Programming with R

Reminder: Group Information Submission

Each group consists of at most 6 students (and at least 1).

You need to choose a name for your group, e.g., "Marketers", "Fantastic", "A Plus"…

Submit your group form on Moodle before Sep 16 (Class 1A) or Sep 20 (Class 1B) --- deadline extended.

Let the TA Yana Lo, (yanalo@hku.hk) know if you cannot find a group.

What is R?

R is a *programming language*. It is *not* a statistics program like SPSS, SAS, JMP or Minitab, and doesn't wish to be one. The official R Project describes R as "a language and environment for statistical computing and graphics."

Why R?

- R offers the largest and most diverse set of analytic tools and statistical methods.
- There is a community.
- Compared with other programming languages like Python, R is relatively easier to learn, especially for beginners.
- R is free.

Let's Download R.

Your installation path must not contain any non-English characters. Otherwise, you will have troubles using it.

安装路径必须为纯英文，否则运行可能出错。

Let's Download RStudio.

Your installation path must not contain any non-English characters. Otherwise, you will have troubles using it.

安装路径必须为纯英文，否则运行可能出错。

Data Structures

Let's start with a simple dataset (in R, a vector), consisting of three numbers, 1, 2 and 4.

```
1 x <- c(1, 2, 4)
2 x
```

Here, $\leftarrow$ is the assignment operator. It means we are assigning values on the right-hand side to x. You can also use $=$ as the assignment operator. c stands for concatenate. Here, we are concatenating the numbers 1, 2, and 4 to create a vector.

If you want to select the third element of x, try the followings:

```
1 x <- c(1, 2, 4)
2 x[3]
```

If you want to select the second to the third element of x, try the followings:

```
1 x <- c(1, 2, 4)
2 x[2:3]
```

Scalars

Scalars, or individual numbers, do not really exist in R. As mentioned earlier, what appear to be individual numbers are actually one-element vectors.

```
1 x <- 8
2 x
```

Here, x is a vector that contains a single element, which is 8.

Scalars

You can perform numerical operations on scalars. See the following examples.

```
x <- 100
y <- 50.5
z1 <- x + y
z2 <- x - y
z3 <- x*y
z4 <- x/y
z5 <- y^2
z6 <- sqrt(x)
print(c(z1,z2,z3,z4,z5,z6))
```

Character Strings

Character strings are actually single-element vectors of mode character.

```
1 x <- "abc"
2 y <- c("abc", "29")
3 x
4 y
```

Here, x is a vector that contains a single string, whereas y is a vector containing two strings. Here, the quoted 29 is a string, not a number.

String Operations

There are also operations defined on strings. See the following example:

```
1 x <- "Welcome"
2 y <- "to"
3 z <- "Marketing"
4 result <- paste(x, y, z)
5 result0 <- paste0(x, y, z)
6 print(c(result, result0))
```

Data Frame

Data Frame

Data frame is the most important data structure in R. It is similar to a table which contains rows and columns. Let us consider the following table:

Name	Salary	Job
Alice	20,000	IT
Bob	19,000	Sales
Carol	23,000	Finance
Denis	22,000	IT

To create this data frame, we can adopt the following code:

```
employees <- data.frame(
  name = c('Alice', 'Bob', 'Carol', 'Denis'),
  salary = c(20000,19000,23000,22000),
  job = c('IT', 'Sales', 'Finance', 'IT' ))
```

When you want to choose a specific column of the data frame, try the following code:

```
employees$job
```

When there are missing values in your data frame, you can use NA to represent them.

```
employees <- data.frame(
  name = c('Alice', 'Bob', 'Carol','Denis'),
  salary = c(20000,NA,23000,22000),
  job = c('IT', 'Sales', NA, 'IT'))
```

Simple Statistics

You can easily obtain simple statistics of your data, such as mean, median, variance:

```
vector <- c(0, 8, 4, 6, 7, 9, 5)
length(vector)
mean(vector)
median(vector)
var(vector)   # Variance
sd(vector)    # Standard Deviation
max(vector)   # Maximum
min(vector)   # Minimum
sort(vector)  # Sort, in increasing order
```

stands for comments: The part after # is note for programmers and is not processed by R.

Seeking Help

If you forget the meaning of a function, you can ask R for help. Just type help(name). See the following examples.

```
help(length)
help(paste)
help(data.frame)
```

The shortcut for help() is a question mark (?)

```
?length
?paste0
?data.frame
```

Another way to seek help is to use the example() function. For instance, if you do not know how to use the paste function, try the following:

```
example(paste)
```

You can also use help.search() function to perform a Google style search on R:

```
help.search("scattered plot")
```

Packages

An R package is a collection of R functions, data, and documentation that is bundled together for a specific purpose or to provide a specific set of functionalities.

For instance, the ggplot2 package is used for visualization while the stringi package is used for string analysis.

Suppose that you want to install the package “ggplot2”, you can enter the following in your R:

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

And R will download the package from the default online platform. Sometimes, the default platform is down, and you want to download from another platform. In this case, specify the URL of your platform directly:

```
1 install.packages("stargazer",
2                  repos = "http://cran.us.r-project.org")
```

If your path contains non-English letters or characters, you may get an error message when installing the package. Here are a few solutions:

- Specify a different path for your package. You can refer to the solution here.
- Use the cloud version RStudio here.
- Uninstall and reinstall your R/RStudio and specify a path that only contains English letters.

When a package is successfully installed, you will see the following message in your R console: "The downloaded source packages are in..."

Each package only needs to be installed once on a computer (as long as you do not remove it from your directory). When you use the package, just inform R by stating:

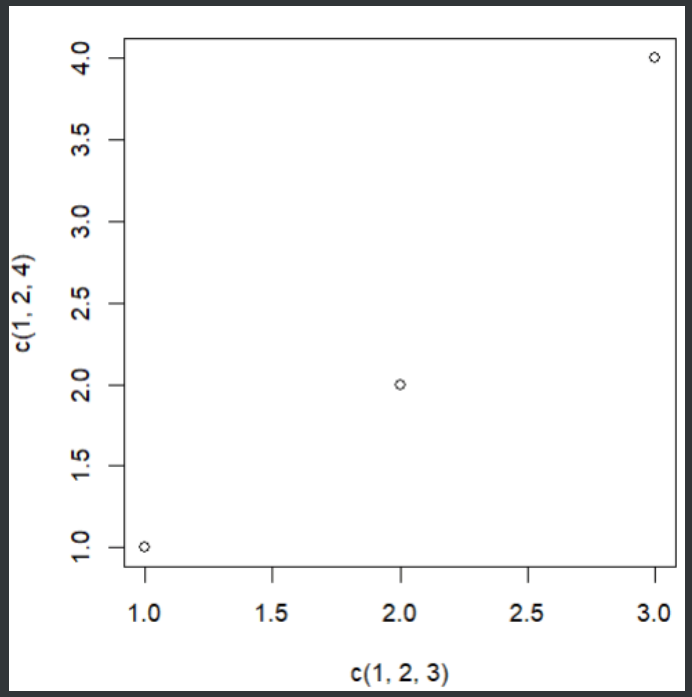
```
1 library(ggplot2)
```

Graphics

We begin with the most simple graphics with the plot() function.

```
1 plot(c(1,2,3), c(1,2,4))
```

You will get three points: $(1, 1), (2, 2), (3, 4)$.



You can use function ablines() to add lines to your graphics. Consider the following code:

```
1 x <- c(1,2,3)
2 y <- c(1,3,8)
3 plot(x, y)
4 lmout <- lm(y~x)
5 abline(lmout)
```

Here, we first show the scatter plots of x and y. Then, we use the regression function lm(y~x), which will be covered later, to generate a regression line, and add the fitted regression line to the previous figure.

When creating your plot, you can also specify the type, lty, and pch of your plots.

- The type parameter determines the type of plot to be created (e.g., points only, line only, or both).
- The lty (line type) parameter sets the line type for lines plotted in the graph (e.g., solid or dashed line).
- The pch (plotting character) parameter determines the type of symbol or shape used to represent data points in a scatter plot (e.g., circle, diamond, or triangle).

```r
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
y <- c(9, 7, 7, 4.5, 6, 7.8, 7, 3, 6, 2, 4)
plot(x, y)
plot(x, y, type = "b")
plot(x, y, pch = 17)
plot(x, y, pch = 2, lty = 2, type = "b")
```

Click here for the meaning of type.

Click here for the meaning of pch.

Click here for the meaning of lty.

We consider next the visualization of a data frame. We first install and use the library ggplot2, and then create a data frame containing two columns, x, y.

```
1 library(ggplot2)
2 x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
3 y <- c(9, 7, 7, 4.5, 6, 7.8, 7, 3, 6, 2, 4)
4 data <- data.frame(x, y)
5 ggplot(data, aes(x, y)) + geom_point()
```

We next add gender information to the data frame and use colors to represent different gender.

```r
library(ggplot2)

data <- data.frame(
  x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11),
  y <- c(9, 7, 7, 4.5, 6, 7.8, 7, 3, 6, 2, 4),
  gender = c("M", "F", "F", "M", "M", "F", "F", "M", "F", "F", "M")
)

# Specify colors for each gender
colors <- c("red", "blue")

ggplot(data = data, aes(x, y, color = gender)) +
  geom_point() + scale_color_manual(values = colors)
```

Logical Operatorations

In R, we use “if-else” logic to construct logical operations. See the following example used for detecting the sign of a number.

```r
x <- 0
if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else
  print("Zero")
```

For Loop

Suppose that you want have a vector and you want to print all elements in the vector. You can do the followings.

```
1 vector = c(1, 3, 5, 7)
2 print(vector)
```

This prints all elements together. If, however, you want to print each element one by one, what should you do?

We can implement the following code to print each element one by one.

```
1 vector = c(1, 3, 5, 7)
2 for (item in vector)
3   print(item)
```

Here, item points to each element in your vector one by one, and then we print the value of item.

You can also add more operations in your for loop!

```
1 vector = c(1, 3, 5, 7)
2 for (item in vector){
3   item = item^2 + 1
4   print(item)}
```

Input/Output

File I/O

Previously, we specify all our data in the RStudio directly. This is inconvenient when you have a very large table: You don't want to type millions of data points one by one. Also, when we get the result, we simply print it on the screen, which is also inefficient when you have large output. Next, we see how we can read and write files to your local computer.

File I/O

When inputting/outputting data from your disk, you must specify the folder that contains your file or the place your files should be saved. There is a default folder used if you don't specify it. Here is my default folder (on my Windows System):

```
1 getwd()
```

File I/O

You can also choose a different default folder (but make sure this folder exists on your disk):

```
1 setwd('C:/Users/Li Xi/Dropbox/Marketing Classes/Algorithm')
2 getwd()
```

Writing to a text document:

You need to create a file first and use the writeline function to write to the document:

```
1 file1<-file("output.txt")
2 writeLines(c("Big","Data"), file1)
3 close(file1)
```

Writing to a text document:

You can also use sink function to create a file and use cat function to write multiple lines to the document.

```
1 sink("output.txt")  # Creating a document
2 for (i in 1:100){
3 cat(toString(i))    # Converting number into String
4 cat('\n')           # Creating a new line
5 }
6 sink()              # Closing the document
```

In R, the "\n" sequence represents a newline character (in Chinese, “回车键”).

Reading a text document

We can either read a text file from your local folder or from the Internet. Let's see how to read from the Internet first.

```
1 file <- readLines("https://ximarketing.github.io/data/input.txt")
2 print(file)
```

Reading a text document

If the document is in your default folder, you can do the followings to read it.

```
1 file <- readLines("input.txt")
2 print(file)
```

If, however, there is no such a document in your default folder, you will simply receive an error message.

Writing to a CSV file

It is more convenient to write your table to a CSV file (i.e., excel spreadsheet). Here is how to create and write to a CSV file.

```r
data <- data.frame(
  Name = c("John", "Jane", "Michael"),
  Age = c(25, 30, 35),
  City = c("New York", "London", "Paris")
)
write.csv(data, file = "myfile.csv", row.names = FALSE)
```

Reading a CSV file

We can also read data from an existing CSV file, either from the Internet or your local folder. Here is how to read from the Internet:

```
1 url = "https://ximarketing.github.io/class/teachingfiles/r-exercise.csv"
2 mydata <- read.csv(url)
```

The new variable, mydata, is a data frame.

Analyzing a Data Frame

Data frame is the most commonly used data structure in R. As mentioned earlier, a data frame can be viewed as a table with rows and columns. We continue to import the previous CSV file from the Internet and perform data operations on it.

```
1 url = "https://ximarketing.github.io/class/teachingfiles/r-exercise.csv"
2 mydata <- read.csv(url)
```

You can use the head function to show the first a few rows of the dataset.

```
1 head(mydata)              # show first 5 rows
2 head(mydata, n = 10)      # show first n = 10 rows
```

You can use the nrow and ncol functions to show the number of rows and columns in the data frame.

```
1 nrow(mydata)
2 ncol(mydata)
```

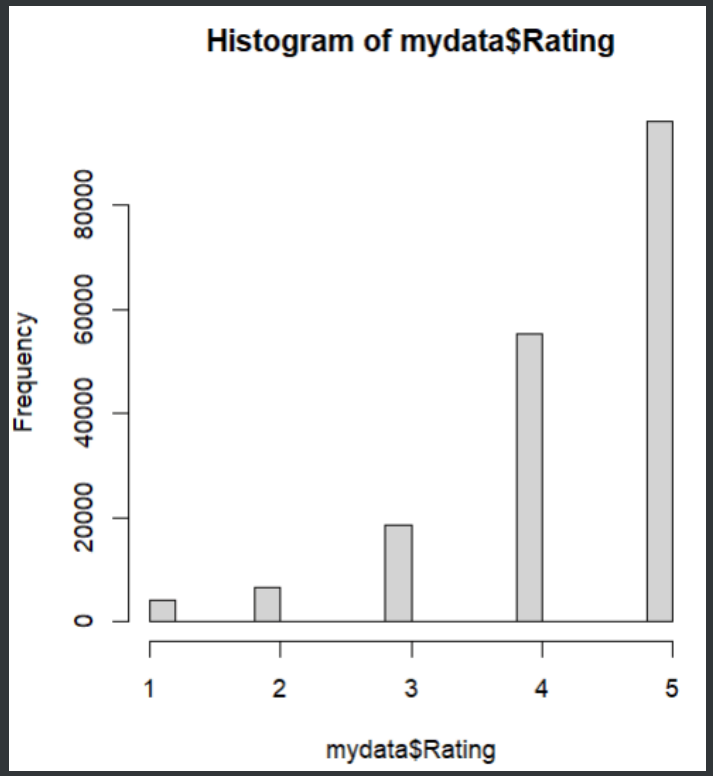
You can use the summary function to obtain the summary statistics of your dataset.

```
1 summary(mydata)
```

```
     Rating        Expertise         Votes             Purpose
Min.   :1.000   Min.   :0.000   Min.   : 0.0000   Length:180635
1st Qu.:4.000   1st Qu.:1.000   1st Qu.: 0.0000   Class :character
Median :5.000   Median :3.000   Median : 0.0000   Mode  :character
Mean   :4.286   Mean   :2.892   Mean   : 0.8217
3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.: 1.0000
Max.   :5.000   Max.   :6.000   Max.   :75.0000
```

You can use the hist function to plot the histogram of a variable (i.e., column) of your data frame.

```
1 hist(mydata$Rating)
```



You can use the subset function to choose a subset of your original data frame. For example, let's choose the rows with review ratings equal to or smaller than 4.

```
1 subdata <- subset(mydata, Rating <= 4)
2 head(subdata)
```

Linear Regression

Linear regression is arguably the most basic and widely used type of data analysis. Image that you want to figure out how one's rating is affected by his or her experience, you can run the following regression:

$$\text{Rating}_i = \alpha + \beta \cdot \text{Experience}_i + e_i,$$

where α is the intercept (constant term), β is the coefficient for experience, i is the individual index and e_i is an error term.

To run the regression, consider the following code:

```
1 result <- lm(Rating ~ Expertise, data = mydata)
2 summary(result)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.332610   0.003872 1118.98   <2e-16 ***
Expertise   -0.016138   0.001091  -14.79   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

$$\text{Rating}_i = 4.33 - 0.016 \cdot \text{Experience}_i + e_i.$$

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.332610   0.003872 1118.98   <2e-16 ***
Expertise   -0.016138   0.001091  -14.79   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The significance (i.e., p-value) of the coefficient for experience is $p < 2 \times 10^{-16}$. Typically, when $p < 5\%$, we say the coefficient is significant. Here, because $p \ll 5\%$, we can state that experience significantly affects your review rating.

To make predictions based on your regression result, try the followings:

```
1 prediction <- predict(result, data.frame(Expertise
  = 4, Rating = 2))
2 prediction
```

Let's now move from simple regression to multiple regression, i.e., a regression with more than one independent variables on the right-hand side.

```
1 result <- lm(Votes ~ Expertise + Rating +
  factor(Purpose), data = mydata)
2 summary(result)
```

Here, Purpose is a string, not a variable, and factor(Purpose) means we are treating Purpose as a fixed effect. What is a fixed effect?

What is a fixed effect?

Instead of treating purpose as one variable, we create multiple variables from it, and include them in our regression. For example, one of the purpose is family. Then, we can create a new variable family, which is defined as follows.

$$\text{family} = \begin{cases} 1 & \text{if purpose is family} \\ 0 & \text{otherwise} \end{cases}$$

```
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               1.405531   0.020426  68.812  < 2e-16 ***
Expertise                 0.004507   0.001960   2.299   0.0215 *
Rating                   -0.180778   0.004238 -42.660  < 2e-16 ***
factor(Purpose)couple     0.176869   0.011199  15.794  < 2e-16 ***
factor(Purpose)family     0.091479   0.012095   7.564 3.94e-14 ***
factor(Purpose)friend     0.065447   0.016380   3.996 6.46e-05 ***
factor(Purpose)solo       0.081461   0.018771   4.340 1.43e-05 ***
factor(Purpose)Unknown    1.130019   0.016878  66.954  < 2e-16 ***
```

Here, we take business as the benchmark and compare other purposes against it. What type of purpose generates most votes?

To make predictions, try:

```
1 prediction <- predict(result, data.frame(Expertise
  = 4, Rating = 2, Purpose = "family"))
2 print(prediction)
```

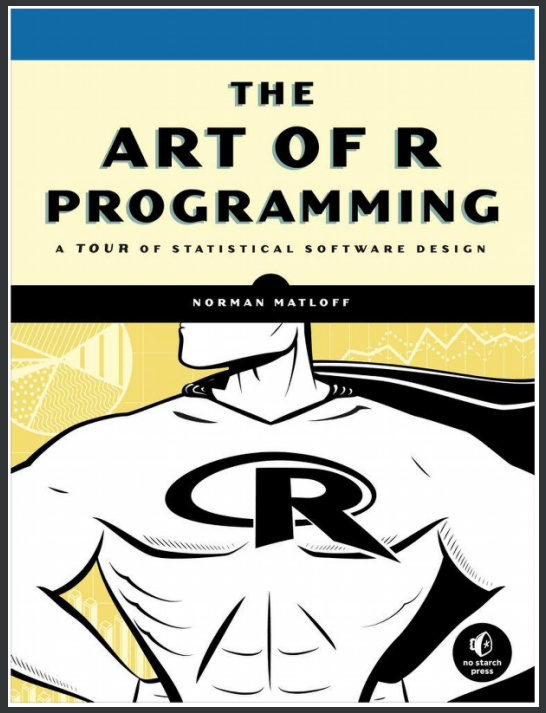
We can use the stargazer package to help organize the output of the above analysis.

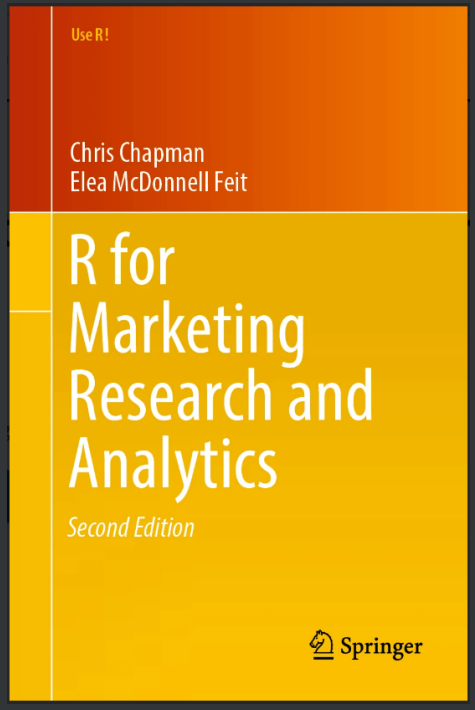
```
1 library(stargazer)
2 result <- lm(Votes ~ Expertise + Rating + factor(Purpose), data
  = mydata)
3 summary(result)
4 stargazer(result, title = "regression output", align = TRUE,
  out = "regression.html", type = "html")
```

We can also put multiple regression results together.

```
result0 <- lm(Votes ~ Expertise, data = mydata)
result1 <- lm(Votes ~ Expertise + Rating, data = mydata)
result2 <- lm(Votes ~ Expertise + Rating + factor(Purpose),
data = mydata)
stargazer(result0, result1, result2, title = "regression
output", align = TRUE, out = "regression.html", type = "html")
```

Optional Reading:





Reminder: Group Information Submission

Each group consists of at most 6 students (and at least 1).

You need to choose a name for your group, e.g., "Marketers", "Fantastic", "A Plus"…

Submit your group form on Moodle before Sep 16 (Class 1A) or Sep 20 (Class 1B) --- deadline extended.

Let the TA Yana Lo, (yanalo@hku.hk) know if you cannot find a group.

Thank you!