

A Brief Intro to R

(with Copilot)

Reminder: Group Information Submission

Each group consists of up to 8 students (and at least 1).

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

Submit your group form to Moodle on or before Dec 13, 23:59. Group form can be found on Moodle.

Let your TA (kevinzyx@hku.hk) know if you cannot find a group, or if you are looking for more group members.

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.

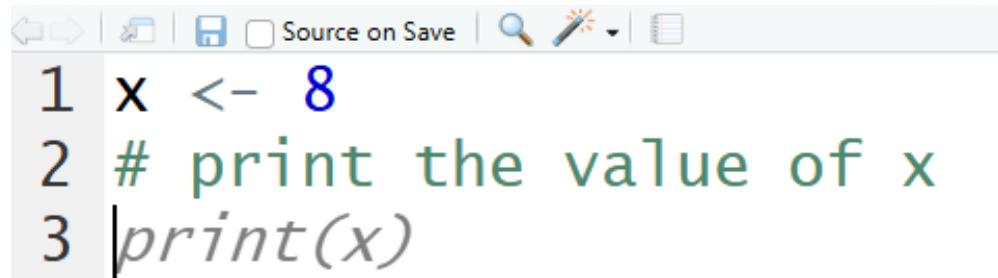
Your installation path must not contain any non-English characters. Otherwise, you will have troubles using it.

安装路径必须为纯英文，否则运行可能出错。

Numbers

It is easy to create and use numbers in R. In R, `←` means “assigning values”. Equivalently, you can write $x = 8$. You can always ask Copilot (AI) for help!

```
● ● ●  
x <- 8  
print(x)
```



A screenshot of a code editor window. The title bar includes icons for back, forward, file, and search, followed by "Source on Save". The main area contains three lines of R code:

```
1 x <- 8  
2 # print the value of x  
3 |print(x)
```

Vectors

Let us create a vector consisting of three numbers. Here, *c* stands for concatenate: we are concatenating the numbers 1, 2, and 4 to create a vector.

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

A screenshot of the RStudio interface showing a code editor and a console window. The code editor contains the R command `x <- c(1, 2, 4)`. The console window below it displays the output: `# x is a vector consisting of 1, 2, 4` followed by the vector `[1] 1 2 4`.

```
1 # x is a vector consisting of 1, 2, 4
2 [1] 1 2 4
```

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

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

```
1 # x is a vector consisting of 1, 2, 4  
2 x <- c(1, 2, 4)  
3 # choose the third element of x  
4 |x[3]
```

Numbers

You can perform numerical operations on numbers. 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

```
x <- "abc"  
y <- c("abc", "29")  
print(x)  
print(y)
```

x is a string containing three characters, a, b, and c. In *y*, the quoted 29 is a string, not a number.

String Operations

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

```
x <- "Welcome"
y <- "to"
z <- "Marketing"
result <- paste(x, y, z)
result0 <- paste0(x, y, z)
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.

```
1 vector <- c(0, 8, 4, 6, 7, 9, 5)
2 # what is the second largest element of vector?
3 second_largest <- sort(unique(vector), decreasing = TRUE)[2]
```

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:

```
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:

```
install.packages("stargazer",
                 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:



```
library(ggplot2)
```

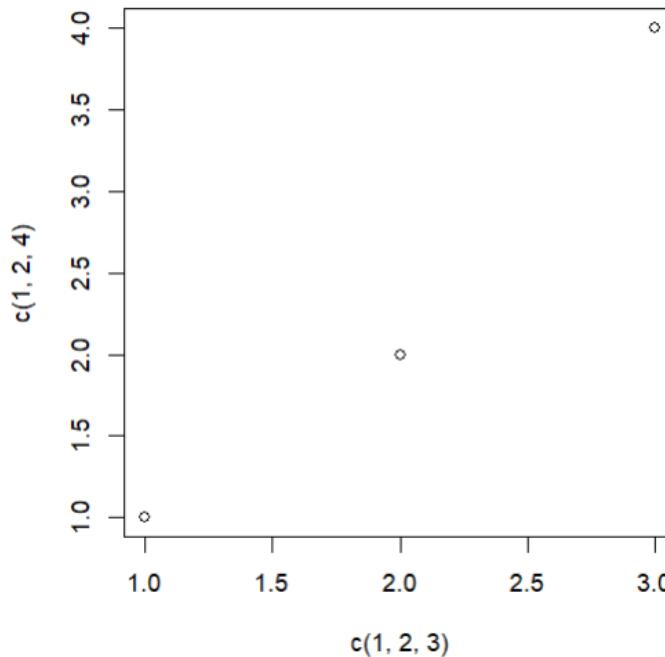
Graphics

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



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

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



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

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

Here, we first show the scatter plots of x and y . Then, we use the regression function $\text{lm}(y \sim x)$, 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).

```
● ● ●  
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 .

```
● ● ●  
library(ggplot2)  
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)  
data <- data.frame(x, y)  
ggplot(data, aes(x, y)) + geom_point()
```

I want to use red to represent female and blue to represent male and plot the points. Try to get help from AI.



```
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")
)
```

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



```
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)
```

For Loop

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

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

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

```
vector = c(1, 3, 5, 7)
# print the elements of vector one by one
for (i in vector) {
  print(i)
}
```



```
vector = c(1, 3, 5, 7)
for (i in vector) {
  print(i)
}
```

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

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):

A screenshot of a dark-themed file explorer window. At the top left, there are three small colored dots: red, yellow, and green. Below them, the text "getwd()" is displayed in white, indicating the current working directory.

File I/O

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

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

On Mac, my path looks like this:

```
● ● ●  
setwd('/Users/xili/Library/CloudStorage/Dropbox/Classes/Digital M')  
getwd()
```

To find the folder path in macOS, you can use several methods:

1. Using Finder:

- Open Finder and navigate to the folder you want.
- Right-click (or Control-click) on the folder.
- Hold down the Option key, and you'll see "Copy 'folder_name' as Pathname". Click it to copy the path to your clipboard.

To find a specific folder in macOS using its path, you can use the following methods:

1. Using Finder:

- Open Finder.
- In the menu bar at the top, click on **Go > Go to Folder...** (or press **Command + Shift + G**).
- In the dialog box that appears, paste the folder path you know and click **Go**. This will take you directly to the folder.

Writing to a text document:

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

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

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.



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

Reading a text document

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

```
● ● ●  
file <- readLines("input.txt")  
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.

```
● ● ●  
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:



```
url = "https://ximarketing.github.io/class/teachingfiles/r-exercise.csv"  
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.



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

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

```
● ● ●  
head(mydata)          # show first 5 rows  
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.

```
● ● ●  
nrow(mydata)  
ncol(mydata)
```

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

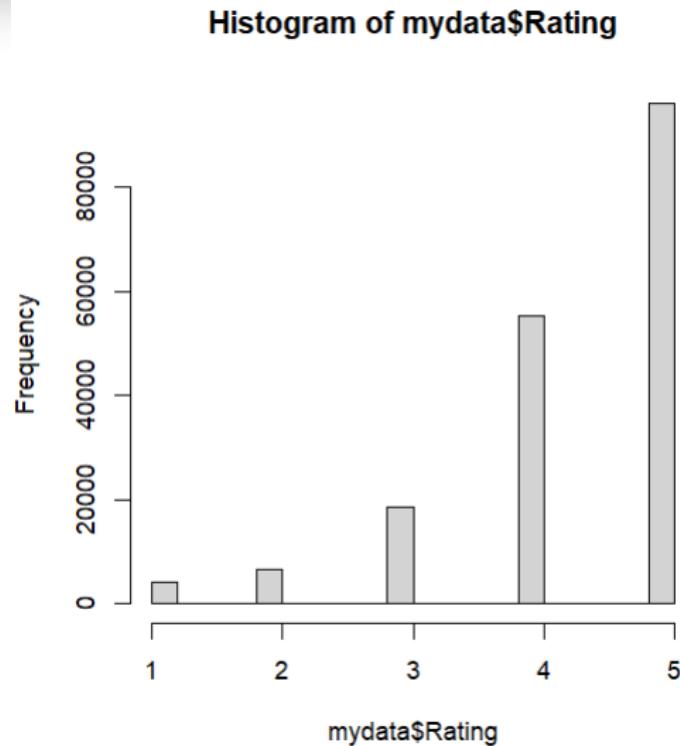
```
● ● ●  
summary(mydata)
```

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.



```
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.

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

Linear Regression

Linear regression is arguably the most basic and widely used type of data analysis. Imagine 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:

```
result <- lm(Rating ~ Expertise, data = mydata)
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.**

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.



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

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



```
library(stargazer)
result <- lm(Votes ~ Expertise + Rating + factor(Purpose), data
= mydata)
summary(result)
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")
```

Chatting with AI:

Rgress votes on rating, expertise, and purpose (purpose is fixed effects, business as benchmark). Show me the result and explain it.



r-exercise.csv
电子表格

Variable	Coefficient	Interpretation
Rating	-0.181*	Higher ratings are associated with <i>fewer</i> votes. Strong negative effect, highly significant.
Expertise	0.0045*	More expertise slightly increases votes, but the effect is tiny.
Purpose: couple	0.177*	Reviews written for "couple" purposes receive about 0.18 more votes than "business", on average.
Purpose: family	0.092*	Family-purpose reviews get ~0.09 more votes relative to business.
Purpose: friend	0.065*	Friend-purpose reviews get ~0.07 more votes relative to business.
Purpose: solo	0.082*	Solo-purpose reviews get ~0.08 more votes relative to business.

Reminder: Group Information Submission

Each group consists of up to 8 students (and at least 1).

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

Submit your group form to Moodle on or before Dec 13, 23:59. Group form can be found on Moodle.

Let your TA (kevinzyx@hku.hk) know if you cannot find a group, or if you are looking for more group members.

Thank you!