับซอฟต์แวร์ดั้งเดิม

[6]ตัวอย่างเช่น http://stat.cmu.edu/S/

2 หลักการบางอย่างก่อนการเริ่มต้น

ทันทีที่ R ได้ถูกติดตั้งบนเครื่องคอมพิวเตอร์และเมื่อเปิดโปรแกรม ซอฟต์แวร์ถูกดำเนินการ โดยจะปรากฏสัญลักษณ์ “>” คือ prompt ที่เป็นค่าเริ่มต้นหรือค่าโดยปริยาย (default) เพื่อบ่งบอกว่า R กำลังรอคำสั่งอยู่ ภายใต้ระบบปฏิบัติการวินโดว์ (Window) ใช้คำสั่ง Rgui.exe (โดยบางคำสั่งซึ่งถูกเข้าถึงได้จากการใช้ on-line help การเปิดไฟล์ หรือด้วยวิธีอื่นๆ) สามารถถูกดำเนินการได้ผ่านทางการเลือกลักษณะที่เกิดจากการคลิกในรายการเลือกแบบดึงลง (pull-down menus) ที่ขั้นตอนนี้ผู้ไม่เคยใช้ R มาก่อนอาจจะสงสัยว่า “ตอนนี้ฉันทำอะไรอยู่” ดังนั้นการเข้าใจถึงแก่นบางอย่างว่า R ทำงานอย่างไรจะเป็นประโยชน์อย่างมากเมื่อ R ถูกใช้เป็นครั้งแรกและต่อไปนี้เป็นสิ่งที่เราจะทำความเข้าใจเกี่ยวกับ R

เริ่มแรกเราจะทำความเข้าใจโดยย่อว่า R ทำงานอย่างไร และจากนั้นผู้เขียนจะอธิบายถึงตัวดำเนินการที่ถูกกำหนดให้ (“assign” operator) ซึ่งยอมให้สร้างอ็อบเจ็กต์ ต่อมาเกี่ยวกับการจัดการอ็อบเจ็กต์ในหน่วยความจำ (memory) ทำอย่างไร และสุดท้ายการใช้ on-line help ทำได้อย่างไรซึ่งเป็นประโยชน์อย่างมากเมื่อใช้งานใน R

2.1 R ทำงานอย่างไร

ความจริงที่ว่า R เป็นภาษาหนึ่งของคอมพิวเตอร์อาจจะขัดขวางผู้ใช้ R บางคนที่คิดว่า “ฉันไม่สามารถเขียนโปรแกรมได้” ข้อนี้อาจไม่เป็นความจริงด้วยเหตุผลสองประการ ประการแรก R เป็นภาษาโปรแกรมที่ทำงานแบบทีละคำสั่งหรือคำสั่งต่อคำสั่ง (interpreted language) ไม่ใช่ภาษาที่แปลทีเดียวตั้งแต่ต้นจนจบ (compiled language) นั่นหมายความว่าคำสั่ง (command) ของ R ทั้งหมดที่ถูกพิมพ์บนคีย์บอร์ด จะถูกดำเนินการโดยตรงซึ่งไม่ต้องสร้างโปรแกรมที่สมบูรณ์ตั้งแต่ต้นจนจบแล้วจึงคอมไพล์โปรแกรม อย่างเช่นในภาษาคอมพิวเตอร์ส่วนใหญ่ เช่น C, Fortran, Pascal เป็นต้น

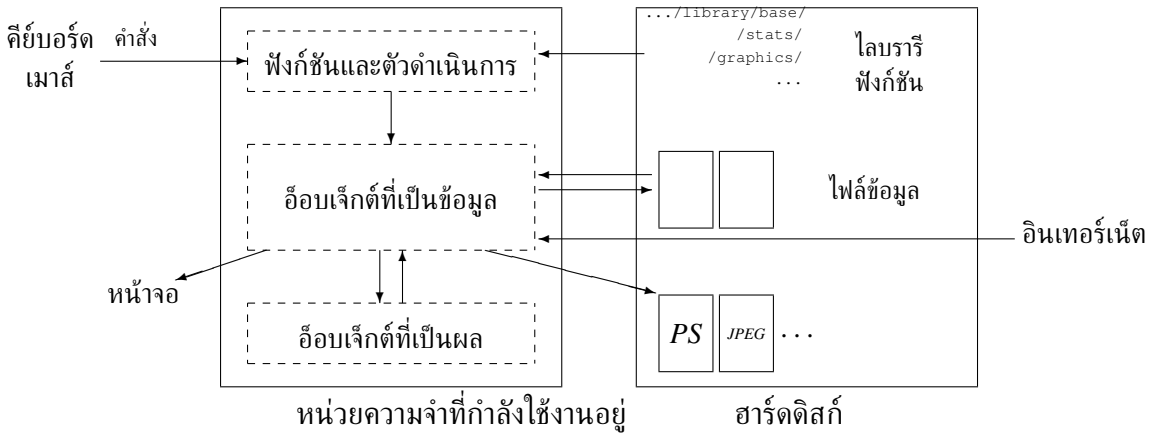
ประการที่สอง syntax ของ R มีความเรียบง่ายและเป็นธรรมชาติมาก ตัวอย่างเช่น ในสมการถดถอยเชิงเส้น (linear regression) สามารถใช้คำสั่ง `lm(y ~ x)` วิเคราะห์ได้ซึ่งหมายถึง “การกำหนดโมเดลเชิงเส้นโดยที่ y เป็นตัวแปรตามและ x เป็นตัวแปรพยากรณ์หรือตัวแปรอิสระ” ใน R เพื่อให้ถูกดำเนินการได้ ฟังก์ชันจำเป็นต้องถูกเขียนภายในเครื่องหมายวงเล็บ (parentheses) *เสมอ* หรือแม้กระทั่งไม่มีฟังก์ชันเลยใน parentheses เช่น `ls()` ถ้าผู้ใช้ R พิมพ์เพียงแต่ชื่อของฟังก์ชันโดยไม่มี parentheses แล้ว R จะแสดงเนื้อหาของฟังก์ชันนั้นๆ แทน ในเอกสารนี้ชื่อฟังก์ชันโดยทั่วไปถูกเขียนอยู่กับ parentheses เพื่อที่จะได้ใช้แยกแยะฟังก์ชันเหล่านี้ออกจากอ็อบเจ็กต์อื่นๆ ยกเว้นว่ามีข้อความระบุชัดเจนว่าเป็นอย่างไร

ขณะที่ R กำลังทำงานอยู่ ตัวแปร ข้อมูล ฟังก์ชัน ผลและอื่นๆ ถูกเก็บในหน่วยความจำที่กำลังใช้งานอยู่ (active memory) ของเครื่องคอมพิวเตอร์ในรูปแบบของ *อ็อบเจ็กต์* ซึ่งมีชื่อ *name* ผู้ใช้ R สามารถดำเนินการกระทำกับอ็อบเจ็กต์เหล่านี้ด้วยตัวดำเนินการ (*operator*) เช่น ทางเลขคณิต(arithmetic) ทางตรรกะ (logical) ทางการเปรียบเทียบ (comparison) และอื่นๆ เป็นต้น และด้วยฟังก์ชันต่างๆ (*functions*) ซึ่งเป็นอ็อบเจ็กต์โดยตัวมันเอง การใช้ตัวดำเนินการเหล่านี้ค่อนข้างเป็นธรรมชาติซึ่งเราจะได้เห็นในรายละเอียดถัดไป (หน้า 30) ฟังก์ชันใน R ฟังก์ชันหนึ่งอาจมีโครงสร้างดังต่อไปนี้

อาร์กิวเมนต์สามารถเป็นอ็อบเจ็กต์ได้ เช่น ข้อมูล (data) สูตร (formulae) นิพจน์ (expressions) เป็นต้น อ็อบเจ็กต์บางอ็อบเจ็กต์ นี้อาจถูกนิยามโดยปริยายในฟังก์ชันเนื่องจากค่าปริยายอาจถูกปรับเปลี่ยนโดยผู้ใช้ซึ่งทำการกำหนดในรายละเอียดเอง ฟังก์ชันใน R ฟังก์ชันหนึ่งอาจไม่ต้องการอาร์กิวเมนต์ก็ได้ เนื่องจากอาร์กิวเมนต์ทั้งหมดถูกนิยามโดยปริยายหากไม่กำหนดเป็นค่าอื่น และค่าเหล่านี้อาจถูกเปลี่ยนแปลงได้ด้วยตัวเลือก (options) หรืออาจเนื่องมาจากไม่มีอาร์กิวเมนต์ที่ได้นิยามไว้ในฟังก์ชัน รายละเอียดเพิ่มเติมเราสามารถดูได้ในส่วนการใช้และการสร้างฟังก์ชันทำได้อย่างไรในหน้า 80) การอธิบายในที่นี้เพียงพอสำหรับการทำความเข้าใจว่า R ทำงานอย่างไร

ปฏิบัติการทั้งหมดของ R ถูกกระทำในรูปแบบอ็อบเจ็กต์ ซึ่งถูกเก็บในหน่วยความจำที่กำลังใช้งานอยู่ (active memory) ของเครื่องคอมพิวเตอร์โดยไม่มีการใช้ไฟล์ชั่วคราวดังรูปที่ 1 การอ่านและการเขียนของไฟล์ (file) ที่ถูกใช้สำหรับการรับ (input) และการแสดงผล (ouput) ของข้อมูลและผลลัพธ์ (ภาพกราฟิกหรืออื่นๆ) ผู้ใช้ R กระทำการตามฟังก์ชันผ่านทางคำสั่งบางคำสั่ง ผลที่ได้ถูกแสดงผ่านหน้าจอโดยตรงโดยถูกเก็บในรูปของอ็อบเจ็กต์หรือถูกเขียนลงบนแผ่นดิสก์(โดยเฉพาะในส่วนภาพกราฟิก) เนื่องจากผลลัพธ์ (results) จะเป็นอ็อบเจ็กต์โดยตัวมันเอง ดังนั้นผลนี้สามารถถูกทำให้เป็นข้อมูลและนำไปวิเคราะห์ได้ ไฟล์ข้อมูลสามารถถูกอ่านจาก local disk ซึ่งเป็น hard disk ภายในเครื่องคอมพิวเตอร์ที่ใช้งานหรือจาก การเข้าถึงเครื่องแม่ข่ายระยะไกล (remote server) จากอินเทอร์เน็ต

ฟังก์ชันที่ผู้ใช้ R ใช้งานได้ถูกเก็บในไลบรารี (library) ที่อยู่บนดิสก์ในไดเรกทอรี่ (directory) ซึ่งเรียกว่า R_HOME/library (R_HOME คือ directory ที่ R ถูกติดตั้ง) ไดเรกทอรี่นี้ประกอบด้วยแพ็กเกจ (*packages*) ของฟังก์ชันต่างๆ ซึ่งได้ถูกสร้างขึ้นเองในไดเรกทอรี่โดยแพ็กเกจหลักชื่อว่า base และประกอบไปด้วยฟังก์ชันพื้นฐานของภาษาคอมพิวเตอร์ R โดยเฉพาะในการอ่านและการจัดการข้อมูล แต่ละแพ็กเกจมีไดเรกทอรี่ที่มีชื่อ R กับชื่อไฟล์คล้ายกับแพ็กเกจ ตัวอย่างเช่น แพ็กเกจชื่อว่า base มีไฟล์คือ R_HOME/library/base/R/base



รูปที่ 1: ภาพแผนผังแสดงว่า R ทำงานอย่างไร

โดยไฟล์นี้มีฟังก์ชันทั้งหมดของแพ็กเกจนี้

หนึ่งในชุดคำสั่งที่ง่ายที่สุดคือการพิมพ์ชื่อของอ็อบเจ็กต์หนึ่งเพื่อแสดงเนื้อหาของมัน ตัวอย่างเช่น ถ้า อ็อบเจ็กต์หนึ่งคือ `n` มีค่าคือ 10 การแสดงผลจะเป็นดังนี้

```
> n
[1] 10
```

อธิบายได้ว่า ตัวเลข 1 ในเครื่องหมายวงเล็บเหลี่ยม (bracket) เป็นตัวบ่งบอกถึงจุดเริ่มต้นของการแสดงค่าที่เป็นองค์ประกอบหรืออีลีเมนท์ (element) แรกของ n ในคำสั่งนี้เป็นการใช้ที่บอกเป็นนัยว่าหมายถึงการใช้ฟังก์ชัน `print` และในตัวอย่างข้างต้นมีความคล้ายคลึงกับ `print(n)` ซึ่งในบางสถานการณ์ฟังก์ชัน `print` ต้องถูกใช้อย่างชัดเจนอย่างเช่นภายในฟังก์ชันหนึ่งหรือแบบวนรอบ (loop) หนึ่ง

สำหรับชื่อของอ็อบเจ็กต์ ตัวอักษรแรกต้องเริ่มด้วย ตัวใดตัวหนึ่งของ A–Z หรือ a–z และสามารถประกอบด้วยตัวอักษร ตัวเลข (0-9) จุด (.) และเครื่องหมาย underscores (_) ได้ R แสดงความแตกต่างระหว่างตัวอักษรตัวพิมพ์ใหญ่และตัวอักษรตัวพิมพ์เล็กในชื่อของอ็อบเจ็กต์ได้ ดังนั้น `x` และ `X` สามารถเป็นชื่อของอ็อบเจ็กต์สองอ็อบเจ็กต์ที่แตกต่างกัน (แม้แต่การทำงานภายใต้ระบบปฏิบัติการวินโดว์)

2.2 การสร้าง การทำรายการและการลบ อ็อบเจ็กต์ในหน่วยความจำ

อ็อบเจ็กต์หนึ่งสามารถถูกสร้างด้วยตัวดำเนินการที่กำหนดให้ (“assign"" operator) ซึ่งถูกเขียนด้วยสัญลักษณ์ลูกศรกับเครื่องหมายลบ และเครื่องหมายวงเล็บเหลี่ยมซึ่งสัญลักษณ์นี้สามารถเริ่มจากซ้ายไปขวาหรือกลับกันได้ ดังตัวอย่าง

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

ถ้าอ็อบเจ็กต์นั้นมีอยู่แล้วค่าของอ็อบเจ็กต์ก่อนหน้านี้จะถูกลบไปซึ่งการเปลี่ยนแปลงมีผลเฉพาะ อ็อบเจ็กต์ในหน่วยความจำที่กำลังใช้งานอยู่ไม่มีผลต่อข้อมูลบนดิสก์ ค่าที่ถูกกำหนดไว้ด้วยวิธีนี้จะเป็นผลลัพธ์ของการดำเนินการและ/หรือฟังก์ชันหนึ่งดังตัวอย่าง

```
> n <- 10 + 2
> n
[1] 12
> n <- 3 + rnorm(1)
> n
[1] 2.208807
```

จากตัวอย่างข้างต้น ฟังก์ชัน `rnorm(1)` ซึ่งสร้างตัวแปรสุ่มปกติ (normal random variate) กับ mean zero และ variance unity (หน้า 21) ข้อสังเกตคือผู้ใช้ R สามารถเพียงแต่พิมพ์นิพจน์ (expression) หนึ่งโดยไม่มีการกำหนดค่าของนิพจน์ไปยังอ็อบเจ็กต์ได้ ซึ่งผลลัพธ์จะถูกแสดงยังหน้าจอแต่ไม่ถูกเก็บไว้ในหน่วยความจำ ดังตัวอย่าง

```
> (10 + 2) * 5
[1] 60
```

จากตัวอย่างข้างต้น การกำหนดค่าให้ (assignment) จะถูกละไว้ถ้าไม่มีความจำเป็นเพื่อการทำความเข้าใจ

ฟังก์ชัน `ls` เป็นฟังก์ชันที่ทำบัญชีรายชื่ออย่างง่ายของ อ็อบเจ็กต์ ในหน่วยความจำโดยรายชื่อของอ็อบเจ็กต์เท่านั้นที่ถูกแสดง ดังตัวอย่าง

```
> name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5
> ls()
[1] "m"    "n1"   "n2"   "name"
```

ข้อสังเกตสำหรับการใช้เครื่องหมาย semi-colon เพื่อแยกชุดคำสั่งที่ต่างกันในบรรทัดเดียวกัน หากต้องการทำรายชื่อของอ็อบเจ็กต์ ซึ่งประกอบไปด้วยอักขระหรืออักษรที่ให้มา (given character) เท่านั้นในชื่อของอ็อบเจ็กต์ ด้วยตัวเลือกนี้สามารถใช้ ฟังก์ชัน `pattern` ซึ่งสามารถเขียนย่อว่า `pat` ได้ ดังตัวอย่าง

```
> ls(pat = "m")
[1] "m"    "name"
```

สำหรับกรณีที่ต้องการจำกัดรายชื่อของอ็อบเจ็กต์ที่มีชื่อขึ้นต้นด้วยตัวอักขระนี้ สามารถทำได้ดังต่อไปนี้

```
> ls(pat = "^m")
[1] "m"
```

ส่วนฟังก์ชัน `ls.str` เป็นฟังก์ชันที่แสดงรายละเอียดบางอย่างของอ็อบเจ็กต์ในหน่วยความจำ ดังตัวอย่าง

```
> ls.str()
m :  num 0.5
n1 :  num 10
n2 :  num 100
name :  chr "Carmen"
```

ตัวเลือกสำหรับฟังก์ชัน `pattern` สามารถถูกใช้ในลักษณะเดียวกันกับฟังก์ชัน `ls` ตัวเลือกหนึ่งที่เป็นประโยชน์ของฟังก์ชัน `ls.str` คือ `max.level` ซึ่งเจาะจงไปที่ระดับของรายละเอียดสำหรับการแสดง composite อ็อบเจ็กต์ สำหรับค่าโดยปริยายหากไม่กำหนดไว้เป็นอย่างอื่น ฟังก์ชัน `ls.str` แสดงรายละเอียดอ็อบเจ็กต์ทั้งหมดในหน่วยความจำ รวมทั้งคอลัมน์ (column) ของ data frame, matrices และ lists ซึ่งอาจจะแสดงผลที่ยาวมาก เราสามารถเลี่ยงการแสดงรายละเอียดผลเหล่านี้ทั้งหมดโดยใช้ฟังก์ชันตัวเลือกคือ `max.level = -1` ดังต่อไปนี้

```
> M <- data.frame(n1, n2, m)
> ls.str(pat = "M")
M : 'data.frame':       1 obs. of  3 variables:
 $ n1: num 10
 $ n2: num 100
 $ m : num 0.5
> ls.str(pat="M", max.level=-1)
M : 'data.frame':       1 obs. of  3 variables:
```

เพื่อจะลบอ็อบเจ็กต์ออกไปจากหน่วยความจำนั้นเราสามารถใช้ ฟังก์ชัน `rm`: `rm(x)` ลบ อ็อบเจ็กต์ `x` ในขณะที่ `rm(x,y)` ลบทั้ง อ็อบเจ็กต์ `x` และ อ็อบเจ็กต์ `y` ส่วน `rm(list=ls())` ลบ อ็อบเจ็กต์ทั้งหมดในหน่วยความจำ และสำหรับฟังก์ชัน `ls()` ที่สามารถถูกใช้เพื่อลบอ็อบเจ็กต์บางอ็อบเจ็กต์ที่ต้องการเลือกลบคือการใช้คำสั่ง `rm(list=ls(pat="^m"))`

2.3 การช่วยเหลือการใช้งานออนไลน์ (on-line help)

On-line help ของ R ให้ข้อมูลที่เป็นประโยชน์อย่างมากว่าฟังก์ชันใช้งานได้อย่างไร โดยการช่วยเหลือนี้ใช้ได้โดยตรงผ่านทางฟังก์ชันที่ให้ไว้ ยกตัวอย่างดังต่อไปนี้

```
> ?lm
```

ซึ่งฟังก์ชันนี้จะแสดงภายใน R ในหน้าช่วยเหลือ (help page) ของฟังก์ชัน `lm()` (*linear model*) สำหรับคำสั่ง `help(lm)` และ `help("lm")` ให้ผลแบบเดียวกันกับ `?lm` และคำสั่งสุดท้ายที่มีความจำเป็นในการเข้าถึงการช่วยเหลือโดยใช้อักขระแบบพิเศษ (non-conventional characters) ดังต่อไปนี้

```
> ?*
Error: syntax error
> help("*")
Arithmetic            package:base            R Documentation

Arithmetic Operators
...
```

การเรียกขอความช่วยเหลือจะเปิดหน้าแสดงผลอีกหน้า (ทั้งนี้ขึ้นอยู่กับระบบปฏิบัติการ) ด้วยข้อมูลทั่วไปในบรรทัดแรกเช่น ชื่อของแพ็กเกจซึ่งเป็นฟังก์ชันหรือตัวดำเนินการที่เขียนบันทึกไว้ และต่อจากหัวข้อตามด้วยส่วนที่เป็นข้อมูลที่ให้รายละเอียดเกี่ยวกับหัวข้อนั้น ดังต่อไปนี้

Description: เป็นการบรรยายโดยย่อ

Usage: การใช้งานของฟังก์ชันตามชื่อที่ให้ไว้กับอากิวเมนต์ของมันทั้งหมดและตัวเลือกที่เป็นไปได้ (ร่วมกับค่าโดยปริยายหากไม่ได้กำหนดเป็นค่าอื่นที่ตรงกัน) สำหรับตัวดำเนินการนั้นให้การใช้แบบมาตรฐาน

Arguments: อากิวเมนต์สำหรับฟังก์ชันหนึ่งที่มีรายละเอียดของแต่ละอากิวเมนต์

Details: การบรรยายลงในรายละเอียด

Value: ถ้ามี เป็นชนิดของอ็อบเจ็กต์ที่ถูกรีเทิร์นค่ากลับโดยฟังก์ชันหรือตัวดำเนินการ

See Also: หน้าช่วยเหลืออื่นๆ ที่ใกล้เคียงหรือคล้ายกับหน้าที่กำลังดูอยู่

Examples: ตัวอย่างบางอย่างซึ่งสามารถถูกดำเนินการได้ทั่วไปโดยไม่ต้องเปิดหน้าช่วยเหลือ โดยการใช้ฟังก์ชัน `example`

สำหรับผู้เริ่มต้นแล้วเป็นเรื่องที่ดีในการเข้าไปดูที่ส่วนของ **Examples** โดยทั่วไปแล้วมันมีประโยชน์ที่จะอ่านอย่างละเอียดในส่วนของ **Arguments** ในส่วนอื่นๆ ที่ผู้อ่านพบและน่าจะอ่านเช่นกัน ได้แก่ **Note**, **References** หรือ **Author(s)**

โดยปริยาย (by default) หากไม่กำหนดเป็นค่าอื่นแล้ว ฟังก์ชัน `help` ค้นหาเฉพาะแพ็กเกจที่ถูกใส่เข้ามาไว้ในหน่วยความจำ ตัวเลือกคือ `try.all.packages` ซึ่งค่าโดยปริยายคือ `FALSE` ที่ยอมให้ทำการค้นหา แพ็กเกจทั้งหมด ถ้าค่าของของฟังก์ชันเป็น `TRUE` ดังตัวอย่างต่อไปนี้

```
> help("bs")
No documentation for 'bs' in specified packages and libraries:
you could try 'help.search("bs")'
> help("bs", try.all.packages = TRUE)
Help for topic 'bs' is not in any loaded package but
can be found in the following packages:

  Package               Library
  splines               /usr/lib/R/library
```

ข้อสังเกตในกรณีนี้คือหน้าช่วยเหลือ (help page) ของฟังก์ชัน `bs` ไม่ถูกแสดง ผู้ใช้สามารถแสดงหน้าช่วยเหลือจากแพ็กเกจหนึ่งที่ยังไม่ได้ถูกใส่เข้าในหน่วยความจำโดยการใช้ `package` ตัวเลือกดังต่อไปนี้

```
> help("bs", package = "splines")
bs              package:splines             R Documentation

B-Spline Basis for Polynomial Splines

Description:

    Generate the B-spline basis matrix for a polynomial spline.
...
```

การช่วยเหลือ (help) ที่อยู่ในรูปแบบ html (html format) ให้อ่านจาก Netscape (ตัวอย่าง) ซึ่งถูกเรียกดูได้โดยการพิมพ์ฟังก์ชันต่อไปนี้

```
> help.start()
```

การค้นหาผ่านคำสำคัญ (keywords) สามารถทำได้โดยการใช้ html help ดังกล่าวข้างต้น ในส่วนของ **See Also** นี้มี hypertext links เชื่อมไปยังหน้าช่วยเหลือของฟังก์ชันอื่น การค้นหานี้ด้วยคำสำคัญสามารถทำได้เช่นกันใน R โดยใช้ฟังก์ชัน `help.search` โดยหลังจากนั้นค้นหาหัวข้อที่เฉพาะเจาะจงให้เป็นข้อความอักขระ (character string) ในหน้าช่วยเหลือของ packages ทั้งหมดที่ได้ติดตั้งไว้ในเครื่องคอมพิวเตอร์ ตัวอย่างเช่น ฟังก์ชัน `help.search("tree")` จะแสดงรายการของฟังก์ชันซึ่งในหน้าช่วยเหลือกล่าวถึง “tree” ข้อสังเกตคือถ้า แพ็กเกจบางแพ็กเกจเพิ่งได้ถูกลงติดตั้งใหม่ จะเป็นประโยชน์ถ้าทำการ refresh ฐานข้อมูลใหม่ที่ถูกใช้โดย `help.search` ผ่านฟังก์ชันตัวเลือกคือ `rebuild` ตัวอย่างเช่น `help.search("tree", rebuild=TRUE)`

ฟังก์ชัน `apropos` ใช้ค้นหาฟังก์ชันทั้งหมดซึ่งชื่อประกอบด้วยข้อความอักขระที่ถูกกำหนดให้เป็นอาร์กิวเมนต์และ แพ็กเกจที่ได้ถูกนำเข้าแล้วในหน่วยความจำเท่านั้นที่จะถูกค้นหาตามตัวอย่างดังต่อไปนี้

```
> apropos(help)
[1] "help"          ".helpForCall" "help.search"
[4] "help.start"
```

3 ข้อมูลด้วย R

3.1 อ็อบเจ็กต์

เราได้ทราบแล้วว่า R ทำงานด้วยอ็อบเจ็กต์ ซึ่งนอกจากว่าถูกกำหนดลักษณะโดยชื่อและเนื้อหาของมันแล้วยังถูกกำหนดโดย *ลักษณะเฉพาะ* (*attributes*) ซึ่งระบุชนิดของข้อมูลซึ่งถูกทำให้เป็นตัวแทนของอ็อบเจ็กต์หนึ่ง เพื่อให้เข้าใจประโยชน์ของลักษณะเฉพาะเหล่านี้ให้พิจารณาตัวแปรหนึ่งที่มีค่าเป็น 1, 2 หรือ 3 ตัวแปรที่มีค่านี้อาจจะเป็นตัวแปรจำนวนเต็มตัวอย่างเช่น จำนวนไข่ที่อยู่ในรัง หรือการทำรหัสของ categorical variable ตัวอย่างเช่นเพศในประชากรบางประชากรของสัตว์กลุ่ม crustaceans ชนิดหนึ่ง ซึ่งประกอบด้วย เพศผู้ เพศเมียหรือกะเทย

เป็นที่ชัดเจนว่าการวิเคราะห์สถิติของตัวแปรข้างต้นจะไม่เหมือนกันในทั้งสองกรณี อธิบายได้ว่าใน R ลักษณะเฉพาะของอ็อบเจ็กต์ให้ข้อมูลที่จำเป็นหรืออธิบายได้มากกว่านี้ว่าในทางเทคนิคหรือโดยทั่วไปแล้วการกระทำของฟังก์ชันต่ออ็อบเจ็กต์หนึ่งขึ้นอยู่กับลักษณะเฉพาะของที่ตามมาในภายหลัง

ทุกๆ อ็อบเจ็กต์มีสองลักษณะเฉพาะภายในที่สำคัญ (*intrinsic attributes*) คือ *โหมด* (*mode*) และ *ความยาว* (*length*) โหมดคือชนิดพื้นฐานขององค์ประกอบหรืออีลีเมนท์ (elements) ของอ็อบเจ็กต์ ซึ่งมีอยู่สี่โหมดหลักๆ คือ ตัวเลข (numeric) ตัวอักษรหรืออักขระ (character) คอมเพล็กซ์ (complex)[7] และ ทางตรรกะ (logical (`FALSE` หรือ `TRUE`)) ส่วนโหมดอื่นๆ ก็มีแต่โหมดเหล่านี้ไม่ได้เป็นตัวแทนข้อมูล ตัวอย่างเช่น ฟังก์ชันหรือนิพจน์ สำหรับความยาวคือ จำนวนของ elements ของอ็อบเจ็กต์ เพื่อที่จะแสดงโหมดและความยาวของอ็อบเจ็กต์หนึ่ง ต่อไปนี้เป็นอ็อบเจ็กต์ที่ใช้ฟังก์ชัน `mode` และ `length` ตามลำดับ

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
```

[7] โหมดคอมเพล็กซ์จะไม่ถูกกล่าวถึงในเอกสารนี้

```
[1] "complex"
```

อะไรก็ตามที่เป็นโหมด ข้อมูลที่หายไป (missing data) ถูกแทนด้วย `NA` (*not available*) ส่วนค่าตัวเลขที่มีจำนวนที่ใหญ่มากจะถูกกำหนดด้วยสัญกรณ์ยกกำลังหรือเลขชี้กำลัง (exponential notation) ดังต่อไปนี้

```
> N <- 2.1e23
> N
[1] 2.1e+23
```

R แสดงค่าตัวเลขที่ไม่จำกัด (non-finite numeric values) ให้เหมาะสม เช่น $\pm\infty$ ด้วย `Inf` และ `-Inf` หรือค่าที่ไม่เป็นจำนวนด้วย `NaN` (*not a number*) ดังต่อไปนี้

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

ค่าหนึ่งของโหมดอักขระคือ input กับ double quotes (") โดยที่เป็นไปได้ที่รวมอักขระที่ตามมาภายหลังในค่านั้นถ้าค่านั้นตามหลังเครื่องหมาย backslash (\) เมื่อสอง character นี้ (\")อยู่ด้วยกันจะถูกทำให้เป็นวิธีเฉพาะโดยฟังก์ชันบางฟังก์ชัน เช่น ฟังก์ชัน `cat` สำหรับแสดงบนหน้าจอ หรือฟังก์ชัน `write.table` ที่เขียนบนดิสก์ (หน้า 17 มีฟังก์ชัน `qmethod` เป็นตัวเลือกของฟังก์ชันนี้)

```
> x <- "Double quotes \" delimitate R's strings."
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.
```

อีกทางเลือกหนึ่ง ตัวแปรของโหมดอักขระสามารถถูกทำให้เป็น delimited ด้วย single quotes (') ในกรณีนี้เครื่องหมาย double quotes ไม่ต้องตามหลังเครื่องหมาย backslashes แต่ต้องมีเครื่องหมาย single quotes ดังตัวอย่างต่อไปนี้

```
> x <- 'Double quotes " delimitate R\'s strings.'
> x
[1] "Double quotes \" delimitate R's strings."
```

ตารางต่อไปนี้แสดงภาพรวมชนิดของอ็อบเจ็กต์ที่แสดงข้อมูล

อ็อบเจ็กต์	โหมด	อ็อบเจ็กต์เดียวกันเป็นได้หลายโหมดหรือไม่
vector	numeric, character, complex หรือ logical	ไม่ใช่
factor	numeric หรือ character	ไม่ใช่
array	numeric, character, complex หรือ logical	ไม่ใช่
matrix	numeric, character, complex หรือ logical	ไม่ใช่
data frame	numeric, character, complex หรือ logical	ใช่
ts	numeric, character, complex หรือ logical	ไม่ใช่
list	numeric, character, complex, logical, function, expression, ...	ใช่

เวกเตอร์ (vector) ถือว่าเป็นตัวแปรชนิดหนึ่งในความหมายที่เป็นที่ยอมรับโดยทั่วไป ส่วน factor เป็น categorical variable ชนิดหนึ่ง สำหรับอาร์เรย์ (array) หมายถึงตารางที่มี k dimensions และ เมทริกซ์(matrix) ก็คือกรณีเฉพาะของอาร์เรย์ที่ $k = 2$ ข้อสังเกตคือ elements ของอาร์เรย์หนึ่งหรือของเมทริกซ์หนึ่ง คือโหมดเดียวกันทั้งหมด กรณีกรอบข้อมูล (data frame) คือตารางหนึ่งที่ประกอบด้วย หนึ่งเวกเตอร์หรือหลายเวกเตอร์ และ/หรือ factors ทั้งหมดที่มีความยาวเท่ากัน แต่อาจจะเป็นของโหมดที่แตกต่างกัน 'ts' เป็นชุดข้อมูลที่เป็น time series ดังนั้นจึงประกอบด้วยลักษณะเฉพาะ (attributes) เพิ่มเติมเช่น ความถี่และวันต่างๆ สุดท้าย list สามารถประกอบด้วยชนิดใดก็ได้ของอ็อบเจ็กต์รวมถึง lists ด้วย

สำหรับเวกเตอร์หนึ่ง โหมดและความยาวของมันเพียงพอที่จะอธิบายข้อมูลนั้นๆ สำหรับ อ็อบเจ็กต์และสารสนเทศ (information) อื่นๆ มีความจำเป็นและถูกกำหนดโดยลักษณะเฉพาะภายในที่ไม่สำคัญ (*non-intrinsic* attributes) โดยระหว่างลักษณะเฉพาะเหล่านี้ เราสามารถอ้างถึงฟังก์ชัน *dim* ซึ่งตรงกับมิติ (dimensions) ของอ็อบเจ็กต์หนึ่ง ตัวอย่างเช่น เมทริกซ์หนึ่งที่ประกอบด้วยสองบรรทัดและสองคอลัมน์มีฟังก์ชัน `dim` เป็นหนึ่งคู่ของค่า [2,2] แต่ความยาวของมันคือ 4

3.2 การอ่านข้อมูลในไฟล์

สำหรับการอ่านและการเขียนในไฟล์ R ใช้ working directory เพื่อที่จะหา working directory นี้ คำสั่ง `getwd()` (*get working directory*) จะถูกใช้ และ working directory จะถูกเปลี่ยนไปโดย `setwd("C:/data")` หรือ (`setwd("/home/~paradis/R")`) มันมีความจำเป็นที่ต้องให้เส้นทาง (path) ไปยังไฟล์ถ้าไฟล์นั้นไม่ได้อยู่ใน working directory[8]

[8]Under Windows, it is useful to create a short-cut of Rgui.exe then edit its properties and change the directory in the field "Start in:" under the tab "Short-cut": this directory will then be the working

R สามารถอ่านข้อมูลที่ถูกเก็บใน text (ASCII) files ด้วยฟังก์ชันดังนี้ `read.table` (ซึ่งมีหลาย variants ดูตามข้างล่าง) `scan` และ `read.fwf` R สามารถอ่านไฟล์ในรูปแบบอื่นได้ด้วย ตัวอย่างเช่น Excel, SAS, SPSS เป็นต้น และ R สามารถเข้าถึงฐานข้อมูลประเภท SQL แต่ฟังก์ชันที่จำเป็นนี้ไม่อยู่ใน แพ็กเกจ base ฟังก์ชันเหล่านี้เป็นประโยชน์มากสำหรับการใช้ R ในขั้นสูงมากขึ้น อย่างไรก็ตามเราจะจำกัดการใช้เหล่านี้โดยการอ่านไฟล์ในรูปแบบ ASCII

ฟังก์ชัน `read.table` มีผลสำหรับการสร้าง data frame และนี่เป็นวิธีสำคัญที่จะอ่านข้อมูลในรูปแบบตาราง (tabular form) ต่อไปนี้เป็นตัวอย่างที่สมมติว่าไฟล์หนึ่งมีชื่อว่า name.dat และมีคำสั่งดังนี้

```
> mydata <- read.table("data.dat")
```

จากข้างบน R จะสร้าง data frame มีชื่อว่า `mydata` และแต่ละตัวแปร (variable) จะถูกให้ชื่อโดยปริยาย (by default) เช่น `V1, V2, ...` และสามารถทำให้เข้าถึงแต่ละตัวแปรโดยคำสั่ง `mydata$V1, mydata$V2,...` หรือเช่น `mydata["V1"], mydata["V2"], ...`หรือโดยคำสั่ง `mydata[,1], mydata[,2],...`[9] มีหลายตัวเลือกที่เป็นค่าปริยาย ตัวอย่างเช่น ค่าเหล่านี้ถูกใช้โดย R ถ้าค่าเหล่านี้ถูกละไว้โดยผู้ใช้ตามรายละเอียดในตารางต่อไปนี้

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
         row.names, col.names, as.is = FALSE, na.strings = "NA",
           colClasses = NA, nrows = -1,
         skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#")
```

directory if R is started from this short-cut.

[9]มีข้อแตกต่างดังต่อไปนี้ `mydata$V1` และ `mydata[, 1]` เป็นเวกเตอร์ ขณะที่ `mydata["V1"]` เป็นกรอบข้อมูล (data frame) โดยสามารถเห็นรายละเอียดมากกว่านี้ในการจัดการอ็อบเจ็กต์ในหน้า 22

file	ชื่อของไฟล์ (ภายในเครื่องหมาย "" หรือตัวแปรหนึ่งของโหมดอักขระ) อาจจะมีทางที่ไปถึงไฟล์นั้นด้วย (สัญลักษณ์ \ ไม่สามารถใช้ได้ต้องถูกแทนที่ด้วยสัญลักษณ์ / แม้ภายใต้ระบบปฏิบัติการวินโดว์) หรือการเข้าถึงไฟล์จากทางไกลประเภท URL (http://....)
header	การบ่งบอกทางตรรกะ (FALSE หรือ TRUE) ถ้าไฟล์นั้นประกอบด้วยชื่อของตัวแปรในบรรทัดแรกของตัวไฟล์นั้น
sep	ตัวแยกเขตข้อมูลที่ถูกใช้ในไฟล์นั้น ตัวอย่างเช่น sep="\t" ถ้าไฟล์นั้นเป็นตาราง
quote	อักขระที่ใช้ในการอ้างถึงตัวแปรของโหมดอักขระ
dec	อักขระที่ใช้สำหรับจุดทศนิยม
row.names	เวกเตอร์หนึ่งซึ่งมีชื่อของบรรทัดซึ่งอาจเป็นเวกเตอร์ของโหมดอักขระหรืออาจเป็นตัวเลข (หรือชื่อ) ของตัวแปรหนึ่งของไฟล์ (โดยปริยาย: 1, 2, 3, ...)
col.names	เวกเตอร์หนึ่งซึ่งมีชื่อของตัวแปร (โดยปริยาย: V1, V2, V3, ...)
as.is	ควบคุมการเปลี่ยนของตัวแปรอักขระตามปัจจัย (ถ้า FALSE) หรือยังคงตัวแปรเป็นอักขระ (TRUE) ซึ่ง as.is สามารถเป็นได้ทั้งเวกเตอร์ตรรกะ ตัวเลขหรืออักขระซึ่งระบุให้ตัวแปรถูกเก็บเป็นอักขระ
na.strings	ค่าที่กำหนดให้เป็นข้อมูลที่หายไป (ถูกเปลี่ยนเป็น NA)
colClasses	เวกเตอร์ของโหมดอักขระที่กำหนดให้คลาสเป็นลักษณะเฉพาะไปที่คอลัมน์
nrows	จำนวนสูงสุดของบรรทัดที่อ่านค่าได้ (ค่าลบถูกละทิ้ง)
skip	จำนวนของบรรทัดที่ถูกข้ามไปก่อนการอ่านข้อมูล
check.names	ถ้าเป็นจริง ตรวจสอบว่าตัวแปรเป็นชื่อที่ใช้ได้สำหรับ R
fill	ถ้าเป็นจริงและทุกบรรทัดไม่มีจำนวนเดียวกันของตัวแปร “blanks” ถูกเพิ่ม
strip.white	ถ้าเป็นจริง (เงื่อนไขต่อ sep) ให้ลบช่องว่างเพิ่มเติมก่อนและหลังตัวแปรอักขระ
blank.lines.skip	ถ้าเป็นจริงละทิ้งบรรทัดว่าง (“blank”)
comment.char	อักขระหนึ่งที่นิยามคำอธิบายในไฟล์ข้อมูล ที่เหลือของบรรทัดภายหลังอักขระนี้ถูกละทิ้ง (ทำให้อาร์กิวเมนต์นี้ไม่สามารถใช้ได้ ใช้ comment.char = "")

ฟังก์ชันที่แตกต่างกันของ read.table เป็นประโยชน์เนื่องจากฟังก์ชันเหล่านี้มีค่าปริยายที่แตกต่างกันดังต่อไปนี้

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, ...)
read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          fill = TRUE, ...)
read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".",
           fill = TRUE, ...)
read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",",
            fill = TRUE, ...)
```

ฟังก์ชัน scan เป็นฟังก์ชันที่มีความยืดหยุ่นมากกว่าฟังก์ชัน read.table ความแตกต่างอย่างหนึ่งคือฟังก์ชัน scan สามารถระบุโหมดของตัวแปรดังตัวอย่าง

```
> mydata <- scan("data.dat", what = list("", 0, 0))
```

จากคำสั่งข้างบนเป็นการอ่านในไฟล์ data.dat มี 3 ตัวแปร ตัวแปรแรกคือโหมดอักขระ และตัวแปรสองตัวที่อยู่ถัดมาคือโหมดตัวเลข ลักษณะพิเศษที่สำคัญอีกอย่างหนึ่งของฟังก์ชัน `scan()` คือสามารถถูกใช้เพื่อสร้างอ็อบเจ็กต์เวกเตอร์ เมทริกซ์ data frames, list และอื่นๆ ที่แตกต่างกันได้ ในตัวอย่างข้างบน `mydata` เป็น list หนึ่งของ 3 เวกเตอร์ โดยค่าปริยายนี้เป็นสิ่งที่ละไว้ ฟังก์ชัน `scan()` สร้างเวกเตอร์ตัวเลข ถ้าการอ่านข้อมูลไม่สอดคล้องกับรูปแบบที่คาดหมาย (ไม่เป็นค่าปริยายหรือไม่เป็นค่าที่ถูกระบุไว้อย่างใดอย่างหนึ่ง) ข้อความผิดพลาด (error message) ถูกตอบกลับมา ต่อไปนี้เป็นตัวเลือก

`file`	ชื่อของไฟล์ (ภายในเครื่องหมาย `""`)อาจจะมีทางที่ไปถึงไฟล์นั้นด้วย (สัญลักษณ์ \ ไม่สามารถใช้ได้ต้องถูกแทนที่ด้วยสัญลักษณ์ / แม้ภายใต้ระบบปฏิบัติการวินโดว์) หรือการเข้าถึงไฟล์จากทางไกลประเภท URL (http://....) ถ้า `file=""`แล้วข้อมูลถูกนำเข้าผ่านทางคีย์บอร์ด (การนำเข้าสิ้นสุดลงโดยบรรทัดว่าง)
`what`	ระบุโหมดของข้อมูล (ค่าปริยายคือโหมดตัวเลข)
`nmax`	จำนวนของข้อมูลที่อ่านหรือถ้า `what` เป็นรายการ (list) หนึ่ง จำนวนของบรรทัดที่อ่าน (โดยค่าปริยาย `scan` อ่านข้อมูลถึงจุดสิ้นสุดของไฟล์)
`n`	จำนวนของข้อมูลที่อ่าน (ไม่มีข้อจำกัดสำหรับค่าปริยาย)
`sep`	ตัวแยกเขตข้อมูลที่ถูกใช้ในไฟล์
`quote`	อักขระที่ถูกใช้ในการอ้างอิงตัวแปรของโหมดอักขระ
`dec`	อักขระที่ถูกใช้เป็นจุดทศนิยม
`skip`	จำนวนของบรรทัดที่ถูกข้ามไปก่อนการอ่านข้อมูล
`nlines`	จำนวนของบรรทัดที่อ่าน
`na.string`	ค่าที่กำหนดให้เป็นข้อมูลที่หายไป (ถูกเปลี่ยนเป็น `NA`)
`flush`	โหมดอักขระหนึ่ง ถ้าเป็นจริง ดูบรรทัดถัดไปทันทีเมื่อถึงจำนวนคอลัมน์ที่ได้รับ (ยอมให้ผู้ใช้ R เพิ่มคำอธิบายในไฟล์ข้อมูล)
`fill`	ถ้าเป็นจริงและทุกบรรทัดไม่มีจำนวนเดียวกันของตัวแปร “blanks” ถูกเพิ่ม
`strip.white`	ถ้าเป็นจริง (เงื่อนไขต่อ `sep`) ให้ลบช่องว่างเพิ่มเติมก่อนและหลังตัวแปรอักขระ
`quiet`	โหมดตรรกะ ถ้าเป็นเท็จ พิจารณาบรรทัดที่แสดงส่วนที่ถูกอ่าน
`blank.lines.skip`	ถ้าเป็นจริงละทิ้งบรรทัดว่าง (“blanks”)
`multi.line`	ถ้า `what` คือรายการหนึ่ง ให้ระบุ ถ้าตัวแปรของตัวเดียวกันอยู่บนบรรทัดเดียวในไฟล์ (เป็นเท็จ)
`comment.char`	อักขระหนึ่งที่นิยามคำอธิบายในไฟล์ข้อมูล ที่เหลือของบรรทัดภายหลังอักขระนี้ถูกละทิ้ง (ค่าปริยายคือปิดการใช้งาน)
`allowEscapes`	ระบุว่าการหลีกเลี่ยง C-style escapes (ตัวอย่างเช่น ‘\t’) ถูกดำเนินการหรือไม่ (ค่าปริยาย) หรือถูกอ่านตามทุกตัวอักษร

ฟังก์ชัน `read.fwf` สามารถถูกใช้เพื่ออ่านไฟล์หนึ่งได้ โดยข้อมูลบางอย่างอยู่ในรูป *fixed width format* ดังตัวอย่าง

```
read.fwf(file, widths, header = FALSE, sep = "\t",
         as.is = FALSE, skip = 0, row.names, col.names,
```

```
         n = -1, buffersize = 2000, ...)
```

ตัวเลือกของฟังก์ชันนี้เหมือนกับฟังก์ชัน `read.table()` ยกเว้น**ความกว้าง** (`widths`) ซึ่งระบุให้ความกว้างของ fields (`buffersize` เป็นจำนวนบรรทัดสูงสุดที่อ่านพร้อมกัน) ตัวอย่างเช่น ถ้าไฟล์หนึ่งที่ให้ชื่อว่า data.text มีข้อมูลตามที่บอกทางด้านขวามือ ข้อมูลนี้สามารถถูกอ่านได้ด้วยคำสั่งดังต่อไปนี้

```
A1.501.2
A1.551.3
B1.601.4
B1.651.5
C1.701.6
C1.751.7
```

```
> mydata <- read.fwf("data.txt", widths=c(1, 4, 3))
> mydata
  V1   V2  V3
1  A 1.50 1.2
2  A 1.55 1.3
3  B 1.60 1.4
4  B 1.65 1.5
5  C 1.70 1.6
6  C 1.75 1.7
```

3.3 การบันทึกข้อมูล

ฟังก์ชัน `write.table` เขียนไฟล์หนึ่งเป็นอ็อบเจ็กต์ ซึ่งโดยทั่วไปเป็นแบบกรอบข้อมูล (data frame) อย่างไรก็ตามฟังก์ชันนี้อาจเขียนเป็น อ็อบเจ็กต์ชนิดอื่นได้ เช่น vector, matrix เป็นต้น โดยที่อาร์กิวเมนต์และตัวเลือกคือ

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
          eol = "\n", na = "NA", dec = ".", row.names = TRUE,
          col.names = TRUE, qmethod = c("escape", "double"))
```

x	ชื่อของอ็อบเจ็กต์ที่ถูกเขียนขึ้น
file	ชื่อของไฟล์ (โดยค่าปริยาย อ็อบเจ็กต์ถูกแสดงบนหน้าจอ)
append	ถ้าเป็นจริงเพิ่มข้อมูลโดยปราศจากการลบข้อมูลเหล่านี้ที่อาจจะคงอยู่ในไฟล์
quote	เวกเตอร์เชิงตรรกะหรือเชิงตัวเลข ถ้าเป็นจริงตัวแปรของโหมดอักขระและปัจจัยถูกเขียนภายในเครื่องหมาย "" มิฉะนั้นเวกเตอร์เชิงตัวเลขบ่งบอกถึงจำนวนของตัวแปรที่เขียนภายใน เครื่องหมาย "" (ในทั้งสองกรณีชื่อของตัวแปรถูกเขียนภายในเครื่องหมาย "" ถ้าไม่ได้เขียนเป็น quote = FALSE)
sep	ตัวแยกเขตข้อมูลที่ถูกใช้ในไฟล์
eol	อักขระที่ถูกใช้เมื่อสิ้นสุดในแต่ละบรรทัด ("\n" คือ ตัวนำการนำย้อนกลับ)
na	อักขระที่ถูกใช้สำหรับข้อมูลที่หายไป
dec	อักขระที่ถูกใช้สำหรับจุดทศนิยม
row.names	โหมดตรรกะที่บ่งบอกว่าชื่อของบรรทัดถูกเขียนในไฟล์หรือไม่
col.names	id. สำหรับชื่อของคอลัมน์
qmethod	การกำหนดเฉพาะ ถ้า quote=TRUE เครื่องหมาย double quotes " ซึ่งรวมตัวแปรของโหมดอักขระถูกกระทำอย่างไร ถ้า "escape" (หรือ "e", เป็นค่าปริยาย) แต่ละ " ถูกแทนที่โดย \" ถ้า แต่ละ "d" ถูกแทนที่โดย ""

เพื่อเขียนอ็อบเจ็กต์สำหรับไฟล์หนึ่งในวิธีที่ง่ายขึ้น คำสั่ง `write(x,file="data.txt")` สามารถถูกใช้ได้โดยที่ x คือชื่อของอ็อบเจ็กต์ ซึ่งอาจจะเป็น vector, matrix หรือ array หนึ่ง) ที่มีสองตัวเลือกให้ใช้คือ nc (หรือ ncol) ซึ่งกำหนดจำนวนของคอลัมน์ในไฟล์นั้น (ค่าโดยปริยายคือ `nc = 1` ถ้า x เป็นของโหมดอักขระ, `nc = 5` สำหรับโหมดอื่นๆ) และใช้ append (a logical) เพื่อเพิ่มข้อมูลโดยปราศจากการลบข้อมูลที่อาจมีอยู่แล้วในไฟล์นั้น (TRUE) หรือการลบข้อมูลถ้าไฟล์นั้นมีอยู่ก่อนแล้ว (FALSE, ซึ่งเป็นค่าโดยปริยาย)

เพื่อบันทึกกลุ่มของอ็อบเจ็กต์ของชนิดใดก็ตาม เราสามารถใช้คำสั่ง `save(x,y,z,file="xyz.RData")` เพื่อลบการโอนย้ายของข้อมูลระหว่างเครื่องคอมพิวเตอร์ที่ต่างกันได้สะดวกขึ้น ตัวเลือกที่-ใช้ได้คือ `ascii = TRUE` (ข้อมูลนี้ซึ่งตอนนี้ถูกเรียกว่า *workspace* ใน R's jargon) สามารถถูกส่งเข้าหน่วยความจำได้ภายหลังด้วยฟังก์ชัน `load("xyz.RData")` ส่วนฟังก์ชัน `save.image()` เป็น shortcut เพื่อบันทึก `(list=ls(all=TRUE), file=".RData")`

3.4 การสร้างข้อมูล

3.4.1 ลำดับปกติ (regular sequences)

ลำดับปกติของจำนวนเต็ม (integers) ตัวอย่างเช่น จาก 1 ถึง 30 สามารถถูกสร้างได้ดังนี้

```
> x <- 1:30
```

จากผลดังกล่าวข้างต้นได้ว่า x มี 30 องค์ประกอบหรือ elements ส่วน operator ':' มีลำดับความสำคัญเป็นพิเศษในการเป็นตัวดำเนินการเลขคณิต (arithmetic operators) ก่อน ภายในนิพจน์ดังต่อไปนี้

```
> 1:10-1
 [1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

ฟังก์ชัน seq สามารถสร้างลำดับของจำนวนจริงได้ดังต่อไปนี้

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

จากฟังก์ชันข้างบน ตัวเลขแรกบ่งบอกถึงลำดับเริ่มต้นส่วนตัวเลขที่สองหมายถึงตัวเลขสิ้นสุดของลำดับและตัวเลขที่สามหมายถึงจำนวนตัวเลขที่เพิ่มขึ้นที่ถูกใช้ในการสร้างลำดับ นอกจากนั้นยังสามารถใช้ฟังก์ชันดังต่อไปนี้ได้

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

นอกจากนี้สามารถพิมพ์ค่าที่ใช้โดยตรงโดยการใช้ฟังก์ชัน c ดังนี้

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

เป็นไปได้เช่นกันถ้าผู้ใช้ต้องการใส่ข้อมูลบางอย่างบนคีย์บอร์ดโดยใช้ฟังก์ชัน scan โดยเพียงแค่ใช้ตัวเลือกค่าปริยาย ดังตัวอย่างต่อไปนี้

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10:
Read 9 items
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

ตัวอย่างข้างล่างเป็นฟังก์ชันrep ที่สร้างเวกเตอร์หนึ่งโดยมี elements ที่เหมือนกันทั้งหมด

```
> rep(1, 30)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

ฟังก์ชัน sequence สร้างอนุกรมของลำดับของจำนวนเต็มโดยที่ตัวเลขสิ้นสุดแต่ละตัวคือจำนวนที่ให้ไว้ตามอาร์กิวเมนต์ดังนี้

```
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(c(10,5))
 [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
```

ฟังก์ชัน `gl` (*generate levels*) เป็นฟังก์ชันที่มีประโยชน์มากเนื่องจากเป็นฟังก์ชันที่สร้างอนุกรมปกติของ factors การใช้งานของฟังก์ชันนี้คือ `gl(k,n)` โดยที่ `k` คือจำนวนของ levels (หรือ classes) และ `n` เป็นการทำซ้ำในแต่ละ level มีสองตัวเลือกอาจถูกที่ใช้ได้คือ ใช้ `length` เพื่อระบุจำนวนตัวเลขของข้อมูลที่ถูกสร้างและ `labels` เพื่อระบุชื่อของ levels ของ factors ดังตัวอย่างต่อไปนี้

```
> gl(3, 5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(3, 5, length=30)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels:  1 2 3
> gl(2, 6, label=c("Male", "Female"))
 [1] Male   Male   Male   Male   Male   Male
 [7] Female Female Female Female Female Female
Levels:  Male Female
> gl(2, 10)
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels:  1 2
> gl(2, 1, length=20)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels:  1 2
> gl(2, 2, length=20)
 [1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels:  1 2
```

สุดท้ายฟังก์ชัน `expand.grid()` สร้างกรอบข้อมูลร่วมกับการรวมของเวกเตอร์หรือของ factors ทั้งหมดที่ถูกกำหนดให้ตามอาร์กิวเมนต์ ดังตัวอย่างต่อไปนี้

```
> expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
   h   w    sex
1 60 100   Male
2 80 100   Male
3 60 300   Male
4 80 300   Male
```

```
5 60 100 Female
6 80 100 Female
7 60 300 Female
8 80 300 Female
```

3.4.2 ลำดับแบบสุ่ม (random sequences)

กฎ (law)	**ฟังก์ชัน**
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

มันเป็นประโยชน์ในทางสถิติที่สามารถสร้างข้อมูลแบบสุ่มได้และ R สามารถทำแบบนั้นได้สำหรับจำนวนขนาดใหญ่ของฟังก์ชันความหนาแน่นและความน่าจะเป็น ฟังก์ชันเหล่านี้อยู่ในรูปของ `r`*`func`*`(n, p1, p2, ...)` โดยที่ *`func`* บ่งบอกถึงการแจกแจงความน่าจะเป็น (probability distribution) ส่วน `n` เป็นจำนวนของข้อมูลที่ถูกสร้างและ `p1, p2, ...`เป็นค่าพารามิเตอร์ของการแจกแจงนั้น จากตารางข้างบนได้ให้รายละเอียดของแต่ละการแจกแจงและค่าปริยายที่เป็นไปได้ (ถ้าไม่มีค่าปริยายบ่งบอกไว้นั่นหมายความว่าพารามิเตอร์ต้องถูกระบุโดยผู้ใช้ R)

ฟังก์ชันส่วนใหญ่มีฟังก์ชันที่เป็นคู่กันหรือใกล้เคียงกันโดยได้มาจากการแทนที่ตัวอักษร `r` ด้วยตัวอักษร `d`, `p` หรือ `q` ได้ตามลำดับดังนี้คือ the probability density (`d`*`func`*`(x,`

`...`)), the cumulative probability density(`pfunc(x, ...)`) และ the value of quantile (`qfunc(p, ...)`, with $0 < p < 1$) สองชุดสุดท้ายของฟังก์ชันสามารถถูกใช้เพื่อหาค่าวิกฤตหรือค่า P-values ของการทดสอบทางสถิติ ตัวอย่างเช่น ค่าวิกฤตสำหรับการทดสอบแบบสองทาง (two-tailed) ตามการแจกแจงตามปกติ (normal distribution) ที่ระดับนัยสำคัญเท่ากับ 0.05 ดังตัวอย่างต่อไปนี้

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

สำหรับกรณีการทดสอบแบบเดียวกันในรูปแแบบทางเดียว (one-tailed) สามารถใช้ `qnorm(0.05)` หรือใช้ `1-qnorm(0.95)` ซึ่งการใช้ขึ้นอยู่กับรูปแบบของสมมติฐานตัวเลือก (alternative hypothesis)
สำหรับค่า P-value ของการทดสอบหนึ่งกล่าวได้ว่า $\chi^2 = 3.84$ ที่ $df = 1$ ดังตัวอย่างข้างล่างนี้

```
> 1 - pchisq(3.84, 1)
[1] 0.05004352
```

3.5 การจัดการอ็อบเจ็กต์ (Manipulating objects)

3.5.1 การสร้างอ็อบเจ็กต์

เราได้เห็นวิธีที่แตกต่างกันก่อนหน้านี้ในการสร้างอ็อบเจ็กต์โดยการใช้ assign operator ซึ่งเป็นโหมดและชนิดของอ็อบเจ็กต์ ที่โดยทั่วไปถูกสร้างตามที่กำหนดให้แบบแน่นอน มีความเป็นไปได้ในการสร้างอ็อบเจ็กต์หนึ่งและระบุโหมด ความยาว ชนิดและอื่นๆ ให้กับอ็อยเจ็กต์นั้น การเข้าถึงในการสร้างอ็อบเจ็กต์นี้เป็นเรื่องน่าสนใจในภาพรวมของการจัดการอ็อบเจ็กต์ ตัวอย่างเช่น ผู้ใช้ R สามารถสร้าง 'empty'อ็อบเจ็กต์ และต่อจากนั้นดัดแปลงอีลีเมนท์ของมันให้สำเร็จได้ซึ่งมีประสิทธิภาพมากกว่าการใส่อีลีเมนท์ทั้งหมดไปด้วยกันด้วยฟังก์ชัน `c()` สำหรับระบบการทำดัชนี (indexing system) อาจถูกใช้ได้ในส่วนนี้ตามที่เราจะได้เห็นต่อไป (หน้า 32)

การสร้างอ็อบเจ็กต์จากอ็อบเจ็กต์อื่นมักมีความสะดวกมากกว่า ตัวอย่างเช่น ถ้าผู้ใช้ R ต้องการสร้างอนุกรมของแบบจำลองที่เหมาะสม สามารถทำได้ง่ายที่จะใส่สูตรในรายการและถัดจากนั้นสกัดหรือดึงอีลีเมนท์ ออกมาให้สำเร็จเพื่อที่จะใส่อีลีเมนท์เหล่านั้นในฟังก์ชัน `lm`

ในขั้นตอนนี้ของการเรียนรู้ของเราในเรื่อง R ความสนใจในการเรียนรู้การใช้ฟังก์ชันต่อจากนี้ไม่เพียงแต่ใช้งานในทางปฏิบัติได้จริงแต่ยังสามารถนำไปใช้ในการสอนได้ด้วย การสร้างอ็อบเจ็กต์อย่างรู้จริงทำให้สามารถเข้าใจโครงสร้างเหล่านี้ได้และทำให้เราไปได้ไกลมาก

ขึ้นในเรื่องแนวคิดบางอย่างของ R ที่ได้กล่าวไว้ก่อนหน้านี้

Vector ฟังก์ชัน `vector` เป็นฟังก์ชันที่มี 2 อาร์กิวเมนต์คือ mode (โหมด) และ length (ความยาว) การสร้างเวกเตอร์หนึ่งซึ่งอีลีเมนท์มีค่าหนึ่งที่ขึ้นอยู่กับโหมดที่ถูกระบุให้เป็นอาร์กิวเมนต์หนึ่ง ดังนี้ 0 ถ้าเป็นตัวเลข `FALSE` ถ้าเป็นทางตรรกะ หรือ `""` ถ้าเป็นอักขระ ฟังก์ชันต่อไปนี้ให้ผลแบบเดียวกันอย่างแน่นอน และมีสำหรับอาร์กิวเมนต์เดียวที่มีความยาวของเวกเตอร์คือ `numeric()`, `logical()` และ `character()`

Factor factor หนึ่งไม่ได้หมายรวมถึงเพียงค่าของตัวแปรที่เป็น corresponding categorical เท่านั้นแต่หมายรวมถึง levels ที่เป็นไปได้ต่าง ๆ ของตัวแปรนั้น (หรือแม้แต่พวกมันไม่ปรากฏในข้อมูล) ฟังก์ชัน `factor` สร้าง factor หนึ่งด้วยตัวเลือกดังต่อไปนี้

```
factor(x, levels = sort(unique(x), na.last = TRUE),
      labels = levels, exclude = NA, ordered = is.ordered(x))
```

จากข้างบน `levels` ระบุ levels ที่เป็นไปได้ของ factor (โดยค่าปริยาย ค่านี้มีความเฉพาะของเวกเตอร์ x) `labels` กำหนดชื่อของ levels `exclude` ค่าของ `x` หมายถึงการเอาออกไปจาก levels และ `ordered` เป็นอาร์กิวเมนต์ทางตรรกะในการระบุว่า levels ของ factor ถูกจัดเรียงลำดับหรือไม่ เพื่อระลึกหรือย้อนกลับไปดูว่า `x` เป็นของโหมดตัวเลขหรือโหมดอักขระ ต่อไปนี้เป็นบางตัวอย่าง

```
> factor(1:3)
[1] 1 2 3
Levels:  1 2 3
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels:  1 2 3 4 5
> factor(1:3, labels=c("A", "B", "C"))
[1] A B C
Levels:  A B C
> factor(1:5, exclude=4)
[1] 1  2  3  NA 5
Levels:  1 2 3 5
```

ฟังก์ชัน `levels` สกัด levels ที่เป็นไปได้ของ factor หนึ่ง ดังตัวอย่างต่อไปนี้

```
> ff <- factor(c(2, 4), levels=2:5)
> ff
[1] 2 4
```

```
Levels:  2 3 4 5
> levels(ff)
[1] "2" "3" "4" "5"
```

เมทริกซ์ Matrix เมทริกซ์หนึ่ง จริง ๆ แล้วคือเวกเตอร์หนึ่งที่มีลักษณะเฉพาะเพิ่มเติม (dim) ซึ่งโดยตัวมันเองคือ เวกเตอร์เชิงตัวเลขกับความยาวที่มีค่าเท่ากับ 2 และถูกนิยามว่าเป็นจำนวนของแถวและคอลัมน์ของเมทริกซ์นั้น เมทริกซ์หนึ่งสามารถถูกสร้างด้วยฟังก์ชัน `matrix` ดังตัวอย่างต่อไปนี้

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

โดยตัวเลือก ฟังก์ชัน `byrow` บ่งบอกว่าค่าที่ให้มาโดยคำสั่ง `data` ต้องใส่คอลัมน์ (โดยค่าปริยาย) หรือแถว (ถ้าเป็นจริง `TRUE`) ตามลำดับหรือไม่แสดงดังข้างล่าง โดยตัวเลือกฟังก์ชัน `dimnames` ทำให้แถวและคอลัมน์ถูกให้ชื่อ

```
> matrix(data=5, nr=2, nc=2)
     [,1] [,2]
[1,]    5    5
[2,]    5    5
> matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, 2, 3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

อีกวิธีหนึ่งในการสร้างเมทริกซ์หนึ่งคือการให้ค่าที่เหมาะสมต่อลักษณะเฉพาะ (dim) (ซึ่งเป็นค่า `NULL` ในตอนแรก)

```
> x <- 1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> dim(x)
NULL
> dim(x) <- c(5, 3)
> x
```

```
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

กรอบข้อมูล (Data frame) เราได้เห็นอย่างชัดเจนว่ากรอบข้อมูลถูกสร้างโดยฟังก์ชัน `read.table` มีความเป็นไปได้ที่จะสร้างกรอบข้อมูลด้วยฟังก์ชัน data.frame ดังนั้นเวกเตอร์ที่ประกอบอยู่ในกรอบข้อมูลต้องเป็น เวกเตอร์ที่มีความยาวเท่ากัน หรือถ้าเวกเตอร์หนึ่งเวกเตอร์ใดสั้นกว่า เวกเตอร์นั้นจะถูก “recycled” เป็นจำนวนทั้งหมดของครั้งต่อไปนี้คือตัวอย่าง

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
  x  n
1 1 10
2 2 10
3 3 10
4 4 10
> data.frame(x, M)
  x  M
1 1 10
2 2 35
3 3 10
4 4 35
> data.frame(x, y)
Error in data.frame(x, y) :
    arguments imply differing number of rows: 4, 3
```

ถ้า factor หนึ่งเป็นส่วนหนึ่งของกรอบข้อมูลแล้ว factor นี้ต้องมีความยาวเดียวกันกับเวกเตอร์ มีความเป็นไปได้ที่จะเปลี่ยนชื่อของคอลัมน์ อย่างเช่น ด้วยฟังก์ชัน `data.frame(A1=x, A2=n)` ผู้ใช้ R สามารถให้ชื่อของแถว โดยใช้อีกฟังก์ชันที่เป็นอีกหนึ่งตัวเลือกได้เช่นเดียวกันคือ `row.names` ซึ่งแน่นอนว่าต้องเป็นเวกเตอร์หนึ่งของโหมดอักขระและความยาวเท่ากับจำนวนบรรทัดของกรอบข้อมูล ท้ายที่สุดเป็นที่สังเกตว่ากรอบข้อมูลมีลักษณะเฉพาะ (dim) ที่มีความคล้ายคลึงกันกับเมทริกซ์

รายการ (List) รายการหนึ่งถูกสร้างขึ้นในทางที่คล้ายคลึงกับกรอบข้อมูลด้วยฟังก์ชัน `list` ไม่มีข้อจำกัดบนอ็อบเจ็กต์ซึ่งสามารถถูกรวบรวมไว้ได้ ในทางตรงกันข้ามกับฟังก์ชัน

data.frame() ชื่อต่างๆ ของอ็อบเจ็กต์ในฟังก์ชัน list ไม่ได้ถูกได้มาโดยค่าปริยาย ต่อไปนี้เป็นตัวอย่างการใช้ฟังก์ชัน list ที่มีเวกเตอร์ x และเวกเตอร์ y ของตัวอย่างก่อนหน้านี้

```
> L1 <- list(x, y); L2 <- list(A=x, B=y)
> L1
[[1]]
[1] 1 2 3 4

[[2]]
[1] 2 3 4

> L2
$A
[1] 1 2 3 4

$B
[1] 2 3 4

> names(L1)
NULL
> names(L2)
[1] "A" "B"
```

อนุกรมเวลา(Time-series) ฟังก์ชัน ts สร้างอ็อบเจ็กต์หนึ่งของ class "ts" จากเวกเตอร์หนึ่ง (single time-series) หรือเมทริกซ์หนึ่ง (multivariate time-series) และตัวเลือกบาง ตัวเลือกซึ่งอธิบายลักษณะอนุกรมนั้น ๆ ต่อไปนี้เป็นตัวเลือกกับค่าปริยาย

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class, names)
```

data	เวกเตอร์หรือเมทริกซ์
start	เวลาของการสังเกตครั้งแรก ซึ่งไม่เป็นตัวเลขก็เป็นเวกเตอร์ของจำนวนเต็มสองจำนวน (ดูตัวอย่างข้างล่าง)
end	เวลาของการสังเกตครั้งสุดท้ายซึ่งระบุเป็นวิธีเดียวกันกับ start
frequency	จำนวนของการสังเกตต่อหน่วยเวลา
deltat	เศษส่วนของช่วงเวลาของการสุ่มตัวอย่างระหว่างการสังเกตอย่างต่อเนื่อง (เช่น 1/12 ในข้อมูลรายเดือน) ซึ่งจะกำหนดเฉพาะ frequency หรือ deltat เท่านั้น
ts.eps	ความทนทานสำหรับการเปรียบเทียบของอนุกรม ความถี่ถูกพิจารณาว่าเท่ากันถ้าความแตกต่างของความถี่มีค่าน้อยกว่า ts.eps
class	คลาสที่ส่งผ่านไปยังอ็อบเจ็กต์ ค่าปริยายคือ "ts" สำหรับอนุกรมเดี่ยว และ c("mts", "ts") สำหรับอนุกรมหลายตัวแปร
names	เวกเตอร์ของโหมดอักขระพร้อมกับชื่อของอนุกรมแต่ละตัว ในกรณีของอนุกรมหลายตัวแปร ชื่อของคอลัมน์ ของ data หรือ Series 1, Series 2, ... จะเป็นค่าโดยปริยาย

ต่อไปนี้เป็นการยกตัวอย่างบางตัวอย่างที่ถูกสร้างขึ้นด้วยฟังก์ชัน ts:

```
> ts(1:10, start = 1959)
Time Series:
Start = 1959
End = 1968
Frequency = 1
 [1]  1  2  3  4  5  6  7  8  9 10
> ts(1:47, frequency = 12, start = c(1959, 2))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1959       1   2   3   4   5   6   7   8   9  10  11
1960  12  13  14  15  16  17  18  19  20  21  22  23
1961  24  25  26  27  28  29  30  31  32  33  34  35
1962  36  37  38  39  40  41  42  43  44  45  46  47
> ts(1:10, frequency = 4, start = c(1959, 2))
     Qtr1 Qtr2 Qtr3 Qtr4
1959         1    2    3
1960    4    5    6    7
1961    8    9   10
> ts(matrix(rpois(36, 5), 12, 3), start=c(1961, 1), frequency=12)
         Series 1 Series 2 Series 3
Jan 1961        8        5        4
```

```
Feb 1961        6        6        9
Mar 1961        2        3        3
Apr 1961        8        5        4
May 1961        4        9        3
Jun 1961        4        6       13
Jul 1961        4        2        6
Aug 1961       11        6        4
Sep 1961        6        5        7
Oct 1961        6        5        7
Nov 1961        5        5        7
Dec 1961        8        5        2
```

นิพจน์ (Expression) อ็อบเจ็กต์ของนิพจน์ของโหมดมีหน้าที่พื้นฐานหนึ่งใน R ซึ่งนิพจน์หนึ่งเป็นอนุกรมของอักขระที่เป็นเหตุเป็นผลสำหรับ R ชุดคำสั่งที่ใช้งานได้ทั้งหมดคือนิพจน์ เมื่อคำสั่งหนึ่งถูกพิมพ์โดยตรงบนคีย์บอร์ดคำสั่งนั้นจะถูกประเมิน*ประเมิน (evaluated)* โดย R และถูกดำเนินการ (executed) ถ้ามันเป็นคำสั่งที่ใช้การได้ ในหลายสถานการณ์มันเป็นประโยชน์ที่จะสร้างนิพจน์หนึ่งโดยปราศจากการประเมินมัน นั่นคือสิ่งที่ฟังก์ชัน `expression` ถูกสร้างขึ้นมา แน่นอนว่ามันเป็นไปได้ที่จะประเมินนิพจน์ได้ในภายหลังด้วยฟังก์ชัน `eval()`.

```
> x <- 3; y <- 2.5; z <- 1
> exp1 <- expression(x / (y + exp(z)))
> exp1
expression(x/(y + exp(z)))
> eval(exp1)
[1] 0.5749019
```

นิพจน์สามารถถูกใช้ร่วมกับสิ่งอื่นเพื่อประกอบเป็นสมการในกราฟ (หน้า 50) นิพจน์หนึ่งสามารถถูกสร้างจากตัวแปรหนึ่งของโหมดอักขระ บางฟังก์ชันนำนิพจน์มาเป็นอาร์กิวเมนต์ ยกตัวอย่างดังต่อไปนี้คือ `D` ซึ่งส่งกลับ (return) อนุพันธ์ย่อย (partial derivatives)

```
> D(exp1, "x")
1/(y + exp(z))
> D(exp1, "y")
-x/(y + exp(z))^2
> D(exp1, "z")
-x * exp(z)/(y + exp(z))^2
```

3.5.2 การปรับเปลี่ยนอ็อบเจ็กต์ (converting objects)

แน่นอนเราได้ตระหนักแล้วว่าความแตกต่างระหว่างบางชนิดของอ็อบเจ็กต์มีเล็กน้อย ดังนั้นมันเป็นเหตุเป็นผลว่าเป็นไปได้ที่เปลี่ยนจากอ็อบเจ็กต์หนึ่งจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่งโดยการเปลี่ยนแปลงบางอย่างของ ลักษณะเฉพาะ (attributes) ของมัน การเปลี่ยนแปลงนี้จะถูกกระทำด้วยฟังก์ชันชนิดหนึ่ง `as.`*`something`*. R (version 2.1.0) ในแพ็กเกจ base และ utils มีถึง 98 ฟังก์ชัน ดังนั้นเราจะไม่ลงให้ลึกในรายละเอียดในที่นี้

ผลของปรับเปลี่ยนอ็อบเจ็กต์หนึ่งแน่นอนว่าขึ้นอยู่กับลักษณะเฉพาะของอ็อบเจ็กต์ที่ถูกปรับเปลี่ยน (converted object) โดยทั่วไปการปรับเปลี่ยนทำตามหลักการพื้นฐาน ตารางต่อไปนี้สรุปเงื่อนไขการปรับเปลี่ยนของโหมดต่าง ๆ

การปรับเปลี่ยนโหมดเป็น	ฟังก์ชัน	เกณฑ์
โหมดตัวเลข	`as.numeric`	`FALSE` → 0
		`TRUE` → 1
		`"1", "2", ...` → 1, 2, ...
		`"A", ...` → `NA`
โหมดตรรกะ	`as.logical`	0 → `FALSE`
		other numbers → `TRUE`
		`"FALSE", "F"` → `FALSE`
		`"TRUE", "T"` → `TRUE`
		other characters → `NA`
โหมดอักขระ	`as.character`	1, 2, ... → `"1", "2", ...`
		`FALSE` → `"FALSE"`
		`TRUE` → `"TRUE"`

มีฟังก์ชันต่างๆ ที่ปรับเปลี่ยนชนิดของอ็อบเจ็กต์ ตัวอย่างเช่น (`as.matrix`, `as.ts`, `as.data.frame`, `as.expression` เป็นต้น ฟังก์ชันเหล่านี้จะมีผลต่อลักษณะเฉพาะมากกว่ามีผลต่อโหมดในระหว่างการปรับเปลี่ยน ผลของการปรับเปลี่ยนนี้ตามที่กล่าวไว้ก่อนหน้าโดยทั่วไปเป็นไปตามหลักการพื้นฐาน ในสถานการณ์หนึ่งที่มักพบบ่อยคือการปรับเปลี่ยนของ factors ให้เป็นค่าตัวเลข ในกรณีนี้ R ทำการปรับเปลี่ยนด้วยการทำรหัสตัวเลขของ levels ของ factor ดังตัวอย่างต่อไปนี้

```
> fac <- factor(c(1, 10))
> fac
[1] 1 10
Levels:  1 10
```

```
> as.numeric(fac)
[1] 1 2
```

เมื่อทำการพิจารณาถึง factor ของโหมดอักขระ ตัวอย่างต่อไปนี้จะดูเป็นเหตุเป็นผล

```
> fac2 <- factor(c("Male", "Female"))
> fac2
[1] Male   Female
Levels: Female Male
> as.numeric(fac2)
[1] 2 1
```

ข้อสังเกตคือผลที่ได้ข้างต้นไม่เป็น `NA` ตามที่ถูกคาดหมายไว้ตามตารางข้างต้น

เพื่อปรับเปลี่ยน factor ของโหมดตัวเลขให้กลายเป็นเวกเตอร์ตัวเลขแต่ยังคงเก็บ levels ไว้ตามที่ถูกระบุไว้ในรูปแบบดั้งเดิม ผู้ใช้ R ต้องปรับเปลี่ยนให้กลายเป็นอักขระ character ตั้งแต่แรกก่อนแล้วจึงปรับเปลี่ยนเป็นตัวเลขในลำดับถัดมา

```
> as.numeric(as.character(fac))
[1] 1 10
```

กระบวนการนี้เป็นประโยชน์มากถ้าในไฟล์หนึ่งประกอบด้วยตัวแปรตัวเลขที่มีค่าที่ไม่เป็นตัวเลข (non-numeric values) อยู่ด้วย เราได้เห็นว่า `read.table()` ในเงื่อนไขนี้จะเป็นค่าปริยายโดยอ่านคอลัมน์นี้เป็น factor

3.5.3 ตัวดำเนินการ (operators)

เราได้เห็นแล้วก่อนหน้านี้ว่ามีตัวดำเนินการ 3 ชนิดหลักใน R[10] ตารางต่อไปนี้เป็นรายการของตัวดำเนินการ

ตัวดำเนินการ					
ทางเรขาคณิต		ทางการเปรียบเทียบ		ทางตรรกะ	
+	บวก	<	น้อยกว่า	! x	ตรรกะ ไม่
-	ลบ	>	มากกว่า	x & y	ตรรกะ และ
*	คูณ	<=	น้อยกว่าหรือเท่ากับ	x && y	เหมือนคำอธิบายด้านบน
/	หาร	>=	มากกว่าหรือเท่ากับ	x | y	ตรรกะ หรือ
^	ยกกำลัง	==	เท่ากับ	x || y	เหมือนคำอธิบายด้านบน
%%	การหาเศษที่เหลือจากการหาร	!=	ไม่เท่ากับ	xor(x, y)	เฉพาะ หรือ
%/%	การหารให้ปัดเศษเป็นจำนวนเต็ม				

[10] ลักษณะต่อไปนี้เป็นตัวดำเนินการเช่นกันสำหรับ R `$`, `@`, `[`, `[[`, `:`, `?`, `<-`, «`-`, `=`, `::` ตารางของตัวดำเนินการซึ่งถูกอธิบายเกี่ยวกับหลักสำคัญเกี่ยวกับตัวดำเนินการสามารถหาได้จาก `?Syntax`.

ตัวดำเนินการเลขคณิตและเปรียบเทียบกระทำการต่อ elements 2 ตัว (`x + y`, `a < b`)ตัวดำเนินการเลขคณิต ไม่เพียงแต่กระทำการต่อตัวแปรของโหมดตัวเลขหรือโหมดเชิงซ้อนเท่านั้นแต่ยังกระทำการต่อตัวแปรตรรกะด้วย ในกรณีหลังค่าตรรกะถูกบังคับให้กลายเป็นตัวเลข ตัวดำเนินการเปรียบเทียบอาจถูกนำไปใช้กับโหมดใด ๆ ก็ได้หมายความว่าตัวดำเนินการนี้ให้ผลกลับมาเป็นค่าตรรกะค่าหนึ่งหรือหลายค่า

ตัวดำเนินการตรรกะถูกนำไปใช้ต่ออ็อบเจ็กต์หนึ่งอ็อบเจ็กต์(`!`) หรือสองอ็อบเจ็กต์ของโหมดตรรกะและส่งกลับมาเป็นค่าตรรกะหนึ่งค่าหรือหลายค่า ตัวดำเนินการ “AND” และ “OR” มีอยู่สองรูปแบบหมายความว่า กรณีรูปแบบเดี่ยว (single) ดำเนินการต่อ elements แต่ละตัวของอ็อบเจ็กต์และได้ผลกลับคืนเป็นค่าตรรกะหลายค่า ตามการเปรียบเทียบที่ได้ถูกกระทำ และกรณีรูปแบบคู่ (double) ตัวดำเนินการกระทำต่อ element ตัวแรกสุดของอ็อบเจ็กต์

โดยมีความจำเป็นที่ต้องใช้ตัวดำเนินการ “AND” เพื่อระบุความไม่เท่ากันของชนิด $0 < x < 1$ ซึ่งถูกทำรหัสด้วย `0 < x & x < 1` นิพจน์ `0 < x < 1` ใช้ได้และถูกต้องแต่จะไม่ส่งค่ากลับตามที่คาดหมาย หมายความว่าเนื่องจากตัวดำเนินการทั้งคู่เป็นสิ่งเดียวกันและจะถูกดำเนินการตามลำดับจากซ้ายไปขวา การเปรียบเทียบ `0 < x` จะถูกกระทำแรกสุดและได้ผลกลับเป็นค่าตรรกะซึ่งหลังจากนั้นจะถูกเปรียบเทียบกับ 1 (`TRUE` or `FALSE < 1`) ในสถานการณ์นี้หมายความว่า ค่าตรรกะถูกบังคับทางอ้อมให้กลายเป็นค่าตัวเลข (`1` หรือ `0 < 1`) ดังตัวอย่างต่อไปนี้

```
> x <- 0.5
> 0 < x < 1
[1] FALSE
```

ตัวดำเนินการเปรียบเทียบดำเนินการต่ออีลีเมนท์แต่ละตัวของ 2 อ็อบเจ็กต์ซึ่งถูกเปรียบเทียบ (การนำค่ากลับมาใช้ของค่าที่น้อยที่สุดถ้าจำเป็น) และหลังจากนั้นจะส่งค่าอ็อบเจ็กต์ที่มีขนาดเดียวกันกลับมา เพื่อเปรียบเทียบอ็อบเจ็กต์ 2 อ็อบเจ็กต์ทั้งหมด มี 2 ฟังก์ชันที่ใช้ได้คือ `identical` และ `all.equal` ดังตัวอย่างต่อไปนี้

```
> x <- 1:3; y <- 1:3
> x == y
[1] TRUE TRUE TRUE
> identical(x, y)
[1] TRUE
> all.equal(x, y)
[1] TRUE
```

`identical` เปรียบเทียบการเป็นตัวแทนภายในของข้อมูลและนำค่ากลับเป็น `TRUE` ถ้าอ็อบเจ็กต์เหมือนกันอย่างสมบูรณ์ และไม่เช่นนั้นแล้วเป็น `FALSE` ส่วน `all.equal` เปรียบเทียบ “near equality” ของ 2 อ็อบเจ็กต์และนำกลับมาเป็น `TRUE` หรือแสดงสรุป

ของความแตกต่าง สำหรับฟังก์ชัน `all.equal` นำการประมาณของกระบวนการคำนวณมาพิจารณาด้วย เมื่อทำการเปรียบเทียบค่าตัวเลข การเปรียบเทียบของค่าตัวเลขในคอมพิวเตอร์บางครั้งเป็นเรื่องน่าประหลาดใจ

```
> 0.9 == (1 - 0.1)
[1] TRUE
> identical(0.9, 1 - 0.1)
[1] TRUE
> all.equal(0.9, 1 - 0.1)
[1] TRUE
> 0.9 == (1.1 - 0.2)
[1] FALSE
> identical(0.9, 1.1 - 0.2)
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance = 1e-16)
[1] "Mean relative  difference: 1.233581e-16"
```

3.5.4 การเข้าถึงค่าของอ็อบเจ็กต์: ระบบการทำดัชนี (indexing system)

ระบบการทำดัชนี (indexing system) เป็นวิธีที่มีประสิทธิภาพและยืดหยุ่นในการเข้าถึงอีลีเมนท์ที่เลือกไว้ของ อ็อบเจ็กต์ซึ่ง ไม่เป็น *ตัวเลข* ก็ต้องเป็น *ตรรกะ* เพื่อที่จะเข้าถึงค่า ตัวอย่างเช่น ค่าที่สามของเวกเตอร์ `x` เราเพียงแต่พิมพ์ `x[3]` ซึ่งสามารถถูกใช้ในกรณีเพื่อสกัดเอาข้อมูลหรือเพื่อเปลี่ยนค่านั้น ๆ ดังตัวอย่างต่อไปนี้

```
> x <- 1:5
> x[3]
[1] 3
> x[3] <- 20
> x
[1]  1  2 20  4  5
```

ดัชนีโดยตัวมันเองสามารถเป็นเวกเตอร์ของโหมดตัวเลขได้ ดังตัวอย่าง

```
> i <- c(1, 3)
> x[i]
[1]  1 20
```

ถ้า `x` เป็นเมทริกซ์หรือกรอบข้อมูลหนึ่งแล้วค่าของบรรทัดที่ i และคอลัมน์ที่ j จะถูกเข้าถึงด้วย `x[i,j]` เพื่อเป็นการเข้าถึงค่าทั้งหมดของแถวหรือคอลัมน์ที่กำหนดให้ ผู้ใช้ R ต้องละดัชนีที่เหมาะสมแต่จะต้องไม่ลืมเครื่องหมาย , ดังตัวอย่างต่อไปนี้

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[, 3] <- 21:22
> x
     [,1] [,2] [,3]
[1,]    1    3   21
[2,]    2    4   22
> x[, 3]
[1] 21 22
```

เราได้สังเกตเห็นอย่างแน่ชัดแล้วว่าผลลัพธ์สุดท้ายเป็นเวกเตอร์ไม่ใช่เมทริกซ์ โดยลักษณะการตอบสนองที่เป็นโดยปริยายของ R คือการคืนค่าอ็อบเจ็กต์ที่มีมิติ (dimension) น้อยที่สุดเท่าที่จะเป็นไปได้กลับมา กรณีนี้สามารถเปลี่ยนแปลงได้ด้วยการเลือกใช้ฟังก์ชัน `drop` ซึ่งมีค่าโดยปริยายเป็น `TRUE`ดังตัวอย่างต่อไปนี้

```
> x[, 3, drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

ระบบการทำดัชนีนี้ ถูกทำให้เป็นทั่วไปโดยง่ายโดยการใช้อาร์เรย์ ได้ดัชนีมากมายเท่ากับจำนวนของมิติของอาร์เรย์ ตัวอย่างเช่น อาร์เรย์ 3 มิติ (`x[i, j, k]`, `x[, , 3]`, `x[, , 3, drop = FALSE]`) และต่อไปเรื่อย ๆ มันเป็นประโยชน์ที่จะจำว่าการทำดัชนีถูกทำโดยเครื่องหมายวงเล็บเหลี่ยม (square brackets) ขณะที่วงเล็บกลม (parentheses) ถูกใช้สำหรับอาร์กิวเมนต์ของฟังก์ชัน

```
> x(1)
Error: couldn't find function "x"
```

การทำดัชนียังสามารถถูกใช้เพื่อยกเลิก (suppress) จำนวนแถวหรือคอลัมน์ทั้งหนึ่งหรือหลายจำนวนโดยการใช้ค่าลบ ตัวอย่างเช่น `x[-1, ]` จะยกเลิกแถวที่หนึ่งขณะที่ `x[-c(1, 15), ]` จะกระทำการเหมือนกันสำหรับแถวที่ 1 และแถวที่ 15 ส่วนการใช้เมทริกซ์ ถูกกำหนดดังข้างต้น

```
> x[, -1]
     [,1] [,2]
[1,]    3   21
```

```
[2,]    4   22
> x[, -(1:2)]
[1] 21 22
> x[, -(1:2), drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

สำหรับเวกเตอร์ เมทริกซ์ และอาร์เรย์ นั้นเป็นไปได้ที่จะเข้าถึงค่าของอีลีเมนท์หนึ่งด้วยนิพจน์เปรียบเทียบ ดังดัชนีต่อไปนี้

```
> x <- 1:10
> x[x >= 5] <- 20
> x
 [1]  1  2  3  4 20 20 20 20 20 20
> x[x == 1] <- 25
> x
 [1] 25  2  3  4 20 20 20 20 20 20
```

การใช้ในทางปฏิบัติของการทำดัชนีตรรกะ ตัวอย่างเช่น ความเป็นไปได้ที่จะเลือก elements ของจำนวนเต็มที่เป็นเลขคู่ ดังต่อไปนี้

```
> x <- rpois(40, lambda=5)
> x
 [1]  5  9  4  7  7  6  4  5 11  3  5  7  1  5  3  9  2  2  5  2
[21]  4  6  6  5  4  5  3  4  3  3  3  7  7  3  8  1  4  2  1  4
> x[x %% 2 == 0]
 [1] 4 6 4 2 2 2 4 6 6 4 4 8 4 2 4
```

ดังนั้น ระบบการทำดัชนีนี้ใช้ค่าตรรกะกลับคืนมาแล้วดังในตัวอย่างข้างต้นโดยใช้ตัวดำเนินการเปรียบเทียบ ค่าตรรกะเหล่านี้สามารถถูกคำนวณก่อนแล้วค่าเหล่านี้ถูกนำกลับไปใช้ถ้าจำเป็น ดังตัวอย่างต่อไปนี้

```
> x <- 1:40
> s <- c(FALSE, TRUE)
> x[s]
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

การทำดัชนีตรรกะสามารถถูกใช้กับกรอบข้อมูลได้ด้วยแต่มีข้อระวังเนื่องจากคอลัมน์ที่แตกต่างกันของกรอบข้อมูลอาจจะมีโหมดที่แตกต่างกัน

สำหรับการแสดงรายการ (lists) ในการเข้าถึง elements ที่แตกต่างกันสามารถเป็นอ็อบเจ็กต์ชนิดใดก็ได้ที่ถูกกระทำโดยมี square brackets เดี่ยวหรือไม่ก็มี square brackets คู่ หมายความว่าความแตกต่างนี้คือ single brackets ทำให้ได้รายการกลับคืนมาในขณะที่ double brackets *สกัด (extract)* อ็อบเจ็กต์จากรายการ เมื่อนั้นผลที่ได้นี้สามารถถูกทำให้เป็นดัชนีด้วยตัวเองตามที่ได้เห็นก่อนหน้านี้สำหรับเวกเตอร์ เมทริกซ์และอื่น ๆ ตัวอย่างเช่น ถ้าอ็อบเจ็กต์ที่ 3 ของรายการหนึ่งเป็นเวกเตอร์หนึ่ง ค่าที่ i ของมันสามารถถูกเข้าถึงได้โดยการใช้ `my.list[[3]][i]` และถ้าเป็นอาร์เรย์ 3 มิติ `my.list[[3]][i, j, k]` และอื่นๆ ความแตกต่างอีกอย่างหนึ่งคือว่า `my.list[1:2]` จะนำรายการหนึ่งกลับเข้ามาด้วยอีลีเมนท์ที่หนึ่งและที่สองของรายการดั้งเดิม ขณะที่ `my.list[[1:2]]` จะให้ผลลัพธ์ที่ไม่ได้ตามที่คาดหมาย

3.5.5 การเข้าถึงค่าต่าง ๆ ของอ็อบเจ็กต์โดยชื่อ

ชื่อ names เป็นการลงรายการของอีลีเมนท์ของอ็อบเจ็กต์หนึ่งดังนั้นจึงเป็นโหมดอักขระ โดยทั่วไปชื่อเหล่านี้เป็นลักษณะเฉพาะที่เป็นตัวเลือก (optional attributes) โดยที่มีชนิดของชื่อหลายชนิด ตัวอย่างเช่น *names*, *colnames*, *rownames* และ *dimnames*

ชื่อของเวกเตอร์หนึ่งถูกเก็บในเวกเตอร์หนึ่งของอ็อบเจ็กต์ ที่มีความยาวเท่ากันและสามารถถูกเข้าถึงด้วยฟังก์ชัน `names` ได้ ดังตัวอย่างต่อไปนี้

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("a", "b", "c")
> x
a b c
1 2 3
> names(x)
[1] "a" "b" "c"
> names(x) <- NULL
> x
[1] 1 2 3
```

สำหรับเมทริกซ์และกรอบข้อมูลแล้ว *colnames* และ *rownames* คือ การลงรายการของคอลัมน์และแถวตามลำดับซึ่งสามารถถูกเข้าถึงได้ไม่ว่าจะเป็น respective functions ของมัน หรือ `dimnames` ซึ่งเป็นการนำรายการกลับเข้ามาโดยเวกเตอร์ทั้งคู่ ดังตัวอย่างต่อไปนี้

```
> X <- matrix(1:4, 2)
> rownames(X) <- c("a", "b")
> colnames(X) <- c("c", "d")
```

```
> X
  c d
a 1 3
b 2 4
> dimnames(X)
[[1]]
[1] "a" "b"

[[2]]
[1] "c" "d"
```

สำหรับอาร์เรย์ชื่อของมิติสามารถถูกเข้าถึงได้โดย ฟังก์ชัน `dimnames` ดังตัวอย่างต่อไปนี้

```
> A <- array(1:8, dim = c(2, 2, 2))
> A
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

> dimnames(A) <- list(c("a", "b"), c("c", "d"), c("e", "f"))
> A
, , e

  c d
a 1 3
b 2 4

, , f

  c d
```

```
a 5 7
b 6 8
```

ถ้าอีลีเมนท์ของอ็อบเจ็กต์หนึ่งมีชื่อแล้วมันสามารถถูกสกัดออกมาได้โดยการใช้อีลีเมนท์ตามดัชนี ตามจริงแล้วศัพท์ที่ควรเรียกคือ “subsetting” มากกว่า “extraction” เนื่องจากลักษณะเฉพาะของอ็อบเจ็กต์ ต้นฉบับถูกเก็บไว้ ตัวอย่างเช่น ถ้ากรอบข้อมูล (dataframe) `DF` ประกอบด้วยตัวแปร `x`, `y` และ `z` การใช้คำสั่ง `DF["x"]` จะนำ กรอบข้อมูลกลับเข้ามาด้วย `x` เท่านั้นส่วนคำสั่ง `DF[c("x", "y")]` จะนำกรอบข้อมูลกลับเข้าด้วยตัวแปรทั้ง `x` และ `y` การทำงานด้วยรายการก็เช่นกันถ้า elements ในรายการนั้นมีชื่ออยู่

ตามที่ผู้อ่านได้ตระหนักแน่นอนแล้วว่าดัชนีที่ถูกใช้ในที่นี้เป็นเวกเตอร์หนึ่งของโหมดอักขระคล้ายกับเวกเตอร์ตัวเลขหรือเวกเตอร์ตรรกะที่ได้เห็นข้างต้น เวกเตอร์นี้สามารถถูกกำหนดได้ก่อนและต่อมาสามารถถูกใช้สำหรับการสกัด (extraction)

เพื่อสกัดเวกเตอร์หนึ่งหรือ factor หนึ่งจากกรอบข้อมูล ผู้ใช้ R สามารถใช้ตัวดำเนินการ `$` (ตัวอย่างเช่น `DF$x`) การกระทำโดยตัวดำเนินการนี้สามารถใช้กับการทำรายการได้เช่นกัน

3.5.6 ตัวปรับแก้ข้อมูล (data editor)

มันเป็นไปได้ที่จะใช้ตัวปรับแก้ที่คล้ายกับตารางทำกราฟิก (graphical spreadsheet-like editor) เพื่อปรับแก้ อ็อบเจ็กต์ของ “ข้อมูล” ตัวอย่างเช่น ถ้า `X` แป็นเมทริกซ์แล้ว คำสั่ง `data.entry(X)` จะเปิดตัวปรับแก้กราฟิก (graphic editor) และผู้ใช้ R จะสามารถเปลี่ยนแปลงค่าบางค่าโดยการกด (clicking) ไปบนเซลล์ที่เหมาะสมหรือเพิ่มคอลัมน์หรือแถวใหม่ได้

ฟังก์ชัน `data.entry` เปลี่ยนแปลงอ็อบเจ็กต์นั้นโดยตรงตามที่ถูกกำหนดให้โดยอาร์กิวเมนต์โดยไม่ต้องกำหนด (assign) ผลของอาร์กิวเมนต์ ในทางตรงกันข้ามฟังก์ชัน `de` นำรายการกลับเข้ามากับอ็อบเจ็กต์ที่ถูกกำหนดให้ตามอาร์กิวเมนต์และอาจถูกเปลี่ยนแปลงได้ ผลนี้ถูกแสดงบนหน้าจอโดยปริยายแต่ในส่วนของฟังก์ชันส่วนใหญ่สามารถถูกกำหนดไปที่อ็อบเจ็กต์ได้

รายละเอียดของการใช้ตัวปรับแก้ข้อมูลขึ้นกับระบบปฏิบัติการ

3.5.7 เลขคณิตและฟังก์ชันอย่างง่าย (arithmetics and simple functions)

มีหลายฟังก์ชันใน R ที่ใช้จัดการข้อมูลได้ เราได้เห็นแล้วสำหรับฟังก์ชันที่ง่ายที่สุดฟังก์ชันหนึ่งคือ `c` ซึ่งเชื่อมต่อ (concatenate) อ็อบเจ็กต์ที่ถูกลงรายการไว้ในเครื่องหมายวงเล็บ ( ) ดังตัวอย่างต่อไปนี้

```
> c(1:5, seq(10, 11, 0.2))
 [1]  1.0  2.0  3.0  4.0  5.0 10.0 10.2 10.4 10.6 10.8 11.0
```

เวกเตอร์สามารถถูกใช้เพื่อจัดการกับนิพจน์เลขคณิตดั้งเดิมได้ ดังตัวอย่างต่อไปนี้

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
[1] 2 3 4 5
```

เวกเตอร์ที่มีความยาวแตกต่างกันสามารถถูกรวมเข้าด้วยกันได้โดยในกรณีนี้เวกเตอร์ที่สั้นที่สุดถูกนำกลับมาใช้ ดังตัวอย่างนี้

```
> x <- 1:4
> y <- 1:2
> z <- x + y
> z
[1] 2 4 4 6
> x <- 1:3
> y <- 1:2
> z <- x + y
Warning message:
longer object length
 is not a multiple of shorter object length in: x + y
> z
[1] 2 4 4
```

ข้อสังเกตคือ R ให้ผลเป็นข้อความเตือน (warning message) และไม่ใช่ข้อความผิดพลาด (error message) ดังนั้นการดำเนินการได้ถูกกระทำไปแล้ว ถ้าเราต้องการเพิ่ม (หรือคูณ) ค่าที่เหมือนกันแก่ elements ทั้งหมดของเวกเตอร์หนึ่งสามารถทำได้ดังตัวอย่างต่อไปนี้

```
> x <- 1:4
> a <- 10
> z <- a * x
> z
[1] 10 20 30 40
```

ฟังก์ชันที่ใช้ได้ใน R สำหรับการจัดการข้อมูลมีมากเกินกว่าที่จะถูกแสดงรายการในที่นี้ ผู้ใช้ R สามารถหาฟังก์ชันคณิตศาสตร์พื้นฐานทั้งหมด (`log`, `exp`, `log10`, `log2`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `abs`, `sqrt` และอื่นๆ) ฟังก์ชันพิเศษ (`gamma`, `digamma`, `beta`, `besselI` และอื่นๆ) เช่นเดียวกันกับฟังก์ชันต่างๆ ที่เป็นประโยชน์ในทางสถิติ บางฟังก์ชันเหล่านี้ถูกแสดงรายการไว้ในตารางดังต่อไปนี้

sum(x)	ผลรวมของอีลีเมนท์ของ x
prod(x)	ผลที่ได้ของอีลีเมนท์ของ x
max(x)	ค่าสูงสุดของอีลีเมนท์ของ x
min(x)	ค่าต่ำสุดของอีลีเมนท์ของ x
which.max(x)	ได้ผลกลับมาเป็นดัชนีของอีลีเมนท์ที่มากสุดของ x
which.min(x)	ได้ผลกลับมาเป็นดัชนีของอีลีเมนท์ที่น้อยสุดของ x
range(x)	เหมือนคำอธิบายด้านบน than c(min(x), max(x))
length(x)	จำนวนของอีลีเมนท์ใน x
mean(x)	ค่าเฉลี่ยเลขคณิตของอีลีเมนท์ของ x
median(x)	ค่ามัธยฐานของอีลีเมนท์ของ x
var(x) or cov(x)	ความแปรปรวนของอีลีเมนท์ของ x (ถูกคำนวณที่ $n-1$) ถ้า x เป็นเมทริกซ์หรือกรอบข้อมูลแล้วเมทริกซ์ระหว่างความแปรปรวนและความแปรปรวนร่วมถูกคำนวณ
cor(x)	เมทริกซ์ของสหสัมพันธ์ของ x ถ้ามันคือเมทริกซ์หรือกรอบข้อมูล (ค่าเท่ากับ 1 ถ้า x เป็นเวกเตอร์)
var(x, y) or cov(x, y)	ความแปรปรวนร่วมระหว่าง x และ y หรือ ระหว่างคอลัมน์ของ x และ y ถ้าเป็นเมทริกซ์หรือกรอบข้อมูล
cor(x, y)	สหสัมพันธ์เชิงเส้นระหว่าง x และ y หรือเมทริกซ์สหสัมพันธ์ถ้าเป็นเมทริกซ์หรือกรอบข้อมูล

ฟังก์ชันจากตารางข้างต้นให้ค่ากลับมาเป็นค่าเดี่ยว เช่น เวกเตอร์หนึ่งของความยาวหนึ่งค่า ยกเว้นฟังก์ชัน range ซึ่งค่ากลับออกมาเป็นเวกเตอร์หนึ่งของความยาวสองค่า และฟังก์ชัน var, cov และ cor ซึ่งอาจให้ค่ากลับออกมาในรูปของเมทริกซ์หนึ่ง ฟังก์ชันในตารางต่อไปนี้ให้ผลออกมาที่ซับซ้อนมากกว่าฟังก์ชันก่อนหน้านี้

round(x, n)	ทำให้อีลีเมนท์ของ x เป็นทศนิยม n ตำแหน่ง
rev(x)	กลับค่าอีลีเมนท์ของ x
sort(x)	แยกประเภทอีลีเมนท์ของ x ตามลำดับที่เพิ่มขึ้น ถ้าเรียงอีลีเมนท์ของ x ตามลำดับที่ลดลง ใช้ rev(sort(x))
rank(x)	จัดอันดับของอีลีเมนท์ของ x
log(x, base)	คำนวณลอการิทึมของ x ด้วยฐาน base
scale(x)	ถ้า x เป็นเมทริกซ์ เป็นศูนย์กลางและลดข้อมูล โดยถ้าเป็นศูนย์กลางเท่านั้นใช้ตัวเลือก center=FALSE ถ้าเป็นลดข้อมูลเท่านั้นใช้ scale=FALSE (โดยค่าปริยาย center=TRUE, scale=TRUE)
pmin(x,y,...)	เวกเตอร์ซึ่งอีลีเมนท์ที่ i เป็นค่าต่ำสุดของ x[i], y[i], ...
pmax(x,y,...)	เหมือนคำอธิบายด้านบน สำหรับค่าต่ำสุด
cumsum(x)	เวกเตอร์ซึ่งอีลีเมนท์ที่ i เป็นผลรวมจาก x[1] ถึง x[i]
cumprod(x)	เหมือนคำอธิบายด้านบน สำหรับผลที่ได้
cummin(x)	เหมือนคำอธิบายด้านบน สำหรับค่าต่ำสุด
cummax(x)	เหมือนคำอธิบายด้านบน สำหรับค่าสูงสุด

`match(x, y)`	ได้ผลเป็นเวกเตอร์ของความยาวเหมือนกันกับ `x` ด้วยอีลีเมนท์ของ `x` ซึ่งอยู่ใน `y` (`NA` ถ้าเป็นอย่างอื่น)
`which(x == a)`	ได้ผลเป็นเวกเตอร์ของดัชนีของ `x` ถ้าการดำเนินการเปรียบเทียบเป็นจริง (`TRUE`) ในตัวอย่างนี้ค่าของ `i` สำหรับ which `x[i] == a` (อาร์กิวเมนต์ของฟังก์ชันนี้ต้องเป็นตัวแปรของโหมดตรรกะ)
`choose(n, k)`	คำนวณผลรวมของเหตุการณ์ k ครั้งในการทำซ้ำ n ครั้ง = $n!/[(n-k)!k!]$
`na.omit(x)`	ยกเลิกการสังเกตด้วยค่าที่หายไป (`NA`) (ยกเลิกบรรทัดที่ตรงกันถ้า `x` เป็นเมทริกซ์หรือกรอบข้อมูล)
`na.fail(x)`	ได้ผลเป็นข้อความผิดพลาดถ้า `x` ประกอบด้วยอย่างน้อยหนึ่ง `NA`
`unique(x)`	ถ้า `x` เป็นเวกเตอร์หรือกรอบข้อมูลแล้วให้ผลเป็นอ็อบเจ็กต์ที่คล้ายกันแต่อีลีเมนท์ที่เหมือนกันถูกยกเลิก
`table(x)`	ได้ผลเป็นตารางที่มีตัวเลขของค่าที่แตกต่างกันของ `x` (โดยทั่วไปแล้วสำหรับจำนวนเต็มหรือ factors)
`table(x, y)`	ตารางแจกแจงของ `x` และ `y`
`subset(x, ...)`	ได้ผลเป็นการคัดเลือกของ `x` ตามเกณฑ์ที่พิจารณา (`...`, โดยทั่วไปเป็นการเปรียบเทียบ เช่น `x$V1 < 10`) โดยที่ `x` เป็นกรอบข้อมูล ตัวเลือกเป็น `select` และกำหนดให้ตัวแปรถูกเก็บหรือเอาออกไปด้วยการใช้เครื่องหมายลบ
`sample(x, size)`	การสุ่มตัวอย่างอีกครั้งแบบสุ่มและปราศจากการแทนที่อีลีเมนท์ `size` ในเวกเตอร์ `x` ขณะที่ ตัวเลือก `replace = TRUE` ยอมให้สุ่มอีกครั้งโดยการแทนที่

3.5.8 การคำนวณหรือการคณนาเมทริกซ์ (matrix computation)

R มีความง่ายสำหรับการคำนวณและจัดการเมทริกซ์โดยฟังก์ชัน `rbind` และฟังก์ชัน `cbind` เชื่อมเมทริกซ์ตามลักษณะของจำนวนแถวหรือจำนวนคอลัมน์ตามลำดับของฟังก์ชันที่กล่าว ต่อไปนี้คือตัวอย่าง

```
> m1 <- matrix(1, nr = 2, nc = 2)
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
> cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    2
[2,]    1    1    2    2
```

ตัวดำเนินการสำหรับผลผลิตของสองเมทริกซ์คือ '%*%' ตัวอย่างเช่น พิจารณาสองเมทริกซ์คือ m1 และ m2 ข้างต้น ได้ผลดังต่อไปนี้

```
> rbind(m1, m2) %*% cbind(m1, m2)
     [,1] [,2] [,3] [,4]
[1,]    2    2    4    4
[2,]    2    2    4    4
[3,]    4    4    8    8
[4,]    4    4    8    8
> cbind(m1, m2) %*% rbind(m1, m2)
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

การเปลี่ยนตำแหน่ง(transposition) ของเมทริกซ์หนึ่งถูกกระทำได้ด้วยฟังก์ชัน t โดยที่ฟังก์ชันนี้ใช้งานได้กับกรอบข้อมูลได้ด้วย

ฟังก์ชัน diag สามารถถูกใช้เพื่อสกัดค่าหรือเปลี่ยนแปลงแนวทแยงของเมทริกซ์หรือสร้างเมทริกซ์ทแยงมุม (diagonal matrix) ดังตัวอย่างต่อไปนี้

```
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
     [,1] [,2]
[1,]   10    1
[2,]    1   10
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> v <- c(10, 20, 30)
> diag(v)
     [,1] [,2] [,3]
[1,]   10    0    0
[2,]    0   20    0
```

```
[3,]    0    0   30
> diag(2.1, nr = 3, nc = 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.1  0.0  0.0    0    0
[2,]  0.0  2.1  0.0    0    0
[3,]  0.0  0.0  2.1    0    0
```

R ยังมีฟังก์ชันพิเศษบางฟังก์ชันสำหรับการคำนวณเมทริกซ์ ในที่นี้เราจะกล่าวถึง ฟังก์ชัน `solve` สำหรับการกลับด้านเมทริกซ์ ส่วนฟังก์ชัน `qr` สำหรับการแยกเมทริกซ์ ฟังก์ชัน `eigen` สำหรับการคำนวณค่าเฉพาะ (eigenvalues) และเวกเตอร์เฉพาะ (eigenvectors) และสุดท้ายฟังก์ชัน `svd` สำหรับการแยกค่าออกเป็นเดี่ยว ๆ

4 กราฟิกโดย R

R ให้กราฟิกที่มีความหลากหลายเป็นพิเศษ เพื่อที่จะได้หลักการเกี่ยวกับกราฟิกใน R ผู้ใช้สามารถพิมพ์ `demo(graphics)` หรือ `demo(persp)` เป็นไปไม่ได้ที่จะให้รายละเอียดในที่นี้เกี่ยวกับความเป็นไปได้ของ R ในแง่มุมของกราฟิกเนื่องจากโดยเฉพาะแต่ละฟังก์ชันกราฟิกมีตัวเลือกจำนวนมากในการสร้างงานกราฟิกที่มีความยืดหยุ่นมาก

วิธีที่ฟังก์ชันกราฟิกทำงานนั้นเบี่ยงเบนอย่างมากจากแบบที่ร่างไว้ในตอนต้นของเอกสารนี้ โดยเฉพาะผลของฟังก์ชันกราฟิกไม่สามารถถูกกำหนดให้เป็นอ็อบเจ็กต์[11] แต่ว่าจะถูกส่งไปที่ *graphical device* หนึ่งคือวินโดว์กราฟิกหรือไฟล์หนึ่ง

มีฟังก์ชันกราฟิกสองชนิดคือ *ฟังก์ชันการวาดระดับสูง (high-level plotting functions)* ซึ่งสร้างกราฟใหม่ และ *ฟังก์ชันการวาดระดับต่ำ (low-level plotting functions)* ซึ่งเพิ่มอีลีเมนท์ไปที่กราฟที่มีอยู่ กราฟถูกสร้างโดยเกี่ยวข้องกับ *พารามิเตอร์กราฟิก (graphical parameters)* ซึ่งถูกกำหนดโดยค่าปริยายและสามารถถูกเปลี่ยนแปลงได้ด้วยฟังก์ชัน `par`

ในตอนต้นเราจะได้เห็นว่าการจัดการกราฟิกและ graphical device ทำอย่างไร และต่อจากนั้นเราจะลงไปในรายละเอียดของฟังก์ชันกราฟิกและพารามิเตอร์ เราจะได้เห็นตัวอย่างในทางปฏิบัติของการใช้ฟังก์ชันเหล่านี้ในการสร้างกราฟ สุดท้ายเราจะได้ทราบถึงแพ็กเกจ grid และ lattice ซึ่งมีฟังก์ชันที่แตกต่างจากฟังก์ชันที่สรุปไว้ก่อนหน้านี้

4.1 การจัดการกราฟิก

4.1.1 การเปิด graphical devices หลายส่วน (several graphical devices)

เมื่อฟังก์ชันกราฟิกถูกดำเนินงาน ถ้าไม่มี graphical device ถูกเปิดแล้ว R จะเปิดวินโดว์กราฟิกและแสดงกราฟนั้น graphical device อาจจะถูกเปิดด้วยฟังก์ชันที่เหมาะสม รายการของ graphical devices ที่ใช้การได้ขึ้นอยู่กับระบบปฏิบัติการ วินโดว์กราฟิกถูกให้ชื่อว่า `X11` ภายใต้ระบบ Unix/Linux และ `windows` ภายใต้ระบบวินโดว์ ในทุกกรณีผู้ใช้ R สามารถเปิดวินโดว์กราฟิก โดยคำสั่ง `x11()` ซึ่งใช้งานได้เช่นกันภายใต้ระบบวินโดว์ เนื่องจากเป็นนามแฝงหนึ่งที่เกี่ยวกับคำสั่ง `windows()` graphical device ซึ่งเป็นไฟล์หนึ่งจะถูกเปิดด้วยฟังก์ชันหนึ่งที่ขึ้นอยู่กับรูปแบบดังต่อไปนี้ได้แก่ `postscript()`, `pdf()` และ `png()` เป็นต้น รายการของ graphical devices ที่ใช้งานได้สามารถถูกสืบค้นได้ด้วย

[11] มีข้อยกเว้นที่สำคัญบางข้อคือ `hist()` และ `barplot()` ให้ผลออกมาเป็นตัวเลขเช่นกันตามรายการหรือเมทริกซ์

```
?device
```

device ที่เปิดสุดท้ายกลายเป็น graphical device ที่ใช้ได้อยู่ (active) ซึ่งกราฟที่ตามมาภายหลังทั้งหมดถูกแสดงด้วยฟังก์ชัน `dev.list()` ที่แสดงรายการของ open devices ดังต่อไปนี้

```
> x11(); x11(); pdf()
> dev.list()
X11 X11 pdf
  2   3   4
```

รูปที่ถูกแสดงคือจำนวนของ devices ซึ่งต้องถูกใช้เพื่อเปลี่ยน active device เพื่อที่ทราบว่าอะไรคือ active device แสดงได้ดังนี้

```
> dev.cur()
pdf
  4
```

และเพื่อเปลี่ยน active device แสดงดังนี้

```
> dev.set(3)
X11
  3
```

ฟังก์ชัน `dev.off()` ปิด device หมายความว่าโดยค่าปริยาย active device ถูกปิด มิฉะนั้นแล้ว active device ซึ่งจำนวน (number) ถูกกำหนดให้เป็นอาร์กิวเมนต์ไปที่ฟังก์ชันนั้น หลังจากนั้น R แสดงจำนวนของ active device ใหม่ ดังนี้

```
> dev.off(2)
X11
  3
> dev.off()
pdf
  4
```

ลักษณะเฉพาะสองอย่างของเวอร์ชันวินโดว์ ของ R ซึ่งควรค่าต่อการกล่าวถึงคือ Windows Metafile device สามารถถูกเปิดด้วยฟังก์ชัน `win.metafile` และเมนู “History” ถูกแสดงเมื่อวินโดว์กราฟิกถูกเลือกซึ่งยอมให้บันทึกกราฟทั้งหมดที่ถูกวาดระหว่างชุด (session) หนึ่ง (โดยค่าปริยาย ระบบการบันทึกถูกปิด ผู้ใช้ต้องเปิดมันขึ้นมาโดยกดไปที่ “Recording” ในเมนูนี้)

4.1.2 การแบ่งส่วนของกราฟิก (partitioning a graphic)

ฟังก์ชัน `split.screen` แบ่งส่วนของ active graphical device ตัวอย่างดังต่อไปนี้

```
> split.screen(c(1, 2))
```

จากข้างบนแบ่งภายใน device เป็นสองส่วนซึ่งสามารถถูกเลือกด้วย `screen(1)` หรือ `screen(2)` และ `erase.screen()` ลบกราฟที่ถูกวาดครั้งสุดท้าย ส่วนของ device สามารถถูกแบ่งได้โดยตัวมันเองโดย `split.screen()` ซึ่งนำไปสู่ความเป็นไปได้ที่ทำ-ให้เกิดการจัดเรียงที่มีความซับซ้อน

ฟังก์ชันเหล่านี้ไม่เข้ากับฟังก์ชันอื่น (ตัวอย่างเช่น `layout` หรือ `coplot`) และต้องไม่ถูกใช้กับ graphical devices หลายส่วน การใช้นี้ควรเป็นไปอย่างจำกัด ตัวอย่างเช่น การสำรวจกราฟิกของข้อมูล

ฟังก์ชัน `layout` แบ่งวินโดว์กราฟิกที่ active เป็นหลายส่วนโดยที่กราฟจะถูกแสดงลำดับถัดมา อาร์กิวเมนต์หลักของมันคือเมทริกซ์หนึ่งกับจำนวนเต็มที่บ่งบอกถึงจำนวนของ “sub-windows” ตัวอย่างดังต่อไปนี้แสดงการแบ่ง device ภายในออกเป็นสี่ส่วนเท่า ๆ กัน

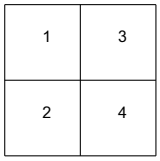
```
> layout(matrix(1:4, 2, 2))
```

เป็นไปได้อย่างแน่นอนที่จะสร้างเมทริกซ์ก่อนหน้านี้โดยการยอมให้เห็นภาพได้ดีขึ้นว่า device ถูกแบ่งได้อย่างไรดังตัวอย่างต่อไปนี้

```
> mat <- matrix(1:4, 2, 2)
> mat
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> layout(mat)
```

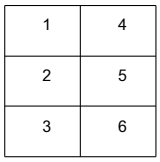
เพื่อที่เห็นภาพจริงที่การแบ่งถูกสร้างขึ้นมา ผู้ใช้ R สามารถใช้ฟังก์ชัน `layout.show` กับจำนวนหน้าต่างย่อย (sub-windows) เป็นอาร์กิวเมนต์ (ในที่นี้คือ 4) ดังตัวอย่างต่อไปนี้

```
> layout.show(4)
```

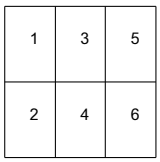


ตัวอย่างดังต่อไปนี้แสดงบางส่วนของความเป็นไปได้ที่ฟังก์ชัน `layout()` ทำได้

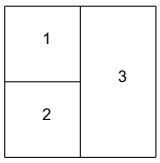
```
> layout(matrix(1:6, 3, 2))
> layout.show(6)
```




```
> layout(matrix(1:6, 2, 3))
> layout.show(6)
```



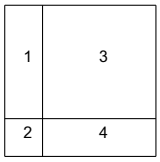

```
> m <- matrix(c(1:3, 3), 2, 2)
> layout(m)
> layout.show(3)
```



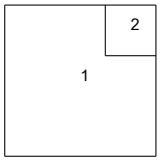
ตัวอย่างเหล่านี้ทั้งหมดยัง ไม่ได้ใช้ตัวเลือกคือ `byrow` ของ `matrix()` ดังนั้นหน้าต่างย่อยข้างต้นมีจำนวนแนวตั้งนับตามคอลัมน์ลงมา (column-wise) โดยผู้ใช้ R สามารถระบุเพียง `matrix(..., byrow=TRUE)` เพื่อที่ว่าหน้าต่างย่อยถูกนับตามแถว (row-wise) นอกจากนี้ตัวเลข ในเมทริกซ์อาจถูกกำหนดให้เป็น order ใดก็ได้ ตัวอย่างเช่น `matrix(c(2, 1, 4, 3), 2, 2)`

โดยปริยาย ฟังก์ชัน `layout()` แบ่ง device ด้วยความสูงและความกว้างที่เป็นปกติ ซึ่งความสูงและความกว้างนี้สามารถถูกปรับเปลี่ยนได้ด้วยตัวเลือก `widths` และ `heights` มิติเหล่านี้ถูกกำหนดให้อย่างสัมพันธ์กัน ดังตัวอย่างต่อไปนี้[12]

```
> m <- matrix(1:4, 2, 2)
> layout(m, widths=c(1, 3),
          heights=c(3, 1))
> layout.show(4)
```



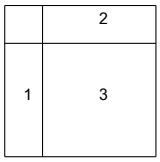

```
> m <- matrix(c(1,1,2,1),2,2)
> layout(m, widths=c(2, 1),
          heights=c(1, 2))
> layout.show(2)
```



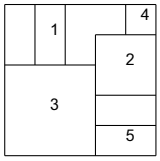
สุดท้ายจำนวนในเมทริกซ์สามารถรวมค่า 0 ที่ทำให้มีความเป็นไปได้ในการแบ่งส่วนของกราฟิกให้ดูซับซ้อนยิ่งขึ้น (หรือแม้กระทั่งเป็น esoterical)

[12]สามารถกำหนดให้อยู่ในหน่วยเซ็นติเมตรได้ ดู `?layout`.

```
> m <- matrix(0:3, 2, 2)
> layout(m, c(1, 3), c(1, 3))
> layout.show(3)
```




```
> m <- matrix(scan(), 5, 5)
1: 0 0 3 3 3 1 1 3 3 3
11: 0 0 3 3 3 0 2 2 0 5
21: 4 2 2 0 5
26:
Read 25 items
> layout(m)
> layout.show(5)
```



4.2 ฟังก์ชันกราฟิก (graphical functions)

ตารางแสดงภาพรวมของฟังก์ชันกราฟิกขั้นสูงใน R

`plot(x)`	พล็อตกราฟของค่า `x` (บนแกน y) ตามค่าบนแกน x
`plot(x, y)`	พล็อตกราฟขอบสองตัวแปรของค่า `x` (บนแกน x) และ `y` (บนแกน y)
`sunflowerplot(x, y)`	เหมือนคำอธิบายด้านบน แต่จุดตามพิกัดคล้ายกันถูกลากเหมือนดอกไม้ซึ่งจำนวนกลีบดอกแทนจำนวนจุด
`pie(x)`	กราฟแบบวงกลม
`boxplot(x)`	กราฟแบบ “box-and-whiskers”
`stripchart(x)`	พล็อตของค่า `x` บนเส้น (อีกทางเลือกสำหรับ `boxplot()` กรณีขนาดตัวอย่างขนาดเล็ก)
`coplot(x~y | z)`	พล็อตกราฟของ `x` และ `y` สำหรับแต่ละค่า (หรือช่วงของค่า) ของ `z`
`interaction.plot (f1, f2, y)`	ถ้า `f1` และ `f2` เป็นปัจจัย (factors) แล้วพล็อตค่าเฉลี่ยของ `y` (บนแกน y) ตามค่าของ `f1` (บนแกน x) และค่าของ `f2` (เส้นโค้งที่ต่างกัน) โดยที่ตัวเลือก `fun` ให้เลือกสถิติโดยสรุปของ `y` (ค่าปริยายคือ `fun=mean`)
`matplot(x,y)`	พล็อตกราฟสองตัวแปรของคอลัมน์แรกของ `x` กับคอลัมน์แรกของ `y` คอลัมน์ที่สองของ `x` กับคอลัมน์ที่สองของ `y` ไปเรื่อย ๆ
`dotchart(x)`	ถ้า `x` เป็นกรอบข้อมูล พล็อตกราฟ Cleveland dot (พล็อตกราฟแบบตั้งขึ้นแบบเส้นต่อเส้นและคอลัมน์ต่อคอลัมน์)
`fourfoldplot(x)`	ทำให้เห็นเป็นสี่ส่วนของวงกลมที่สัมพันธ์ระหว่างตัวแปรสองประเภท (dichotomous variables) สำหรับประชากรที่แตกต่างกัน (`x` ต้องเป็นอาร์เรย์ที่มี `dim=c(2, 2, k)` หรือเมทริกซ์ที่มี `dim=c(2, 2)` ถ้า $k = 1$)
`assocplot(x)`	กราฟ Cohen–Friendly ที่แสดงส่วนเบี่ยงเบนจากความอิสระของแถวและคอลัมน์ในตารางแจกแจงสองมิติ
`mosaicplot(x)`	กราฟ ‘mosaic’ ของ residuals จากการถดถอยเชิง log-linear ของตารางแจกแจง
`pairs(x)`	ถ้า `x` คือเมทริกซ์หรือกรอบข้อมูล วาดกราฟสองตัวแปรที่เป็นไปได้ทั้งหมดระหว่างคอลัมน์ของ `x`
`plot.ts(x)`	ถ้า `x` คืออ็อบเจ็กต์ของคลาส `"ts"` พล็อตกราฟของ `x` ที่แปรตามเวลา `x` อาจเป็นหลายตัวแปรแต่อนุกรมต้องมีความถี่และเวลาเดียวกัน
`ts.plot(x)`	เหมือนคำอธิบายด้านบน แต่ถ้า `x` เป็นหลายตัวแปรแล้วอนุกรมอาจมีเวลาต่างกันได้และต้องมีความถี่เดียวกัน
`hist(x)`	แผนภูมิแท่งของความถี่ของ `x`
`barplot(x)`	แผนภูมิแท่งของค่า `x`
`qqnorm(x)`	ควอร์ไทล์ของ `x` ตามค่าคาดหวังภายใต้กฎปกติ
`qqplot(x, y)`	ควอร์ไทล์ของ `y` ตามควอร์ไทล์ของ `x`
`contour(x, y, z)`	พล็อตกราฟที่แสดงรูปร่าง (ข้อมูลถูกเพิ่มการวาดเส้นโค้ง) `x` และ `y` ต้องเป็นเวกเตอร์และ `z` ต้องเป็นเมทริกซ์เพื่อว่า `dim(z)=c(length(x), length(y))` (`x` และ `y` อาจถูกละไว้)
`filled.contour (x, y, z)`	เหมือนคำอธิบายด้านบน แต่พื้นที่ระหว่างเส้นแสดงรูปเป็นสีและสีของคำอธิบายกราฟถูกวาดเช่นกัน
`image(x, y, z)`	เหมือนคำอธิบายด้านบน แต่ข้อมูลจริงถูกทำให้เป็นตัวแทนด้วยสี

persp(x, y, z)	เหมือนคำอธิบายด้านบน แต่ในภาพรวม
stars(x)	ถ้า x เป็นเมทริกซ์หรือกรอบข้อมูล วาดกราฟโดยแบ่งเป็นส่วนหรือเป็นดาว ที่แต่ละแถวของ x ถูกทำให้เป็นตัวแทนโดยดาวและคอลัมน์คือความยาวของส่วนแบ่ง
symbols(x, y, ...)	วาดที่พิกัดที่ถูกกำหนดให้โดย x และ yสัญลักษณ์ (วงกลม สี่เหลี่ยมจัตุรัส สี่เหลี่ยมผืนผ้า ดาว ที่วัดอุณหภูมิ หรือ “boxplots”) ซึ่งขนาด สีและอื่นๆ ถูกระบุโดยอาร์กิวเมนต์เพิ่มเติม
termplot(mod.obj)	พล็อตกราฟของผล (บางส่วน) ของแบบจำลองการถดถอย (mod.obj)

สำหรับแต่ละฟังก์ชัน ตัวเลือกอาจถูกสืบค้นได้ด้วย the on-line help ใน R บางส่วนของตัวเลือกเหล่านี้เป็นเหมือนกันกับฟังก์ชันกราฟิกหลายฟังก์ชัน ต่อไปนี้จะเป็นฟังก์ชันหลัก (กับค่าปริยายที่เป็นไปได้ของฟังก์ชัน)

add=FALSE	ถ้าเป็นจริง (TRUE) แล้วกำหนดเหนือพล็อตกราฟก่อนหน้านี้ (ถ้ายังคงมีอยู่)
axes=TRUE	ถ้าเป็นเท็จ (FALSE) แล้วไม่วาดแกนและกล่อง
type="p"	ระบุชนิดของพล็อต "p" คือจุด "l" คือเส้น "b" คือจุดที่ถูกเชื่อมต่อโดยเส้น "o" คือ คล้าย "b" แต่เส้นอยู่เหนือจุด "h": คือ เส้นแนวตั้ง "s" คือ ขั้นที่ข้อมูลถูกเป็นตัวแทนโดยส่วนบนสุดของเส้นแนวตั้ง "S" คล้าย "s" แต่ข้อมูลถูกเป็นตัวแทนโดยส่วนล่างสุดของเส้นแนวตั้ง
xlim=, ylim=	ระบุขีดจำกัดล่างและขีดจำกัดบนของแกน ตัวอย่างเช่น xlim=c(1, 10) หรือ xlim=range(x)
xlab=, ylab=	อธิบายแกน ต้องเป็นตัวแปรของโหมดอักขระ
main=	ชื่อเรื่องหลัก ต้องเป็นตัวแปรโหมดอักขระ
sub=	ชื่อเรื่องรอง (ถูกเขียนในตัวอักษรที่เล็กกว่า)

4.3 คำสั่งการวาดระดับต่ำ

R มีชุดของฟังก์ชันกราฟิกซึ่งมีผลต่อกราฟที่มีอยู่แล้ว และถูกให้ชื่อว่า *คำสั่งการวาดระดับต่ำ (low-level plotting commands)* ตารางต่อไปนี้เป็นชุดคำสั่งหลักดังกล่าว

points(x, y)	เติมจุด (ตัวเลือก type= สามารถถูกใช้ได้)
lines(x, y)	เหมือนคำอธิบายด้านบนแต่เป็นการเพิ่มเส้น
text(x, y, labels, ...)	เพิ่มข้อความที่กำหนดโดยฟังก์ชัน labels ที่พิกัด (x,y) การใช้แบบมาตรฐานคือ plot(x, y, type="n"); text(x, y, names)

`mtext(text, side=3, line=0, ...)`	เพิ่มข้อความที่กำหนดโดย ฟังก์ชัน `text` ในขอบที่ถูกระบุโดย ฟังก์ชัน `side` (ดู `axis()` ข้างล่าง) และ ฟังก์ชัน `line` ระบุเส้นจากพื้นที่พล็อตกราฟ
`segments(x0, y0, x1, y1)`	ลากเส้นจากจุด (`x0`,`y0`) ไปที่จุด (`x1`,`y1`)
`arrows(x0, y0, x1, y1, angle=30, code=2)`	เหมือนคำอธิบายด้านบนโดยมีลูกศรที่จุด (`x0`,`y0`) ถ้า `code=2` ที่จุด (`x1`,`y1`) ถ้า `code=1` หรือทั้งคู่ถ้า `code=3` ส่วนฟังก์ชัน `angle` ควบคุมแกนของลูกศรไปที่ขอบของหัวลูกศร
`abline(a,b)`	วาดเส้นของความชัน `b` และมีจุดตัดแกน `a`
`abline(h=y)`	วาดเส้นแนวนอนที่พิกัดที่สอง `y`
`abline(v=x)`	วาดเส้นแนวตั้งที่พิกัดที่หนึ่ง `x`
`abline(lm.obj)`	วาดเส้นการถดถอยที่ถูกกำหนดโดย `lm.obj` (ดูส่วนที่ 5)
`rect(x1, y1, x2, y2)`	วาดสี่เหลี่ยมผืนผ้าซึ่ง ขีดจำกัดซ้าย ขวา ล่างและบนคือ `x1`, `x2`, `y1` และ `y2` ตามลำดับ
`polygon(x, y)`	วาดรูปหลายเหลี่ยมซึ่งเชื่อมจุดด้วยพิกัดที่ถูกกำหนดโดย `x` และ `y`
`legend(x, y, legend)`	เติมคำอธิบายที่จุด (`x`,`y`) ด้วยสัญลักษณ์ที่ถูกกำหนดโดยฟังก์ชัน `legend`
`title()`	เติมชื่อเรื่องและโดยทางเลือกชื่อเรื่องย่อย
`axis(side, vect)`	เติมแกนที่ด้านล่าง (`side=1`) ที่ด้านซ้าย (2) ที่ด้านบนสุด (3) หรือที่ด้านขวา (4) โดยที่ `vect` เป็นทางเลือกที่ให้พิกัดที่หนึ่งหรือพิกัดที่สองเป็นที่วาดเครื่องหมายกำกับ (tick-marks)
`box()`	เพิ่มกล่องรอบพล็อตกราฟปัจจุบัน
`rug(x)`	วาดข้อมูล `x` บนแกน x ให้เหมือนเส้นแนวตั้งขนาดเล็ก
`locator(n, type="n", ...)`	ได้ผลเป็นพิกัด (x, y) ภายหลังผู้ใช้ได้กด `n` ครั้งบนพล็อตด้วยเมาส์ รวมถึงวาดสัญลักษณ์ (`type="p"`) หรือ เส้น (`type="l"`) ตามพารามิเตอร์กราฟิกทางเลือก (`...`) ซึ่งโดยค่าปริยายคือไม่ถูกวาด (`type="n"`)

ข้อสังเกตคือความเป็นไปได้ที่จะเพิ่มนิพจน์ทางคณิตศาสตร์บนการพล็อตหนึ่งด้วย `text(x, y, expression(...))` ที่ซึ่งฟังก์ชัน `expression` เปลี่ยน (transform) อาร์กิวเมนต์ของมันเป็นสมการทางคณิตศาสตร์ ตัวอย่างเช่น

```
> text(x, y, expression(p == over(1, 1+e^-(beta*x+alpha))))
```

จากตัวอย่างข้างต้นจะแสดงบนพล็อตตามสมการดังต่อไปนี้ที่จุดใดจุดหนึ่งของพิกัด (x, y):

$$\mathrm{p} = \frac{1}{1 + \mathrm{e}^{-(\beta \mathrm{x} + \alpha)}}$$

เพื่อที่จะรวมตัวแปรหนึ่งในนิพจน์หนึ่ง เราสามารถใช้ฟังก์ชัน `substitute` และ `as.expression` ตัวอย่างเช่น เพื่อรวมค่าของ R^2 (ก่อนหน้านี้ได้คำนวณและเก็บไว้ในอ็อบเจ็กต์หนึ่ง มีชื่อว่า `Rsquared`) ดังตัวอย่างต่อไปนี้

```
> text(x, y, as.expression(substitute(R^2==r, list(r=Rsquared))))
```

จากข้างต้นจะแสดงบนพล็อตที่จุดใดจุดหนึ่งของพิกัด (x, y) ดังนี้

$$\mathrm{R}^2 = 0.9856298$$

เพื่อให้แสดงเพียงทศนิยมสามตำแหน่ง เราสามารถเปลี่ยนรหัสได้ตามต่อไปนี้

```
> text(x, y, as.expression(substitute(R^2==r,
+                                 list(r=round(Rsquared, 3)))))
```

ซึ่งจะแสดงดังนี้

$$\mathrm{R}^2 = 0.986$$

สุดท้ายเพื่อที่จะเขียนใน R เป็นตัวเอน (italics) ทำได้ดังต่อไปนี้

```
> text(x, y, as.expression(substitute(italic(R)^2==r,
+                                 list(r=round(Rsquared, 3)))))
```

$$R^2 = 0.986$$

4.4 พารามิเตอร์กราฟิก (graphical parameters)

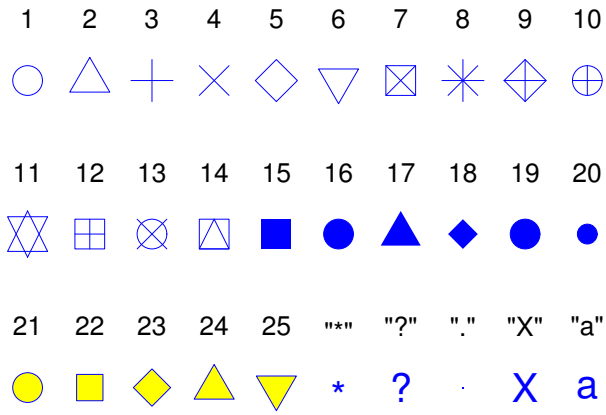
นอกเหนือจากชุดคำสั่งการวาดระดับต่ำแล้ว การนำเสนอของกราฟิกสามารถทำให้ดีขึ้นได้ด้วยพารามิเตอร์กราฟิก พารามิเตอร์นี้สามารถถูกใช้ด้วยตัวเลือกของฟังก์ชันกราฟิก (แต่มันทำงานไม่ได้ทั้งหมด) หรือด้วยฟังก์ชัน `par` เพื่อเปลี่ยนพารามิเตอร์กราฟิกอย่างถาวร เช่น พล็อตที่ภายหลังจะถูกวาดตามแต่ที่พารามิเตอร์ที่ถูกระบุโดยผู้ใช้ R เช่น คำสั่งดังต่อไปนี้

```
> par(bg="yellow")
```

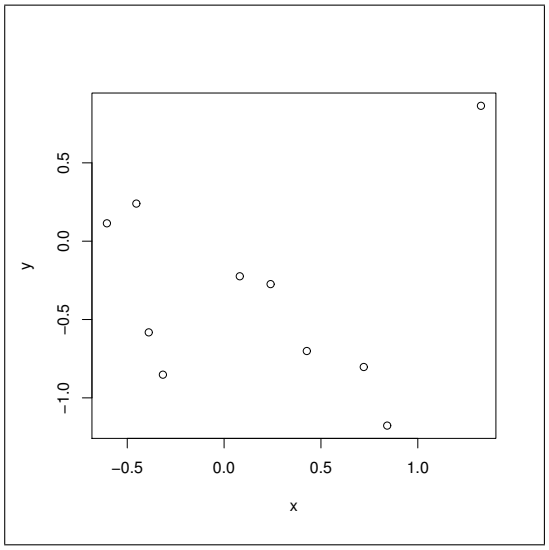
จากข้างต้นจะได้ผลที่ตามมาเป็นการพล็อตที่ถูกวาดด้วยพื้นหลังสีเหลือง มีพารามิเตอร์กราฟิก 73 แบบ บางแบบของ พารามิเตอร์เหล่านี้มีฟังก์ชันที่คล้ายกันมาก รายการที่มากมายของพารามิเตอร์เหล่านี้สามารถหาอ่านได้ด้วยฟังก์ชัน `?par` ซึ่งตารางดังต่อไปนี้ผู้เขียนจะยกมาเฉพาะฟังก์ชันที่ใช้เป็นประจำมากที่สุด

`adj`	ควบคุมการจัดแนวของข้อความตามขอบซ้ายของข้อความโดยที่ `0` คือ การจัดชิดซ้าย `0.5` คือการจัดตรงกลาง `1` คือการจัดชิดขวา ส่วนค่า > 1 คือเคลื่อนข้อความไปทางซ้ายมากขึ้นและค่าลบหมายถึงเคลื่อนข้อความมาทางขวามากขึ้น กรณีที่มีสองค่าถูกกำหนดไว้ (เช่น `c(0, 0)`) ค่าที่สองควบคุมการจัดแนวข้อความตามแนวตั้งตามเส้นฐานของของข้อความ

bg	ระบุสีของพื้น (เช่น bg="red", bg="blue" ส่วนรายการของสีที่มีได้ 657 สีถูกแสดงได้ด้วย colors())
bty	ควบคุมชนิดของกล่องที่ถูกวาดรอบพล็อต ค่าที่ใช้ได้มีดังนี้ "o", "l", "7", "c", "u", "]" (กล่องซึ่งคล้ายลักษณะเป็นตามส่วน) ถ้า bty="n" แล้วกล่องไม่ถูกวาด
cex	ค่าที่ควบคุมขนาดของข้อความและสัญลักษณ์ตามค่าโดยปริยาย ซึ่งพารามิเตอร์ดังต่อไปนี้มีการควบคุมแบบเดียวกัน สำหรับตัวเลขบนแกนคือ cex.axis สำหรับการลงรายการบนแกนคือ cex.lab สำหรับชื่อเรื่องคือ cex.main และสำหรับชื่อเรื่องย่อยคือ cex.sub
col	ควบคุมสีของสัญลักษณ์ ตามที่ cex มี ได้แก่ col.axis, col.lab, col.main และ col.sub
font	จำนวนเต็มซึ่งควบคุมรูปแบบของข้อความ (1 คือ 2 คือตัวเอน 3 คือตัวหนา และ 4 คือตัวเอนที่หนา) ตามที่ cex มี ได้แก่ font.axis, font.lab, font.main และ font.sub
las	จำนวนเต็มซึ่งควบคุมการจัดวางของการลงรายการบนแกน (0 คือขนานกับแกน 1 คือตามแนวนอน 2 คือตามแนวตั้งฉากกับแกน และ 3 คือตามแนวตั้ง)
lty	ควบคุมชนิดของเส้นที่มีค่าเป็นจำนวนเต็ม (1 คือเส้นทึบ 2 คือเส้นประ 3 คือเส้นจุด 4 คือเส้นจุดประ 5 คือเส้นประยาว 6 คือเส้นประ 2 เส้น) หรือเป็นแถวได้ถึง 8 ลักษณะ (ระหว่าง "0" และ "9") ซึ่งระบุความยาวเป็นทางเลือก ในจุดหรือจุดภาพ (pixels) ของอีลีเมนท์ที่ถูกวาดและเป็นที่ว่าง ตัวอย่างเช่น lty="44" จะมีผลแบบเดียวกับ lty=2
lwd	ค่าตัวเลขซึ่งควบคุมความกว้างของเส้น
mar	เวกเตอร์ของค่าตัวเลข 4 ค่าซึ่งควบคุมที่ว่างระหว่างแกนและขอบของกราฟของแบบ c(bottom, left, top, right) โดยที่ค่าปริยาย c(5.1, 4.1, 4.1, 2.1)
mfcol	เวกเตอร์ของแบบ c(nr,nc) ซึ่งแบ่งหน้าต่างกราฟิกคล้ายเมทริกซ์ของ nr แถวและ nc คอลัมน์ และตามด้วยพล็อตถูกวาดในแนวคอลัมน์ (ดูส่วน 4.1.2)
mfrow	เหมือนคำอธิบายข้างบนแต่ตามด้วยพล็อตถูกวาดในแนวแถว (ดูส่วน 4.1.2)
pch	ควบคุมชนิดของสัญลักษณ์โดยไม่เป็นจำนวนเต็มระหว่าง 1 ถึง 25 ก็เป็นอักขระเดี่ยวที่อยู่ภายในเครื่องหมาย "" (รูปที่ 2)
ps	จำนวนเต็มซึ่งควบคุมขนาดในจุดของข้อความและสัญลักษณ์
pty	ลักษณะซึ่งระบุชนิดของพื้นที่การพล็อต "s" คือสี่เหลี่ยมจัตุรัส "m" คือใหญ่สุด
tck	ค่าซึ่งระบุความยาวของเครื่องหมายกำกับบนแกนตามเศษส่วนของความกว้างที่เล็กที่สุดหรือความสูงของพล็อต โดย tck=1 ตาตาราง (grid) ถูกวาด
tcl	เหมือนคำอธิบายข้างบนแต่เศษส่วนของความสูงของเส้นของข้อความ (ค่าปริยาย tcl=-0.5)
xaxt	ถ้า xaxt="n" แล้วแกน x ถูกจัดวางแต่ไม่ถูกวาด (มีประโยชน์ในการเชื่อมด้วย axis axis(side=1, ...))
yaxt	ถ้า yaxt="n" แล้วแกน y ถูกจัดวางแต่ไม่ถูกวาด (มีประโยชน์ในการเชื่อมด้วย axis axis(side=2, ...))



รูปที่ 2: สัญลักษณ์การวาด ใน R (`pch=1:25`) สีได้มาด้วยตัวเลือกเช่น `col="blue"`, `bg="yellow"` ตัวเลือกที่สองมีผลเฉพาะสัญลักษณ์ที่ 21–25 ลักษณะใด ๆ ก็ตามสามารถถูกใช้ได้ (เช่น`pch="*"`, `"?"`, `"."`, และอื่นๆ)



รูปที่ 3: แสดงฟังก์ชัน `plot` ที่ถูกใช้โดยไม่ใช้ตัวเลือก

4.5 ตัวอย่างภาคปฏิบัติ

เพื่อที่จะแสดงการใช้ฟังก์ชันกราฟิกใน R เรามาพิจารณาตัวอย่างที่ง่ายของกราฟสองตัวแปร (bivariate graph) ของ 10 คู่ของตัวแปรสุ่ม ค่าเหล่านี้ถูกสร้างได้ด้วยฟังก์ชันต่อไปนี้

```
> x <- rnorm(10)
> y <- rnorm(10)
```

กราฟที่ต้องการจะได้มาด้วยฟังก์ชัน `plot()` ซึ่งผู้ใช้ R จะพิมพ์คำสั่งต่อไปนี้

```
> plot(x, y)
```

และกราฟนี้จะถูกพล็อตบน active graphical device ซึ่งผลที่ได้ถูกแสดงดังรูปที่ 3 โดยปริยาย R สร้างกราฟในวิธีที่ฉลาด (“intelligent”) หมายความว่า ช่องว่างระหว่างเครื่องหมายกำกับ (tick-marks) บนแกน การวางตำแหน่งของการลงรายการ (labels) และอื่น ๆ ถูกคำนวณเพื่อว่ากราฟที่แสดงผลเป็นกราฟที่เข้าใจได้ง่ายเท่าที่จะเป็นไปได้

อย่างไรก็ตาม ผู้ใช้ R อาจเปลี่ยนวิธีการนำเสนอกราฟได้ ตัวอย่างเช่น เพื่อให้สอดคล้องกับรูปแบบ pre-defined editorial style หรือเพื่อให้กราฟแสดงออกมาเฉพาะตัวสำหรับการบรรยาย วิธีที่ง่ายที่สุดที่จะเปลี่ยนการนำเสนอของกราฟคือการเพิ่มตัวเลือกซึ่งจะเปลี่ยนแปลงอาร์กิวเมนต์โดยปริยาย ในตัวอย่างต่อไปเราจะเปลี่ยนรูปภาพอย่างมีนัยสำคัญในวิธีดังต่อไปนี้

```
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red",
     bg="yellow", bty="l", tcl=0.4,
    main="How to customize a plot with R", las=1, cex=1.5)
```

จากข้างต้นได้ผลแสดงในรูปที่ 4 ในที่นี้จะกล่าวโดยละเอียดในแต่ละตัวเลือกที่ถูกใช้ อย่างแรก `xlab` และ `ylab` เปลี่ยนการลงรายการของแกนจากที่เป็นค่าปริยายเป็นชื่อของตัวแปร หลังจากนั้น `xlim` และ `ylim` ยอมให้สามารถกำหนด limits ของทั้งสองแกน พารามิเตอร์กราฟิก[13] `pch` ถูกใช้ในที่นี้เพื่อเป็นตัวเลือก หมายความว่า `pch=22` เป็นการระบุให้เป็นรูปสี่เหลี่ยมซึ่งมีเส้นแสดงของรูปสี่เหลี่ยมและสีพื้นอาจแตกต่างกันและตามที่กำหนดให้ตามลำดับโดยการใช้ `col` และ `bg` ตารางของพารามิเตอร์กราฟิกได้ให้ความหมายของการเปลี่ยนแปลงที่ถูกกระทำโดย `bty`, `tcl`, `las` และ `cex` สุดท้ายชื่อกราฟถูกเพิ่มโดยใช้ตัวเลือก main

พารามิเตอร์กราฟิกและฟังก์ชันการวาดระดับต่ำ ยอมให้ผู้ใช้ทำอะไรได้มากขึ้นในการนำเสนอในรูปแบบกราฟ ตามที่ผู้ใช้ได้เห็นแล้วก่อนหน้านี้ พารามิเตอร์กราฟิกบางพารามิเตอร์

[13] By default, R adds 4% on each side of the axis limit. This behaviour may be altered by setting the graphical parameters `xaxs="i"` and `yaxs="i"` (they can be passed as options to `plot()`).

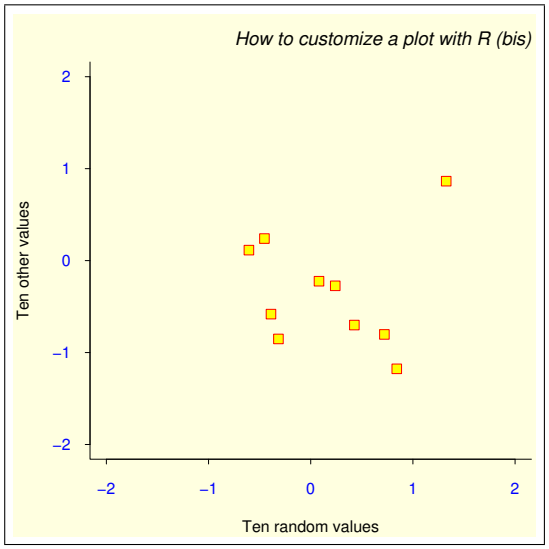
รูปที่ 4: ฟังก์ชัน `plot` ที่ถูกใช้แบบมีตัวเลือก

ไม่สามารถถูกส่งไปยังฟังก์ชันในฐานะอาร์กิวเมนต์ได้ เช่น ฟังก์ชัน `plot` ถึงตรงนี้ จะเปลี่ยนแปลงบางอย่างของพารามิเตอร์เหล่านี้ด้วย `par()` ดังนั้นมีความจำเป็นที่ต้องพิมพ์คำสั่งหลายคำสั่ง เมื่อพารามิเตอร์กราฟิกถูกเปลี่ยน จะเป็นประโยชน์หากทำการบันทึกค่าเริ่มต้นของมันก่อนเพื่อจะได้สามารถกู้กลับมาใช้ในภายหลังได้ ต่อไปนี้เป็นชุดคำสั่งที่ถูกใช้เพื่อให้ได้รูปที่ 5.

```
opar <- par()
par(bg="lightyellow", col.axis="blue", mar=c(4, 4, 2.5, 0.25))
plot(x, y, xlab="Ten random values", ylab="Ten other values",
     xlim=c(-2, 2), ylim=c(-2, 2), pch=22, col="red", bg="yellow",
      bty="l", tcl=-.25, las=1, cex=1.5)
title("How to customize a plot with R (bis)", font.main=3, adj=1)
par(opar)
```

มาดูในรายละเอียดผลของการกระทำจากชุดคำสั่งเหล่านี้ เริ่มจากพารามิเตอร์กราฟิกที่เป็นค่าปริยายถูกคัดลอกในรายการซึ่งในที่นี้เรียกว่า `opar` หลังจากนั้นพารามิเตอร์ 3 ตัวถูกเปลี่ยนแปลง หมายความว่า `bg` สำหรับสีของพื้นหลัง `col.axis` สีของตัวเลขที่อยู่บนแกน และ `mar` สำหรับขนาดของขอบรอบพื้นที่ การพล็อตกราฟนี้ถูกวาดในวิธีที่เกือบคล้ายกับรูปที่ 4 การเปลี่ยนแปลงของกรอบยอมให้ใช้พื้นที่ว่างรอบๆ พื้นที่การพล็อตได้ ในที่นี้ชื่อเรื่องถูกเติมโดยใช้ฟังก์ชั่นการวาดระดับต่ำคือฟังก์ชัน `title` ซึ่งยอมให้บางพารามิเตอร์เป็นอาร์กิวเมนต์โดยไม่เปลี่ยนแปลงบริเวณที่เหลือของกราฟ สุดท้ายค่าพารามิเตอร์กราฟิกเริ่มต้นถูกเก็บด้วยคำสั่งสุดท้าย

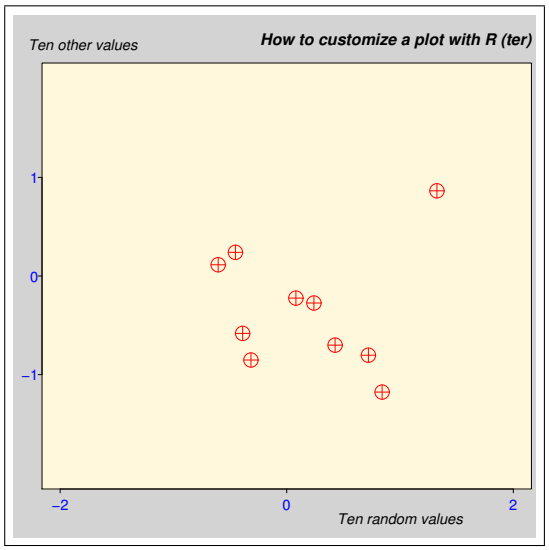
ณ ตอนนี้ อยู่ในการควบคุมทั้งหมด ตามรูปที่ 5 R ยังคงกำหนดบางอย่าง เช่น จำ-



รูปที่ 5: ฟังก์ชัน `par`, `plot` และ `title`

นวนของเครื่องหมายกำกับ (tick marks) บนแกน หรือช่องว่างระหว่างชื่อเรื่องและพื้นที่การพล็อต ในเวลานี้เราเข้าใจแล้วว่าทำอย่างไรที่จะควบคุมทั้งหมดในการนำเสนอกราฟ ในที่นี้วิธีการที่ถูกใช้เริ่มจากการพล็อต กราฟที่ว่าง (“blank” graph) ด้วยคำสั่ง `plot(..., type="n")` แล้วตามด้วยการเติมจุด แกน การเขียนข้อความลงรายการ และอื่น ๆ ด้วยฟังก์ชั่นการวาดระดับต่ำ เราจะทำการปรับค่าบางอย่าง เช่น การเปลี่ยนสีของพื้นที่การพล็อต โดยใช้ชุดคำสั่งต่อไปนี้และกราฟที่ได้จากชุดคำสั่งนี้คือรูปที่ 6

```
opar <- par()
par(bg="lightgray", mar=c(2.5, 1.5, 2.5, 0.25))
plot(x, y, type="n", xlab="", ylab="", xlim=c(-2, 2),
     ylim=c(-2, 2), xaxt="n", yaxt="n")
rect(-3, -3, 3, 3, col="cornsilk")
points(x, y, pch=10, col="red", cex=2)
axis(side=1, c(-2, 0, 2), tcl=-0.2, labels=FALSE)
axis(side=2, -1:1, tcl=-0.2, labels=FALSE)
title("How to customize a plot with R (ter)",
      font.main=4, adj=1, cex.main=1)
mtext("Ten random values", side=1, line=1, at=1, cex=0.9, font=3)
mtext("Ten other values", line=0.5, at=-1.8, cex=0.9, font=3)
mtext(c(-2, 0, 2), side=1, las=1, at=c(-2, 0, 2), line=0.3,
      col="blue", cex=0.9)
mtext(-1:1, side=2, las=1, at=-1:1, line=0.2, col="blue", cex=0.9)
par(opar)
```



รูปที่ 6: กราฟ “hand-made”

เช่นเดียวกับก่อนหน้านี้พารามิเตอร์กราฟิกที่เป็นค่าปริยายถูกบันทึกไว้และสีของพื้นหลังและขอบถูกเปลี่ยนแปลง จากนั้นกราฟถูกวาดด้วย `type="n"` เพื่อที่ไม่พล็อตจุด `xlab=""`, `ylab=""` เพื่อที่ไม่เขียนการลงรายการบนแกน และ `xaxt="n"`, `yaxt="n"` เพื่อที่ไม่วาดแกน ผลในการวาดนี้เป็นเพียงกล่องล้อมรอบพื้นที่การพล็อตและการกำหนดแกนที่ใช้ `xlim` และ `ylim` ข้อสังเกตคือน่าจะใช้ตัวเลือก `axes=FALSE` แต่ในกรณีนี้ทั้งแกนและกล่องจะไม่ถูกวาด

ต่อมาอีลีเมนท์ถูกเติมในพื้นที่การพล็อตซึ่งถูกกำหนดด้วยฟังก์ชันการวาดระดับต่ำบางฟังก์ชัน ก่อนการเติมจุด สีภายในพื้นที่การพล็อตถูกเปลี่ยนด้วย `rect()` หมายความว่าขนาดของสี่เหลี่ยมถูกเลือกเพื่อให้มีขนาดใหญ่มากขึ้นในพื้นที่การพล็อต

จุดถูกพล็อตด้วย `points()` ซึ่งเป็นสัญลักษณ์ใหม่ที่ถูกใช้ แกนถูกเติมด้วย `axis()` หมายความว่า เวกเตอร์ถูกกำหนดให้เป็นอาร์กิวเมนต์ที่สองซึ่งเป็นตัวระบุพิกัดของเครื่องหมายกำกับ ตัวเลือก `labels=FALSE` ระบุว่าต้องไม่มีคำอธิบายประกอบถูกเขียนกับเครื่องหมายกำกับ นอกจากนี้ตัวเลือกนี้ยอมรับเวกเตอร์ของโหมดอักขระด้วย ตัวอย่างเช่น `labels=c("A", "B", "C")`

ชื่อเรื่องถูกเติมด้วย `title()` แต่ตัวอักษรถูกเปลี่ยนเล็กน้อย คำอธิบายประกอบบนแกนถูกเขียนด้วย `mtext()` (*marginal text*) อาร์กิวเมนต์แรกสุดของฟังก์ชันนี้คือเวกเตอร์ของโหมดอักขระซึ่งให้ข้อความที่ถูกเขียนขึ้น ตัวเลือก `line` บ่งบอกถึงระยะห่างจากพื้นที่การพล็อต (โดยค่าปริยาย `line=0`) และ `at` คือพิกัด ส่วนการเรียก `mtext()` ครั้งที่สองใช้ค่าปริยายของ `side` (3) และ อีกสอง `mtext()` ที่เหลือคือการผ่านเวกเตอร์ตัวเลขเป็นอาร์กิวเมนต์แรกที่จะถูกเปลี่ยนให้กลายเป็นอักขระ

4.6 แพ็กเกจ grid และ lattice

แพ็กเกจ grid และ lattice ทำให้เกิดผลกับระบบ grid และ lattice grid เป็นโหมดกราฟิกรูปแบบใหม่พร้อมด้วยระบบของตัวเองของพารามิเตอร์กราฟิกซึ่งแตกต่างจากสิ่งที่เราได้เห็นก่อนหน้านี้ ความแตกต่างหลักสองประการของ grid เมื่อเปรียบเทียบกับกราฟิก base มีดังนี้

- วิธีที่ยืดหยุ่นมากกว่าเพื่อแยก graphical devices โดยการใช้ viewports ซึ่งสามารถทับซ้อนกันได้ (อ็อบเจ็กต์กราฟิก อาจแม้กระทั่งถูกแบ่งส่วนระหว่าง viewports ที่แตกต่างกัน เช่น ลูกศร)
- อ็อบเจ็กต์กราฟิก (grob) อาจถูกเปลี่ยนแปลงหรือเอาออกจากกราฟหนึ่งโดยไม่มีความจำเป็นที่ต้องวาดกราฟทั้งหมดขึ้นใหม่อีกครั้ง (แต่ต้องทำใหม่กับกราฟิก base)

Grid graphics ปกติแล้วไม่สามารถถูกรวมหรือผสมกับกราฟิก base ได้ (แพ็กเกจ gridBase ต้องถูกใช้เพื่อทำเรื่องนี้) อย่างไรก็ตามมีความเป็นไปได้ที่จะใช้โหมดกราฟิกทั้งคู่ใน session เดียวกันตาม graphical device เดียวกัน

Lattice เป็นสิ่งที่จำเป็นในการนำไปใช้ใน R ของกราฟิก Trellis ของ S-PLUS Trellis เป็นการเข้าถึงรูปแบบหนึ่งเพื่อทำให้เกิดภาพของข้อมูลหลายตัวแปร ซึ่งเหมะสมเป็นอย่างยิ่งสำหรับการสำรวจความสัมพันธ์หรือปฏิสัมพันธ์ระหว่างตัวแปร[14] แนวคิดหลักเบื้องหลัง lattice (และรวมถึง Trellis) คือ conditional multiple graphs กล่าวคือ กราฟสองตัวแปรจะถูกแยกออกเป็นหลายกราฟที่เกี่ยวข้องกับค่าของตัวแปรที่สาม ฟังก์ชัน `coplot` ใช้ในแนวทางคล้ายกัน แต่ lattice ให้ช่วงของการทำงานที่กว้างกว่ามาก lattice ใช้โหมดกราฟิกแบบตาราง (grid graphical mode)

ฟังก์ชันส่วนใหญ่ใน lattice มีสูตรที่เป็นอาร์กิวเมนต์หลักของมัน เช่น `y ~ x` [15] สูตร `y ~ x | z` หมายถึงกราฟของ `y` ที่เกี่ยวข้องกับ `x` จะถูกพล็อตเป็นหลายกราฟซึ่งเกี่ยวข้องกับค่าของ `z`

ตารางต่อไปนี้แสดงถึงฟังก์ชันหลักใน lattice สูตรที่กำหนดให้เป็นอาร์กิวเมนต์ที่เป็นสูตรที่จำเป็นโดยทั่วไป แต่ฟังก์ชันเหล่านี้ยอมรับสูตรแบบเงื่อนไข (`y ~ x | z`) ว่าเป็นอาร์กิวเมนต์หลัก ซึ่งในกรณีหลังกราฟพหุคูณ (multiple graph) ที่เกี่ยวข้องกับค่าของ `z` ถูกพล็อตตามตัวอย่างข้างล่างนี้

[14] http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/index.html

[15] `plot()` ยอมรับสูตรเช่นกันตามอาร์กิวเมนต์หลัก หมายความว่า ถ้า `x` และ `y` คือเวกเตอร์สองเวกเตอร์ของความยาวเดียวกันแล้ว `plot(y ~ x)` และ `plot(x, y)` จะให้กราฟที่เหมือนกัน

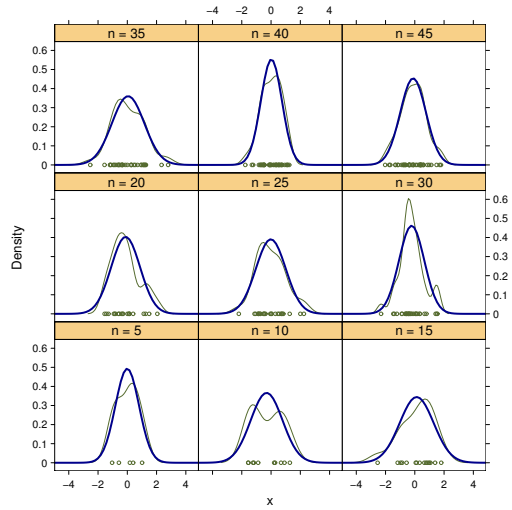
`barchart(y ~ x)`	แผนภูมิแท่งของค่า `y` ตามค่าของ `x`
`bwplot(y ~ x)`	พล็อตแบบ “box-and-whiskers”
`densityplot(~ x)`	พล็อต density functions
`dotplot(y ~ x)`	พล็อต Cleveland dot (พล็อตแบบตั้ง เส้นต่อเส้น และคอลัมน์ต่อคอลัมน์)
`histogram(~ x)`	แผนภูมิแท่งของความถี่ `x`
`qqmath(~ x)`	ควอร์ไทล์ของ `x` ตามค่าที่คาดหมายภายใต้การกระจายตามทฤษฎี
`stripplot(y ~ x)`	พล็อตหนึ่งมิติ โดย `x` ต้องเป็นตัวเลข `y` อาจเป็นปัจจัย (factor)
`qq(y ~ x)`	ควอร์ไทล์ที่เปรียบเทียบสองการกระจาย โดย `x` ต้องเป็นตัวเลข `y` อาจเป็นตัวเลข อักขระหรือ factor ก็ได้แต่ต้องมีสอง ‘levels’
`xyplot(y ~ x)`	พล็อตสองตัวแปร (กับฟังก์ชันมากมาย)
`levelplot(z ~ x*y)` `contourplot(z ~ x*y)`	พล็อตสีของค่าของ `z` ที่พิกัดที่กำหนดให้โดย `x` และ `y` (`x`, `y` และ codez ทั้งหมดมีความยาวเดียวกัน)
`cloud(z ~ x*y)`	พล็อตภาพรวมแบบ 3-D (จุด)
`wireframe(z ~ x*y)`	พล็อตภาพรวมแบบ 3-D (ผิว)
`splom(~ x)`	เมทริกซ์ของพล็อตสองตัวแปร
`parallel(~ x)`	พล็อตพิกัดที่ขนานกัน

ตอนนี้เรามาทำความเข้าใจตัวอย่างบางตัวอย่างเพื่อที่จะอธิบายประกอบในแง่มุมบางอย่างของแพ็กเกจ lattice แพ็กเกจนี้ต้องถูกใส่เข้าในไปความจำด้วยคำสั่ง `library(lattice)` เพื่อให้ฟังก์ชันสามารถถูกเข้าถึงได้

เริ่มต้นด้วยกราฟของฟังก์ชันความหนาแน่น กราฟนี้สามารถถูกทำได้โดยง่ายด้วย `densityplot(~ x)` ซึ่งจะพล็อตส่วนโค้งของฟังก์ชันความหนาแน่นที่ได้จากการสังเกต (empirical density function) ด้วยจุดที่ตรงกับการสังเกตบนแกน x (คล้ายกับ `rug()`) ตัวอย่างนี้ยากกว่าปกติเล็กน้อยเนื่องด้วยมีการซ้อนในแต่ละพล็อตของเส้นโค้งของความหนาแน่นที่ได้จากการสังเกตและค่าที่เกิดจากการคาดการณ์จากกฎปกติ (normal law) จึงมีความจำเป็นที่ต้องใช้อาร์กิวเมนต์ `panel` ซึ่งกำหนดสิ่งที่ถูกวาดในแต่ละพล็อต ต่อไปนี้คือชุดคำสั่ง

```
n <- seq(5, 45, 5)
x <- rnorm(sum(n))
y <- factor(rep(n, n), labels=paste("n =", n))
densityplot(~ x | y,
            panel = function(x, ...) {
              panel.densityplot(x, col="DarkOliveGreen", ...)
                 panel.mathdensity(dmath=dnorm,
                                   args=list(mean=mean(x), sd=sd(x)),
                                         col="darkblue")
            })
```

สามบรรทัดแรกของคำสั่งสร้างตัวอย่างสุ่มของตัวแปรปกติอิสระ ซึ่งถูกแยกในตัวอย่างย่อยของขนาดเท่ากับ 5, 10, 15, . . . , และ 45 หลังจากนั้นในส่วนของการเรียก `densityplot`



รูปที่ 7: ฟังก์ชัน `densityplot`

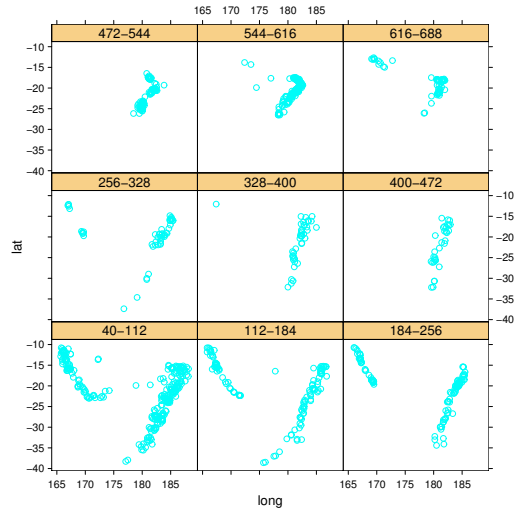
จะทำการสร้างพล็อตหนึ่งพล็อตเพื่อแต่ละตัวอย่างย่อย `panel` ใช้อาร์กิวเมนต์เป็นฟังก์ชันในตัวอย่างนี้ได้กำหนดฟังก์ชันหนึ่งซึ่งเรียกอีก 2 ฟังก์ชันซึ่งฟังก์ชันนิยามก่อนในแพ็กเกจ lattice `panel.densityplot` เป็นฟังก์ชันความหนาแน่นที่วาดจากข้อมูลจริง ส่วน `panel.mathdensity` เป็นฟังก์ชันความหนาแน่นที่ได้จากการคาดการณ์จากเกณฑ์มาตรฐาน ฟังก์ชัน `panel.densityplot` ถูกให้ชื่อว่า `panel` โดยค่าปริยายถ้าไม่ถูกกำหนดโดยอาร์กิวเมนต์ หมายความว่า คำสั่ง `densityplot (~ x | y)` จะให้ผลลัพธ์เป็นกราฟเช่นเดียวกับรูปที่ 7 แต่ไม่มีเส้นโค้งสีน้ำเงิน

ตัวอย่างถัดไปถูกดัดแปลงไม่มากก็น้อยจาก help pages ของแพ็กเกจ lattice และชุดข้อมูลบางชุดหาได้ใน R โดยเป็นชุดข้อมูลของที่ตั้งของเหตุการณ์ที่เกี่ยวข้องกับแผ่นดินไหว (seismic) ใกล้หมู่เกาะฟิจิและทำการวัดดอกไม้บางลักษณะของพืชไอริส (iris) สามชนิด

รูปที่ 8 แสดงที่ตั้งทางภูมิศาสตร์ของเหตุการณ์ที่เกิดแผ่นดินไหว seismic โดยพิจารณาตามความลึก ชุดคำสั่งที่จำเป็นสำหรับกราฟนี้มีดังต่อไปนี้

```
data(quakes)
mini <- min(quakes$depth)
maxi <- max(quakes$depth)
int <- ceiling((maxi - mini)/9)
inf <- seq(mini, maxi, int)
quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)),
                    labels=paste(inf, inf + int, sep="-"))
xyplot(lat ~ long | depth.cat, data = quakes)
```

คำสั่งแรกโหลดข้อมูล `quakes` เข้ามาในหน่วยความจำ จากนั้นคำสั่งห้าชุดถัดมาสร้าง



รูปที่ 8: ฟังก์ชัน `xyplot` กับข้อมูล “quakes”

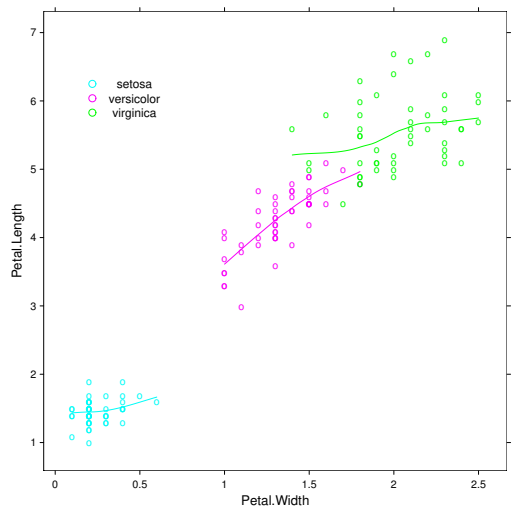
factor หนึ่งโดยการแบ่งความลึก (ตัวแปร `depth`) ให้มีการจัดแนวที่เท่ากันเก้าช่วง หมายความว่า ระดับของ factor นี้ถูกลงรายการด้วยค่าขอบเขตต่ำและค่าขอบเขตสูงของช่วงเหล่านี้ หลังจากนั้นจึงเพียงพอที่จะเรียกฟังก์ชัน `xyplot` กับสูตรที่เหมาะสมและอาร์กิวเมนต์ `data` ซึ่งบ่งบอกถึงตำแหน่งที่ `xyplot` จำเป็นในการหาตัวแปร[16]

ด้วยข้อมูล `iris` ความซ้อนกันระหว่างสปีชีส์ที่แตกต่างกันเล็กน้อยที่มากพอสามารถถูกวาดได้ดังรูปที่ 9) โดยชุดคำสั่งมีดังต่อไปนี้

```
data(iris)
xyplot(
  Petal.Length ~ Petal.Width, data = iris, groups=Species,
   panel = panel.superpose,
   type = c("p", "smooth"), span=.75,
   auto.key = list(x = 0.15, y = 0.85)
)
```

การเรียกฟังก์ชัน `xyplot` ในตัวอย่างนี้มีความซับซ้อนมากกว่าตัวอย่างก่อนหน้านี้เล็กน้อยและใช้ตัวเลือกได้หลากหลายซึ่งสามารถให้รายละเอียดดังนี้ ตัวเลือก `groups` โดยตามชื่อของมันเป็นการนิยามกลุ่มที่ถูกใช้โดยตัวเลือกอื่น จะเห็นได้ว่าตัวเลือก `panel` ซึ่งนิยามกลุ่มที่แตกต่างกันจะถูกแทนบนกราฟได้อย่างไร ซึ่งอธิบายได้ว่า ในที่นี้ได้ใช้ฟังก์ชันที่เป็น pre-defined คือ `panel.superpose` เพื่อที่จะจัดกลุ่มแบบต่อยอดในพล็อตเดียวกัน ไม่มีตัวเลือกถูกผ่านไปที่ `panel.superpose` ด้วยสีต่าง ๆ โดยปริยายถูก

[16] `plot()` ไม่สามารถมีอาร์กิวเมนต์ `data` ได้ ตำแหน่งของตัวแปรต้องถูกกำหนดให้อย่างชัดเจน ตัวอย่างเช่น `plot(quakes$long ~ quakes$lat)`



รูปที่ 9: ฟังก์ชัน `xyplot` กับข้อมูล “iris”

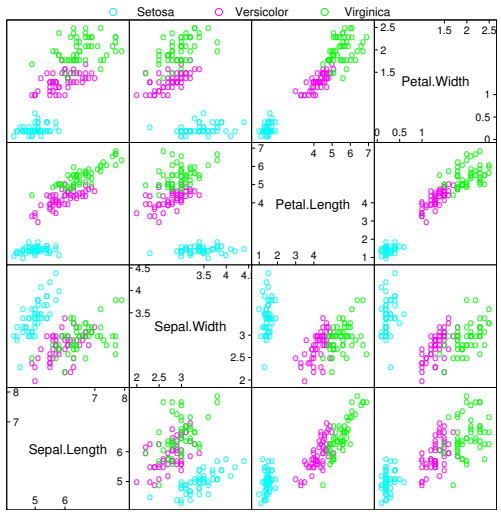
ใช้เพื่อแยกกลุ่มต่าง ๆ ต่อมาตัวเลือก `type` คล้ายใน `plot()` เป็นการระบุว่าข้อมูลถูกเป็นตัวแทนได้อย่างไร แต่ในที่นี้สามารถให้อาร์กิวเมนต์ที่หลากหลายเป็นเวกเตอร์หนึ่งอธิบายได้ว่า `"p"` ใช้วาดจุดต่างๆ และ `"smooth"` เพื่อวาดเส้นโค้งที่เรียบ ซึ่งระดับของความเรียบถูกระบุโดย `span` หลังจากนั้นตัวเลือก `auto.key` เติมคำอธิบายสัญลักษณ์ (legend) ไปที่กราฟนี้ ซึ่งมีความจำเป็นเพียงเพื่อแสดงรายการเกี่ยวกับพิกัดที่คำอธิบายสัญลักษณ์ถูกวาดขึ้น ข้อสังเกตในที่นี้คือ พิกัดมีความสัมพันธ์กับขนาดของพล็อต (เช่น ในตำแหน่ง [0,1])

ในที่นี้จะได้เห็นฟังก์ชัน `splom` กับข้อมูล iris เดียวกัน ซึ่งชุดคำสั่งต่อไปนี้ถูกใช้ในการสร้างรูปที่ 10:

```
splom(
  ~iris[1:4], groups = Species, data = iris, xlab = "",
  panel = panel.superpose,
  auto.key = list(columns = 3)
)
```

อาร์กิวเมนต์หลักในตอนนี้เป็น เมทริกซ์หนึ่ง (คอลัมน์สี่คอลัมน์แรกของข้อมูล `iris`) ผลที่ได้คือ ชุดของการพล็อตของตัวแปรสองตัวที่เป็นไปได้ระหว่างคอลัมน์ของเมทริกซ์นี้ ซึ่งคล้ายกับฟังก์ชันมาตรฐาน `pairs` โดยค่าปริยาย `splom` เติมข้อความ “Scatter Plot Matrix” ใต้แกน x ดังนั้น เพื่อหลีกเลี่ยงข้อความนี้ `xlab=""` เป็นตัวเลือกที่ถูกใช้ ตัวเลือกอื่น ๆ มีความคล้ายกันตามตัวอย่างก่อนหน้านี้ยกเว้น `columns = 3` สำหรับ `auto.key` ถูกระบุดังนั้นคำอธิบายสัญลักษณ์ถูกแสดงในคอลัมน์สามคอลัมน์

รูปที่ 10 อาจถูกสร้างได้โดยใช้ฟังก์ชัน `pairs()` แต่ฟังก์ชันนี้ในลำดับถัดมาไม่สามารถสร้างกราฟโดยมีเงื่อนไขคล้ายกับรูปที่ 11 ได้ ต่อไปนี้เป็นโค้ดที่ถูกใช้ในการสร้าง



รูปที่ 10: ฟังก์ชัน `splom` กับข้อมูล “iris” (1)

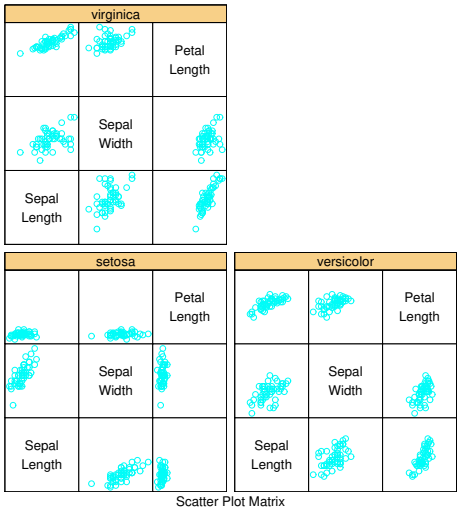
กราฟนี้ซึ่งค่อนข้างง่าย

```
splom(~iris[1:3] | Species, data = iris, pscales = 0,
     varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"))
```

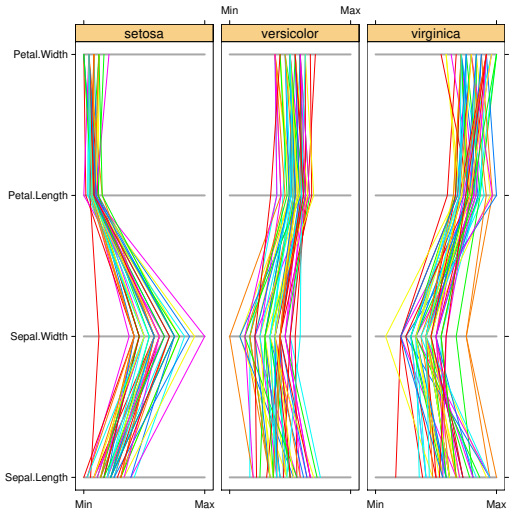
เนื่องจากกราฟย่อยซึ่งอยู่ในกราฟนี้มีขนาดค่อนข้างเล็ก จึงได้เพิ่มสองตัวเลือกเพื่อให้รูปนี้มีความง่ายมากขึ้นในการอ่านดังนี้ `pscales = 0` เอาเครื่องหมายกำกับบนแกนออก (กราฟย่อยทั้งหมดถูกวาดด้วยอัตราส่วนเหมือนกัน) และชื่อของตัวแปรถูกกำหนดอีกครั้งเพื่อแสดงค่าของพวกมันเป็นสองบรรทัด (`"\n"` เป็นรหัสเพื่อการแบ่งแถวในสายอักขระ (character string) หนึ่ง ๆ)

ตัวอย่างสุดท้ายใช้วิธีการของพิกัดแบบขนาน เพื่อการวิเคราะห์เชิงสำรวจ (exploratory analysis) ของข้อมูลหลายตัวแปร ตัวแปรเหล่านี้ถูกกำหนดอยู่บนแกนหนึ่ง (เช่น แกน y) และค่าที่สังเกตได้ถูกวาดบนอีกแกนหนึ่ง (ตัวแปรเหล่านี้ถูกวัดคล้ายกัน เช่น การทำตัวแปรเหล่านี้ให้ได้มาตรฐาน) ค่าที่แตกต่างกันของตัวเดียวกันถูกเชื่อมด้วยเส้นเส้นหนึ่ง รหัสต่อไปนี้เป็นรหัสที่ใช้สร้างรูปที่ 12 โดยใช้ข้อมูล `iris`

```
parallel(~iris[, 1:4] | Species, data = iris, layout = c(3, 1))
```



รูปที่ 11: ฟังก์ชัน `splom` กับข้อมูล “iris” (2)



รูปที่ 12: ฟังก์ชัน `parallel` กับข้อมูล “iris”

5 การวิเคราะห์ทางสถิติด้วย R

นอกจากกราฟิกแล้ว ในที่นี้เป็นไปไม่ได้ที่จะลงรายละเอียดของความเป็นไปได้ที่นำเสนอโดย R ที่เกี่ยวกับการวิเคราะห์ทางสถิติ เป้าหมายของผู้เขียนในที่นี้เพื่อให้หลักสำคัญบางประการพร้อมจุดมุ่งหมายในการได้แนวคิดของลักษณะของ R ที่ใช้แสดงการวิเคราะห์ข้อมูล

แพ็กเกจ stats ประกอบด้วยกลุ่มฟังก์ชันมากมายของการวิเคราะห์สถิติพื้นฐาน ได้แก่ การทดสอบแบบดั้งเดิม (classical tests) ตัวแบบเชิงเส้น (linear models) ซึ่งรวมถึงการถดถอยแบบกำลังสองน้อยสุด (least-squares regression) ตัวแบบเชิงเส้นนัยทั่วไป (generalized linear models) และการวิเคราะห์ความแปรปรวน (analysis of variance) การแจกแจง (distributions) ค่าทางสถิติโดยสรุป (summary statistics) การจัดกลุ่มแบบลำดับชั้น (hierarchical clustering) การวิเคราะห์อนุกรมเวลา (time-series analysis) กำลังสองน้อยที่สุดแบบไม่เป็นเชิงเส้น (nonlinear least squares) และการวิเคราะห์หลายตัวแปร (multivariate analysis) ส่วนวิธีทางสถิติอื่น ๆ หาได้ในแพ็กเกจที่มีเป็นจำนวนมากใน R บางแพ็กเกจมีให้แล้วในการติดตั้งพื้นฐานของโปรแกรม R และถูกแสดงข้อความว่า *แนะนำ (recommended)* และมีแพ็กเกจอื่น ๆ อีกมากมายที่เป็นแพ็กเกจ *สนับสนุน (contributed)* และจำเป็นต้องได้รับการติดตั้งโดยผู้ใช้

โดยจะเริ่มด้วยตัวอย่างที่ง่ายซึ่งไม่ต้องการแพ็กเกจใด ๆ นอกเหนือจาก stats เพื่อที่จะแนะนำการนำไปสู่การวิเคราะห์ข้อมูลโดยทั่วไปใน R หลังจากนั้นจะลงในรายละเอียดของความรู้บางประการ *สูตร (formulae)* และ *ฟังก์ชันทั่วไป (generic functions)* ซึ่งเป็นประโยชน์ทั้งสิ้น ไม่ว่าจะดำเนินการวิเคราะห์โดยวิธีใดก็ตาม และสุดท้ายเราจะสรุปด้วยภาพรวมของแพ็กเกจ

5.1 ตัวอย่างอย่างง่ายของการวิเคราะห์ความแปรปรวน

ฟังก์ชันเพื่อการวิเคราะห์ความแปรปรวนในแพ็กเกจ stats คือ `aov` เพื่อที่จะลองทำตัวอย่างนี้ ให้เอาชุดข้อมูลที่มีอยู่แล้วใน R คือ `InsectSprays` ชุดข้อมูลนี้เป็นการทดสอบยาฆ่าแมลง 6 ชนิดในภาคสนาม ตัวแปรตาม (observed response) คือ จำนวนของแมลงซึ่งยาฆ่าแมลงแต่ละชนิดถูกทดสอบ 12 ครั้ง ดังนั้นมีข้อมูลที่ได้จากการสังเกต 72 ชุด ในที่นี้จะไม่พิจารณาสำรวจดูข้อมูลทางกราฟิกแต่จะเน้นไปที่การวิเคราะห์ความแปรปรวนอย่างง่ายของการตอบสนองของแมลงต่อยาฆ่าแมลง ภายหลังจากการใส่ข้อมูลลงไปในหน่วยความจำด้วยฟังก์ชัน `data` การวิเคราะห์ถูกดำเนินการภายหลังการแปลงเป็นค่ารากที่สอง (square-root transformation) ของตัวแปรตามดังต่อไปนี้

```
> data(InsectSprays)
```

```
> aov.spray <- aov(sqrt(count) ~ spray, data = InsectSprays)
```

อาร์กิวเมนต์หลัก (และจำเป็น) ของ aov คือสูตรหนึ่งที่ระบุตัวแปรตามทางด้านซ้ายมือของสัญลักษณ์ตัวหนอน ~ และตัวแปรพยากรณ์ (predictor) ทางด้านขวามือ ส่วนตัวเลือก data = InsectSprays ระบุว่าตัวแปรจำต้องถูกพบในกรอบข้อมูล (data frame) InsectSprays วากยสัมพันธ์นี้ (syntax) ที่เท่าเทียมกันมีดังต่อไปนี้

```
> aov.spray <- aov(sqrt(InsectSprays$count) ~ InsectSprays$spray)
```

หรือ syntax เหมือนดังตัวอย่างต่อไปนี้ก็ได้ ถ้าทราบจำนวนคอลัมน์ของตัวแปร

```
> aov.spray <- aov(sqrt(InsectSprays[, 1]) ~ InsectSprays[, 2])
```

syntax แรกถูกเลือกใช้มากกว่าเนื่องจากมันมีความชัดเจนมากกว่า

ผลที่ได้เหล่านี้ไม่ถูกแสดงเนื่องจากถูกกำหนดให้เป็นอ็อบเจ็กต์หนึ่งที่เรียกว่า aov.spray ซึ่งจะถูกนำไปใช้ในบางฟังก์ชัน เพื่อแยกผลเหล่านี้ออกมาในลำดับถัดไป ตัวอย่างเช่น ฟังก์ชัน print เพื่อแสดงสรุปโดยย่อของการวิเคราะห์ (ส่วนใหญ่เป็นค่าพารามิเตอร์ที่ถูกประมาณค่า) และ summary เพื่อแสดงรายละเอียดที่มากขึ้น (รวมถึงการทดสอบทางสถิติ) ดังตัวอย่างต่อไปนี้

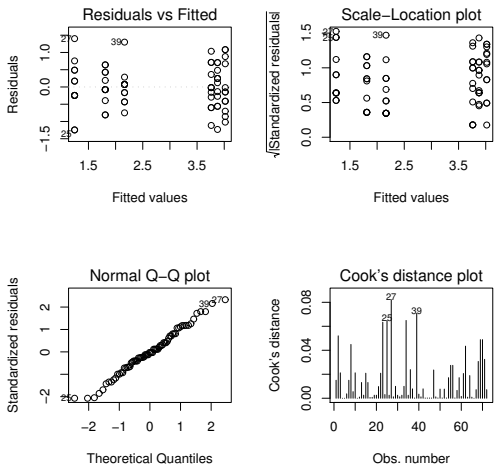
```
> aov.spray
Call:
   aov(formula = sqrt(count) ~ spray, data = InsectSprays)

Terms:
                    spray Residuals
Sum of Squares  88.43787  26.05798
Deg. of Freedom        5        66

Residual standard error: 0.6283453
Estimated effects may be unbalanced
> summary(aov.spray)
            Df Sum Sq Mean Sq F value    Pr(>F)
spray        5 88.438  17.688  44.799 < 2.2e-16 ***
Residuals   66 26.058   0.395
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

อาจเตือนความจำด้วยการพิมพ์ชื่อของอ็อบเจ็กต์เป็นคำสั่งหนึ่งที่คล้ายกับคำสั่ง print(aov.spray) ได้ โดยการแสดงทางกราฟิกของผลสามารถถูกทำได้โดย plot() หรือ termplot()

ก่อนการพิมพ์ `plot(aov.spray)` จะแบ่งกราฟิกภายในเป็นสี่ส่วนเพื่อว่าพล็อตที่วิเคราะห์ทั้งสี่ส่วนนี้จะถูกดำเนินการบนกราฟเดียวกัน ต่อไปนี้คือชุดคำสั่งดังกล่าว

```
> opar <- par()
> par(mfcol = c(2, 2))
> plot(aov.spray)
> par(opar)
> termplot(aov.spray, se=TRUE, partial.resid=TRUE, rug=TRUE)
```

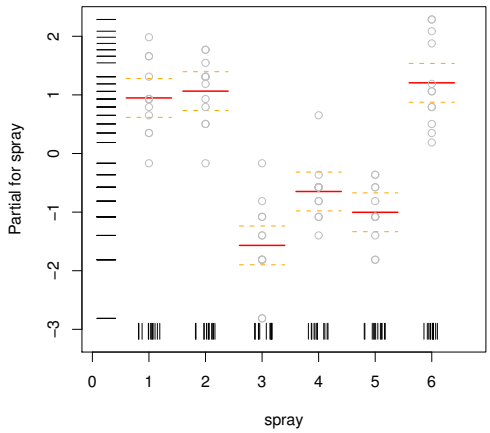
และกราฟิกของผลเหล่านี้อยู่ในรูปที่ 13 และ 14



รูปที่ 13: การแสดงทางกราฟิกของผลจากฟังก์ชัน `aov` ด้วย `plot()`

5.2 สูตร (formulae)

สูตรเป็นอีลีเมนท์ (element) ที่สำคัญอย่างหนึ่งในการวิเคราะห์ทางสถิติด้วย R โดยคำอธิบายที่ถูกใช้เป็นเหมือนกันหรือเกือบเหมือนกันสำหรับทั้งหมดของฟังก์ชัน โดยทั่วไปสูตรหนึ่งเป็นของรูปแบบ `y ~ model` ซึ่ง `y` เป็นตัวแปรตามที่ถูกวิเคราะห์ และ *`model`* เป็นชุดของเงื่อนไขสำหรับค่าพารามิเตอร์บางค่าถูกประมาณค่า เงื่อนไขเหล่านี้ถูกแยกด้วยสัญลักษณ์คำนวณ แต่สัญลักษณ์เหล่านี้มีความหมายเฉพาะดังต่อไปนี้



รูปที่ 14: การแสดงทางกราฟิกของผลจากฟังก์ชัน `aov` ด้วย `termplot()`

`a+b`	ผลแบบรวมกัน (additive effects) ของ `a` และ `b`
`X`	ถ้า `X` คือเมทริกซ์หนึ่งแล้วสิ่งนี้ระบุผลการบวกของคอลัมน์ของมัน ตัวอย่างเช่น `X[,1]+X[,2]+...+X[,ncol(X)]` ในบางกรณีคอลัมน์อาจถูกเลือกด้วยดัชนีตัวเลข ตัวอย่างเช่น `X[,2:4]`
`a:b`	ผลแบบมีปฏิสัมพันธ์กัน (interactive effects) ระหว่าง `a` และ `b`
`a*b`	ผลทั้งแบบรวมกันและมีปฏิสัมพันธ์กัน (เท่ากับ `a+b+a:b`)
`poly(a, n)`	พหุนาม (polynomials) ของ `a` ถึงดีกรี n
`^n`	หมายรวมถึงปฏิสัมพันธ์ทั้งหมดจนถึงระดับ n ตัวอย่างเช่น `(a+b+c)^2` ได้ผลเท่ากับ `a+b+c+a:b+a:c+b:c`
`b %in% a`	ผลของ `b` ถูกซ้อนอยู่ใน `a` (ได้ผลเท่ากับ `a+a:b` หรือ `a/b`)
`-b`	เอาผลของ `b` ออก ตัวอย่างเช่น `(a+b+c)^2-a:b` ได้ผลเท่ากับ `a+b+c+a:c+b:c`
`-1`	`y~x-1` คือสมการถดถอยผ่านจุดกำเนิด (เหมือนกับ `y~x+0` หรือ `0+y~x`)
`1`	`y~1` ที่เหมาะกับตัวแบบ (model) โดยไม่มีผลใด ๆ มีเพียงแต่จุดตัด (intercept)
`offset(...)`	เพิ่มผลไปที่ตัวแบบโดยไม่มีการประมาณค่าพารามิเตอร์ใด ๆ ตัวอย่างเช่น `offset(3*x)`

เราจะเห็นได้ว่าตัวดำเนินการเลขคณิต (arithmetic operators) ของ R มีความหมายที่ต่างกันมากกว่าที่มีในนิพจน์ (expressions) ตัวอย่างเช่น จากสูตร `y~x1+x2` กำหนดตัวแบบ $y = \beta_1 x_1 + \beta_2 x_2 + \alpha$ ไม่ใช่ $y = \beta(x_1 + x_2) + \alpha$ (แม้ว่าตัวดำเนินการ + จะมีความ-

หมายตามเครื่องหมายของมันเอง) เพื่อรวมการดำเนินการเลขคณิตในสูตร เราสามารถใช้ฟังก์ชัน `I` โดยสูตร `y~I(x1+x2)` กำหนดตัวแบบ $y = \beta(x_1 + x_2) + \alpha$ ในทำนองเดียวกันนั้นเพื่อกำหนดตัวแบบเป็น $y = \beta_1 x + \beta_2 x^2 + \alpha$ เราจะใช้สูตร `y ~ poly(x, 2)` (แต่ไม่ใช่ `y ~ x + x^2`) อย่างไรก็ตาม เป็นไปได้ที่จะรวมฟังก์ชันหนึ่งในสูตรเพื่อที่จะแปลงตัวแปรหนึ่งตามที่ปรากฏก่อนหน้านี้ในการวิเคราะห์ความแปรปรวนในการฉีดยาฆ่าแมลง

สำหรับการวิเคราะห์ความแปรปรวนนั้น `aov()` ยอมรับ syntax เฉพาะเพื่อกำหนดอิทธิพลแบบสุ่ม (random effects) ตัวอย่างเช่น `y ~ a + Error(b)` หมายถึงอิทธิพลแบบรวมกันของพจน์กำหนด (fixed term) `a` และพจน์สุ่ม `b`.

5.3 ฟังก์ชันทั่วไป (generic functions)

เราได้เรียนรู้แล้วว่าฟังก์ชันของ R กระทำการโดยการเชื่อมโยงคุณลักษณะของอ็อบเจ็กต์ซึ่งอาจผ่านไปทางอาร์กิวเมนต์ *คลาส (class)* ของต้นแบบฟังก์ชันนี้เป็นคุณลักษณะที่สมควรให้ความสนใจในที่นี้ เป็นเรื่องปกติมากที่ฟังก์ชันทางสถิติของ R ส่งอ็อบเจ็กต์หนึ่งของคลาสกลับมาด้วยชื่อเดียวกัน (ตัวอย่างเช่น `aov` ส่งกลับ อ็อบเจ็กต์ของคลาส `"aov"` หรือ `lm` ส่งกลับอ็อบเจ็กต์ของคลาส `"lm"`) ฟังก์ชันเหล่านี้สามารถใช้ในลำดับถัดมาเพื่อสกัดผลลัพธ์นี้จะกระทำการที่เฉพาะในส่วนของคลาสของอ็อบเจ็กต์นั้น ฟังก์ชันเหล่านี้ถูกเรียกว่า *generic*

ตัวอย่างเช่น ฟังก์ชันซึ่งน่าจะถูกใช้มากที่สุดเพื่อสกัดผลลัพธ์จากการวิเคราะห์คือ `summary` ซึ่งจะแสดงผลลัพธ์โดยละเอียด ไม่ว่าอ็อบเจ็กต์ที่ถูกกำหนดให้ตามอาร์กิวเมนต์ของคลาสจะเป็นอย่างไรเช่น `"lm"`(ตัวแบบเชิงเส้น) หรือ `"aov"` (การวิเคราะห์ความแปรปรวน) ชัดเจนว่าข้อมูลที่แสดงจะไม่เหมือนกัน นี้เป็นข้อดีของฟังก์ชันทั่วไปที่ว่า syntax เป็นอย่างเดียวกันในทุกกรณี

อ็อบเจ็กต์หนึ่งที่ประกอบด้วยผลลัพธ์ของการวิเคราะห์หนึ่งโดยทั่วไปอยู่ในรูปหนึ่งของการแสดงรายการ (list) และเป็นวิธีที่ถูกแสดงเพื่อระบุคลาสของมันด้วย ได้เห็นแล้วว่าแนวคิดนี้เป็นการดำเนินการของฟังก์ชันหนึ่งขึ้นอยู่กับชนิดของอ็อบเจ็กต์ที่ถูกกำหนดให้ตามอาร์กิวเมนต์ซึ่งเป็นลักษณะทั่วไปของ R[17] ตารางต่อไปนี้แสดงฟังก์ชันทั่วไปที่สำคัญ ซึ่งสามารถถูกใช้สกัดข้อมูลจากอ็อบเจ็กต์ซึ่งเป็นผลจากการวิเคราะห์หนึ่งได้ การใช้ที่เป็นตัวอย่างของฟังก์ชันเหล่านี้มีดังต่อไปนี้

```
> mod <- lm(y ~ x)
> df.residual(mod)
[1] 8
```

[17]มีฟังก์ชันทั่วไปมากกว่า 100 ฟังก์ชันใน R

print	ได้ผลเป็นสรุปอย่างย่อ
summary	ได้ผลเป็นสรุปอย่างละเอียด
df.residual	ได้ผลเป็นจำนวนของระดับส่วนเหลือความเป็นอิสระ
coef	ได้ผลเป็นค่าสัมประสิทธิ์ที่ประมาณการ (บางครั้งมาพร้อมกับค่าคลาดเคลื่อนมาตรฐาน (stanadard error) ของค่าสัมประสิทธิ์เอง)
residuals	ได้ผลเป็นส่วนเหลือ (residuals)
deviance	ได้ผลเป็นค่าสถิติดิเวียนส์ (deviances)
fitted	ได้ผลเป็นค่าที่เหมาะสม (fitted values)
logLik	คำนวณลอการิทึมของความควรจะเป็น (likelihood) และจำนวนของค่าพารามิเตอร์
AIC	คำนวณเกณฑ์ข้อมูล Akaike หรือ AIC (ขึ้นกับ logLik())

ฟังก์ชันหนึ่งเช่น aov หรือ lm ได้ผลเป็นรายการพร้อมด้วยส่วนประกอบที่แตกต่างกันของฟังก์ชันของมันตรงตามผลลัพธ์ของการวิเคราะห์ หากใช้กรณีของเราในการวิเคราะห์ความแปรปรวนด้วยข้อมูลจาก InsectSprays สามารถดูโครงสร้างของอ็อบเจ็กต์ที่ถูกส่งค่ากลับมาโดย aov ดังต่อไปนี้

```
> str(aov.spray, max.level = -1)
List of 13
 - attr(*, "class")= chr [1:2] "aov" "lm"
```

อีกวิธีหนึ่ง เพื่อจะดูโครงสร้างนี้คือการแสดงชื่อต่างๆ ของอ็อบเจ็กต์ดังต่อไปนี้

```
> names(aov.spray)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "contrasts"
[10] "xlevels"       "call"          "terms"
[13] "model"
```

หลังจากนั้น (elements) สามารถถูกสกัดออกมาตามที่ได้เห็นแล้วดังต่อไปนี้

```
> aov.spray$coefficients
(Intercept)      sprayB      sprayC      sprayD
  3.7606784   0.1159530  -2.5158217  -1.5963245
     sprayE      sprayF
 -1.9512174   0.2579388
```

summary() สร้างรายการ (list) ได้เช่นกัน ซึ่งในกรณีของ aov() นั้นรายการเป็นตารางอย่างง่ายของการทดสอบดังต่อไปนี้

```
> str(summary(aov.spray))
List of 1
 $ :Classes anova  and 'data.frame':   2 obs. of  5 variables:
  ..$ Df     : num [1:2] 5 66
  ..$ Sum Sq : num [1:2] 88.4 26.1
  ..$ Mean Sq: num [1:2] 17.688  0.395
  ..$ F value: num [1:2] 44.8   NA
  ..$ Pr(>F) : num [1:2] 0 NA
 - attr(*, "class")= chr [1:2] "summary.aov" "listof"
> names(summary(aov.spray))
NULL
```

ฟังก์ชันทั่วไปโดยปกติไม่กระทำการใด ๆ บนอ็อบเจ็กต์ หมายความว่าฟังก์ชันเหล่านี้เรียกฟังก์ชันที่เหมาะสมในส่วนของคลาสของอาร์กิวเมนต์นั้น ๆ ฟังก์ชันหนึ่งที่ให้ชื่อว่าเป็นทั่วไป (generic) เป็น *วิธีหรือเมธอด (method)* หนึ่งในศัพท์เทคนิคเฉพาะ (jargon) ของ R โดยตามแบบแผนแล้วเมธอดหนึ่งถูกสร้างตาม `generic.cls` ที่ซึ่ง `cls` เป็นคลาสของอ็อบเจ็กต์นั้น ตัวอย่างเช่น ในกรณีของ `summary` เราสามารถแสดง เมธอดซึ่งสอดคล้องกันได้ดังต่อไปนี้

```
> apropos("^summary")
 [1] "summary"                "summary.aov"
 [3] "summary.aovlist"        "summary.connection"
 [5] "summary.data.frame"     "summary.default"
 [7] "summary.factor"         "summary.glm"
 [9] "summary.glm.null"       "summary.infl"
[11] "summary.lm"             "summary.lm.null"
[13] "summary.manova"         "summary.matrix"
[15] "summary.mlm"            "summary.packageStatus"
[17] "summary.POSIXct"        "summary.POSIXlt"
[19] "summary.table"
```

เราสามารถเห็นความแตกต่างสำหรับฟังก์ชันทั่วไปนี้ในกรณีของสมการถดถอยเชิงเส้นที่ถูกเปรียบเทียบกับการวิเคราะห์ความแปรปรวนโดยตัวอย่างจำลองย่อยหนึ่งดังต่อไปนี้

```
> x <- y <- rnorm(5)
> lm.spray <- lm(y ~ x)
> names(lm.spray)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
```

```
[10] "call"          "terms"         "model"
> names(summary(lm.spray))
 [1] "call"          "terms"         "residuals"
 [4] "coefficients"  "sigma"         "df"
 [7] "r.squared"     "adj.r.squared" "fstatistic"
[10] "cov.unscaled"
```

ตารางดังต่อไปนี้แสดงฟังก์ชันทั่วไปบางฟังก์ชันที่แสดงการวิเคราะห์เพิ่มขึ้น (supplementary analyses) จากอ็อบเจ็กต์หนึ่งซึ่งเป็นผลลัพธ์จากการวิเคราะห์หนึ่งและอาร์กิวเมนต์หลักคือ อ็อบเจ็กต์ที่อยู่ถัดมา แต่ในบางกรณีอาร์กิวเมนต์เพิ่มเติมนั้นมีความจำเป็นดังเช่น `predict` หรือ `update`.

`add1`	ทดสอบพจน์ทั้งหมดตามลำดับที่สามารถถูกเพิ่มไปที่ตัวแบบหนึ่ง
`drop1`	ทดสอบพจน์ทั้งหมดตามลำดับที่สามารถถูกเอาออกจากตัวแบบหนึ่ง
`step`	เลือกตัวแบบหนึ่งด้วย AIC (เรียก `add1` และ `drop1`)
`anova`	คำนวณตารางความแปรปรวนหรือ deviance สำหรับตัวแบบหนึ่งหรือหลายรูปแบบ
`predict`	คำนวณค่าพยากรณ์สำหรับข้อมูลให้จากตัวแบบที่เหมาะสม
`update`	ปรับปรุงตัวแบบอีกครั้งด้วยสูตรใหม่หรือข้อมูลใหม่

นอกจากนี้มีฟังก์ชันที่เป็นประโยชน์มากมายที่สกัดข้อมูลจากอ็อบเจ็กต์ที่เป็นตัวแบบหรือสูตร เช่น ฟังก์ชัน `alias` ซึ่งหาพจน์ไม่เป็นอิสระเชิงเส้นในตัวแบบสมการเชิงเส้นซึ่งถูกระบุโดยสูตรหนึ่ง สุดท้ายมีความจำเป็นที่ต้องมีฟังก์ชันกราฟิก เช่น `plot` ที่แสดงผลให้เห็นชัดขึ้นเพื่อการตรวจสอบข้อผิดพลาดต่าง ๆ หรือ `termplot` (ดูตัวอย่างก่อนหน้านี้) ถึงแม้ว่าฟังก์ชัน `termplot` ไม่เป็น generic แต่เรียกว่า `predict`.

5.4 Packages

ตารางต่อไปนี้แสดงรายการแพ็กเกจมาตรฐาน (standard) ซึ่งมีอยู่ในการติดตั้งพื้นฐานของ R บางแพ็กเกจถูกนำเข้าใส่ในหน่วยความจำเมื่อ R เริ่มใช้ ซึ่งสามารถแสดงได้ด้วยการใช้ฟังก์ชัน `search` ดังต่อไปนี้

```
> search()
[1] ".GlobalEnv"        "package:methods"
[3] "package:stats"     "package:graphics"
[5] "package:grDevices" "package:utils"
[7] "package:datasets"  "Autoloads"
[9] "package:base"
```

แพ็กเกจอื่น ๆ อาจจะถูกใช้ภายหลังถูกนำเข้าได้ดังเช่น

```
> library(grid)
```

รายการของฟังก์ชันในแพ็กเกจหนึ่งสามารถถูกแสดงได้ด้วย

```
> library(help = grid)
```

หรือโดยการเรียกดู help ในรูปแบบ html ข้อมูลที่สัมพันธ์ในแต่ละฟังก์ชันสามารถถูกเข้าถึงได้ตามที่เห็นก่อนหน้านี้ (หน้า 8).

แพ็กเกจ	คำอธิบาย
base	base R functions
datasets	base R datasets
grDevices	graphics devices for base and grid graphics
graphics	base graphics
grid	grid graphics
methods	definition of methods and classes for R objects and programming tools
splines	regression spline functions and classes
stats	statistical functions
stats4	statistical functions using S4 classes
tcltk	functions to interface R with Tcl/Tk graphical user interface elements
tools	tools for package development and administration
utils	R utility functions

แพ็กเกจที่มีส่วนสนับสนุน (contributed packages) จำนวนมากเพิ่มมากขึ้นในรายการของวิธีการทางสถิติที่ใช้ได้ใน R แพ็กเกจเหล่านี้ถูกส่งแยกกันและจำเป็นต้องถูกติดตั้งและนำเข้าใน R การแสดงรายการที่สมบูรณ์ของแพ็กเกจเหล่านี้พร้อมคำอธิบายอยู่บนเว็บไซต์ของ CRAN[18] หลายแพ็กเกจถูกแนะนำเนื่องจากแพ็กเกจเหล่านี้ครอบคลุมวิธีการทางสถิติซึ่งมักถูกใช้ในการวิเคราะห์ข้อมูล แพ็กเกจที่ถูกแนะนำเหล่านี้มักถูกส่งมาพร้อมกับการติดตั้งพื้นฐานของ R ซึ่งมีคำอธิบายโดยย่อของแต่ละแพ็กเกจในตารางต่อไปนี้

[18] http://cran.r-project.org/src/contrib/PACKAGES.html

แพ็กเกจ	คำอธิบาย
boot	resampling and bootstraping methods
class	classification methods
cluster	clustering methods
foreign	functions for reading data stored in various formats (S3, Stata, SAS, Minitab, SPSS, Epi Info)
KernSmooth	methods for kernel smoothing and density estimation (including bivariate kernels)
lattice	Lattice (Trellis) graphics
MASS	contains many functions, tools and data sets from the libraries of "Modern Applied Statistics with S" by Venables & Ripley
mgcv	generalized additive models
nlme	linear and non-linear mixed-effects models
nnet	neural networks and multinomial log-linear models
rpart	recursive partitioning
spatial	spatial analyses ("kriging", spatial covariance, . . .)
survival	survival analyses

คลังเก็บข้อมูลหลักอีกสองที่ของ R แพ็กเกจคือ Omegahat Project for Statistical Computing[19]ซึ่งเน้นการประยุกต์ใช้งานบนเว็บไซต์และส่วนต่อประสานระหว่างซอฟต์แวร์และภาษา ส่วนอีกคลังเก็บข้อมูลคือ Bioconductor Project[20] ซึ่งเชี่ยวชาญทางด้านการประยุกต์ใช้ทางชีวสารสนเทศโดยเฉพาะสำหรับการวิเคราะห์ข้อมูลเกี่ยวกับ microarray

ขั้นตอนที่ติดตั้งแพ็กเกจหนึ่งขึ้นกับระบบปฏิบัติการและ R ได้ถูกติดตั้งจากต้นทางหรือ pre-compiled binaries หรือไม่ ในระยะหลังมานี้แพ็กเกจที่เป็น pre-compiled ที่มีในเว็บไซต์ของ CRAN ถูกแนะนำให้ใช้ ภายใต้วินโดว์ binary Rgui.exe มีรายการเลือก (menu) "Packages" ซึ่งยอมให้ติดตั้งแพ็กเกจผ่านทางอินเทอร์เน็ตจากเว็บไซต์ CRAN หรือจากซิปไฟล์ (zipped files) ไปที่จานบันทึกเฉพาะที่ (local disk)

ถ้า R ได้ถูกแปลคำสั่ง (compiled) แล้วแพ็กเกจหนึ่งสามารถถูกติดตั้งจากต้นทาง (source) ของมันซึ่งถูกส่งเป็นไฟล์ '.tar.gz' ตัวอย่างเช่น ถ้าเราต้องการติดตั้งแพ็กเกจ gee เราจะต้องดาว์นโหลด (download) ไฟล์ gee_4.13-6.tar.gz ลงก่อน (เลข 4.13-6 บ่งบอกรุ่นหรือเวอร์ชัน (version) ของแพ็กเกจ ซึ่งโดยทั่วไปแล้วมีเพียงหนึ่งรุ่นที่มีได้ใน CRAN) หลังจากนั้นเราจะพิมพ์คำสั่งจากระบบ (และไม่ใช่ใน R) ดังนี้

```
R CMD INSTALL gee_4.13-6.tar.gz
```

[19] http://www.omegahat.org/R/
[20] http://www.bioconductor.org/

มีฟังก์ชันที่เป็นประโยชน์มากมายเพื่อจัดการแพ็กเกจ เช่น `installed. packages` `CRAN.packages` หรือ `download.packages` รวมทั้งยังเป็นประโยชน์ด้วยในการพิมพ์คำสั่งต่อไปนี้อย่างสม่ำเสมอ

```
> update.packages()
```

ซึ่งเป็นการตรวจสอบรุ่นของแพ็กเกจที่ถูกติดตั้งกับรุ่นที่มีใน CRAN (คำสั่งนี้สามารถถูกเรียกจากรายการเลือก “Packages” ในวินโดว์ หลังจากนั้นผู้ใช้สามารถปรับ (update) แพ็กเกจให้เป็นปัจจุบันด้วยรุ่นที่ใหม่กว่าโดยการติดตั้งบนคอมพิวเตอร์

6 การเขียนโปรแกรมด้วย R ในทางปฏิบัติ

เราได้แสดงให้เห็นภาพรวมของการทำงานต่าง ๆ ของ R ถึงตรงนี้เราจะกลับมาที่ภาษาการเขียนโปรแกรมซึ่ง เราจะได้ทราบแนวคิดอย่างง่ายบางอย่างที่อาจจะถูกใช้ในทางปฏิบัติ

6.1 การวนรอบและและการใช้เวกเตอร์ (loops and vectorization)

ข้อดีของ R เมื่อเปรียบเทียบกับซอฟต์แวร์ที่มีรายการเลือกแบบดึงลง (pull-down menus) คือความเป็นไปได้ในการเขียนโปรแกรมได้อย่างง่ายของลำดับของการวิเคราะห์ซึ่งจะถูกดำเนินการอย่างต่อเนื่อง คุณลักษณะนี้พบได้ทั่วไปในภาษาคอมพิวเตอร์ แต่ R มีลักษณะที่เฉพาะบางอย่างซึ่งการเขียนโปรแกรมทำให้ผู้ที่ไม่ได้เป็นผู้เชี่ยวชาญเข้าใจได้ง่ายขึ้น

คล้ายภาษาคอมพิวเตอร์อื่น ๆ R มี *โครงสร้างควบคุม (control structure)* บางอย่างซึ่งคล้ายกับโครงสร้างของภาษา C สมมติเรามี เวกเตอร์ `x` และสำหรับแต่ละ element ของ `x` กับค่า `b` เราต้องการที่จะให้ค่า 0 ไปยังตัวแปรอีกตัวคือ `y` มิฉะนั้นค่าเป็น 1 ก่อนอื่นเราต้องสร้างเวกเตอร์ `y` โดยที่มีความยาวเดียวกันกับ `x` ดังนี้

```
y <- numeric(length(x))
for (i in 1:length(x)) if (x[i] == b) y[i] <- 0 else y[i] <- 1
```

มีหลายวิธีที่สามารถถูกกระทำการถ้าการกระทำเหล่านั้นอยู่ภายในวงเล็บ ดังเช่น

```
for (i in 1:length(x)) {
    y[i] <- 0
    ...
}

if (x[i] == b) {
    y[i] <- 0
    ...
}
```

เงื่อนไขที่เป็นไปได้อีกแบบคือกระทำการตามคำสั่งตราบใดที่เงื่อนไขนั้นเป็นจริง ดังเช่น

```
while (myfun > minimum) {
    ...
}
```

อย่างไรก็ตาม แบบเป็นรอบและโครงสร้างควบคุมสามารถถูกหลีกเลี่ยงได้ในเงื่อนไขส่วนมาก ซึ่งต้องขอบคุณในคุณลักษณะของ R คือการใช้เวกเตอร์ *vectorization* การใช้เวกเตอร์ทำให้การวนรอบถูกต้องในนิพจน์และเราได้เห็นแล้วจากหลายกรณี ต่อไปนี้เรามาพิจารณาการบวกของสองเวกเตอร์

```
> z <- x + y
```

การบวกนี้อาจถูกเขียนด้วยแบบวนรอบตามที่ถูกกระทำในภาษาคอมพิวเตอร์ส่วนใหญ่ดังนี้

```
> z <- numeric(length(x))
> for (i in 1:length(z)) z[i] <- x[i] + y[i]
```

ในกรณีนี้ มีความจำเป็นในการสร้างเวกเตอร์ `z` ก่อนเนื่องจากการใช้ของระบบการทำดัชนี (indexing system) เราได้เห็นว่าการวนรอบที่ชัดเจนนี้จะใช้งานได้ก็ต่อเมื่อ `x` และ `y` มีความยาวเท่ากัน นั่นหมายความว่ามันต้องถูกเปลี่ยนถ้าไม่เป็นความจริง ขณะที่นิพจน์แรกจะใช้งานได้ในทุกเงื่อนไข

การกระทำการแบบมีเงื่อนไข (`if ... else`) สามารถถูกเลี่ยงได้ด้วยการใช้การทำดัชนีเชิงตรรกะ (logical indexing) ดังนี้ โดยให้กลับไปที่ตัวอย่างก่อนหน้านี้

```
> y[x == b] <- 0
> y[x != b] <- 1
```

นอกจากง่ายขึ้นแล้ว นิพจน์ถูกทำให้เป็นเวกเตอร์ที่มีประสิทธิภาพมากกว่าในเชิงคำนวณโดยเฉพาะกับปริมาณของข้อมูลมากๆ

ยังมีหลายฟังก์ชันด้วยเช่นกันของชนิด ‘apply’ ซึ่งหลีกเลี่ยงการเขียนแบบเป็นวนรอบ `apply` กระทำการบนแถวและ/หรือคอลัมน์ของเมทริกซ์ซึ่ง syntax ของมันคือ `apply(X, MARGIN, FUN, ...)` โดยที่ `X` คือเมทริกซ์หนึ่ง `MARGIN` บ่งบอกว่าแถว (`1`) คอลัมน์ (`2`) หรือทั้งสอง (`c(1, 2)`) ที่พิจารณา `FUN` เป็นฟังก์ชันหนึ่ง (หรือเป็นตัวดำเนินการ แต่ในกรณีนี้มันต้องถูกระบุอยู่ในเครื่องหมายวงเล็บ) เพื่อ apply และ `...` เป็นอาร์กิวเมนต์ตัวเลือกที่เป็นไปได้สำหรับ `FUN` ตัวอย่างอย่างง่ายมีดังต่อไปนี้

```
> x <- rnorm(10, -5, 0.1)
> y <- rnorm(10, 5, 2)
> X <- cbind(x, y)  # the columns of X keep the names "x" and "y"
> apply(X, 2, mean)
```

```
        x         y
-4.975132  4.932979
> apply(X, 2, sd)
        x         y
0.0755153 2.1388071
```

lapply() กระทำการบนรายการหนึ่งโดยที่ syntax ของมันมีความคล้ายคลึงกับ apply และจะได้ผลกลับมาเป็นรายการดังนี้

```
> forms <- list(y ~ x, y ~ poly(x, 2))
> lapply(forms, lm)
[[1]]

Call:
FUN(formula = X[[1]])

Coefficients:
(Intercept)            x
     31.683        5.377



[[2]]

Call:
FUN(formula = X[[2]])

Coefficients:
(Intercept)  poly(x, 2)1  poly(x, 2)2
     4.9330       1.2181      -0.6037
```

sapply() เป็นรูปแบบหนึ่งของ lapply() ที่แตกต่างออกไปแต่มีความยืดหยุ่น ซึ่งสามารถมีเวกเตอร์หรือเมทริกซ์ได้ทำนองเดียวกับอาร์กิวเมนต์หลัก และได้ผลลัพธ์กลับมาในรูปแบบที่โดยทั่วไปเป็นตารางซึ่งอาจใช้งานง่ายกว่า

6.2 การเขียนโปรแกรใน R

โดยปกติแล้วโปรแกรม R ถูกเขียนในไฟล์ที่ถูกบันทึกในรูปแบบ ASCII และถูกให้ชื่อกับนามสกุลของไฟล์ (extension) ว่า '.R' ในเงื่อนไขทั่วไปซึ่งโปรแกรมหนึ่งมีประโยชน์ก็ต่อเมื่อผู้ใช้งานต้องการทำงานเดียวกันในหลายช่วงเวลา ในตัวอย่างแรกนี้ผู้เขียนต้องการทำ

การพล็อตแบบเดียวกันสำหรับนกที่ต่างกันสามชนิด ข้อมูลอยู่ในไฟล์ที่แยกกันสามไฟล์ ผู้เขียนจะดำเนินการเป็นขั้น ๆ และจะเห็นวิธีที่ต่างกันเพื่อเขียนโปรแกรมสำหรับประเด็นที่ง่ายมากนี้

ก่อนอื่นเราต้องทำโปรแกรมของเราเองในวิธีทางที่ง่ายต่อการใช้และเข้าใจได้มากที่สุดโดยการกระทำการตามลำดับโดยคำสั่งที่จำเป็นซึ่งรวมทั้งทำการแบ่งส่วนของกราฟิก (graphical device) ล่วงหน้าด้วย ดังตัวอย่าง

```
layout(matrix(1:3, 3, 1))            # partition the graphics
data <- read.table("Swal.dat")          # read the data
plot(data$V1, data$V2, type="l")
title("swallow")                        # add a title
data <- read.table("Wren.dat")
plot(data$V1, data$V2, type="l")
title("wren")
data <- read.table("Dunn.dat")
plot(data$V1, data$V2, type="l")
title("dunnock")
```

อักขระ '#' ถูกใช้เพื่อเพิ่มความเห็นในโปรแกรม จากนั้น R จะไปที่บรรทัดถัดไป

ปัญหาข้างต้นของการเขียนโปรแกรมแรกนี้คือว่ามันอาจดูค่อนข้างยาวถ้าเราต้องการเพิ่มสปีชีส์อื่น ยิ่งไปกว่านั้นบางคำสั่งถูกกระทำการหลายครั้งดังนั้นคำสั่งเหล่านี้สามารถจัดกลุ่มกันและดำเนินการภายหลังการเปลี่ยนอาร์กิวเมนต์บางอย่าง กลยุทธที่ถูกใช้ในที่นี้คือการใส่อาร์กิวเมนต์เหล่านี้ในเวกเตอร์ของโหมดอักขระและหลังจากนั้นใช้การทำดัชนี (indexing) เพื่อเข้าถึงค่าที่แตกต่างกันเหล่านี้ดังนี้

```
layout(matrix(1:3, 3, 1))              # partition the graphics
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
for(i in 1:length(species)) {
    data <- read.table(file[i])          # read the data
    plot(data$V1, data$V2, type="l")
    title(species[i])                    # add a title
}
```

สังเกตได้ว่าไม่มีเครื่องหมายอัญประกาศ (double quotes) ล้อมรอบ `file[i]` ใน `read.table()` เนื่องจากอาร์กิวเมนต์นี้เป็นโหมดอักขระ

โปรแกรมของเราในตอนนี้มีความกระชับมากขึ้น ง่ายขึ้นต่อการเพิ่มสปีชีส์อื่น ๆ เนื่องจากเวกเตอร์เหล่านั้นที่ประกอบด้วยสปีชีส์และชื่อไฟล์อยู่ในส่วนต้นของโปรแกรม

โปรแกรมเหล่านี้ที่ได้กล่าวก่อนหน้าจะทำงานได้อย่างถูกต้องถ้าไฟล์ข้อมูล '.dat' ถูกทำให้อยู่ในตำแหน่งการทำงาน (working directory) ของ R มิฉะนั้นแล้วผู้ใช้จำเป็นต้องเปลี่ยนตำแหน่งการทำงานหรือระบุเส้นทาง (path) ให้ในโปรแกรม (ตัวอย่างเช่น `file <- "/home/paradis/data/Swal.dat"`) ถ้าโปรแกรมถูกเขียนในไฟล์ Mybirds.R แล้วมันสามารถถูกเรียกออกมาได้โดยการพิมพ์ดังต่อไปนี้

```
> source("Mybirds.R")
```

คล้ายข้อมูลนำเข้า (input) ใด ๆ จากไฟล์หนึ่ง มันมีความจำเป็นในการกำหนดเส้นทางเพื่อเข้าถึงไฟล์นั้นถ้าไฟล์นั้นไม่ได้อยู่ในตำแหน่งการทำงาน

6.3 การเขียนฟังก์ชันด้วยตัวเอง

เราได้เห็นแล้วว่าการทำงานของ R ส่วนมากเกี่ยวข้องกับฟังก์ชันซึ่งอาร์กิวเมนต์ถูกกำหนดให้อยู่ภายในเครื่องหมายวงเล็บ ผู้ใช้สามารถเขียนฟังก์ชันได้ด้วยตัวเองและแน่นอนฟังก์ชันเหล่านี้จะมีคุณสมบัติเช่นเดียวกับฟังก์ชันอื่น ๆ ใน R

การเขียนฟังก์ชันด้วยตัวเองทำให้การใช้ R มีประสิทธิภาพ ยืดหยุ่นและสมเหตุสมผล กลับมาที่ตัวอย่างของเราของการอ่านข้อมูลบางข้อมูลซึ่งตามมาด้วยการพล็อตกราฟ ถ้าเราต้องการทำการปฏิบัติการในเงื่อนไขที่แตกต่างกันมันอาจจะเป็นแนวคิดที่ดีที่เขียนฟังก์ชันหนึ่งขึ้นมาดังตัวอย่างต่อไปนี้

```
myfun <- function(S, F)
{
    data <- read.table(F)
    plot(data$V1, data$V2, type="l")
    title(S)
}
```

เพื่อให้ถูกกระทำได้ ฟังก์ชันนี้ต้องถูกนำเข้าในหน่วยความจำและการนำเข้าสามารถกระทำได้ในหลายวิธี ในบรรทัดของฟังก์ชันนี้สามารถถูกพิมพ์เข้าโดยตรงผ่านทางแผนแป้นอักขระ (keyboard) คล้ายกับคำสั่งอื่น ๆ หรือสำเนา (copy) และวาง (paste) จากเอดิเตอร์ (editor) ถ้าฟังก์ชันได้ถูกบันทึกไว้ในไฟล์ข้อความ (text file) แล้วมันสามารถถูกนำเข้าได้ด้วยฟังก์ชัน `source()` ในทำนองเดียวกับโปรแกรมอื่น ถ้าผู้ใช้ต้องการบางฟังก์ชันให้ถูกนำเข้าในแต่ละครั้งที่ R เริ่มทำงาน ฟังก์ชันเหล่านี้สามารถถูกบันทึกไว้ใน workspace `.RData` ซึ่งจะถูกนำเข้าในหน่วยความจำถ้ามันอยู่ในไดเร็กทอรี่ที่กำลังทำงาน ความเป็นไปได้อีกทางหนึ่งคือกำหนดไฟล์ให้เหมาะสมด้วย '.Rprofile' หรือ 'Rprofile' (ดู `?Startup`

สำหรับรายละเอียด) สุดท้ายมีความเป็นไปได้ที่จะสร้างแพ็กเกจแต่การสร้างแพ็กเกจจะไม่ถูกกล่าวถึงในที่นี้ (ดูคู่มือ “Writing R Extensions”)

ทันทีที่ฟังก์ชันถูกนำเข้า เราจะสามารถใช้คำสั่งเดี่ยวในการอ่านข้อมูลและพล็อตกราฟได้ ตัวอย่างเช่นกับ `myfun("swallow", "Swal.dat")` ดังนั้นในตอนนี้เรามีโปรแกรมของเราเป็นเวอร์ชันที่สามดังต่อไปนี้

```
layout(matrix(1:3, 3, 1))
myfun("swallow", "Swal.dat")
myfun("wren", "Wrenn.dat")
myfun("dunnock", "Dunn.dat")
```

นอกจากนี้เราอาจจะใช้ `sapply()` นำไปสู่เวอร์ชันที่สี่ของโปรแกรมของเราดังนี้

```
layout(matrix(1:3, 3, 1))
species <- c("swallow", "wren", "dunnock")
file <- c("Swal.dat" , "Wren.dat", "Dunn.dat")
sapply(species, myfun, file)
```

ใน R มันไม่มีความจำเป็นต้องแจ้งตัวแปรที่ถูกใช้ภายในฟังก์ชันหนึ่ง เมื่อฟังก์ชันนั้นถูกกระทำการแล้ว R ใช้กฎที่เรียกว่า *lexical scoping* เพื่อตัดสินว่าอ็อบเจ็กต์เป็น local ต่อฟังก์ชันนั้นหรือเป็น global เพื่อที่จะเข้าใจกลไกนี้ ให้เราพิจารณาฟังก์ชันที่ง่ายมากข้างล่างนี้

```
> foo <- function() print(x)
> x <- 1
> foo()
[1] 1
```

ชื่อ `x` ไม่ได้ถูกใช้เพื่อสร้างอ็อบเจ็กต์ภายใน `foo()` ดังนั้น R จะหาในสภาพแวดล้อมที่ *enclosing* ถ้ามีอ็อบเจ็กต์ที่ถูกเรียกว่า `x` และจะพิมพ์ค่าของมัน (มิฉะนั้นแล้วข้อความที่ระบุความผิดพลาดจะถูกแสดงและทำให้หยุดกระทำการ)

ถ้า `x` ถูกนำไปใช้เช่นชื่อของอ็อบเจ็กต์ภายในฟังก์ชัน ค่าของ `x` ในสภาพแวดล้อม global จะไม่ถูกใช้

```
> x <- 1
> foo2 <- function() { x <- 2; print(x) }
> foo2()
[1] 2
> x
[1] 1
```

จากตัวอย่างข้างต้น `print()` ใช้อ็อบเจ็กต์ `x` ซึ่งถูกกำหนดให้ไว้ภายในสภาพแวดล้อมคือ สภาพแวดล้อมของ `foo2`

คำว่า "*enclosing*" ข้างต้นเป็นคำสำคัญ ในสองฟังก์ชันตัวอย่างนี้มีสองสภาพแวดล้อม คือ global และอีกหนึ่งสภาพแวดล้อมของฟังก์ชัน `foo` หรือ `foo2` ถ้ามีสามหรือมากกว่าสามสภาพแวดล้อมที่ซ้อนข้างใน (nested environment) แล้วการค้นหาสำหรับอ็อบเจ็กต์ถูกทำได้ดีขึ้นจากสภาพแวดล้อมที่ถูกกำหนดให้ถึงสภาพแวดล้อม enclosing ไปเรื่อย ๆ จนถึงสภาพแวดล้อม global

มีสองวิธีเพื่อระบุอาร์กิวเมนต์ไปยังฟังก์ชันซึ่งทำโดยตำแหน่งหรือโดยชื่อของอาร์กิวเมนต์ (*tagged arguments* เป็นอีกชื่อที่ถูกเรียก) ตัวอย่างเช่น ถ้าเราพิจารณาฟังก์หนึ่งที่มีสามอาร์กิวเมนต์ดังต่อไปนี้

```
foo <- function(arg1, arg2, arg3) {...}
```

`foo()` สามารถถูกกระทำการได้โดยปราศจากการใช้ชื่อ `arg1, ...` ถ้าอ็อบเจ็กต์ที่เชื่อมโยงถูกใส่ในตำแหน่งที่ถูกต้อง ตัวอย่างเช่น `foo(x, y, z)` อย่างไรก็ตามตำแหน่งไม่มีความสำคัญหากชื่อของอาร์กิวเมนต์ถูกนำไปใช้ ตัวอย่างเช่น `foo(arg3 = z, arg2 = y, arg1 = x)` คุณลักษณะอีกอย่างหนึ่งของฟังก์ชัน R คือความเป็นไปได้ในการใช้ค่าโดยปริยายในนิยามของมัน ดังตัวอย่างต่อไปนี้

```
foo <- function(arg1, arg2 = 5, arg3 = FALSE) {...}
```

คำสั่ง `foo(x)`, `foo(x, 5, FALSE)` และ `foo(x, arg3 = FALSE)` จะได้ผลลัพธ์เดียวกันอย่างสมบูรณ์ การใช้ค่าโดยปริยายในนิยามของฟังก์ชันเป็นประโยชน์มาก โดยเฉพาะเมื่อถูกใช้กับ tagged arguments (เช่น เพื่อเปลี่ยนเพียงค่าโดยปริยายค่าหนึ่งคือ `foo(x, arg3 = TRUE)`).

เพื่อสรุปในส่วนนี้ พิจารณาอีกหนึ่งตัวอย่างซึ่งไม่ได้เป็นตัวอย่างทางสถิติจริงๆ แต่ตัวอย่างนี้แสดงถึงความยืดหยุ่นของ R เมื่อพิจารณาว่าต้องการศึกษาพฤติกรรมของตัวแบบไม่เชิงเส้น (non-linear model) ตัวแบบนี้ของ Ricker ถูกนิยามไว้ดังสมการต่อไปนี้

$$N_{t+1} = N_t \exp\left[r\left(1 - \frac{N_t}{K}\right)\right]$$

ตัวแบบนี้ถูกใช้อย่างกว้างขวางในพลวัตประชากรโดยเฉพาะของปลา ซึ่งต้องการใช้ฟังก์ชันหนึ่งเพื่อสร้างตัวแบบจำลองสถานการณ์ในส่วนของอัตราการเติบโต (r) จำนวนประชากรเริ่มต้น (N_0) (ค่าความจุของประชากร (carrying capacity: K มักได้ค่าเท่ากับ 1 และค่านี้จะถูกทำให้เป็นค่าโดยปริยาย) ซึ่งผลลัพธ์จะถูกแสดงเป็นการพล็อตของจำนวนที่เกี่ยวข้องกับเวลา โดยจะเพิ่มตัวเลือกเพื่อทำให้ผู้ใช้แสดงเพียงแค่จำนวนในระยะเวลาช่วงท้ายเล็กน้อยเท่านั้น (ค่าโดยปริยายคือผลลัพธ์ทั้งหมดจะถูกพล็อต) ฟังก์ชันต่อไปนี้สามารถทำการวิเคราะห์ตัวเลขของตัวแบบ Ricker ได้

```
ricker <- function(nzero, r, K=1, time=100, from=0, to=time)
{
    N <- numeric(time+1)
    N[1] <- nzero
    for (i in 1:time) N[i+1] <- N[i]*exp(r*(1 - N[i]/K))
    Time <- 0:time
    plot(Time, N, type="l", xlim=c(from, to))
}
```

โดยสามารถทำด้วยตัวเองดังตัวอย่างนี้

```
> layout(matrix(1:3, 3, 1))
> ricker(0.1, 1); title("r = 1")
> ricker(0.1, 2); title("r = 2")
> ricker(0.1, 3); title("r = 3")
```

7 เอกสารเกี่ยวกับ R

คู่มือ (manuals) คู่มือมากมายเกี่ยวกับ R ถูกส่งไปใน R_HOME/doc/manual/:

- *An Introduction to R* [R-intro.pdf],
- *R Installation and Administration* [R-admin.pdf],
- *R Data Import/Export* [R-data.pdf],
- *Writing R Extensions* [R-exts.pdf],
- *R Language Definition* [R-lang.pdf].

ไฟล์เหล่านี้อาจอยู่ในรูปแบบแตกต่างกัน (pdf, html, texi, . . .) โดยขึ้นอยู่กับชนิดของการติดตั้ง

เอฟเอคิว (FAQ) R ถูกส่งมาพร้อมกับคำถามที่พบบ่อย (*Frequently Asked Questions*) ซึ่งอยู่ที่ไดเรกทอรี่ (directory) R_HOME/doc/html/ เวอร์ชันของ R-FAQ จะถูกปรับให้เป็นปัจจุบันเสมอบนเว็บไซต์ของ CRAN ดังต่อไปนี้

http://cran.r-project.org/doc/FAQ/R-FAQ.html

ทรัพยากรในระบบเครือข่าย (on-line resources) เว็บไซต์ของ CRAN เก็บเอกสารมากมาย ทรัพยากรทางบรรณานุกรมและการเชื่อมโยงไปยังเว็บไซต์อื่น ๆ นอกจากนี้ยังแสดงรายการของการตีพิมพ์หรือเผยแพร่ (หนังสือหรือบทความ) เกี่ยวกับ R หรือวิธีการทางสถิติ[21] และเอกสารบางอย่างรวมทั้งการสอนซึ่งถูกเขียนโดยผู้ใช้ R[22].

รายชื่อและที่อยู่ผู้ได้รับข้อมูล (mailing lists) มีสี่รายการของการสนทนาในการแลกเปลี่ยนเรียนรู้เรื่องของ R ซึ่งเกี่ยวข้องกับการสมัครสมาชิก การส่งข้อความและการอ่านการเก็บถาวร (archives) ดูได้ที่ http://www.R-project.org/mail.html.

'r-help' เป็นรายการของการสนทนาทั่วไปหนึ่งที่เป็นแหล่งของข้อมูลที่น่าสนใจสำหรับผู้ใช้ R (อีกสามรายการถูกเสนอให้แจ้งในเวอร์ชันใหม่และเป็นรายการสำหรับนักพัฒนาโปรแกรม) ผู้ใช้ R มากมายได้ส่งฟังก์ชันหรือโปรแกรมไปที่ 'r-help' ซึ่งสามารถถูกพบได้ในการเก็บถาวร ถ้าพบปัญหาเกี่ยวกับ R แล้ว ควรจะดำเนินการตามลำดับต่อไปนี้ก่อนส่งข้อความไปที่ 'r-help'

1. อ่าน online help อย่างรอบคอบ (อาจใช้เครื่องค้นหา (search engine)

[21] http://www.R-project.org/doc/bib/R-publications.html
[22] http://cran.r-project.org/other-docs.html

2. อ่าน R-FAQ
3. ค้นหาการเก็บถาวรของ 'r-help' ซึ่งที่อยู่ด้านบนหรือโดยการใช้หนึ่งในเครื่องค้นหาซึ่งถูกพัฒนาโดยบางเว็บไซต์[23]
4. อ่าน "posting guide"[24] ก่อนส่งคำถาม

ข่าว R (R News) วารสารอิเล็กทรอนิกส์ *R News* มีจุดมุ่งหมายเพื่อเข้าถึงความรู้เกี่ยวกับ R หรือให้บริการเพื่อเติมช่องว่างระหว่างการสนทนาแลกเปลี่ยนเรียนรู้ทางอิเล็กทรอนิกส์ (electronic discussion) และการตีพิมพ์ทางวิทยาศาสตร์แบบดั้งเดิม ข่าว R ฉบับแรกถูกเผยแพร่ในเดือนมกราคม 2001 [25]

การอ้างอิง R ในการตีพิมพ์และเผยแพร่ (citing R in a publication) สุดท้ายถ้ามีการกล่าวถึง R ในการตีพิมพ์และเผยแพร่ การอ้างอิงดังต่อไปนี้ต้องถูกกล่าวถึง

> R Development Core Team (2005). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL: http://www.R-project.org.

[23] ที่อยู่ของที่ตั้งเหล่านี้ถูกแสดงรายการที่ http://cran.r-project.org/search.html
[24] http://www.r-project.org/posting-guide.html
[25] http://cran.r-project.org/doc/Rnews/