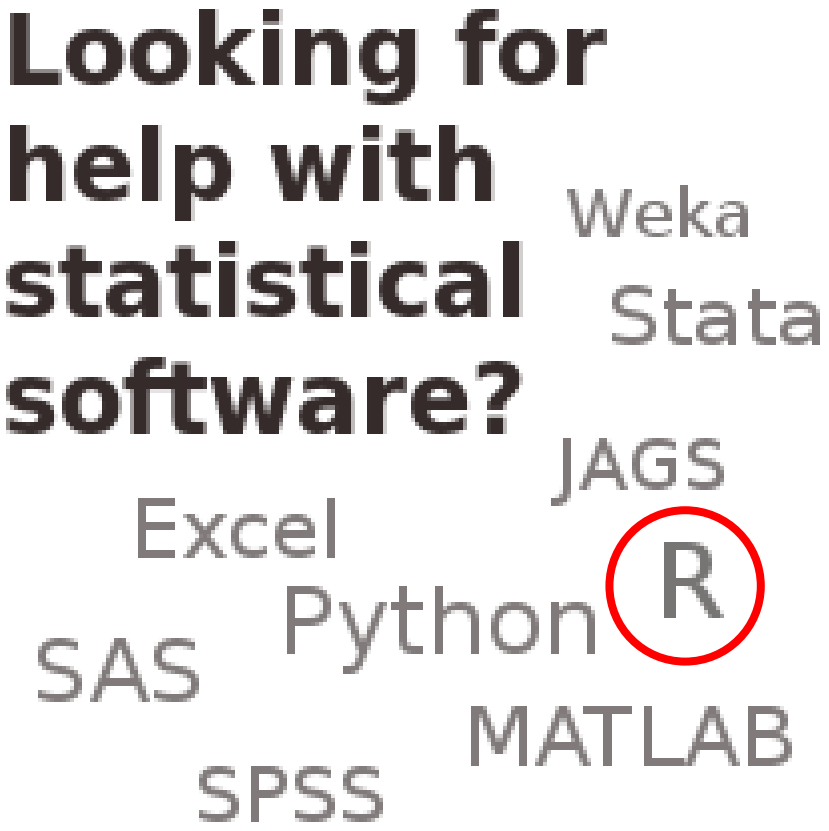
R Tutorial

Let’s meet R

王豐緒

資訊工程學系


Looking for help with statistical software?
Weka
Stata
JAGS
Excel
R
Python
SAS
MATLAB
SPSS

What is R?

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed.

Cited from: https://www.r-project.org/about.html

Learning Resources

The R Manuals

edited by the R Development Core Team.

The following manuals for R were created on Debian Linux and may differ from the manuals for Mac or Windows on platform-specific pages, but most parts will be identical for all platforms. The correct version of the manuals for each platform are part of the respective R installations. The manuals change with R, hence we provide versions for the most recent released R version (R-release), a very current version for the patched release version (R-patched) and finally a version for the forthcoming R version that is still in development (R-devel).

Here they can be downloaded as PDF files, EPUB files, or directly browsed as HTML:

Manual	R-release	R-patched	R-devel
An Introduction to R is based on the former "Notes on R", gives an introduction to the language and how to use R for doing statistical analysis and graphics.	HTML | PDF | EPUB	HTML | PDF | EPUB	HTML | PDF | EPUB
R Data Import/Export describes the import and export facilities available either in R itself or via packages which are available from CRAN.	HTML | PDF | EPUB	HTML | PDF | EPUB	HTML | PDF | EPUB
R Installation and Administration	HTML | PDF | EPUB	HTML | PDF | EPUB	HTML | PDF | EPUB
Writing R Extensions covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.	HTML | PDF | EPUB	HTML | PDF | EPUB	HTML | PDF | EPUB

Cited from: https://cran.r-project.org/manuals.html

Available Packages

R logo

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Contributed Packages

Available Packages

Currently, the CRAN package repository features 8316 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 33 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, OS X, Solaris and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Writing Your Own Packages

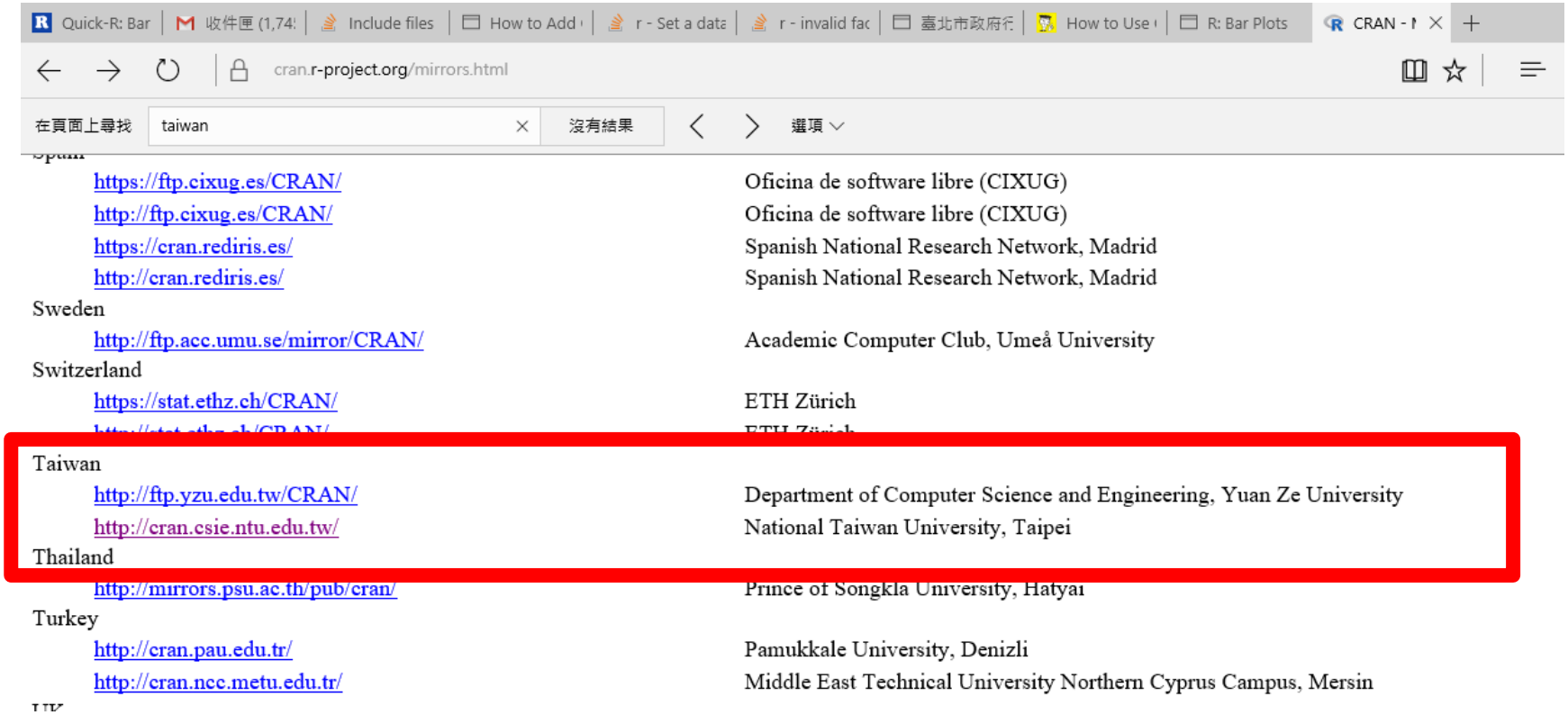
The manual Writing R Extensions (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Cited from: https://www.r-project.org/

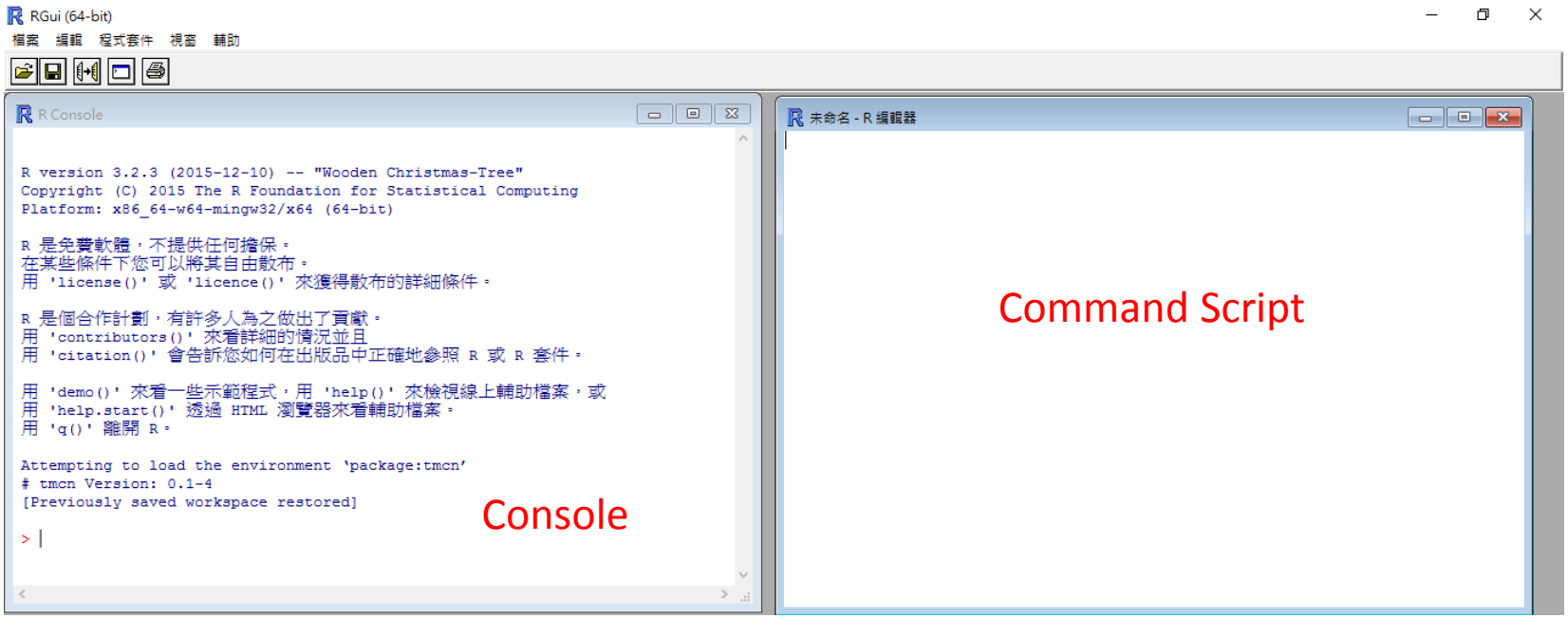
Let’s know a bit about R

DownLoad R and Install

https://www.r-project.org/



Get Familiar with the R Interface

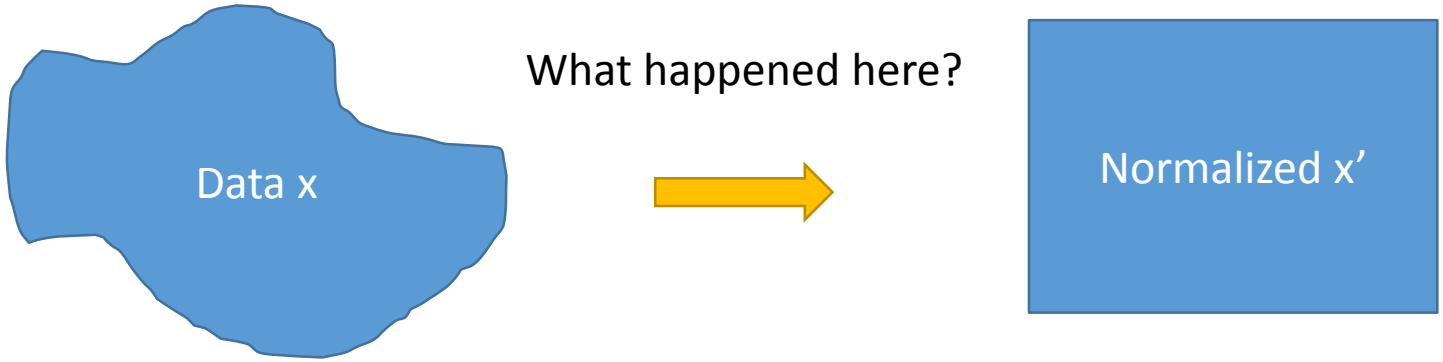


A Simple Sample

```
#Example Data
x <- sample(0:100,60) #random generate 60  numbers between 0 and 100

#Normalized Data
normalized <- (x-min(x))/(max(x)-min(x))

#Histogram of example data and normalized data
par(mfrow=c(1,2)) #set up the plot area as 1 by 2 row-based

hist(x,xlab="Data",col="lightblue",main="TEST1")

hist(normalized,xlab="Normalized Data",col="lightblue",main="TEST2")
```

Do you notice?

```
#Example Data
x <- sample(0:100,60) #random generate 60  numbers between 0 and 100

#Normalized Data
normalized <- (x-min(x))/(max(x)-min(x))
```



To know basics of R, You need to know THREE things…

- Concepts and Characteristics of DATA
- Vector-oriented Operation
- Specific Functions

Basic Data Classes

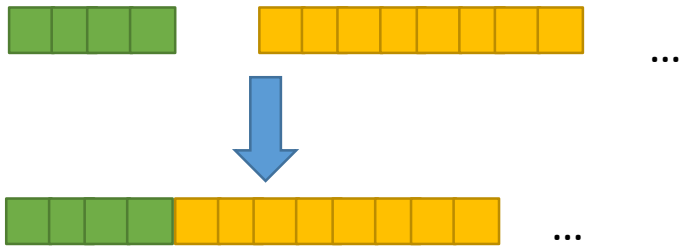
- Vector
- Matrix and Array (Sup.)
- List
- Data Frame

VECTOR

A data structure that consists of ordered objects of the same type!

Number and Vectors

- Number is a vector of length 1
  - X = 1
- N1**:** N2
  - Auto generate a vector consisting of a sequence of number N1, …, N2
  - Y=2:10
- **seq**(n1, n2, step) # generate a vector of a sequence of numbers from *n1* to *n2* with an increment of *step*
- Vector Concatenation
  - Z = ***c***(X, Y, …) # ***c*** command combines a series of vectors



Vector Arithmetic

- Exercise
  - X=1
  - Y=2:10
  - Z=c(X, Y)
  - 1/Z
  - Z^2+1
  - Z + (1:10)* 10
  - ***log***(Z) #log() is a vector operation: element-wise operation
  - ***sapply***(Z, ***log***): apply the function **log** to each element of Z, and return a resulting vector
  - 計算Z的變異數
    - ***sum***((Z-*mean*(Z))^2) / (**length**(Z) - 1)
    - ***var(Z)***

Boolean and Vectors

- Vector Boolean Operations
  - Z>5
  - Z > 5 **&** Z < 10 # & is the *and* logical operator
  - Z > 5 + Z < 10 # what error happens? + is an arithmetic operator
  - (Z > 5) + (Z < 10) #arithmetic on Boolean vectors: TRUE: 1, FALSE: 0
  - ***as.logical***((Z > 5) + (Z < 10)) # type conversion: zero: FALSE, non-zero: TRUE
  - (Z > 5) - (Z < 10) #arithmetic on Boolean vectors: TRUE: 1, FALSE: 0
  - ***as.logical***((Z > 5) - (Z < 10)) # type conversion: zero: FALSE, non-zero: TRUE

Subset a Vector

- X[*index vector*]
  - Return a vector of the elements of X indexed by the given index vector
- X[*Boolean exp*]
  - Return a vector of the elements of X satisfying the given Boolean expression
- Exercises
  - Z[1]
  - Z[c(1,3,5,7,9)]
  - Z[***as.logical***((Z > 5) + (Z < 10))]
  - Z[***as.logica***l((Z > 5) - (Z < 10))]
  - Z[***log***(Z) < 2]
- Extend and Cut a vector
  - **length**(Z) = 20 #extend a vector, adding default value NA
  - Z
  - Z[20]=1
  - Z
  - **length**(Z) = 10 #cut a vector
  - Z

A key concept on Vector–based Operation is **Auto-Expanding!**

Modification of Vector

- Modify subset of vector
  - Z=(1:10) * 2 #c(2, 4, 6, …, 20)
  - Z[log(Z) < 2] = 3 #update the specified subset
  - Z
- Extend and Cut a vector
  - **length**(Z) = 20 #extend the size of a vector of default value NA
  - Z
  - Z[20]=1
  - Z
  - **length**(Z) = 10 #cut the vector back to length 10
  - Z

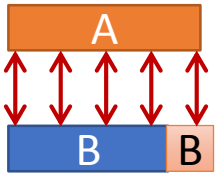
Create and Assign Named Vectors

- Vector element can be accessed by indices, and **names** as well
- **names**(a_vec) = *a_name_vector*

```
>m=1:12
>names(m) = month.name   #default vector of R
>names(m)
[1] "January"   "February"  "March"     "April"     "May"       "June"
[7] "July"      "August"    "September" "October"   "November"  "December"
> m["July"]  #access elements by name
July
  7
```

Vector of Characters

- Delimited by " " 或 ' '
- Exercises
  - **paste**("a", "b") #concatenate strings with default inserted with blank
  - **paste**("a", "b", sep=" ") #add spaces between elements
  - **paste**("a", "b", sep="") #inserted immediately without adding extra separators
  - A= c("a", "b", "c")
  - B=c("d", "e")
  - **paste**(A, B)
  - X=c("a", 'b', "\\", "\t", "\n", "\"") #special characters
  - X
  - Y=c("t","h", "i", "s")
  - Y
  - Z= **paste**(X, Y, sep="") #concatenate element by element
  - Z



auto-expanding

Matrix and Array

A data structure that consists of multi-dimensional objects of the same type!

Create a Matrix: matrix() and array()

- ***matrix***(init_data_vector, rows, cols) #two dimensional
  - A=**matrix**(0, 4, 5)
  - A

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
```

  - B=**matrix**(1:20, 4, 5) #column-majored
  - B

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

- ***array***(init_data_vector, dim=<rows_columns_vector>) #multi-dimensional
  - C=**array**(1:20, dim=c(2,5, 2))
  - C

Subsectioning a matrix

- Subsectioning
  - A[*row_index_vector*, *column_index_vector*]
  - B[c(1,4), c(2, 3)] = 1 #assign a sub-matrix to be 1's
  - A[4,1:5]
  - A[4,] #get the row 4
  - A[4,1:5, drop=FALSE] #return a result of matrix
    - A[4, ,drop=FALSE] #the same as above
  - A[4,1:5, drop=TRUE] #return a result of vector

Combine MATRICES

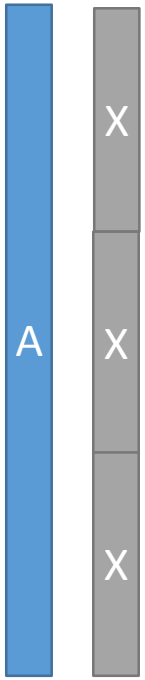
- Column bind
  - ***cbind***(A1, A2)

- Row bind
  - ***rbind***(A1, A2)

- Examples
  - cbind(A, B)
  - rbind(A, B)

Matrix operations (1)

- Transpose a matrix
  - ***t***(A)
- Matrix Arithmetic (column-majored)
  - A+1
  - x=1:4
  - A*x #element-wise operation
  - y=1:2
  - A+y
  - y=1:5
  - A+y



A最好是X長度的倍數
否則會出現警告訊息

Assign Names to Matrix

- **rownames**() and **colnames**()

```
m0 <- matrix(NA, 4, 0)    #initial a matrix
rownames(m0)    #return NULL (no names assigned)

m2 <- cbind(1, 1:4)   #create a 4 by 2 matrix
colnames(m2, do.NULL = FALSE)  #return a default col names for m2
colnames(m2) <- c("X ", " Y ") #set col names for m2
rownames(m2) <- rownames(m2, do.NULL = FALSE, prefix = “Obs.” ) #set row names for m2 with Obs.1, Obs.2, …
m2
```

LIST

A data structure that consists of ordered objects of different types!

Crete a List

- **list**(key1=obj1, key2=obj2,…)
  - A list of key-object pairs
  - 通常用來儲存程式的結果 (a named list)
- Example: create a List
  - V=1:20
  - M=**matrix**(1, 4, 5)
  - S=**c**("n1", "n2")
  - B=**matrix**(1:20, 4, 5)
  - L = **list**(e1=V, e2=M, e3=S)
  - L
- ***length***(L): list的元素個數

Access List Elements

- L**$**key
  - L**$**e1
  - L**$**e2
  - L**$**e3
- L**[[**index**]]** or L**[[key]]**
  - L[[1]]
  - L[[2]]
  - L[[3]]
  - L[["e1"]]
  - L[["e2"]]
  - L[["e3"]]
- Get and set the names of the keys of a list
  - **names**(L): return a vector of the names of the keys
  - **names**(L) = c ("o1", "o2", "o3")

It is important to note the difference between

L[1]: the first key-object pair

L[[1]]: the first object

Data Frame

A data table containing vectors of different types, arranged in a table form

Create a Data Frame

- **data.frame**(vector1, vector2,…)
  - 各欄位建議以變數命名以建立欄位名稱
- Examples
  - V=1:3
  - M=**c**("a", "b", "c")
  - S=**factor**(c("m", "m", "f")) #a factor (category) vector
  - df = **data.frame**(V,M,S)
  - df2=**data.frame**(1:3, **c**("a", "b", "c"), **factor**(**c**("m", "m", "f")), stringsAsFactors=FALSE)
- ***nrow***(D): number of row in D
- ***ncol***(D): number of column in D

Subsectioning a Data Frame

- Use matrix form: D[r, c]
  - r: row index, c: column name
- Use D**$**c #get some column
- Examples
  - df[1,2] #get object at cell[1, 2]
  - df[, "M"] #get the objects at column “M”
  - df[2,] #get row no. 2
  - df**$**M #get the objects at column “M”
  - df[df**$**V>1.0,] #use Boolean expr to filter data which has V > 1
    - Recall that df**$**V >1 returns a Boolean vector

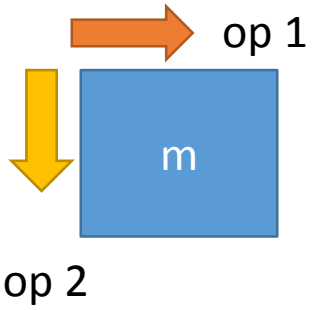
Tips: Always use **str**() to see an object structure

It’s important to get familiar with some operation patterns on data..

- **apply**()
- **lappl**y(): list apply
- **sapply**(): simple apply
- **tapply**()
- **by**()

Operational Patterns: APPLY, SAPPLY, LAPPLY

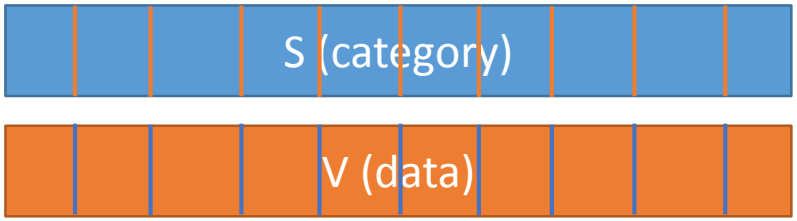
- **sapply**(*data-vector*, *function*): apply function on each vector element
  - return a named vector
  - Example
    - m = InsectSprays #a default data frame
    - sapply(1:2, function(x) mean(m[,x]))
- **lapply**(*data-vector*, *function*) : apply function on each vector element
  - return a named list
  - lapply(1:2, function(x) mean(m[,x]))
- **apply()**
  - apply(*data-frame*, 1, mean)
  - apply(*data-frame*, 2, mean)



Operational Patterns: TAPPLY

- tapply(*data-vectors*, *category-vectors*, *function*)
  - apply function on each **categorical** vector elements mapped position by position
  - return a named vector
  - **tapply**(df**$**V, df**$**S, **mean**)
    - m = InsectSprays #a default data frame
    - tapply(m$count, m$spray, mean)

```
>head(m)
  count spray
1    10     A
2     7     A
3    20     A
```




```
>tapply(m$count, m$spray, mean)
       A        B        C        D        E        F
14.500000 15.333333  2.083333  4.916667  3.500000 16.666667
```

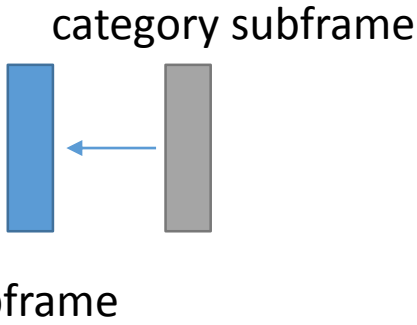
Operational Patterns: BY

- by(*data-vectors*, *category-vectors*, *function*)
  - apply function on each **categorical** vector elements mapped position by position
  - return a named list
  - Example
    - m = InsectSprays #a default data frame
    - res=by(m$count, m$spray, mean)

  returns a named list with *dim* attribute and *dimnames* attribute
  dim: the vector sizes
  dimnames: the named list of the dimname

  res[["fac-value"]]:
  e.g., res[[“0”, “1”]][[1]]

  res[[i]]



Now, let’s talk about I/O …

Read Table

- ***read.table***("filename", header=TRUE|FALSE)
  - header: 預設FALSE, 表示沒有標題列(但實際上指有列標籤)
  - header: TRUE,表示有標題列(但實際上指沒有列標籤)
- 檔案格式
  - **read.table**("datafile", header=FALSE)　read.table("datafile", header=TRUE)

有列名字和行標籤的輸入檔格式：

	ID	Name	Score	Pass
01	S001	XXXX	83	yes
02	S002	YYYY	34	no
03	S003	ZZZZ	56	no

沒有行標籤的輸入檔格式：

ID	Name	Score	Pass
S001	XXXX	83	yes
S002	YYYY	34	no
S003	ZZZZ	56	no

Read CSV file

- df4=***read.csv***("D:\\R\\samples\\datafile3.csv", head=TRUE, sep=",")
- df4

Write CSV

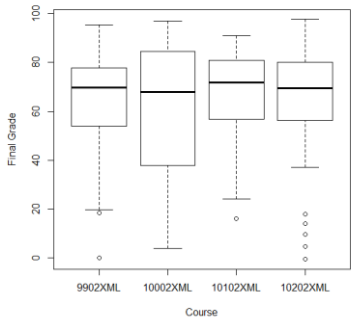
- # Write CSV in R
- **write.csv**(*my-data-frame*, file = "MyData.csv")

ON, let’s play R around…

變異數分析 :ANOVA (Analysis of variance)

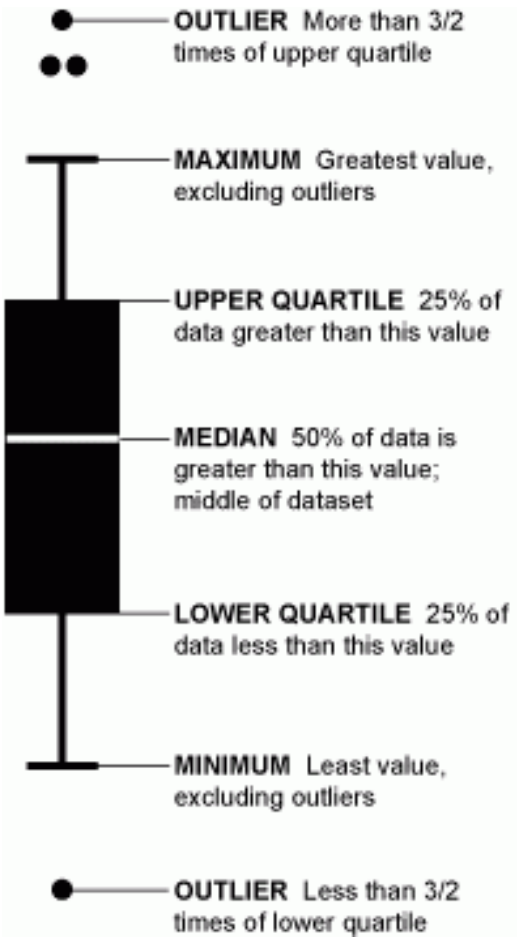
- 同質性檢定
  - levene.test()
  - bartlett.test()

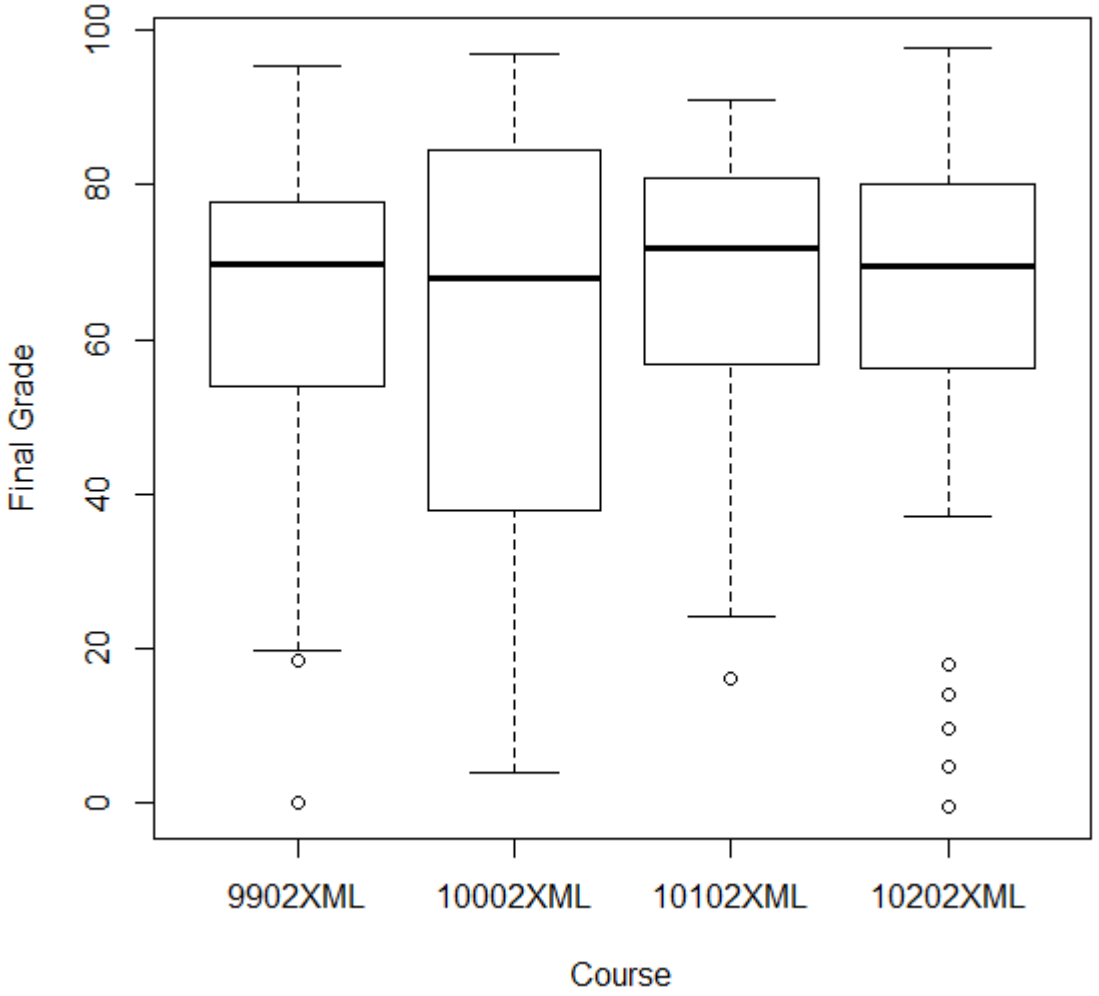
```
#First, test the variances whether equal?
library(lawstat)
levene.test(mdf$score, mdf$course)
boxplot(mdf$score ~ mdf$course)
```




```
#load train data
df1=read.csv("D:\\R\\LIPSamples\\CouseBasicAnalysis\\CoursesBasicInformation_9902XMLScore.csv", head=TRUE, sep=",")
df2=read.csv("D:\\R\\LIPSamples\\CouseBasicAnalysis\\CoursesBasicInformation_10002XMLScore.csv", head=TRUE, sep=",")
df3=read.csv("D:\\R\\LIPSamples\\CouseBasicAnalysis\\CoursesBasicInformation_10102XMLScore.csv", head=TRUE, sep=",")
df4=read.csv("D:\\R\\LIPSamples\\CouseBasicAnalysis\\CoursesBasicInformation_10202XMLScore.csv", head=TRUE, sep=",")
#just use subset of the df
myvars = c( "ID", "Name", "Score")
df1.db = data.frame(
    course = rep("9902XML", nrow(df1)),   #repeat strings for a number of times
    name = df1$Name,
    score = df1$Score
)
df2.db = data.frame(
    course = rep("10002XML", nrow(df2)),
    name = df2$Name,
    score = df2$Score
)
df3.db = data.frame(
    course = rep("10102XML", nrow(df3)),
    name = df3$Name,
    score = df3$Score
)
df4.db = data.frame(
    course = rep("10202XML", nrow(df4)),
    name = df4$Name,
    score = df4$Score
)
mdf = rbind(df1.db, df2.db, df3.db, df4.db) #merge by rows (adding rows)
```

Understanding boxplot

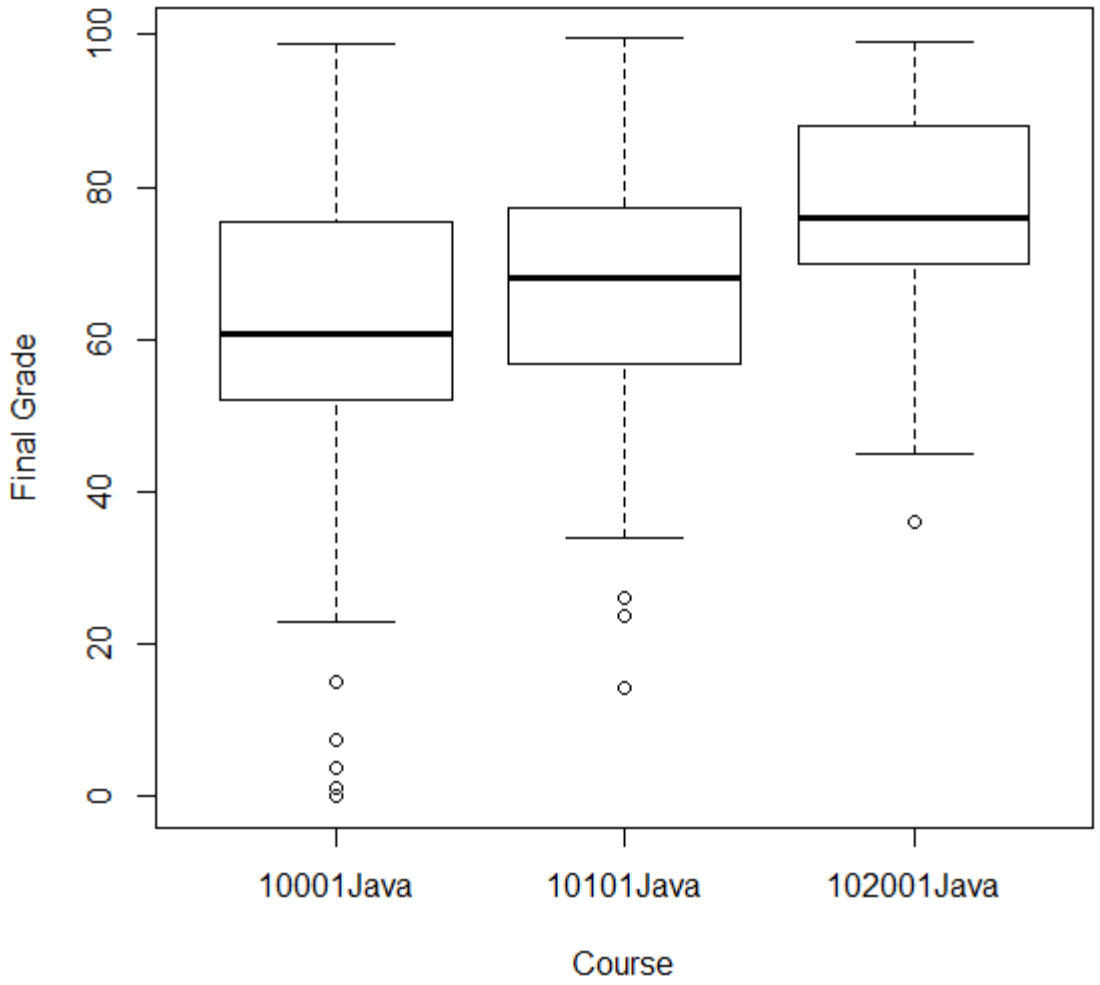




modified robust Brown-Forsythe Levene-type test based on the absolute deviations from the median

data: mdf$score
Test Statistic = 2.4197, p-value = 0.06811

We conclude that there is insufficient evidence to claim that the variances are not equal.



modified robust Brown-Forsythe Levene-type test based on the
absolute deviations from the median

data: mdf$score
Test Statistic = 2.2339, p-value = 0.1104

We conclude that there is insufficient evidence to claim that the variances are not equal.

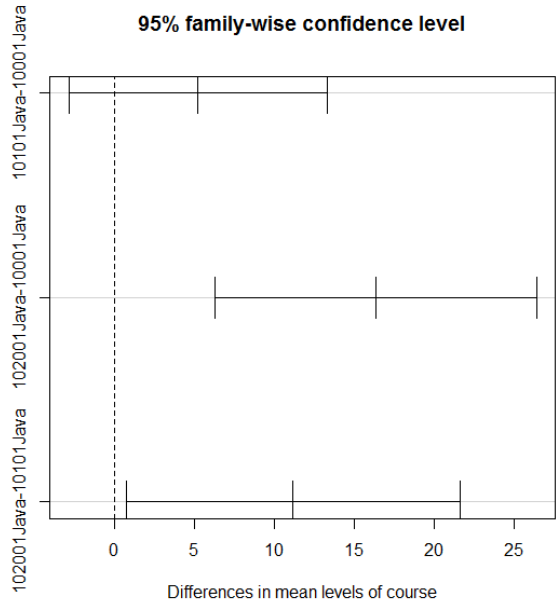
ANOVA TEST

- One-way analysis of means (not assuming equal variances with Welch's correction applied (Data output is simple)
  - **oneway.test**(mdf$score ~ mdf$course)
    - data: mdf$score and mdf$course
    - F = 10.394, num df = 2.000, denom df = 91.275, p-value = 8.575e-05
    - (For Java courses)
- Use **aov**() (Data is more complete)
  - aov.out = **aov**(score ~ course, data=mdf)
  - **summary**(aov.out)

```
Df Sum Sq Mean Sq F value   Pr(>F)
course       2   5718  2858.8   7.352 0.000879 ***
Residuals   162  62998   388.9
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Further Investigation

- If Significant
  - t = **TukeyHSD**(aov.out)
  - plot(t)



Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = score ~ course, data = mdf)

$course	diff	lwr	upr	p adj
10101Java-10001Java	5.177711	-2.9016962	13.25712	0.2861404
102001Java-10001Java	16.320271	6.2434800	26.39706	0.0005319
102001Java-10101Java	11.142560	0.7120906	21.57303	0.0331910

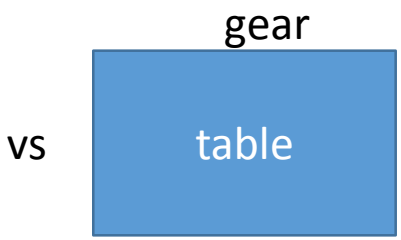
Basic Graphics

plot()

- plot() 繪製各式的散佈圖 (依據第一個參數物件的型態而定)

barplot()

```
counts <- table(mtcars$vs, mtcars$gear)  #create a table vs  w.r.t. gear
barplot(counts, main="Car Distribution by Gears and VS",
   xlab="Number of Gears", col=c("darkblue","red"),
   legend = rownames(counts), beside=TRUE)  #Aside Bar
```




```
counts <- table(mtcars$vs, mtcars$gear)
barplot(counts, main="Car Distribution by Gears and VS",
   xlab="Number of Gears", col=c("darkblue","red"),
   legend = rownames(counts), beside=FALSE) #Stacked Bar
```

Graphics Applications

- Is your data normalized?
  - Histogram is not good!
  - Use the Quantile-Quantile plot

```
qqnorm(x.norm) #creat a Quantile-Quantile plot

qqline(x.norm) #draw a line to make it easy to tell if the data is normalized

plot(density(x.norm))  #create a density plot using the density function
```

Have Fun!