R Foundation

Part I: Installation

Platform: We need R and RStudio. RStudio is the IDE that we will use for writing R codes throughout the bridging course.

How to install R?

For **Windows**, you need to download R and then RStudio following the 2 steps below. Download R-4.0.2 for Windows at https://cran.r-project.org/bin/windows/base/

Save **R-4.0.2-win.exe** and run the file as instructed.

R i386 4.0.2 and R x64 4.0.2 appear on your desktop.

For **MacOS**, download **R-4.0.2.pkg** (for Mac) at https://cran.r-project.org/bin/macosx/

- Done! -

How to install RStudio?

For **Windows**, go to https://rstudio.com/products/rstudio/download/#download

Choose DOWNLOAD RSTUDIO FOR WINDOWS. Save **RStudio-1.3.1073.exe** and run the file as instructed. RStudio appears on your desktop.

For **MacOS**, go to the above page, and scroll it down. Click **RStudio-1.3.1073.dmg** to install.

- Done! -

Remark: *If you are using Chinese window, you may need to open R by choosing **Run as administrator**.*

Part II: R Environment

It contains 4 panels: (1) R Script for your own coding, (2) Console showing the codes you run and the output, (3) Environment that stores the output of the variables you assign in the R script and (4) Miscellaneous items including the figures generated by the R script (Under Plots), available packages for advanced functions (Under Packages) and Help tag allowing you to search for functions.
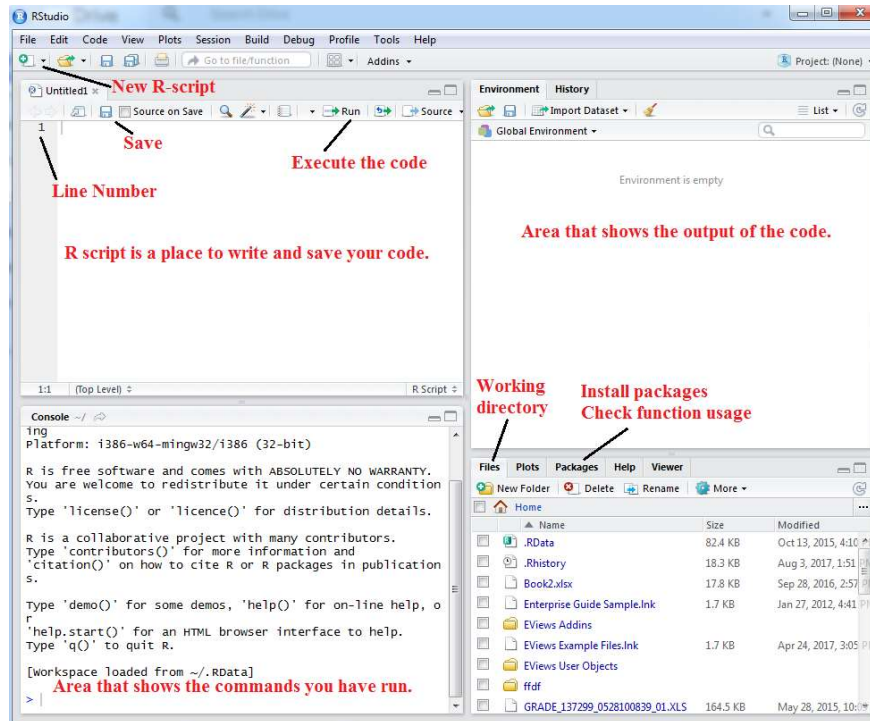
Part III: Programming

R as a Calculator

An **algorithm** is a list of instructions (logic structure of the program) that, upon correct execution, lead to a wanted result. **Variables** are made to store information. They are *case-sensitive*, and order of variables matters. Either "=" or "<-" can be used to assign value to the variable.

```
# Assignment Operator
a = 21
A <- 22
a = 10
```

Q: What is the value of a and A?

Arithmetic Operators and Logical Operators

An **operator** is a sign linking two values to produce a result.

Operator	Description	Operator	Description
+	Addition	==	Exactly equal to
-	Subtraction	!=	Not equal to
*	Multiplication	!x	Not x
/	Division	x|y	x OR y
^ or **	Exponentiation	x & y	x AND y
<	Less than	isTRUE(x)	Test if x is TRUE
<=	Less than or equal to	%*%	Matrix multiplication

Source: https://www.statmethods.net/management/operators.html

```
2+3

## [1] 5 (Appear in Console but it is not stored in the Environment)

n=2
n^5

## [1] 32 (Appear in Console and n is stored in the Environment)
```

Statistical Functions

The general form is *function.name*(*inputs*).

```
sqrt(9)
exp(4)
x1 = seq(from=1,to=10,by=1)
x2 = c(1:10)
mean(x1)
var(x2)
log(x1)
log(x1,2)
```

Converting Data Type to Fit Functions

There are many data types such as list, factor, matrix, numeric, integer, data frame. Some functions only work for a particular data type. In order to convert to the appropriate data type, we can use the function *as.datatype(data)*.

```r
is.matrix(x1) (Check data type of x1)
y = as.matrix(x1) (Change data type of x1)
apply(y,2,mean)
```

Data Type: Matrix

There are many data types such as **list, factor, matrix, numeric, integer, data frame**. Excel spreadsheets containing different columns can be treated as matrix. We can perform arithmetic operations on its rows or columns, subset the data, or draw random sample from the original data. *Standard bracket* `()` is used with functions. *Square bracket* `[,]` is used for indexing.

```r
x = matrix(c(10,20,30,40),2,2,byrow=T) (c() is a collector function)
class(x)
rowSums(x)
colSums(x)
rowMeans(x)
colMeans(x)
sample(x,10,replace=T)
nrow(x)
ncol(x)
dim(x)
x[1,]
x[,2]
```

Data Type: Character

Characters are used to represent string values. They can be put in a vector.

```r
cousins = c("David", "Rachael", "Louie", "Cindy", "Paul")
class(cousins)
cousins[1]
cousins[-5] (Adding – sign means dropping the 5th element)
cousins[c(2,4)]
cousins[c(4,2)]
```

Data Type: Factor

Factors are variables which take on a limited number of different values, and thus categorical. We can convert characters into factors in numeric form.

```
myvec = c("poor","gd","gd","avg","avg","gd", "poor", "gd","avg","gd")
class(myvec)
unique(myvec) (unique(x) lists the unique elements in x)
myvec.factor = factor(myvec)
myvec.ord1 = ordered(myvec,levels=c("poor","avg","gd"))
myvec.ord2 = ordered(myvec,levels=c("gd","avg","poor"))
```

Data Type: Data Frame

A data frame is used for storing data tables. It is a list of vectors of equal length. Each column can be of different data type.

```
df = data.frame(
  gender=c("M","F","F"),
  height=c(180,160,170),
  weight = c(75,60,62),
  age=c(40,25,10))
class(df)
is.data.frame(df)
df[,c(1,3)]
df[,c("gender", "age")]
subset(df, gender=="F")
subset(df, age >= 40 & height > 160)
subset(df, age >= 40 & height > 160, select = c("weight"))
```

Missing Values

Missing values can be of different forms: "NA", "999", or "?".

```
x   = c(13,18,13,14,13,16,14,21,13,NA)
mean(x, na.rm = TRUE) (na.rm is an option: it removes na)
mean(x, na.rm = FALSE) # What is the difference?
median(x, na.rm = TRUE)
quantile(x, na.rm = TRUE)
var(x, na.rm = TRUE)
sd(x, na.rm = TRUE)
range(x, na.rm = TRUE)
summary(x, na.rm = TRUE)
table(x)
```

```
p = c(64, 65, 68, 69, 70, 71, 72, 72, 75, 75, 80, 81, 83, 85, NA)
is.na(p)
complete.cases(p)
p[!is.na(p)]
p[complete.cases(p)]

p = c(64, 65, 68, 69, 70, 71, 72, 72, 75, 75, 80, 81, 83, 85, 999)
p[p==999] = NA
p[complete.cases(p)]

p = c(64, 65, 68, 69, 70, 71, 72, 72, 75, 75, 80, 81, 83, 85, "?")
p[p=="?"] = NA
p[complete.cases(p)]
```
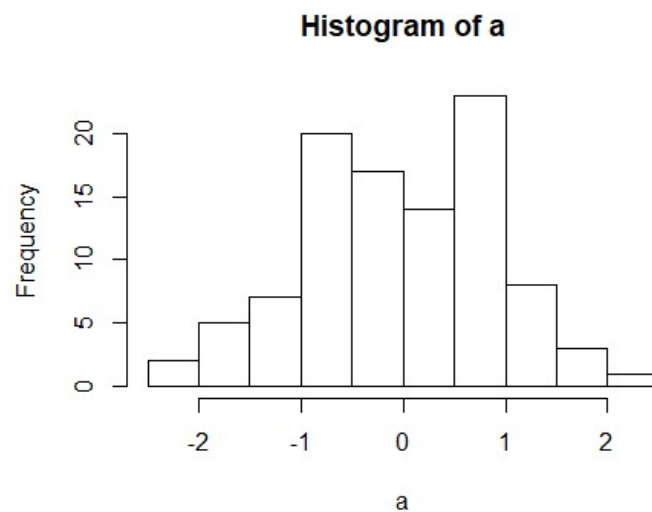
Data Visualization

R can handle simple plots as follows. There is also an advanced data visualization package in R that helps plot elegant figures. https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

```
a = rnorm(100,mean=0,sd=1)
hist(a)
```
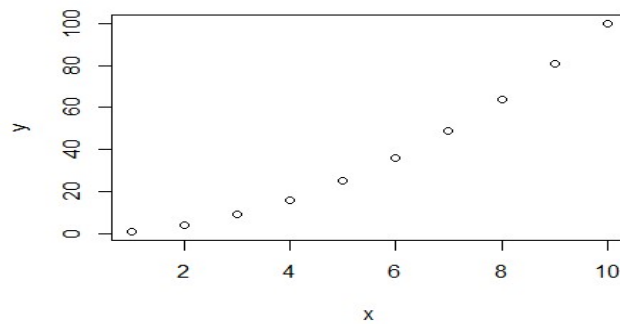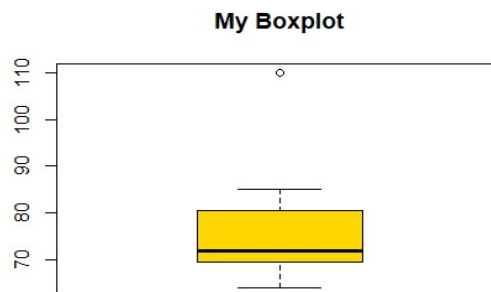
Histogram of a

```r
y = c(64, 65, 68, 69, 70, 71, 72, 72, 75, 75, 80, 81, 83, 85)
hist(y,main="Equal-width Binning",xlim=c(64,85),col="lightskyblue",brea
ks=seq(64,85,by=3))
```


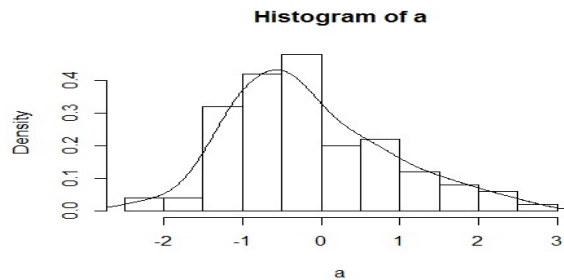Equal-width Binning

```r
x=1:10
y=x^2
plot(x,y)
```
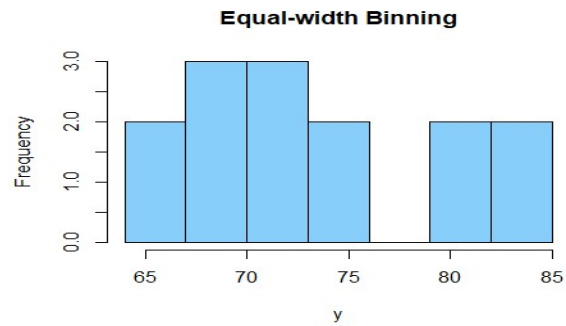


```r
z = c(64, 65, 68, 69, 70, 71, 72, 72, 75, 75, 80, 81, 83, 85, 110)
boxplot(z,main="My Boxplot", col="gold")
```


My Boxplot

Random Sampling

```r
set.seed(1234)
a = rnorm(100,0,1)
hist(a,prob=TRUE)
lines(density(a))
```


Histogram of a

Linear Regression

```r
x = rnorm(100,0,1)
a = runif(1,0,1)
b = runif(1,-2,2)
u = rnorm(100,0,1)
y = a+b*x+u


out = lm(y~x)
out

result = summary(out)
result$coefficients
```

Using Libraries/ Packages

Requesting pre-installed data in the package

There are some datasets pre-installed in the package *library(package.name)* for demonstrating the use of the functions in the package. To use the variables in the data, use *data.set$variable.name*. Linear regression line $y = a + bx$ can be obtained using `lm(y ~ x)` and added to the figure using `abline`.

```r
library(car)
data(Duncan)
colnames(Duncan)

# More plots for multivariate
plot(Duncan$income, Duncan$education)
abline(lm(Duncan$income ~ Duncan$education))
```
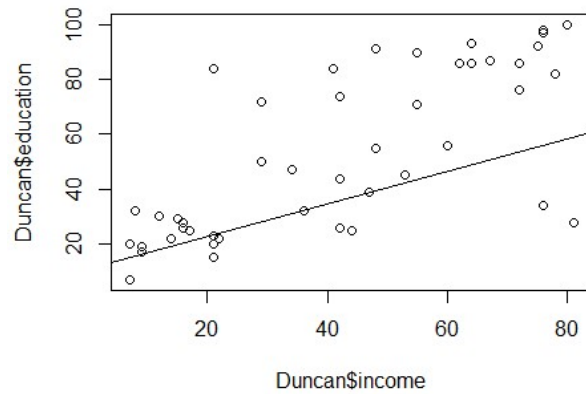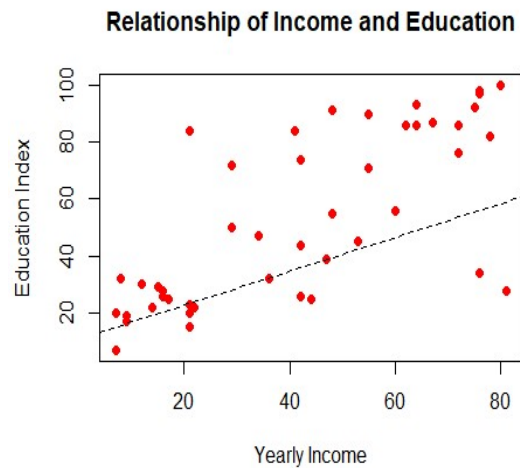
```
plot(Duncan$income, Duncan$education,
     main="Relationship of Income and Education",
     ylab="Education Index", xlab="Yearly Income",
     pch=16, col="red")

abline(lm(Duncan$income~Duncan$education), lty=2)
```


Setting Working Directory

```
rm(list=ls())
setwd("G:/My Drive/Teaching/Bridging/R/") (For Windows)
setwd("/Volumes/GoogleDrive/My Drive/Teaching/Bridging/R") (For Mac)
```

Exporting Data

The data stored in the named variables can be exported to excel. If a working directory is specified before, then the files will be automatically stored there. Otherwise, a route can be specified in `file="your route/file.name.csv"`.

```r
x = rnorm(100,mean=10,sd = 5)
y = rep(1,100)
data = cbind(x,y)
write.csv(data,file="mydata.csv")
```

Importing Data

The icon ![Import Dataset] in the Environment panel can be used to import the data files. Usually, different data formats require different packages to open it. If you do not want to use `$` sign to call a variable in a data set each time, you can use the function `attach(dataset1)`. When you want to use another set which may share common variable names, a good practice is to use `detach(dataset1)` before starting to use another data set.

```r
library(foreign)
grel = read.spss("G:/My Drive/Teaching/Big_Data_Analytics/Data/NILT2012 GR.sav", to.data.frame = TRUE) (For Windows)

debt = read.dta("/Volumes/GoogleDrive/My Drive/Teaching/Bridging/R/debt ratio.dta") (For Mac)


attach(grel)
table(nkids)
detach(grel)

library(foreign)
debt = read.dta("G:/My Drive/Teaching/Big_Data_Analytics/Data/debtratio. dta") (For Windows)
debt = read.dta("/Volumes/GoogleDrive/My Drive/Teaching/Bridging/R/debt ratio.dta") (For Mac)


library(readr)
Wholesale = read_csv("G:/My Drive/Teaching/Bridging/R/Wholesale custome rs data.csv") (For Windows)
Wholesale = read_csv("/Volumes/GoogleDrive/My Drive/Teaching/Bridging/R /Wholesale customers data.csv") (For Mac)
```

```
library(readxl)
Predictor = read_excel("G:/My Drive/Teaching/Bridging/R/PredictorData20
14.xlsx", sheet = "Quarterly") (For Windows)
Predictor = read_excel("/Volumes/GoogleDrive/My Drive/Teaching/Bridging
/R/PredictorData2014.xlsx", sheet = "Quarterly") (For Mac)

library(RWeka) (For Windows)
churn = read.arff("G:/My Drive/Teaching/Bridging/R/churn.arff") (For Wi
ndows)

library(foreign) (For Mac)
churn = read.arff(file="/Volumes/GoogleDrive/My Drive/Teaching/Bridging
/R/churn.arff") (For Mac)
```

In general, if you set the working directory at the beginning, you do not need to type the route to the data file.

If-Else Function

Conditional statement is often used. The syntax of if…else statement is:

if (test_expression) {

    statement

}

More levels can be added.

if (test_expression1) {

    statement1

    } else if (test_expression2) {

    statement2

} else {

statement 3

}

```r
balance = 115
employ = "NO"
age = 40

if ((balance>=50) & (employ=="NO") & (age>=45))
{decision = "Write-off"} else {decision = "Not write-off"}

a = 3
ifelse(a>=2,"TRUE","FALSE")
```

For Loop

A loop is a way to repeat a sequence of instructions. Its syntax is:

for (variable in sequence) {

expression

}

If your expression involves storing values in a vector or a matrix, you need to initialize the vector or matrix before starting the **for** loop. The initialization values can be 0 or NA.

```r
score = matrix(0,nrow=5,ncol=1)
math = c(75,83,78,65,91)
eng  = c(85,89,62,78,95)
subjects = cbind(math,eng)
for (i in 1:5){
  score[i] = mean(subjects[i,])
}
```