R For Data Science 2019

By DataRockie

First Edition 25.12.2018

Knowledge Belongs to All Men.

ความรู้คือสิ่งที่เราทุกคนควรได้รับ

ทำไมต้องอ่านหนังสือเล่มนี้

เราเขียนโดยใช้ภาษาที่อ่านง่าย กันเอง 55+ ใช้เวลาไม่นานก็อ่านจบ เนื้อหาครอบคลุมทุกเรื่องสำคัญของ R สำหรับงาน data science เบื้องต้น ไอเดียและเนื้อหามาจากประสบการณ์เรียน และการทำงานของตัวเองในช่วง 4 ปีที่ผ่านมา (2015-2018)

หนังสือเล่มนี้เหมาะสำหรับใคร

นักเรียนไม่จำเป็นต้องมีพื้นฐานการเขียนโปรแกรมใดๆมาก่อน แค่เคยใช้โปรแกรม Microsoft Excel พื้นฐานมาบ้าง ก็สามารถเรียน R สำหรับงาน data science ได้ทันที

เพราะ R เขียนเหมือนกับ Excel เลย โดยใช้ฟังชั่น (function) เป็นหลักในการทำงาน

ใช้เวลาอ่านเท่าไร

เราออกแบบหนังสือเล่มนี้ให้อ่านจบภายในเวลา **2-3 ชั่วโมง** (Self-Learning) มีทั้งหมด **20 บท** พร้อมฝึกเขียนโค้ดใน RStudio เพื่อทบทวนสิ่งที่อ่านจากหนังสือเล่มนี้

ผู้แต่ง

Facebook: DataRockie (แอดทอย)
Senior Manager, Data Science Unit @dtac

วิธีเรียน R อย่างได้ผล

พยายามไม่ copy โค้ดตัวอย่างในหนังสือเล่มนี้ไปแปะใน RStudio แล้วแค่กดรัน ดีที่สุดคือลองเขียนด้วยตัวเอง

ตาอ่าน มือเขียน สมองจำ

ปล. ถ้าต้อง copy โค้ดไปใช้จริงๆ จะติดปัญหานิดหน่อยตรง สัญลักษณ์คำพูด “...” ที่อาจต้องไปเขียนซ้ำใน RStudio อีกครั้งหนึ่ง เพราะว่า font มันไม่ match กัน

ติดปัญหา ทำอย่างไร

ถ้าเขียนแล้วรันโค้ดไม่ออก หรือว่า R โยน error กลับมา ลองนั่งอ่าน error นั้นก่อนว่าต้องแก้ยังไง หรือลองเข้าไปหาคำตอบใน Google

ทุกปัญหาที่เราเคยเจอ มั่นใจได้เลยว่าคนอื่นเคยเจอมาหมดแล้ว 5555+ และคำตอบมีเกือบ 100% ใน Google

ถ้าหาคำตอบไม่ได้จริงๆ หลังไมค์มาถามเราได้ที่ https://m.me/datarockie

คอร์สเรียนออนไลน์ฟรี

เรามีเปิดสอนฟรี คอร์ส R Python SQL SPSS PowerPoint บน https://datarockie.com ลองดู !!

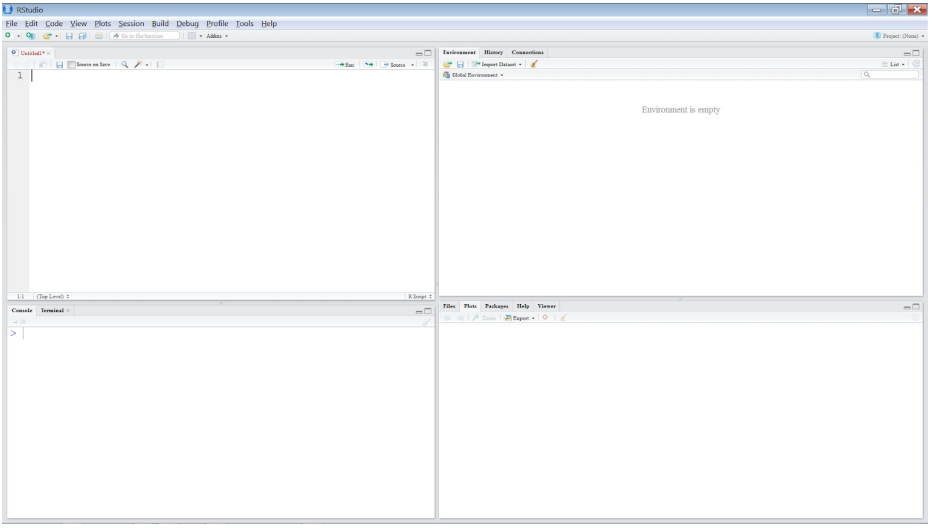
สารบัญ

1. ติดตั้งโปรแกรม R และ RStudio

นักเรียนทุกคนที่อยากทำตาม tutorial ในหนังสือเล่มนี้ ต้องลงโปรแกรมก่อนสองตัว ทั้งสองตัวใช้ฟรี (open-source) และมีคนใช้เยอะมากทั่วโลก สามารถติดตั้งได้ทั้ง windows, mac และ linux OS

- R https://cran.r-project.org/
- RStudio https://www.rstudio.com/products/rstudio/download/

สำหรับนักเรียนที่ใช้ windows เหมือนแอด สามารถกด install โปรแกรมปกติได้เลย ไม่ต้องเปลี่ยนอะไรเป็นพิเศษ พอลงเสร็จแล้ว เปิด RStudio ขึ้นมา จะเห็นหน้าตา interface แบบนี้

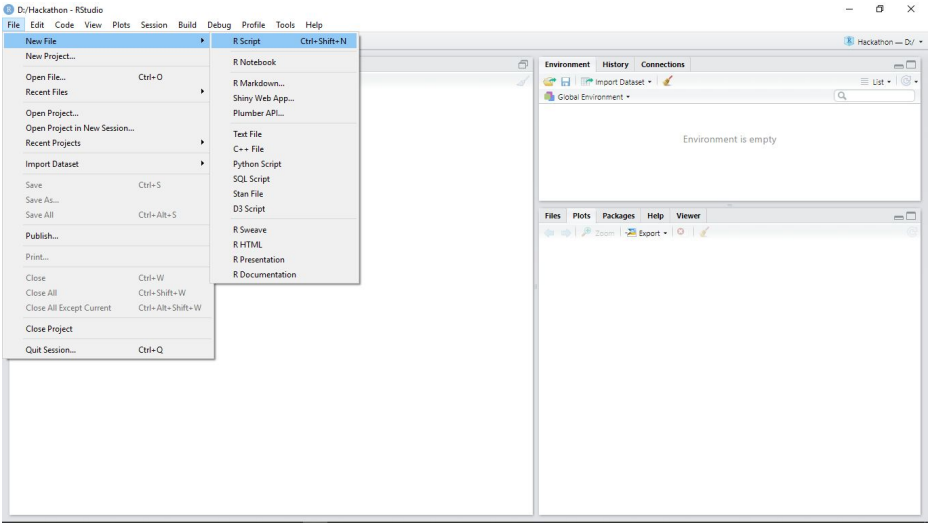


>> RStudio คือ IDE สำหรับงาน Data Science ที่ดีที่สุดในโลก (คหสต !) XD

RStudio จะแบ่งออกเป็น 4 หน้าต่าง

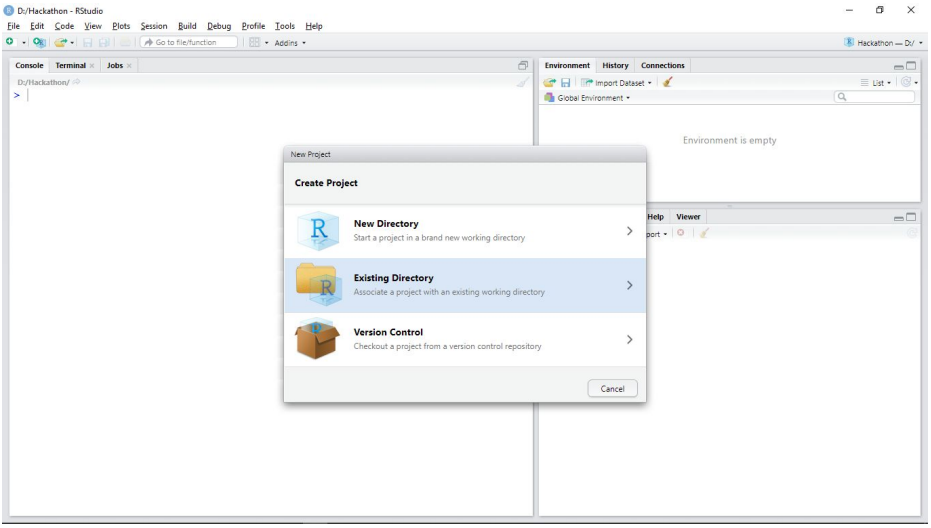
- Script (ซ้ายบน) - เอาไว้เขียน code
- Console (ซ้ายล่าง) - แสดง output ที่โค้ดเรารันออกมา
- Environment (ขวาบน) - แสดงตัวแปรที่เราสร้างขึ้นมา
- Files / Helps (ขวาล่าง) - แสดงกราฟที่เราสร้าง หรืออ่านคู่มือการใช้งานฟังชั่นต่างๆ

ถ้าใครยังไม่เห็นหน้าต่าง Script ซ้ายบน ให้เอาเมาส์ไปคลิกที่ **File > New File > R Script** (หรือกดปุ่ม shortcut CTRL + SHIFT + N)



ให้นักเรียนทุกคนสร้าง folder ใหม่ขึ้นมาที่หน้า desktop ชื่อว่า RDS2019 และไปที่ **File > New Project > Existing Directory > Browse** หา

โฟลเดอร์ที่เราเพิ่งสร้างขึ้นมา แล้วกด Create Project (OK) ได้เลย



RStudio จะ restart ตัวเองครั้งหนึ่ง ให้เพื่อนๆลองพิมพ์ว่า getwd() ลงไปที่ console เพื่อดู path ของ working directory ปัจจุบัน

เท่านี้เราก็ set up เรียบร้อย พร้อมเขียนโค้ดกันแล้ว !

แบบฝึกหัด - ลองพิมพ์ getwd() และ dir() ในหน้าต่าง console เพื่อดูไฟล์ทั้งหมดที่อยู่ใน working directory ของเรา

2. เริ่มเขียน R

R เป็นภาษาที่เรียกว่า **Read Evaluate Print Loop** (REPL) คือเราสามารถเขียนโค้ดแล้วกด **Enter** เพื่อรันผลและดู result ได้เลย

เราสามารถใช้ R เป็นเครื่องคิดเลขได้เหมือนโปรแกรม Excel ทั่วไป แค่พิมพ์ expression ที่เราต้องการลงไปในหน้าต่าง console แล้วกด Enter

```
# basic calculation
1 + 1
2 - 3
2 * 5
2 / 2
2 ** 2
5 %% 2
```

ข้อความที่อยู่หลังเครื่องหมาย # คือ comment เราสามารถเขียนคอมเม้นตรงไหนก็ได้ในโค้ดของเรา ตามหลัง #

R มีฟังชั่นในการคำนวณคณิตศาสตร์เยอะมาก ตัวอย่างเช่น

```
log(100)
sqrt(9)
exp(5)
```

แบบฝึกหัด - นักเรียนทุกคนลองเขียนโค้ดเล่นต่อใน console

3. การสร้างตัวแปร

R มีอีกหนึ่ง concept ที่เราเรียกว่า Object-Oriented Programming โดยเราสามารถสร้าง object หรือ variable (ตัวแปร) โดยใช้ <- เรียกว่า **assign operator**

```
x <- 100
y <- 200
x + y
```

การตั้งชื่อตัวแปรใน R เราแนะนำให้ใช้ตัวพิมพ์เล็กทั้งหมด เพราะว่า R เป็นภาษาแบบ case sensitive (ตัวพิมพ์เล็ก ใหญ่มีผลกับการเขียน R)

```
# create new variables
income2018 <- 50000
pct_increase <- 0.20
income2019 <- income2018 * (1 + pct_increase)

# see our new income in 2019
print(income2019)
```

ถ้าชื่อตัวแปรยาว หรือมีหลายคำ ให้เราใช้ _ หรือ . ในการสร้างชื่อ เช่น my_friends หรือ student_scores_2018

ปล. ชื่อตัวแปรห้ามขึ้นต้นด้วยตัวเลข แฮร่ !!

เราสามารถอัพเดทค่าในตัวแปรได้ ด้วยการ re-assign ค่าใหม่ เช่น

```
x <- 100
print(x)
x <- 500
print(x) # updated value
```

ถ้าต้องการลบตัวแปรออกจาก Environment ของเรา ให้ใช้ฟังชั่น rm()

```
# remove variable x
rm(x)

# remove all variables in current environment
rm(list = ls())
```

โอเคร ! ตอนนี้นักเรียนสามารถเขียนโค้ด R ง่ายๆได้แล้ว รวมถึงการสร้าง อัพเดท และลบตัวแปร

ต่อไปขอให้ทุกคนเขียนโค้ดในหน้าต่าง **Script** ด้านซ้ายบนของ RStudio จะได้ save code เก็บไว้ดูทีหลังได้ โดยไฟล์ Script ที่เราเซฟจะอยู่ใน project working directory ที่เราสร้างไว้ในบทที่ 1

เวลาเขียนโค้ดในหน้าต่าง Script (ซ้ายบน) ถ้าต้องการกดรันโค้ดเพื่อดูผลลัพธ์ สามารถกด shortcut CTRL + ENTER เพื่อรันโค้ด line นั้นๆได้เลย

ถ้าตรงการ clear หน้าจอ console ให้กด shortcut CTRL + L พร้อมกัน

แบบฝึกหัด - ลองเขียนโค้ดในหน้าต่าง Script เช่น 2 ** 20 แล้วกด CTRL + ENTER เพื่อคำนวณค่านี้และดูคำตอบในหน้าต่าง Console เสร็จแล้วกด CTRL + L เพื่อเคลียร์หน้าจอ

4. ทำความรู้จักกับ Vector

Vector คือ data structure รูปแบบที่เราใช้เยอะมากในงาน data science ถ้าใครนึกหน้าตาของ vector ไม่ออก จริงๆมันก็แค่เอาตัวเลข (หรือค่าต่างๆ) มาเรียงต่อกัน เราสามารถสร้าง vector ใน R ด้วยฟังชั่น c()

```
scores <- c(80, 99, 90, 72, 55)
print(scores)
```

เราสามารถบวก ลบ คูณ หาร หรือใช้ฟังชั่นกับ vector ได้เหมือนกับตัวเลขเดี่ยวๆทั่วไปเลย เช่น

```
# try this
scores + 10
scores * 2
updated <- c(5, 2, 3, 5, 7)
new_scores <- scores + updated
```

เราสามารถดึงข้อมูลออกมาจาก vector ได้ด้วยการใช้สัญลักษณ์ [ ] ใน R เรียกว่า Subset โดยตัวเลขใน [1] คือตำแหน่งของข้อมูลที่เราต้องการ

```
new_scores[1] # 85
new_scores[1:3] # 85 101 93
new_scores[3:5] # 93 77 62
```

แบบฝึกหัด - ลองพิมพ์ 1:20 ลงไปใน console เห็นอะไร?

5. ทำความรู้จักกับ Data Types

Data types ที่สำคัญใน R จะมีอยู่ 4 ประเภท ประกอบด้วย

- numeric เช่น 2.5, 3.3, 100, 200, 250, 1000
- character เช่น “hello”, “data science”, “world”
- logical มีแค่สองค่าคือ TRUE, FALSE
- factor คือตัวแปร categorical ทางสถิติใน R เช่น เพศ (male, female) อุณหภูมิ (high, med, low) เป็นต้น

ตั้งแต่บทที่ 1-4 ที่เราเรียนมา เราเห็นข้อมูลแบบ numeric มาเยอะแล้ว เราสามารถตรวจสอบ data type ของตัวแปรนั้นๆด้วยฟังชั่น class() ตามตัวอย่างด้านล่าง

```
# numeric vector
x <- c(100, 200, 300)
class(x)
is.numeric(x)

# character vector
y <- c(“Jason”, “Momoa”, “AquaMan”)
class(y)
is.character(y)

# logical vector
z <- c(TRUE, FALSE)
class(z)
is.logical(z)
```

ส่วนตัวแปร factor จะเป็นตัวแปรทางสถิติใน R ชื่อทางการคือ **categorical**

data (หรือตัวแปรที่ใช้แบ่งกลุ่ม observations ของเรา)

```
# create character vector
animals <- c("Dog", "Cat", "Dog", "Dog", "Hippo")
class(animals)

# convert character to factor
animals <- as.factor(animals)
class(animals)
```

เราสามารถเปลี่ยน character variable ให้กลายเป็น factor ง่ายๆด้วยฟังชั่น as.factor() ตามตัวอย่างด้านบน

ถ้าต้องการนับจำนวน Dog Cat Hippo ให้ใช้ฟังชั่น table() แบบนี้

```
# get frequency table
table(animals)

# get percentage table
table(animals) / length(animals)
```

R จะมี set ของฟังชั่นที่ใช้ในการตรวจสอบ data types และ convert data types เป็นแบบที่เราต้องการ ควรฝึกใช้งานให้คล่อง เพราะเราใช้บ่อยมาก!

ตรวจสอบ data type	**เปลี่ยน data type**
is.numeric()	as.numeric()
is.character()	as.character()
is.logical()	as.logical()

is.factor()	as.factor()

ลองดูตัวอย่างการใช้งานคร่าวๆ

```
# convert x to character
x <- 100
is.numeric(x) # TRUE
x <- as.character(x)

# convert logical to numeric
y <- TRUE
is.logical(y)
y <- as.numeric(y) # 1
```

สิ่งที่ควรรู้คือ TRUE, FALSE ใน R สามารถ convert กลายเป็นตัวเลขได้ โดย TRUE = 1 และ FALSE = 0

```
x <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

# we can use sum and mean functions with logical vector
sum(x) # 3
mean(x) # 0.6 computed from 3/5
```

แบบฝึกหัด - ลองสร้างตัวแปร factor เช่นเพศ gender - Male, Female

6. ทำความรู้จักกับ Data Frame

Data Frame คือ data structure รูปแบบที่สำคัญมาก (ที่สุด) ใน R สำหรับงาน data science เลย ถ้าใครนึกไม่ออกว่ามันหน้าตาเป็นยังไง จริงๆมันคือ Excel Table ทั่วไป มี column และ row แบบนี้

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

บทนี้เรามาลองดูวิธีการสร้าง dataframe ด้วยโค้ดง่ายๆ สมมติว่าเราเป็นอาจารย์มหาวิทยาลัย และต้องการทำ dataframe เพื่อเก็บคะแนนสอบของนักเรียนในห้องเรา

นักเรียนในห้องมีทั้งหมด 5 คน

```
# create three vectors
students <- c("Joe", "Jason", "Momoa", "Anna", "Mary")
scores <- c(69, 89, 92, 56, 87)
grades <- c("C+", "B+", "A", "C", "B+")

# create a single dataframe
df <- data.frame(students, scores, grades)
print(df)
```

สรุปง่ายๆ Data Frame คือการนำ vector ที่เราสร้างมาประกอบร่างรวมกัน โดยโค้ดตัวอย่างด้านบนจะสร้าง df ที่มีขนาด 5 rows x 3 columns

R มีฟังชั่นเยอะมากที่ใช้ในการจัดการ Data Frame เช่น

```
# try this in RStudio
str(df)
head(df)
tail(df)
summary(df)
```

R ยังมาพร้อมกับ built-in dataframe เยอะมาก ให้เราลองเล่น แค่พิมพ์โค้ด data() ลงไปใน console โดย tutorial ที่เราจะสอนต่อจากนี้จะใช้ dataframe ที่ชื่อว่า mtcars เป็นหลัก

```
# load mtcars into our Environment
data("mtcars")
```

7. ฟังชั่นที่เราใช้ทำงานกับ Data Frame

ทุกครั้งที่เรามี dataframe ใหม่ โหลดเข้ามาใน RStudio เราควรต้องดูหน้าตาของ df นั้นก่อน 5 ฟังชั่นด้านล่างเป็นตัวที่แอดใช้ประจำเลย

```
# load mtcars dataframe
data("mtcars")

# review structure of dataframe
str(mtcars)

# print first six rows
head(mtcars)

# print last six rows
tail(mtcars)

# summary statistics
summary(mtcars)
```

ถ้าต้องการเรียกดู column names และ row names (ถ้ามี) แค่พิมพ์

```
colnames(mtcars)
rownames(mtcars)
```

เราสามารถดึงข้อมูลบางส่วนออกมาจาก dataframe (หรือ vector) ด้วยการใช้เครื่องหมาย [ ] มาช่วย ดูตัวอย่างด้านล่าง

```
# subset rows 1-10, all columns
mtcars[1:10, ]
```

```
# subset rows 1-10, columns 1-5
mtcars[1:10, 1:5]

# subset all rows, columns 1-5
mtcars[ , 1:5]

# subset by column names
mtcars[ , "mpg"]
mtcars[ , c("mpg", "hp", "wt")]
```

เราสามารถเซฟ dataframe ที่เรา subset ออกมาใน variable ใหม่ แค่ใช้ assign operator แบบนี้

```
subset_mtcars <- mtcars[1:10, c("mpg", "hp", "wt")]
```

อีก task หนึ่งที่เราทำประจำเลย คือการตรวจสอบว่า df ของเรามี missing value หรือเปล่า? ใน R เราใช้ฟังชั่น complete.cases() เพื่อหาค่า missing value หรือ NA ใน R

```
# return logical vector TRUE for complete rows
# FALSE for incomplete rows i.e. have NA
complete.cases(mtcars)

# if below code = 1, data is complete
mean(complete.cases(mtcars))

# subset only complete rows
clean_df <- na.omit(mtcars)
```

ถ้าเราใช้ฟังชั่น mean(complete.cases(df)) กับ data frame แล้วได้ค่าเฉลี่ยเท่ากับ 1 แปลว่าทุก rows ใน df นั้นสะอาด ไม่มี NA เลย

แต่ถ้า mean(complete.cases(df)) ไม่เท่ากับ 1 แปลว่ามี NA อยู่ โดยเราสามารถลบทุก rows ที่มี NA ง่ายๆด้วยฟังชั่น na.omit()

มาลองดูตัวอย่าง missing value (NA) อีกนิดนึง

```
x <- c(100, 200, 300, NA, 400, 500, NA)
is.na(x)
```

เราใช้ฟังชั่น is.na() เพื่อตรวจสอบ NA ได้เช่นกัน ในโค้ดตัวอย่างได้บนจะได้ผลลัพธ์ออกมาเป็น logical vector เท่ากับ F, F, F, T, F, F, T โดยที่ T=TRUE และ F=FALSE

ถ้าเราต้องการลบ NA ออกจาก vector นี้ ก็ใช้วิธีการ subset ได้เลย เราใช้เครื่องหมาย ! เพื่อ reverse ค่า TRUE → FALSE หรือ FALSE → TRUE

```
x <- x[! is.na(x)]
```

แบบฝึกหัด - ลองสร้าง logical vector อะไรก็ได้ เช่น x <- c(TRUE, FALSE) แล้วลองพิมพ์ !x ลงไปใน console

8. อ่านไฟล์ csv เข้าสู่ RStudio

R สามารถอ่านไฟล์ข้อมูลได้หลายประเภท แต่ที่เราใช้กันเยอะที่สุดคือไฟล์ plain text อย่าง csv

หน้าตาของ csv เป็นแบบนี้ สามารถเปิดได้ด้วยโปรแกรม Excel หรือ text editor ทั่วไป วิธีการ import ไฟล์ csv เข้าสู่ RStudio ให้เรานำไฟล์ csv นั้นไปใส่ใน working folder ของเรา (ที่เราตั้งไว้ในบทที่ 1) และพิมพ์โค้ดด้านล่างได้เลย โดย argument คือชื่อไฟล์ csv ที่เราต้องการ

```
df <- read.csv("filename.csv")
str(df)
```

ฟังชั่น read.csv() มี argument อีกเยอะมาก ให้เราปรับใช้งาน ลองพิมพ์ help(read.csv) เพื่ออ่านวิธีการใช้แบบละเอียด หรือลองดูใน Google ก็ได้

หรือถ้าต้องการ export dataframe ออกจาก RStudio ก็สามารถทำได้ง่ายๆ ด้วยฟังชั่น write.csv() แบบนี้ โดย argument ที่สองคือชื่อไฟล์ที่เราเซฟในโฟลเดอร์ เช่น “mtcars.csv”

```
write.csv(mtcars, "mtcars.csv")
dir()
```

โดยไฟล์ที่เรา export จะไปอยู่ใน working directory ของเรา สามารถเรียกดูไฟล์นั้นได้ด้วยฟังชั่น dir()

9. ติดตั้ง Package เพิ่มความสามารถให้ R

R มีฟังชั่นให้เราเลือกใช้งานมากกว่า 2 ล้านฟังชั่น (ธันวาคม 2018) แต่ฟังชั่นทั้งหมดยังไม่ได้ถูกโหลดเข้ามาที่ laptop ของเรา

เราสามารถเลือก download และ install package (functions) ที่เราต้องการใช้งานได้ด้วยคำสั่ง install.packages("package_name") โดย package ที่เราแนะนำให้ทุกคนติดตั้ง และใช้ให้คล่องคือ **dplyr** ของทีมงาน RStudio

```
# install only once
install.packages("dplyr")

# use library every time we open RStudio
library(dplyr)
```

เราจะใช้คำสั่ง install.packages( ) ติดตั้ง package แค่ครั้งเดียว แต่ทุกครั้งที่เราเปิดโปรแกรม RStudio ขึ้นมาใหม่ และต้องการใช้งาน package dplyr ให้เราพิมพ์ library(dplyr) เสมอ

website ที่ให้ข้อมูลเกี่ยวกับ packages และ functions ทั้งหมดใน R คือ https://www.rdocumentation.org/ ลองเข้าไปดูเล่นๆได้

แบบฝึกหัด - นักเรียนลองติดตั้ง package `ggplot2` ในเครื่องของตัวเอง

10. Data Wrangling Part I

Data Wrangling (n.) อ้างอิงจาก wikipedia คือการ transform data ให้อยู่ในรูปแบบที่เหมาะสมกับงาน data analytics แบบต่างๆ

dplyr มี 5 functions หลักที่เราใช้ในการ wrangling data frame และปรับเปลี่ยนหน้าตาของข้อมูลเราได้อย่างรวดเร็ว ประกอบด้วย

- select()
- filter()
- arrange()
- mutate()
- summarise()

เลือกคอลั่มด้วย select()

เราสามารถเลือกคอลั่มที่เราต้องการด้วยฟังชั่น select() มาลองฝึกกับ mtcars อีกครั้งหนึ่ง

```
# load library
library(dplyr)

# select columns 1-5
select(mtcars, 1:5)

# select columns mpg, hp, wt
select(mtcars, mpg, hp, wt)
```

โดย argument แรกของฟังชั่น select() และฟังชั่นอื่นๆของ dplyr ที่เราจะเรียนด้วยกันวันนี้คือชื่อ data frame เสมอ

ฟิลเตอร์ records ด้วย filter()

เราสามารถเขียนเงื่อนไขง่ายๆเพื่อใช้ filter records ที่เราต้องการได้

```
# filter cars with hp <= 100
filter(mtcars, hp <= 100)

# filter cars with am == 0
filter(mtcars, am == 0)
```

โดยเงื่อนไขที่เราสามารถเขียนได้ใน R มีดังต่อไปนี้ เราเรียกว่า **equality condition** สิ่งที่ต่างจากภาษาอื่นๆคือ R ใช้ == เพื่อเปรียบเทียบสองฝั่งของสมการว่าเท่ากันหรือเปล่า? และใช้ != ที่แปลว่าไม่เท่ากับ

- ==
- !=
- >=
- <=
- >
- <

มาลองเขียน condition ง่ายๆใน R ด้วย vector

```
# see if any numbers <= 300
numbers <- c(100, 200, 300, 50)
numbers <= 300
```

```
# this will return FALSE because R is case-sensitive
“hello” == “Hello”
```

จัดเรียงข้อมูลด้วย arrange()

เราสามารถ sort ข้อมูลจากน้อยไปมาก หรือมากไปน้อยได้ด้วยฟังชั่นนี้

```
# sort column hp from low to high
arrange(mtcars, hp)

# sort column hp from high to low
arrange(mtcars, desc(hp))
```

%>% (Pipe Operator)

dplyr มีสัญลักษณ์พิเศษที่ใช้ในการเชื่อมโค้ดเข้าด้วยกัน และทำให้การทำ data wrangling ใน R ง่ายขึ้นเป็นกอง **%>%** เรียกว่า pipe operator

```
# these two lines get the same result
head(mtcars)
mtcars %>% head()
```

โค้ดทั้งสองไลน์ด้านบนได้ผลลัพธ์เหมือนกัน การใช้ %>% จะมีประโยชน์มาก ถ้าเราต้องเชื่อมโค้ดเราหลายๆต่อ เช่น

dataframe %>%
    function( ) %>%
    function( ) %>% ...

เราสามารถเขียน pipe เพื่อเชื่อมฟังชั่น select() filter() arrange() ได้แบบนี้

```
mtcars %>%
    select(mpg, hp, wt) %>%
    filter(hp <= 100) %>%
    arrange(desc(hp))
```

Code ด้านบนทำสามอย่างนี้ตามลำดับ

1. เลือกสามคอลั่ม mpg hp wt
2. เลือกมาเฉพาะรถยนต์ที่ hp น้อยกว่าหรือเท่ากับ 100
3. เรียงข้อมูลด้วยคอลั่ม hp จากมากไปน้อย

สร้าง DataFrame ใหม่

เราสามารถ save ผลลัพธ์ที่ได้จาก dplyr เป็น data frame ใหม่ (subset_df) แบบนี้

```
subset_df <- mtcars %>%
                 select(mpg, hp, wt) %>%
                 filter(hp <= 100) %>%
                 arrange(desc(hp))

str(subset_df)
```

แบบฝึกหัด - ให้นักเรียนลองเขียน dplyr เพื่อดึงคอลั่ม mpg, hp, wt, am, gear และฟิลเตอร์เฉพาะรถยนต์ที่มี am == 0 เสร็จแล้วเรียงข้อมูลตามคอลั่ม mpg จากค่ามากเป็นค่าน้อย และเลือกมาเฉพาะ 5 คันแรก i.e. %>% head(5)

11. Data Wrangling Part II

บทที่ 10 เราได้เรียนสามฟังชั่นหลักในการ wrangling dataframe ไปแล้ว คราวนี้มาต่อกันกับสองฟังชั่นที่เหลือ เริ่มกันที่ mutate()

เราใช้ mutate() ในการสร้างคอลั่มใหม่

```
# create new column hp_double
mtcars %>%
    mutate(hp_double = hp * 2)

# create multiple columns
mtcars %>%
    mutate(hp_double = hp * 2,
           log_hp = log(hp),
           exp_hp = exp(hp))
```

และฟังชั่นสุดท้าย summarise() ใช้ในการสรุปผลสถิติเบื้องต้น

```
# summarise column
mtcars %>%
    summarise(avg_hp = mean(hp),
              sd_hp = sd(hp),
              min_hp = min(hp),
              max_hp = max(hp),
              n = n())
```

ฝึกใช้ทั้ง 5 functions นี้ให้คล่อง เพื่อนๆจะสามารถปรับหน้าตาของ dataframe ได้อย่างรวดเร็ว และมีประสิทธิภาพ

สรุปกันอีกครั้งหนึ่ง

- select() ใช้เลือกคอลั่มที่เราต้องการ
- filter() ใช้ฟิลเตอร์ records หรือ rows
- arrange() ใช้เรียงข้อมูล
- mutate() ใช้สร้างคอลั่มใหม่
- summarise() ใช้สรุปผลสถิติ
- %>% ใช้ในการเชื่อมโค้ดเข้าด้วยกันอย่างรวดเร็ว

ตัวอย่างสุดท้าย

```
mtcars %>%
    select(mpg, hp, wt, am) %>%
    filter(hp >= 100) %>%
    arrange(hp) %>%
    mutate(hp_double = hp * 2) %>%
    head(10)
```

dplyr ยังทำอะไรได้อีกเยอะมาก ลองศึกษาเพิ่มเติมได้ที่นี่ https://dplyr.tidyverse.org/

ปล. Hadley Wickham ผู้สร้าง dplyr เรียก package นี้ว่า “The Grammar of Data Manipulation” โคตรเจ๋ง !!

12. Exploratory Data Analysis

มาเรียนวิธีการสร้างกราฟเพื่อทำ Exploratory Data Analysis กันบ้าง เรียกสั้นๆว่า EDA

หลักการเลือกใช้กราฟง่ายๆ มีแค่สองข้อเอง

1. จำนวนตัวแปรที่ต้องการจะ plot
2. ประเภทของตัวแปร i.e. data types

Data Types ใน R จะมีอยู่ 4 ประเภทที่เราเรียนไปในบทที่ 5 เราสามารถจับกลุ่มเป็นสองประเภทใหญ่ๆได้ดังนี้

1. ตัวแปรเชิงปริมาณ i.e. Quantitative คือตัวแปรแบบ numeric
2. ตัวแปรเชิงคุณภาพ i.e. Qualitative ได้แก่ factor, logical และ character

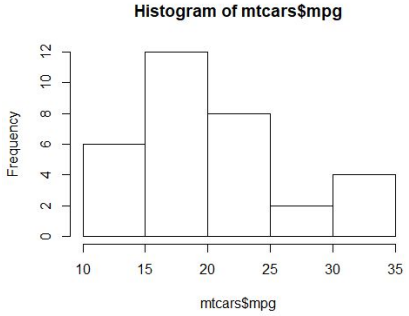
กราฟสำหรับหนึ่งตัวแปร เชิงปริมาณ

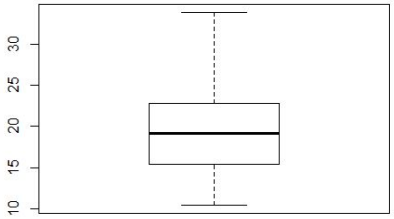
เราใช้ histogram หรือ box plot ในการ visualize ข้อมูลหนึ่งตัวแปร เชิงปริมาณ i.e. numeric โดยใช้ฟังชั่น hist() และ boxplot() ดังนี้

```
data("mtcars")
hist(mtcars$mpg)
boxplot(mtcars$mpg)
```

R จะสร้างกราฟให้เราดูเร็วๆในหน้าต่างขวาล่างของ RStudio เราสามารถ

กดปุ่ม Zoom เพื่อเข้าไปดูกราฟชัดๆ หรือกดปุ่ม Export เพื่อเซฟรูปก็ได้





กราฟสำหรับหนึ่งตัวแปร เชิงคุณภาพ

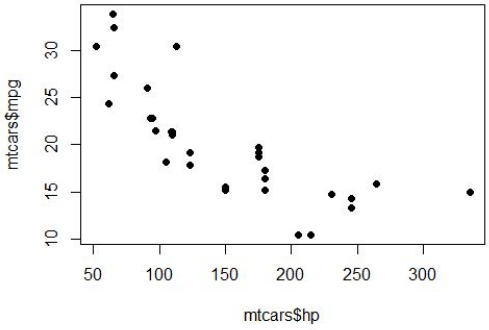
สำหรับตัวแปรเชิงคุณภาพ อย่าง factor ที่เราใช้เยอะๆ หลักๆเราจะ visualize ด้วย bar plot เพื่อดูการกระจายตัว i.e. นักสถิติเรียกว่า distribution

```
t1 <- table(mtcars$am)
barplot(t1)
```

กราฟสำหรับสองตัวแปร เชิงปริมาณทั้งคู่

Scatter Plot คือกราฟที่เราใช้เยอะมากในงานสถิติ เพื่อแสดงความสัมพันธ์ของตัวแปรเชิงปริมาณ i.e. numeric สองตัว

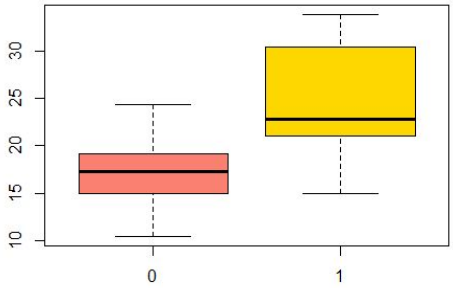
```
plot(mtcars$hp, mtcars$mpg)
plot(mtcars$hp, mtcars$mpg, pch = 16)
```



กราฟสำหรับสองตัวแปร เชิงปริมาณ x เชิงคุณภาพ

อันนี้จะค่อนข้าง tricky สำหรับมือใหม่ เราสามารถใช้ box plot เพื่อแสดง distribution ของตัวแปรเชิงปริมาณ แยกตามกลุ่มได้ด้วยโค้ดนี้

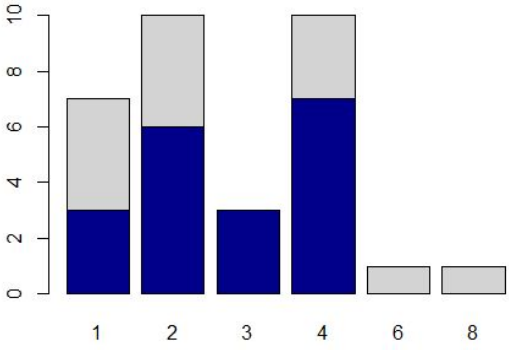
```
boxplot(mtcars$mpg ~ mtcars$am)
boxplot(mtcars$mpg ~ mtcars$am, col = c(“salmon”,
“gold”))
```



กราฟสำหรับสองตัวแปร เชิงคุณภาพทั้งคู่

เรากลับมาใช้ stacked bar plot สำหรับกรณีนี้ ตามรูปด้านล่าง เราสามารถปรับสีในกราฟของเราได้ที่ argument ชื่อว่า col (ย่อมาจาก color)

```
t1 <- table(mtcars$am, mtcars$carb)
barplot(t1, col = c("darkblue", "lightgrey"))
```



สรุปกราฟที่เราสามารถ plot เพื่อดูข้อมูลเราเร็วๆ ขึ้นอยู่กับจำนวนตัวแปรและประเภทของตัวแปร

- histogram - หนึ่งตัวแปร numeric
- box plot ได้ทั้งแบบหนึ่งตัวแปร numeric หรือสองตัวแปร numeric x factor
- bar plot ได้ทั้งแบบหนึ่งหรือสองตัวแปรแบบ factor
- scatter plot - สองตัวแปร numeric x numeric

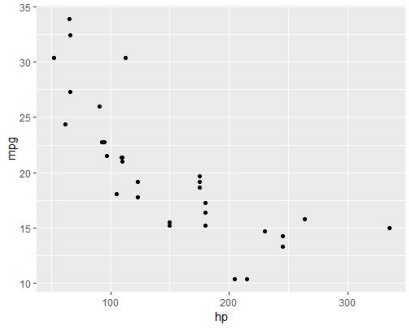
13. สร้างกราฟสวยด้วย ggplot2

package ในตำนานของ R สำหรับทำ graphic สวยๆคือ ggplot2 โดยทีมงาน RStudio เหมือนเดิม โดย gg ย่อมาจาก **The Grammar of Graphics**

```
install.packages("ggplot2")
library(ggplot2)
```

วิธีการเขียน ggplot2 จะเป็น template แบบนี้

```
# scatter plot
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg)) +
    geom_point()
```



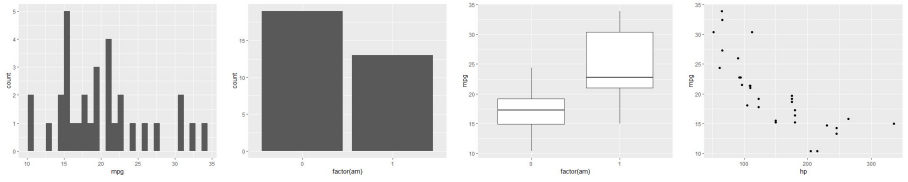
ggplot2 ทำให้การสร้างกราฟใน R ง่ายขึ้น เพราะว่ามันมี template หลักในการ plot กราฟประเภทต่างๆ (ถึงแม้ว่าจะเรียนรู้ยากในช่วงแรก) ลองดูตัวอย่างการสร้าง histogram, barplot, boxplot, scatter plot ได้ด้านล่าง

Example Code

argument แรกในฟังชั่น ggplot คือชื่อ dataframe ส่วน argument ที่สองจะถูก wrap ใน aes( ) คือการเลือกคอลั่มที่เราต้องการใช้สร้างกราฟ

geom_xxx( ) คือ layer ที่เราเรียกว่า geometry เป็นการบอก R ว่าเราต้องการสร้างกราฟแบบไหน เช่น geom_histogram, geom_bar, geom_boxplot หรือ geom_point สำหรับสร้าง scatter plot

```
# histogram
ggplot(mtcars, aes(mpg)) + geom_histogram()

# bar plot
ggplot(mtcars, aes(factor(am))) + geom_bar()

# boxplot
ggplot(mtcars, aes(factor(am), mpg)) + geom_boxplot()

# scatter plot
ggplot(mtcars, aes(hp, mpg)) + geom_point()
```



สำหรับเพื่อนๆที่อยากศึกษา ggplot2 อย่างจริงจัง สามารถเข้าไปที่เว็บไซต์นี้ https://ggplot2.tidyverse.org/ ได้เลย

14. คำนวณสถิติเบื้องต้น

R เกิดมาพร้อมกับฟังชั่นสถิติมากมาย หลายคนถึงกับบอกว่า 99.99% ของ Statistics ในโลกนี้ สามารถทำได้ใน R สบายๆ บทนี้เราจะมารีวิวฟังชั่นสถิติที่ใช้บ่อยๆใน R ให้เพื่อนๆได้ลองเขียนกัน

โดยตัวอย่างในบทนี้ยังใช้ dataframe `mtcars` เหมือนเดิม

เราสามารถเลือกคอลั่มจาก dataframe ได้ด้วยเครื่องหมาย $ (Dollar Sign)

```
# load dataframe
data("mtcars")

# simple statistics
mean(mtcars$mpg)
median(mtcars$mpg)
sd(mtcars$mpg)
var(mtcars$mpg)
min(mtcars$mpg)
max(mtcars$mpg)
quantile(mtcars$mpg)
summary(mtcars$mpg)
```

ถ้าเพื่อนๆอยากจะอ่านเพิ่มเติม เพื่อศึกษาว่าฟังชั่นเหล่านี้ใช้งานยังไง ต้องมีกี่ argument สามารถเรียก help file ขึ้นมาอ่านได้แค่พิมพ์ว่า help(ชื่อฟังชั่น) หรือพิมพ์ ? ตามด้วยชื่อฟังชั่น

```
help(mean)
?mean
```

สำหรับฟังชั่นที่เราเพิ่งเขียนไป จะเหมาะกับคอลั่มที่เป็นแบบ numeric เท่านั้น ถ้าเราต้องการวิเคราะห์ข้อมูลคอลั่มที่เป็นแบบ factor ปกติเราจะทำได้แค่นับความถี่ หรือสร้างตาราง crosstabs ง่ายๆ เช่น

```
# convert columns am and gear to factor
mtcars$am <- as.factor(mtcars$am)
mtcars$gear <- as.factor(mtcars$gear)

# frequency table
table(mtcars$am)

# crosstabs am x gear
table(mtcars$am, mtcars$gear)
```

แบบฝึกหัด - ลองเขียนฟังชั่นเพื่อวิเคราะห์คอลั่ม hp ใน mtcars บ้าง

แบบฝึกหัดพิเศษ - ลอง install package นี้

```
install.packages("skimr")
library(skimr)
library(dplyr)
```

แล้วลองพิมพ์ฟังชั่น skim(mtcars) ใน console
What do you see?

```
#  try this in console
mtcars %>%
    group_by(am) %>%
    skim(mpg, hp, wt)
```

```
Skim summary statistics
 n obs: 32
 n variables: 11
 group variables: am

-- Variable type:numeric -------------------------------------------------------
 am variable missing complete  n   mean    sd    p0    p25    p50    p75   p100     hist
  0       hp       0       19 19 160.26 53.91 62    116.5  175    192.5  245
  0      mpg       0       19 19  17.15  3.83 10.4   14.95  17.3   19.2   24.4
  0       wt       0       19 19   3.77  0.78  2.46   3.44   3.52   3.84   5.42
  1       hp       0       13 13 126.85 84.06 52     66     109    113    335
  1      mpg       0       13 13  24.39  6.17 15     21     22.8   30.4   33.9
  1       wt       0       13 13   2.41  0.62  1.51   1.94   2.32   2.78   3.57
```

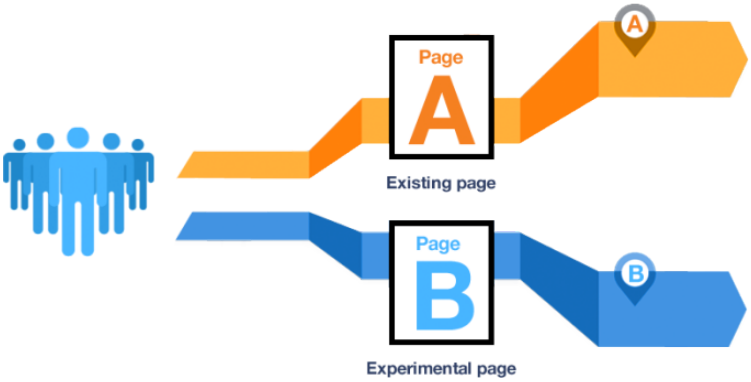
ฟังชั่น skim( ) ช่วยเราคำนวณค่าสถิติแบบเร็วๆ

- คอลั่ม missing บอกจำนวน NA
- คอลั่ม p0 p25 p50 p75 p100 คือค่า percentile ตัวอย่างเช่น p50 `ค่า median` เป็นจุดที่แบ่งข้อมูลออกเป็นสองช่วง 50% เท่ากัน
- คอลั่ม hist ดูการกระจายตัวของตัวแปรเร็วๆ i.e. histogram
- ตัวอย่างด้านบนเราใช้ group_by(am) %>% skim(hp, mp, wt) สามารถคำนวณค่าสถิติแยกตามกลุ่มตัวแปร am ได้ โดย am 0 = `Automatic` และ am 1 = `Manual`

15. AB Testing

งานหนึ่งที่ Data Analyst / Data Scientist ทำเป็นประจำคือการทำทดลอง หรือที่เราเรียกเป็นภาษาอังกฤษว่า Experiment

ที่ใช้กันเยอะสุด เรียกว่าเป็น Gold Standard คือ AB Testing



ตัวอย่างเช่น เรามีหน้าเว็บไซต์ e-commerce และเราต้องการจะเปรียบเทียบระหว่าง background สีส้ม (ปัจจุบัน) และสีฟ้า (ใหม่) ว่าสีไหนจะเปลี่ยนพฤติกรรมลูกค้าได้ดีกว่ากัน เช่น % คนกดซื้อของมากขึ้น เป็นต้น

สมมติวันนี้มี traffic ลูกค้าเข้ามาในหน้าเว็บไซต์ของเรา 200 คน เราจะทำการแบ่งลูกค้าเป็นสองกลุ่มเท่าๆกัน **แบบสุ่ม** ตามรูปด้านบน และบันทึกข้อมูลว่าลูกค้ากดหรือไม่กดซื้อสินค้าของเราบนเว็บ

มาลองดูโค้ดที่เราเขียนใน R เพื่อ simulate data ชุดขึ้นมา

```
# simulated data
set.seed(99)
controls <- sample(c(0,1), size=100, replace=T)

set.seed(99)
tests <- sample(c(0,1), size=100, replace=T,
prob=c(0.3,0.7))
```

AB Testing บางทีเราก็เรียกว่า control vs. test groups โดยกลุ่ม control คือ 100 คนที่เห็นสีส้ม (ปัจจุบัน) และกลุ่ม test คือ 100 คนที่เห็นสีฟ้า (ใหม่)

เราใช้ฟังชั่น sample() ในการ simulate ข้อมูลแบบ 0/1 ขึ้นมาแบบสุ่ม โดยที่ 0 = ลูกค้าไม่กดซื้อ และ 1 = ลูกค้ากดซื้อสินค้า เป็นข้อมูลแบบ binary

ส่วนฟังชั่น set.seed(99) มีไว้เพื่อล็อคผลที่ได้จากฟังชั่น sample() เฉยๆ (เวลานักเรียนทำตามโจทย์นี้ จะได้ค่าที่ simulated ออกมาเหมือนกัน)

Significance Testing

พอเรามีข้อมูลพร้อมแล้ว ก็แค่ใช้ฟังชั่น var.test() และ t.test() เพื่อเปรียบเทียบการกระจายตัวและค่าเฉลี่ยของทั้งสองกลุ่ม

```
# test variances
var.test(controls, tests)

# test means - this is AB testing
t.test(controls, tests, var.equal = TRUE)
```

จะได้ผลลัพธ์ออกมาหน้าตาแบบนี้

```
> # test means - this is AB testing
> t.test(controls, tests, var.equal = TRUE)

        Two Sample t-test

data:  controls and tests
t = -3.441, df = 198, p-value = 0.0007068
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.36181099 -0.09818901
sample estimates:
mean of x mean of y
     0.51      0.74
```

R จัดการคำนวณค่าสถิติทั้งหมดที่เราต้องการ เช่นค่า t = -3.441, df = 198 ค่าเฉลี่ยของกลุ่ม control = 0.51 และค่าเฉลี่ยของกลุ่ม test = 0.74

แล้วเรารู้ได้ยังไงว่า control vs. test แตกต่างกันอย่างมีนัยสำคัญทางสถิติ? ทั่วไปนักสถิติจะใช้ค่า p-value ในการทดสอบสมมติฐานนี้

เราสามารถเรียกดูค่า p-value ของ AB test เราด้วยโค้ดด้านล่าง

```
result <- t.test(controls, tests, var.equal = TRUE)
print(result$p.value)
```

ถ้า p-value <= 0.05 เราจะสรุปผลว่าค่าเฉลี่ย conversion rate ของกลุ่ม control vs. test แตกต่างกันอย่างมีนัยสำคัญทางสถิติ (ไม่ได้เกิดขึ้นแบบฟลุ๊คๆ) ซึ่งเราพยายามจะบอกว่ามันเกิดจากการเปลี่ยนสีเว็บไซต์จากสีส้มเป็นสีฟ้านั่นเอง

แบบฝึกหัด - ลองดูค่า p-value ที่ได้ออกมา เราจะสรุปผลว่าอย่างไร?

16. Monte Carlo Simulation

เอาแบบง่ายๆ Monte Carlo Simulation คือการจำลองสถานการณ์และทำซ้ำเหตุการณ์นั้นไปเรื่อยๆเป็น 100, 1000, 10000, 100000, ล้านครั้ง และมาลองดูความน่าจะเป็นของผลลัพธ์

เรานิยมใช้ Monte Carlo กับเหตุการณ์ที่เกี่ยวข้องกับ randomness หรือความไม่แน่นอนรูปแบบต่างๆ ตัวอย่างเช่น การโยนลูกเต๋า เราไม่รู้ว่าลูกเต๋าหน้าต่อไปจะออกเลขอะไร?

บทนี้เราจะสอนเขียน Monte Carlo ง่ายๆใน R เพื่อจำลองการโยนลูกเต๋า 100 ครั้ง 1000 ครั้ง ไปจนถึง 1,000,000 ครั้ง และหาความน่าจะเป็นของการเกิดเลข {1, 2, 3, 4, 5, 6}

เราจะใช้ฟังชั่น sample( ) เหมือนเดิม เพื่อจำลองการโยนลูกเต๋า

```
# roll a die
sample(1:6, 1)

# roll 100 dices
result1 <- sample(1:6 , size = 100, replace = T)

# roll 1000 dices
result2 <- sample(1:6, size = 1000, replace = T)

# roll 1000000 dices
result3 <- sample(1:6, size = 1000000, replace = T)
```

มาลองสร้าง bar plot เพื่อดู distribution ความน่าจะเป็นของการเกิดเลข 1, 2, 3, 4, 5, 6 จากการโยนลูกเต๋า 100 ครั้ง 1000 ครั้ง และ 1 ล้านครั้ง

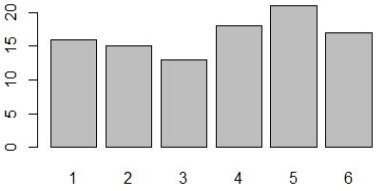
```
barplot(table(result1))
barplot(table(result2))
barplot(table(result3))
```

เห็นอะไรใน distribution

เราอาจจะคิดว่าการโยนลูกเต๋า 100 ครั้ง โอกาสที่เราจะเห็นเลข 1, 2, 3, 4, 5, 6 จะมีเท่ากันหมดเลย เท่ากับ 1 / 6 = 16.67% → bar chart ของเราควรจะมีขนาดเท่าๆกัน

แต่จากการทำ simulation จะเห็นว่าจริงๆแล้วการโยนลูกเต๋าแค่ 100 ครั้ง เรามีโอกาสที่จะเห็นเลข 5 หรือเลข 4 (หรือจะเป็นเลขอะไรก็ได้) มากกว่าเลข

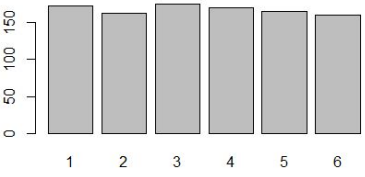
อื่นๆ ได้ผลแบบนี้แปลว่าลูกเต๋าของเรา **ไม่สมมาตร** i.e. biased?

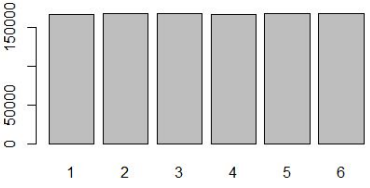


คำตอบคือ อาจจะใช่ หรืออาจจะไม่ใช่

เรายังตอบไม่ได้จนกว่าจะลองโยนลูกเต๋าต่อไปอีก การโยนลูกเต๋า แต่ละครั้งคือการเก็บข้อมูลเพิ่มขึ้น จนตอนที่เรามีข้อมูลมากพอ เราถึงจะตัดสินใจว่าลูกเต๋านี้เบี้ยว หรือสมมาตร

ยิ่งเราโยนลูกเต๋ามากขึ้น เป็น 1000 ครั้ง และ 1000000 ครั้ง เราจะพบว่า bar chart ของเราเริ่มจะมีขนาดเท่ากันขึ้นเรื่อยๆ ที่หนึ่งล้านครั้ง เราตอบได้อย่างมั่นใจว่าลูกเต๋านี้ **สมมาตร** มั่นใจได้เลยที่ 99.9999%



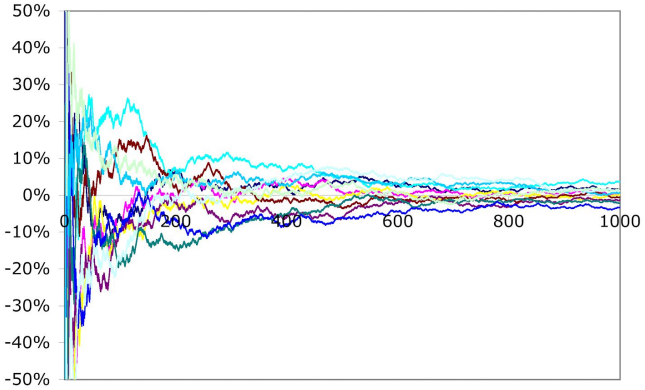


เพราะความน่าจะเป็นของลูกเต๋าออกหน้า {1, 2, 3, 4, 5, 6} ได้เท่ากันหมด

เลยที่ 16.6% และการทำ Monte Carlo Simulation ช่วยเราหาคำตอบว่าลูกเต๋านั้นแท้จริงแล้วเป็นอย่างไร

หัวใจสำคัญของ Monte Carlo คือ

- ยิ่งเก็บข้อมูลมากขึ้น ยิ่งมั่นใจมากขึ้น
- การสุ่มตัวอย่าง (เก็บ data) ต้องเป็นไปอย่าง random
- ในทางสถิติ เราเรียกกฎนี้ว่า **The Law of Large Numbers** ความน่าจะเป็นของเหตุการณ์หนึ่งๆจะ converge เข้าสู่ความจริง ถ้าเราสุ่มตัวอย่างมากขึ้นเรื่อยๆ



R มีอีกหนึ่งฟังชั่นที่ทรงพลังมาก ในการทำ Monte Carlo คือฟังชั่น replicate() ที่สามารถทำซ้ำ **ฟังชั่นอะไรก็ได้** ตามที่เรากำหนด เช่น

```
test_result <- replicate(1000, {
    sample(1:6, 1)
})
```

replicate( ) จะรับสอง argument โดย argument แรกคือจำนวนครั้งที่เราต้องการรัน e.g. 1000 และ argument ที่สองคือฟังชั่น { } ที่เราต้องการรัน

สรุปคือเวลาที่เราเจอกับเหตุการณ์ที่ไม่แน่นอน มีความแรนดอม มีความเสี่ยง เราอาจจะอยากรัน Monte Carlo เพื่อประเมินสถานการณ์ และช่วยให้เราตัดสินใจได้ดีขึ้น (ภายใต้ Uncertainty)

17. Correlation Matrix

Correlation คือการหาความสัมพันธ์ของตัวแปร numeric ตั้งแต่สองตัวขึ้นไป โดยค่า correlation จะมีค่าอยู่ระหว่าง -1 ไปจนถึง +1

- ค่ายิ่งเข้าใกล้ | 1 | แปลว่าความสัมพันธ์ยิ่ง Strong ปกติถ้าเกิน 0.7 ขึ้นไปถือว่าสูงแล้ว
- เครื่องหมาย + - แค่บอกทิศทางความสัมพันธ์ ถ้า + แปลว่าตัวแปรสองตัวเคลื่อนที่ในทิศทางเดียวกัน เช่น x เพิ่ม y เพิ่ม หรือ x ลด y ลด

ใน R ใช้แค่ฟังชั่น cor( ) ง่ายๆก็ได้คำตอบเลย

```
# two variables
cor(mtcars$mpg, mtcars$hp)

# five columns
cor(mtcars[, 1:5])

# all numeric columns in dataframe
cor(mtcars)
```

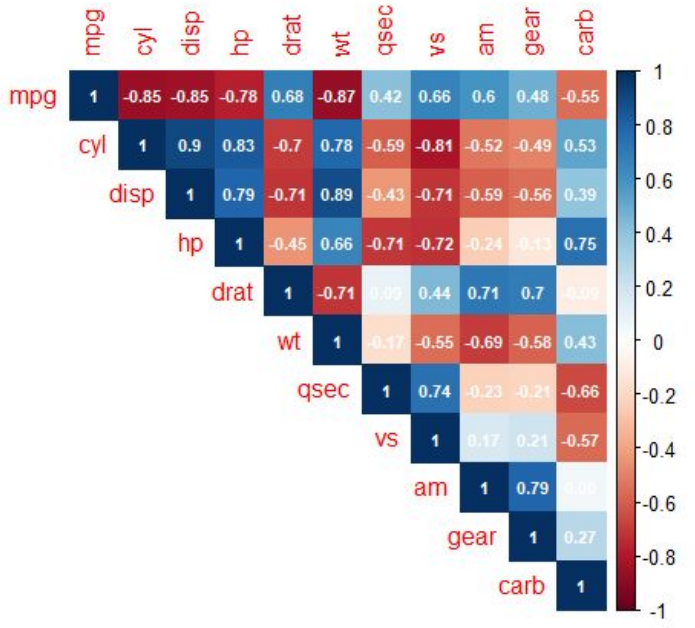
เราสามารถ visualize สร้างพล๊อทสวยๆได้จาก correlation matrix ต้องลง package เพิ่มอีกตัวนึงตามนี้

```
install.packages("corrplot")
library(corrplot)
cor_mat <- cor(mtcars)
corrplot(cor_mat)
```

เราสามารถปรับหน้าตาของ Correlation Plot ด้วยการเปลี่ยน argument ของฟังชั่น corrplot ลองอ่านวิธีการใช้งานด้วยโค้ด help(corrplot)

ปกติแอดจะเขียนแบบนี้เวลาสร้างพล็อท

```
corrplot(cor_mat, method = "color", type = "upper",
addCoef.col = "white", number.cex = 0.7)
```



แบบฝึกหัด - ลองอ่านวิธีการใช้งาน corrplot และลองเปลี่ยน method เป็นแบบอื่นๆ เช่น “number”, “shade”, “circle”, “ellipse” หรือ “square”

18. Linear Regression

Correlation บอกความสัมพันธ์ของตัวแปรเบื้องต้น แต่ถ้าเราอยากจะ quantify ความสัมพันธ์ให้ออกมาเป็นตัวเลข เช่น x เปลี่ยนแปลง 1 หน่วย → y จะเปลี่ยนแปลงกี่หน่วย เราสามารถตอบคำถามนี้ได้ด้วย Regression

R มีฟังชั่น lm() เพื่อใช้สร้าง Linear Model เขียนโค้ดแค่ไลน์เดียวสั้นๆ มาลองสร้างโมเดลเพื่อทำนายตัวแปร mpg (mile per gallon) ใน mtcars

โดยขั้นตอนการสร้างโมเดลแบบ basic มีสามขั้นตอน

1. แบ่งข้อมูลเป็น train และ test sets
2. สร้างโมเดลด้วย train
3. ทดสอบโมเดลด้วย test

```
# step 1
# split data into train and test set
train_df <- mtcars[1:30, ]
test_df <- mtcars[31:32, ]

# step 2
# build model
lm_fit <- lm(mpg ~ wt + hp + am, data = train_df)
summary(lm_fit)

# step 3
# predict and evaluate
p <- predict(lm_fit, newdata = test_df)
rmse <- sqrt(mean((p - test_df$mpg)**2))
```

R มีการเขียนโค้ดแบบพิเศษที่เราเรียกว่า `**Formula**` คือ argument แรกใน ฟังชั่น lm() อธิบายภาษาคนได้แบบนี้

```
lm_fit <- lm(mpg ~ wt + hp + am, data = train_df)
```

mpg ~ wt + hp + am เราบอก R ให้ใช้ตัวแปร wt, hp, am ไปทำนายตัวแปร mpg หรืออ่านแบบนักคณิตศาสตร์ว่า mpg = f(wt, hp, am)

พอเราสร้างโมเดลเสร็จแล้ว เราสามารถเรียกดูค่าสถิติสำคัญของโมเดลด้วย ฟังชั่น `summary(lm_fit)` จะได้ผลลัพธ์ด้านล่างใน console

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 33.92422    2.70423  12.545 1.56e-12 ***
wt          -2.55611    0.93601  -2.731 0.011192 *
hp          -0.04457    0.01156  -3.855 0.000681 ***
am           2.28426    1.45416   1.571 0.128310
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.541 on 26 degrees of freedom
Multiple R-squared:  0.8471,    Adjusted R-squared:  0.8294
F-statistic:    48 on 3 and 26 DF,  p-value: 9.715e-11
```

ใน R เราใช้ฟังชั่น predict() เพื่อนำโมเดลของเราไปทำนาย unseen data ที่โมเดลเราไม่เคยเห็นมาก่อน ในตัวอย่างของเราก็คือ test_df นั่นเอง สำหรับ Regression เราจะใช้ RMSE เพื่อดูว่าโมเดลเราทำนายได้แม่นยำแค่ไหน ยิ่งมีค่าเข้าใกล้ศูนย์ แปลว่าโมเดลเราทำนายดีมาก

ปล. Linear Regression ใช้ทำนายตัวแปรตาม (mpg) ที่เป็นแบบ numeric

กฎเหล็กในการสร้างและทดสอบโมเดล

Never use the same data to test your model

อธิบาย - นี่คือเหตุผลที่เราต้องแบ่งข้อมูลเป็น train และ test ก่อนทุกครั้ง หรืออาจจะมีอีกก้อนเพื่อ validate ก็ได้

19. Logistic Regression

ความแตกต่างระหว่าง Linear และ Logistic Regression คือ

- Linear ทำนายตัวแปรตามแบบ numeric
- Logistic ทำนายตัวแปรตามแบบ binary (1/0) ใน R คือตัวแปร factor

ข้อมูลที่เราจะใช้ในตัวอย่างนี้ มาจาก package `mlbench` โดยเราจะสร้างโมเดลเพื่อทำนายว่าผู้ป่วยคนนี้เป็นโรคเบาหวาน (diabetes) หรือเปล่า?

```
# install and load dataset
install.packages("mlbench")
library(mlbench)
data("PimaIndiansDiabetes")
df <- PimaIndiansDiabetes
head(df)

# inspect diabetes variable
table(df$diabetes)
```

ตัวแปรที่เราต้องการทำนายชื่อว่า diabetes เป็น binary (pos/ neg) โดยขั้นตอนการสร้างโมเดล มีสามขั้นตอนเหมือนเดิม เริ่มจากการ split dataset

```
# split data
set.seed(99)
idx <- sample(nrow(df), 0.8*nrow(df))
train_df <- df[idx, ]
test_df <- df[-idx, ]
```

เราแบ่ง train 80% และ test 20% (ratio สามารถเปลี่ยนได้) เสร็จแล้วก็สร้างโมเดลด้วยฟังชั่น glm() ย่อมาจากคำว่า `Generalized Linear Model`

```
glm_fit <- glm(diabetes ~ ., data = train_df, family =
"binomial")
summary(glm_fit)
```

Formula ในฟังชั่น glm( ) เราเขียนแบบนี้

diabetes ~ .

คือการบอกให้ R ใช้ตัวแปรต้นทั้งหมดใน train_df ไปทำนายตัวแปร diabetes เทียบเท่ากับการเขียน Formula แบบนี้

diabetes ~ pregnant + glucose + pressure + triceps + insulin + mass + pedigree + age

เสร็จแล้วเราก็เอาโมเดลไปทำนาย test_df ต่อด้วยโค้ดนี้

```
# prediction
p <- predict(glm_fit, newdata = test_df, type =
"response")
```

เราใช้ type = “response” ในฟังชั่น predict() เพื่อคำนวณค่า probability ที่ผู้ป่วยแต่ละคนจะเป็นโรคเบาหวาน ลองดูความน่าจะเป็นของ 5 คนแรก

```
p[1:5] # 0.85 0.03 0.59 0.80 0.42
```

ปกติเราจะต้องกำหนดค่า threshold เอง (default ของโปรแกรมสถิติหลายๆ ตัวจะอยู่ที่ 0.5) โดยค่า threshold คือตัวกำหนดการทำนายผลของเรา ว่าผู้ป่วยคนนั้นจะเป็นโรคเบาหวานหรือเปล่า

- IF p >= threshold THEN pos
- IF p < threshold THEN neg

เราสามารถเขียนโค้ดใน R เพื่อสร้าง confusion matrix และคำนวณ accuracy ความถูกต้องของโมเดลเราได้แบบนี้

```
# create confusion matrix
conf_mat <- table(p >= 0.5, test_df$diabetes)
print(conf_mat)

# accuracy 74.67%
(84 + 34) / nrow(test_df)
```

ตัวอย่างของ confusion matrix

```
> table(p >= 0.5, test_df$diabetes)

       neg pos
  FALSE  81  28
  TRUE   11  34
```

โดยความถูกต้องของการทำนายผลของเรา (Accuracy) จะคิดจากผลรวมเส้นทแยงมุม (81 + 34) / (81 + 34 + 28 + 11) = 74.67% Very Good !!

20. One Hidden Layer Neural Network

Neural Network เป็นโมเดลที่ทรงพลังมาก และเป็นหนึ่งในโมเดลพื้นฐานสำคัญของการพัฒนา AI ทุกวันนี้

การเขียน nn model ใน R ทำได้ง่าย ไม่ต่างกับการสร้าง linear หรือ logistic regression ที่เราสอนไปในบทก่อนหน้าเลย แค่ดาวน์โหลด package สองตัว

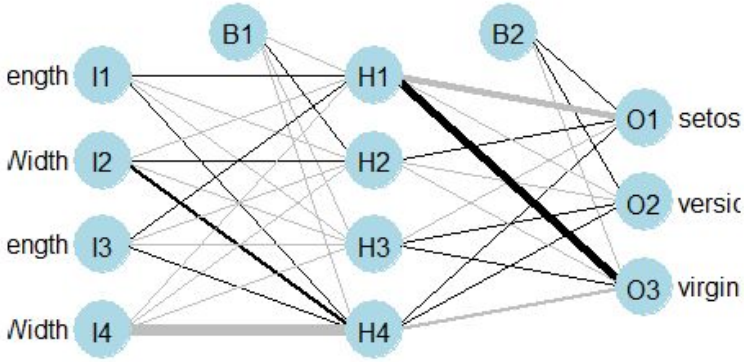
```
install.packages(c("nnet", "NeuralNetTools"))
library(nnet)
library(NeuralNetTools)
```

มาลองสร้าง nn model เพื่อทำนาย Species ของดอกไม้ iris ง่ายๆ โหลดข้อมูลเข้าสู่ RStudio ด้วยโค้ดนี้

```
# load built-in dataframe
data("iris")

# review structure of dataframe
str(iris)

# frequency table
table(iris$Species)
```

สร้างโมเดลและลองทำนาย test_df หลักการจะเหมือนกับตอนที่เราสร้าง linear และ logistic regression

```
# create train and test sets
set.seed(99)
idx <- sample(nrow(iris), 0.8*nrow(iris))
train_df <- iris[idx, ]
test_df <- iris[-idx, ]

# build model using all variables in dataframe
set.seed(99)
nn_fit <- nnet(Species ~ ., data = train_df, size = 4)

# plot neural network
plotnet(nn_fit)
```



คำนวณ Accuracy ของโมเดลเราด้วยโค้ดนี้

```
# predict test_df
p <- predict(nn_fit, newdata = test_df, type = "class")

# accuracy
mean(p == test_df$Species) # Accuracy = 100%
```

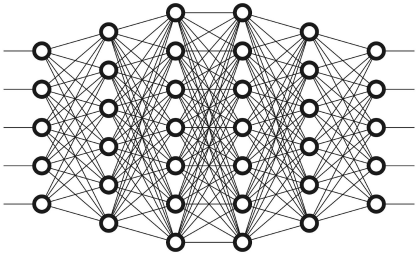
Neural network ที่เราสร้างขึ้นมาจะมีแค่ 3 layers (เรียกว่า shallow network) ประกอบด้วย

- Input layer ซ้ายสุด
- Hidden layer ตรงกลาง
- Output layer ขวาสุด

ส่วนเส้นตรงที่เชื่อมนิวรอน (วงกลม) ทั้งหมดใน network ตามทฤษฎีก็คือ linear regression นั้นเอง แต่ละเส้นจะมีค่า weight (คล้ายๆกับ regression coefficient) เพื่อช่วยกันทำนาย Species ของดอกไม้

เราตั้งค่า size = 4 ในฟังชั่น nnet() คือการกำหนดจำนวนนิวรอนใน hidden layer ยิ่งมีนิวรอนมาก ตามทฤษฎีคือจะช่วยให้ผลการทำนายดีขึ้น แต่ก็จะเสี่ยง overfit และใช้เวลาเทรนโมเดลนานขึ้น

ทุกวันนี้ ML researcher พัฒนา simple neural network ให้มีจำนวนชั้นของ hidden layer มากขึ้น เรียกสั้นๆว่า **Deep Learning** นั่นเอง



พรีวิวการสร้าง ML ด้วย caret

R เก่งมากเวลาสร้างโมเดล machine learning เพราะว่ามันเกิดมาเพื่องานนี้โดยเฉพาะเลย โดย package ที่แอดใช้เป็นประจำในการสร้างโมเดล ML คือ caret โคตรครบเครื่อง !!

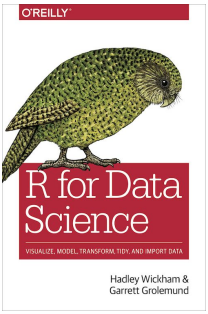
ลองดูตัวอย่างการสร้าง random forest ใน R ด้วย caret ด้านล่าง

```
# load dataset
library(mlbench)
library(dplyr)
data("Sonar")

# review dataframe
glimpse(Sonar)

# install caret package
install.packages("caret")
library(caret)

# random forest using 5 fold cross validation
ctrl <- trainControl(method = "cv", number = 5)
rf_model <- train(Class ~ .,
                  data = Sonar,
                  method = "rf",
                  trControl = ctrl)
```

R is super powerful !! เราสามารถเรียกดูโมเดลของเราแค่พิมพ์ rf_model ลงไปใน console เท่านี้เป็นอันเสร็จเรียบร้อย

แหล่งเรียนรู้ R แบบออนไลน์

แอดรวบรวมเว็บไซต์และหนังสือ R แบบอ่านฟรีออนไลน์มาให้เพื่อนได้ลองศึกษากัน หนังสือเล่มที่เราแนะนำมากที่สุด สำหรับคนที่เพิ่งเริ่ม คือ



อ่านฟรีที่นี่ https://r4ds.had.co.nz/

เขียนโดย Hadley Wickham ผู้พัฒนา package dplyr, ggplot2 และอีกมากมาย เค้าเป็น Chief Data Scientist ของ RStudio เล่มนี้โคตรดี !

Websites ที่แอดนั่งเรียน R ด้วยตัวเอง

- DataCamp - https://www.datacamp.com (เรียนฟรี คอร์ส Introduction ถ้าอยากเรียนคอร์สพรีเมียม จะมีโปรโมชั่น 90$ ต่อปี)
- Coursera - https://www.coursera.org (เรียนฟรี แต่ถ้าอยากได้ใบเซอร์ต้องเสียเงินนิดหน่อย 49$ ต่อคอร์ส หรือต่อเดือน)

DataCamp คือที่สุดของ R Programmers ถ้าใครอยากเรียน R อย่างจริงจัง แอดแนะนำมากๆ แอดเรียนมา 2-3 ปีแล้วบนเว็บนี้

ขอบคุณที่ติดตาม

ชอบกดไลค์ ใช่กดแชร์ให้เพื่อนได้อ่าน เย้!

ถ้าหนังสือเล่มนี้มีประโยชน์ ช่วยกด Recommend เพจ DataRockie ด้วยนะคร้าบที่ https://www.facebook.com/pg/datarockie/reviews/

คนที่ทำงานด้านนี้ ปกติจะมีสองภาษาที่ใช้กันเยอะๆคือ Python กับ R ในแง่การทำงานทั้งสองภาษาสูสีกันมาก ส่วนตัวแอดคิดว่า R เรียนง่ายกว่าเยอะเลย เพราะ R เกิดมาเพื่องานสถิติและ data science ตั้งแต่แรก

ข้อด้อยของ R อาจจะเป็นเรื่อง Deep Learning Framework ที่มีให้เลือกใช้งานน้อยกว่า Python มาก และเรื่อง Production/ Deployment อื่นๆ

แต่นอกนั้น (คหสต) แอดคิดว่า R เก่งกว่า Python เกือบหมดเลย 5555+ ไปล่ะ แอดรักทุกคน ชุ๊บบบบบบ <3