Marketing Analytics

Wei Miao

2021-09-09

Contents

Chapter 1

Preface

This is the online supplement to the MSIN0094 Marketing Analytics Module for the MSc Business Analytics program at UCL School of Management.

Chapter 2

Syllabus: MSIN0094 Marketing Analytics

(Last update on 09 September, 2021)

2.1 Instructor Information

Module Leader:

- Wei Miao[1] (wei.miao@ucl.ac.uk)

Teaching Assistants:

- Christopher Ogbunuzor (christopher.ogbunuzor.14@ucl.ac.uk)
- Henry Bellhouse (henry.bellhouse.18@ucl.ac.uk)

Office:

- S3, Level 38, One Canada Square

Office hours:

- Appointment link
- Please refer to the office hour session for how to make office hour appointments. More details will be confirmed in Week 1's class.

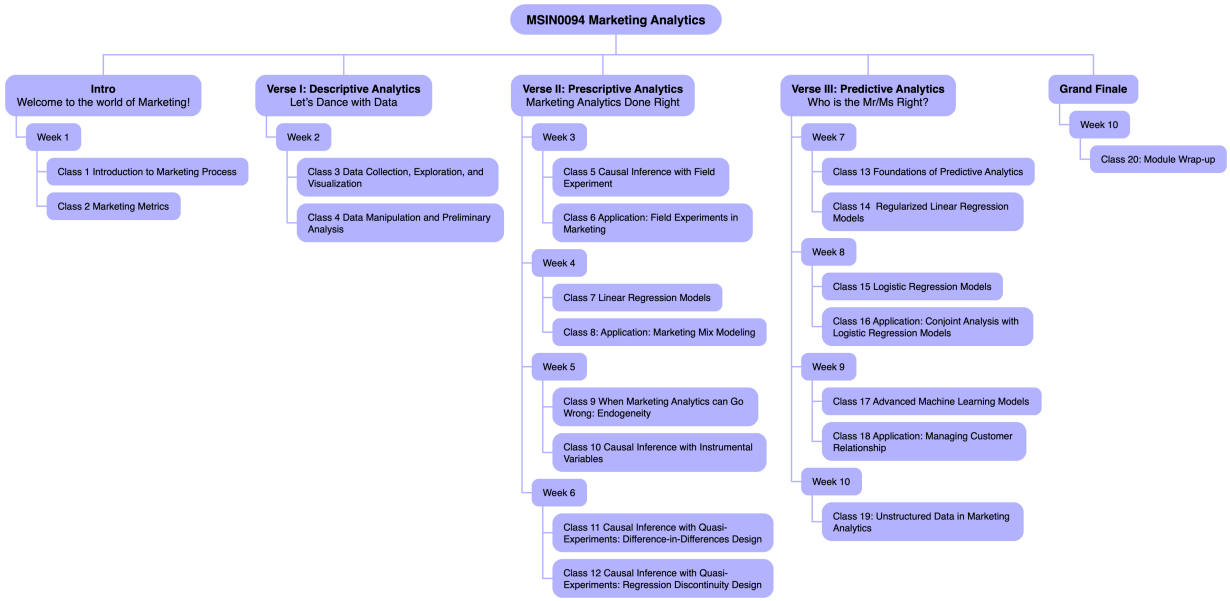
[1]Wei is pronounced as "way", which stands for jade in Mandarin; Miao is my family name, which is exactly pronounced as "meow", the sound of a kitty.

2.2 Course Objective

Marketing analytics addresses how to utilize appropriate analytics tools to guide marketing tactics and strategies in a scientific manner. Driven by recent advances in computing power, software, and other information technologies, the very nature of marketing analytics has evolved. This module provides students with systematic training in applications of statistics, econometrics, machine learning models, and programming tools in solving real-life marketing problems.

More specifically, this module will explore marketing analytics along three inter-related dimensions: descriptive analytics, prescriptive analytics, and predictive analytics. Descriptive analytics involves using visualization tools and statistical tests to provide model-free insights into customer intelligence. Prescriptive analytics teaches how to establish causal inference using both experimental and non-experimental methods. Predictive analytics shed light on how to conduct targeted marketing by taking advantage of state-of-the-art machine learning algorithms. At the end of the module, students will be able to carry out independent marketing research for their dissertation projects and their future jobs.

2.3 Module Structure Overview



2.4 Office Hours

It is a good practice to utilize the Moodle forum as an interactive place to promote peer-to-peer learning. For any questions, please first check the Moodle forum and see if the questions are already posted and answered by any of your classmates there. If not, please post your questions in the Moodle forum. For any forum questions, we will first leave for your classmates to provide potential answers. The teaching team will also monitor the forum and clarify any unanswered questions.

If your question is still unsolved, please make appointments for office hours. I am happy to answer any questions you may have about this module (well, definitely excluding assignment questions before the submission date). Since in the academic year 2021-22, UCL has adopted a blended approach for teaching (online plus face-to-face teaching), I will confirm more details of office hours in the first week's class to accommodate your preference.

To make an appointment, please use this shared Excel form as in this link. Please make sure of the following before/when you make your appointment:

- Please be careful when operating on the Excel form and kindly avoid overwriting/modifying other students' appointments.
- Please use the "comment" function in Excel form to briefly describe the questions you have so that other students sharing similar questions can drop in as well.
- Each session lasts for 15 minutes. If you expect your questions may take a longer time to solve, you can book two consecutive sessions. Each student is entitled to at most two sessions per week to ensure equal opportunities for all students.

2.5 Textbook

We will not rely on any specific textbook in this module. All classes will be based on the lecture notes and supplementary readings I have prepared for you. However, if you would like to further enrich your learning journey and extend your knowledge of marketing analytics, I recommend "Handbook of Marketing Analytics". This book is free for download at UCL's E-library.

2.6 Programming Language R

We will use an open-source programming language R throughout the module. R is one of the most commonly used programming languages by data scientists, economists, and statisticians, and is sometimes called the "golden child" of data science. R is cross-platform compatible on Windows, Mac, and Linux, with thousands of packages ready for use. Undoubtedly, it has one of the richest ecosystems to perform data analytics tasks. More importantly, it is free of charge compared with other commercial software (it is good to start developing cost-benefit concepts now as you are now business school students. And we will learn more on cost-benefit analysis in the module later on)!

The teaching of R will consist of the following part:

- In the induction week, Yongdong and I will provide a tutorial on introduction to R, which covers R basics.
- In the remaining weeks, there will be R tutorials specific to each class's topic.

If you have little prior exposure to R, it is highly recommended that you should read this tutorial, "Introduction to R".

During (and even after) this module, whenever your run into any problems with R, Google (especially Stack Overflow) is always your best place to seek answers for most debugging issues; if you have any specific tasks in mind, the R Task Views page provides comprehensive guidance of which packages on CRAN are relevant for tasks related to a certain topic.

2.7 Classroom Etiquette

- **Punctuality**. Classes will start on time. Please be on time.
- **Class participation**. Please keep your mobile phones and laptops muted at all times unless otherwise instructed. For online students, please remember to mute yourself before joining the Zoom session. And if possible, please join the online session in a quiet room without potential disruptions from others. (Crazy things like these do happen! Be careful not to become the next meme circulated among your classmates :D)

2.8 Assignments and Grading policy

There will be no exams in this module. Learning outcomes will be examined through three individual assignments. Your final grade in this module will be a weighted average of the following:

- First assignment, 30% weight, 1500 words, due on Oct 29
- Second assignment, 40% weight, 2000 words, due on Nov 26
- Third assignment, 30% weight, 1500 words, due on Dec 17

Detailed assignment descriptions will be given in due course.

2.8.1 Code of Conduct and Plagiarism

You will receive training in Student Academic Misconduct Procedure in the induction week, in which you will learn how to properly reference your work and avoid plagiarism. As all assignments in this module are **individual** assignments, you should **NOT** work with other students in any way.

Note that it will be a severe violation of academic integrity if you look for solutions on the Internet or even outsource the assignments to any agency. You may risk failing this module if such a violation is found and investigated by the University.

2.8.2 Word Count

Please note, according to UCL's policy, you should **follow the word limit**. Otherwise, your marks may be reduced. Therefore, try to keep your answer concise and to the point.

2.8.3 For Extenuating Circumstances and Emergencies

Please double check (or even triple check) whether the file to be uploaded is the correct, finalized answer sheet before submission. The final marks will be ONLY based on the file you submit, even though you may be able to show evidence that a wrong file is uploaded. Neither I nor the TAs are authorized to handle ECs or any emergencies related to assignment submission. If anything, please contact the MSc program admin here for assistance.

2.8.4 Marks and Feedback

Your marks and feedback for each assignment will be usually released no later than 4 weeks after the submission date. Each assignment will be carefully marked according to the mark scheme by our teaching assistants, and there will be adequate second-marking procedures in the marking process to ensure the marking quality, as required by UCL. As a result, it is the university policy that students shouldn't question the academic judgment of markers, and **please refrain from emailing teaching assistants for any re-marking**.

2.9 Copyright Issues

All of the materials in this module are copyrighted, either purchased by UCL or written by me. Please refrain from uploading any materials in the module to other external websites or sharing any materials with anyone unauthorized. A copyright violation may be investigated by the school.

Chapter 3

R Tutorials

This is a supplementary teaching material for the MSc Business Analytics program at UCL School of Management. This tutorial aims to provide a comprehensive introduction to the basics of R programming language.

3.1 Install and Setup R

3.1.1 Class Objective

Learning objective:

- Understand R's history and main functionality
- Be able to setup R and Rstudio
- Understand how to use install and load packages in R
- Understand the key concepts of R

3.1.2 What is R

- R is both a programming language and software environment for statistical computing, which is free and open-source (https://www.r-project.org/about.html).
- The ***R Project*** was initiated by Robert Gentleman and Ross Ihaka (University of Auckland) in the early 1990s as a different implementation of the S language, which was developed at Bell Laboratories.
- Since 1997, R has been developed by the ***R Development Core Team***.
- Refer to the Wiki page for a full history of R development.

3.1.3 What is R used for?

R can do almost anything you can think of! Below is a (incomplete) list of what I normally use R for on a daily basis.

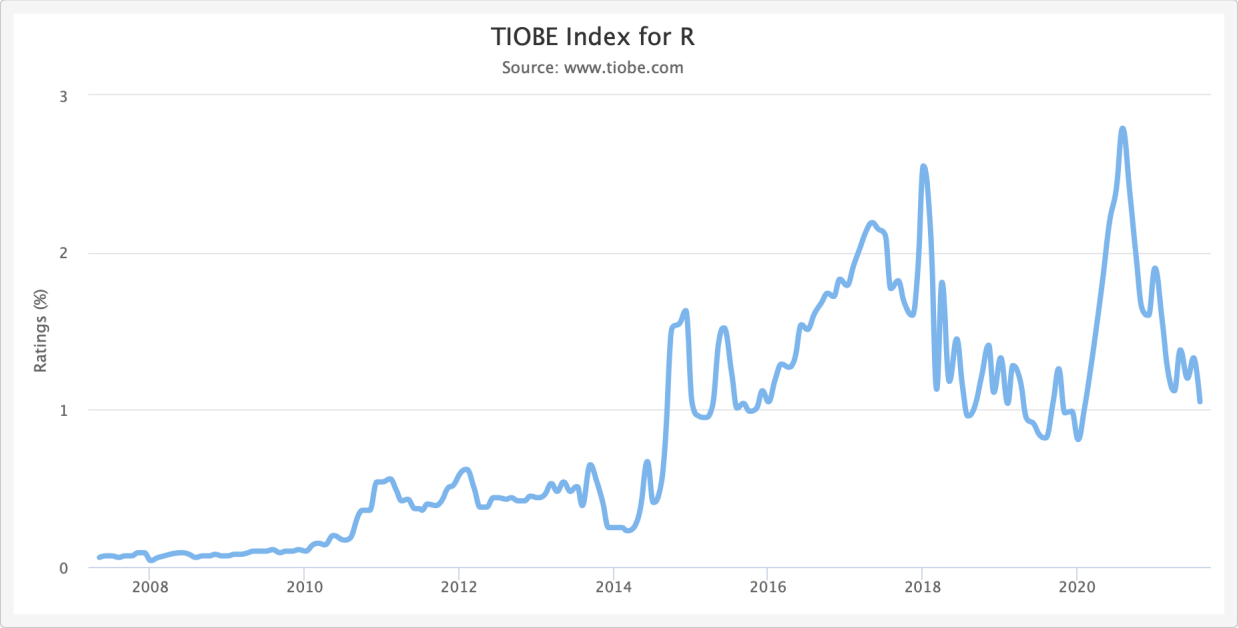
- Data analytics (statistics, data manipulation, machine learning, econometric analysis, etc.)
- Building your own homepage. I build and maintain my personal website with the `blogdown` package completely in R
- Prepare your presentations using R `markdown`
- Write books (such as this whole online supplement) using R `bookdown`

You can even scrawl webpages and send emails with R once you install the right packages.

3.1.4 Popularity of R

R has been one of the most popular programming language used by statisticians, economists, and data analysts.

Below is the popularity index of different languages by tiobe.com. R reached its historic highest in August 2020, ranking the 8th among all.



3.1.5 Install and Setup R on your computers: Step 1

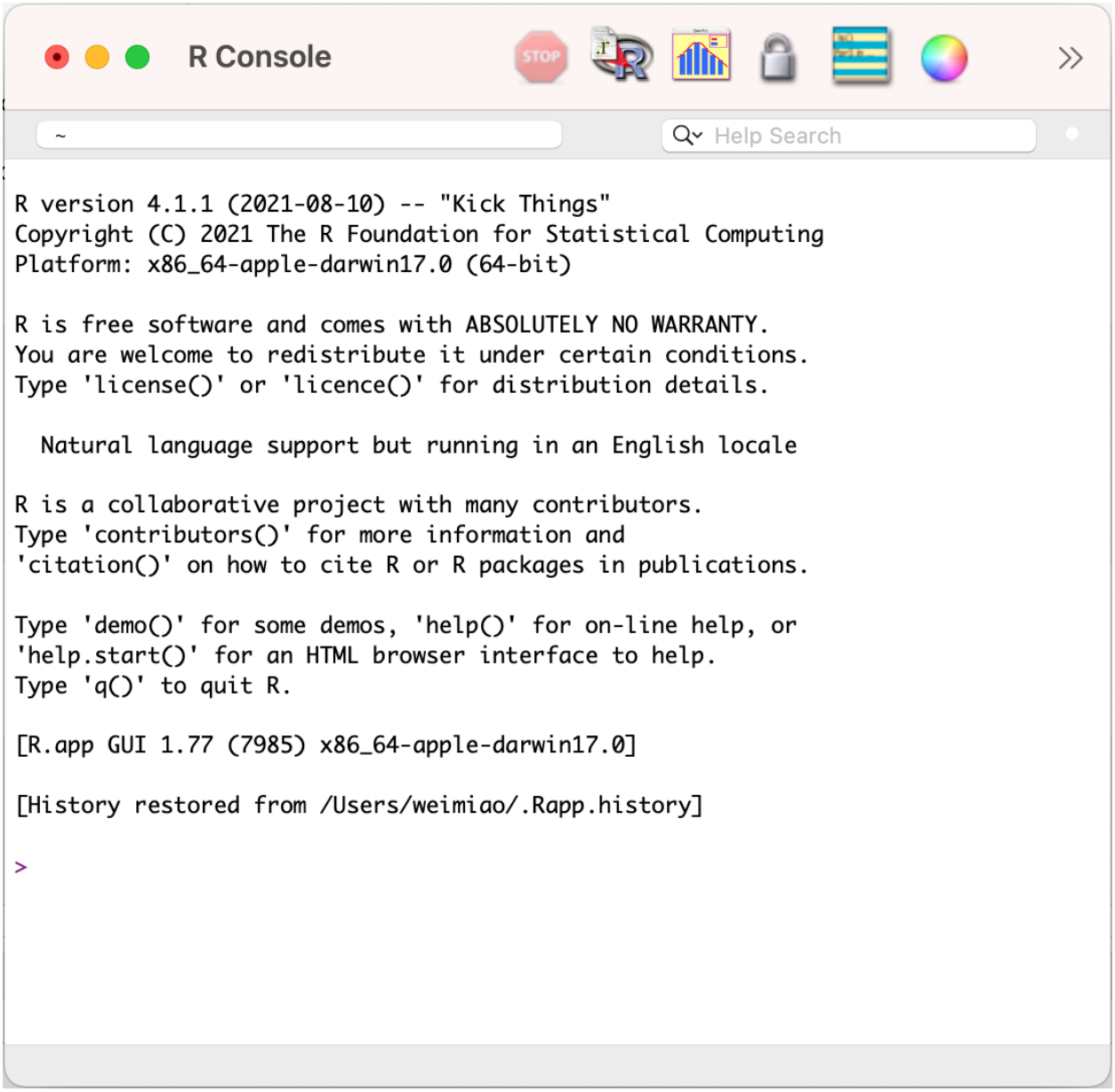
There are alternative ways to install and use R on your computers. Below is the most recommended and hassle free way to install and setup R on your computer.

Step 1: Download R from the official website CRAN.

Tips:
- Note that R version is updated on a regular basis, to fix bugs or add new features. It's good to keep your R up to date with the official website.
- However, sometimes across major versions, some R's base functions may be depreciated and cause breaks in your code, so it's important to read the R News after each update, so that you understand what's been updated in the latest version and if there will be any breaks in your previous code. (For a deeper understanding of this issue, please refer to this article regarding reproducible research with R.)

3.1.6 Screenshot of R Console

R's default user interface:



3.1.7 Install and Setup R on your computers: Step 2

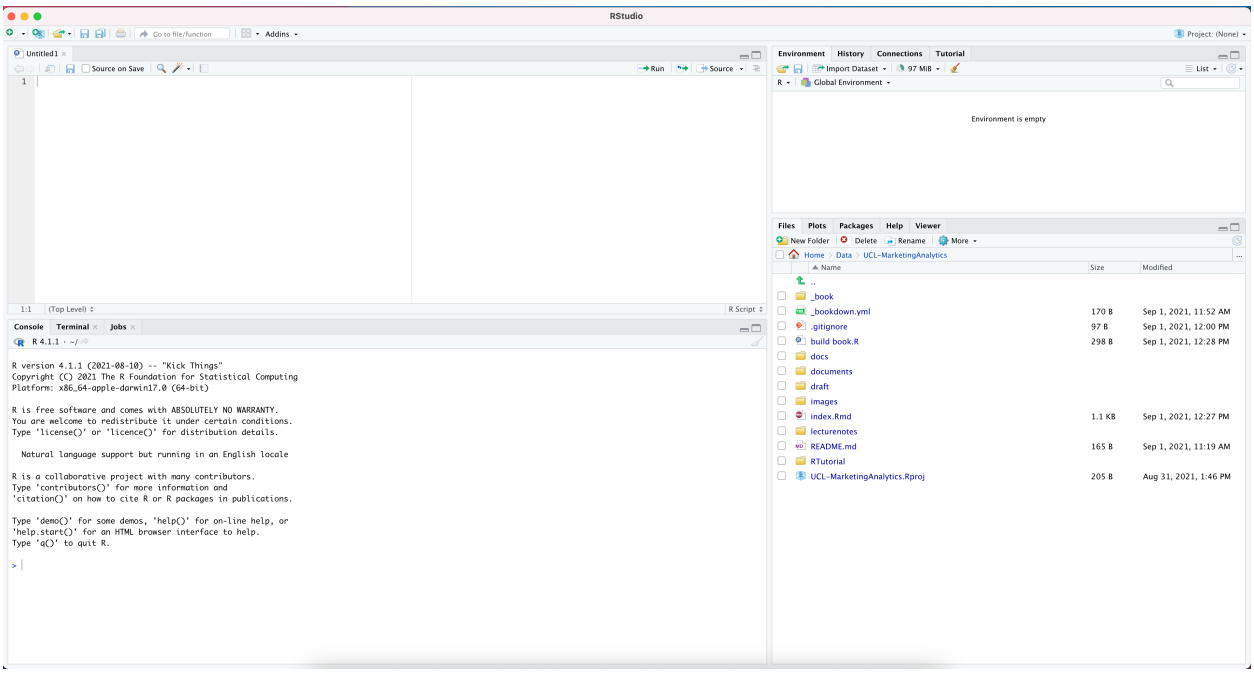
After step 1, you can already open R's own GUI and run R codes. However, we still need an integrated development environment (IDE) to help us better write R codes. Rstudio is currently the best IDE for R.

Step 2: Download and install Rstudio from Rstudio's website.

For a introduction to the Rstudio IDE, refer to this cheat sheet.

3.1.8 Screenshot of RStudio

Our preferred R editor: Rstudio



3.1.9 Glossary

- ***script***: (top left) where you write codes a text-readable file (E.g., 'test.R', 'test.Rmd', etc.), each separated by a new line.
- ***console***: (bottom left) where you can type commands interactively with R and get the results immediately
- ***environment***: (top right) a list of named variables/objects that we have generated; can be checked on the 'environment' panel.
- ***history***: (top right) the list of past commands thaty we have used
- ***help***: (bottom right) a documentation of all the functions available in R (the user manual)
- ***package***: (bottom right) a collection of additional functions and dataset

3.1.10 Use Help

Whenever you don't know how to use a function, you can either

- use `help`. Enter the command `help(log)` in R console

```
?log
help(log)
```

- or you can directly search for the function in RStudio help panel.

If you cannot get an answer:

- (1st) Google your error message or search/ask for help in Stack Overflow
- (2nd) Ask in Moodle Forum
- (3rd) Use office hours

3.1.11 Libraries and packages management: the coventional way

To achieve certain tasks, for instance, to run a regression analysis, we can definitely write our own code from scratch. However, it may take much time if we would like to build everything from zero. That's when the packages/libraries can help us to improve our efficiency by a large margin.

One of the best things about R is that it has tens of thousands of extremely useful packages/libraries. And some are even exclusive to R, for instance, the `grf` package that runs the causal forest model, which I have used for my research.

```
# install.packages('data.table') # install the package
library('data.table') # load the package
```

3.1.12 Libraries and packages management: the simplest way

In R, there is a package called `pacman`, which stands for package management and is probably the most convenient way to manage R packages.

Task: Load `ggplot2` and `data.table` packages; if they are not installed, install them and then load them.

```
# install.packages('pacman')

pacman::p_load(data.table,ggplot2)
```

3.1.13 Comment your codes

You can put a "#" before any line of code, to indicate that this line is your comment, and this line will turn green and not run by R.

```
# print("David is handsome")
```

3.2 R Basics

3.2.1 Use R as a calculator

We can do the basic mathematical calculation in R as follows:

```
8+8
```

```
## [1] 16
```

```
8-8
```

```
## [1] 0
```

```
8*8
```

```
## [1] 64
```

```
2/2
```

```
## [1] 1
```

Common mathematical operations/functions can also be used in R.

```
log(1)
```

```
## [1] 0
```

```
exp(1)
```

```
## [1] 2.718282
```

```
log(exp(1))
```

```
## [1] 1
```

```
sqrt(25)
```

```
## [1] 5
```

3.2.2 Assign value

In R, we normally use the left arrow to assign a value to an object `<-`

```
x <- log(2)
x
```

```
## [1] 0.6931472
```

You can also use equal sign `=` without issues, but it's recommended to stick with R's tradition.

3.2.3 Data types

- Numeric/Double (e.g. 2.5, 1/5, 1.0, etc.)
- Integer (e.g. 1, 2, 3, etc.)
- Complex (e.g. 1 + 2i, etc.)
- Logical (e.g. TRUE, FALSE or NA)
- Character (e.g. "Wei", "UCL", "1 + 1 = 3", "TRUE", etc.)
- Factor/Categorical ("male", "female", etc.)

```
# Character Vector
str1 <- c("R","Python","Java","Scala","Julia")

# Numeric Vector
num2 <- c(1, 2, 3.4, 5, 5.9)

# Integer Vector
int3 <- (1:10)

# Logical Vector
log4 <- c(TRUE, FALSE, FALSE, T, F)

# 5. Complex
com5 <- c(2+3i, 3+5i, 4+5i)
```

- Special type: `NA` which stands for missing values; this is a very tricky type that needs additional attention, as we will show below.

```
missing <- c(1,NA,2,NA)

mean(missing)
```

```
## [1] NA
```

```
mean(missing,na.rm = T) ##### important!!!, do not forget na.rm = T
```

```
## [1] 1.5
```

```
# check missing values
is.na(missing)
```

```
## [1] FALSE  TRUE FALSE  TRUE
```

```
anyNA(missing)
```

```
## [1] TRUE
```

```
sum(is.na(missing))
```

```
## [1] 2
```

3.2.4 Data type: other special values

- `Inf` is infinity. You can have either positive or negative infinity.
- `NaN` means "Not a Number". It's an undefined value.

```
a <- 1/0 #Inf
a
```

```
## [1] Inf
```

```
is.infinite(a)
```

```
## [1] TRUE
```

```
is.na(a)
```

```
## [1] FALSE
```

```
b <- 0/0 #NaN
b
```

```
## [1] NaN
```

```
is.nan(b)
```

```
## [1] TRUE
```

```
is.na(b)
```

```
## [1] TRUE
```

3.2.5 Data type: check data type using class()

We can use `class()` to check the type of an object in R.

```
a <- '1+1'
class(a)
```

```
## [1] "character"
```

```
b <- 1+1
class(b)
```

```
## [1] "numeric"
```

3.2.6 Data type: conversion

```
a <- '1'
class(a)
```

```
## [1] "character"
```

```
b <- as.numeric(a)
class(b)
```

```
## [1] "numeric"
```

3.2.7 Data structures

3.2.8 Vectors: creating vectors

3.2.8.1 Creating vectors: c()

Vector can be created using the command `c()`.

```
x <- c(1, 3, 5, 10)
x
```

```
## [1]  1  3  5 10
```

```
class(x)
```

```
## [1] "numeric"
```

Vectors must contain elements of the same data type. Otherwise, it will convert elements into the same type.

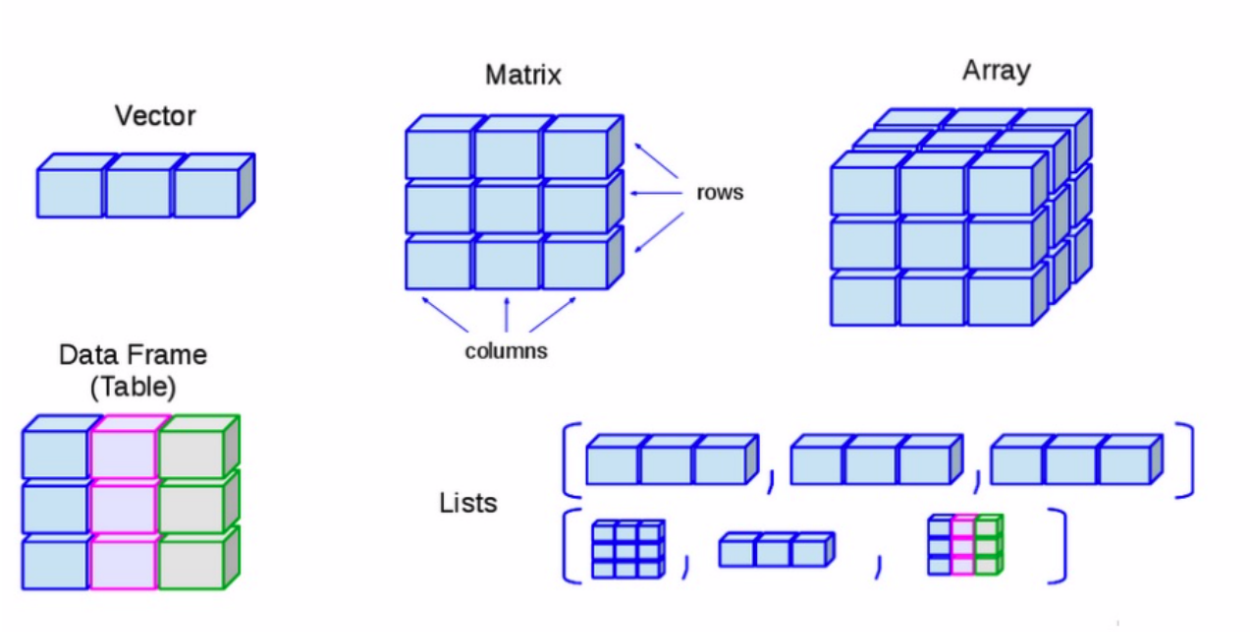


Figure 3.1: Visualization of data structures

```
x <- c(1, "intro", TRUE)

class(x)
```

```
## [1] "character"
```

3.2.8.2 Creating vectors: length()

You can measure the length of a **vector** using the command `length()`

```
x <- c('R',' is', ' fun')
length(x)
```

```
## [1] 3
```

```
y <- c(NULL)
length(y)
```

```
## [1] 0
```

3.2.8.3 Creating vectors: seq() and rep()

It is also possible to easily create sequences

```r
# use colon ':'
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# use seq()
seq(from = 1, to = 2, by = 0.1)
```

```
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```r
# for date object
seq.Date(from = as.Date("2021-09-27"),
         to = as.Date("2021-10-01"),
         by = '1 days')
```

```
## [1] "2021-09-27" "2021-09-28" "2021-09-29" "2021-09-30" "2021-10-01"
```

```r
# replication using rep()
rep("A", times = 5)
```

```
## [1] "A" "A" "A" "A" "A"
```

To understand all usages of a function, use `help()`

3.2.8.4 Creating vectors: combine vectors

You can use `c()` to combine different vectors; this is very commonly used when concatenating vectors.

```r
x <- 1:3 # from 1 to 3
y <- c(10, 15) # 10 and 15
z <- c(x,y) # x first and then y
z
```

```
## [1]  1  2  3 10 15
```

3.2.9 Vectors: subsetting

We put the index of elements we would like to extract in a **square bracket**. Different languages use different ways to index and subset vectors and matrices.
- For instance, Matlab uses parentheses for subsetting vectors and matrices.

```
x <- c(1,3,8,7)
x[2] # which element is in the second position?
```

```
## [1] 3
```

```
x[1:2] # which elements are in the first 2 positions?
```

```
## [1] 1 3
```

```
x[c(1,3,4)] # which elements are in positions 1, 3 and 4?
```

```
## [1] 1 8 7
```

3.2.10 Vectors: relational operation

```
x <- c(1,3,8,7)
x < 6 # is each element lower than 6?
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

```
x == 10 # is the element equal to 10?
```

```
## [1] FALSE FALSE FALSE FALSE
```

```
which(x == 8) # which element equals 8
```

```
## [1] 3
```

```
max(x) # what is the max element
```

```
## [1] 8
```

```
which.max(x) # which is the max element
```

```
## [1] 3
```

```
x[(which.min(x))] #  = min(x)
```

```
## [1] 1
```

```
## logical operation
T & F # and
```

```
## [1] FALSE
```

```
T | F # or
```

```
## [1] TRUE
```

```
!T # not
```

```
## [1] FALSE
```

A special relational operation is `%in%` in R, which tests whether an element exists in the object.

```
3 %in% x
```

```
## [1] TRUE
```

```
4 %in% x
```

```
## [1] FALSE
```

```
3 %in% x & 4 %in% x
```

```
## [1] FALSE
```

3.2.11 Vectors: elementwise operations

Similar to other matrix languages (Matlab), R is a vectorized language, meaning by default it will do vector operation internally.

```
x <- c(1,3,8,7)
x+2
```

```
## [1]  3  5 10  9
```

```
x-2
```

```
## [1] -1  1  6  5
```

```
x*2
```

```
## [1]  2  6 16 14
```

```
x^2
```

```
## [1]  1  9 64 49
```

3.2.11.1 Caveat with vector elementwise operation

Caveat When the length of vectors do not match, R will still do it for your without reporting error but a warning message. As you can see, even if the length of vectors does not match, R can still return an output.

It's important to check the warning messages when there is any!!

```
x <- c(1,3,8,7)

y <- c(1,3,4) # careful!!! does not report error
x + y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object length
```

```
## [1]  2  6 12  8
```

```
z <- c(10,12,13,14)
x + z
```

```
## [1] 11 15 21 21
```

3.2.12 Matrices: creating matrices

3.2.12.1 Creating matrices: matrix()

A matrix can be created using the command `matrix()`
- the first argument is the vector to be converted into matrix
- the second argument is the number of rows
- the last argument is the number of cols (optional)

```
matrix(1:9, nrow = 3, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
matrix(1:9, nrow = 3, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
matrix(1:9, nrow = 3, ncol = 2)
```

```
## Warning in matrix(1:9, nrow = 3, ncol = 2): data length [9] is not a sub-multiple or multipl
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Note: R automatically inserts elements by columns

3.2.12.2 Creating matrices: inserting by row

However, we can ask R to insert by rows by setting the 'byrow' argument.

```
matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

3.2.12.3 Creating matrices: concatenation of matrices cbind() and rbind()

We can use `cbind()` and `rbind()` to concatenate vectors and matrices into new matrices.

```
x <- cbind(1:3, 4:6) # column bind
x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
y <- rbind(7:9, 10:12) # row bind
y
```

```
##      [,1] [,2] [,3]
## [1,]    7    8    9
## [2,]   10   11   12
```

```
cbind(x,x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    1    4
## [2,]    2    5    2    5
## [3,]    3    6    3    6
```

3.2.13 Matrices: indexing and subsetting

```
x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
x[1,2] # the element in the 1st row, 2nd column
```

```
## [1] 4
```

```
x[1,] # all elements in the first row
```

```
## [1] 1 4
```

```
x[,2] # all elements in the second columns
```

```
## [1] 4 5 6
```

```
x[c(1,2),]
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
```

3.2.14 Matrices: operations

Let's use 3 matrices `x`, `y`, and `z`:

```
x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
y
```

```
##      [,1] [,2] [,3]
## [1,]    7    8    9
## [2,]   10   11   12
```

```
z <- x^2
z
```

```
##      [,1] [,2]
## [1,]    1   16
## [2,]    4   25
## [3,]    9   36
```

3.2.14.1 Matrices' operations: matrix addition and multiplication

```
x+z   # elementwise addition
```

```
##      [,1] [,2]
## [1,]    2   20
## [2,]    6   30
## [3,]   12   42
```

```
x*z   # elementwise multiplication
```

```
##      [,1] [,2]
## [1,]    1   64
## [2,]    8  125
## [3,]   27  216
```

```
x%*%y # matrix multiplication
```

```
##      [,1] [,2] [,3]
## [1,]   47   52   57
## [2,]   64   71   78
## [3,]   81   90   99
```

3.2.14.2 Matrices' operations: inverse and transpose

```
t(x) # transpose
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
t(x)%*%x
```

```
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

```
solve(t(x)%*%x) # inverse; must be on a square matrix
```

```
##            [,1]       [,2]
## [1,]  1.4259259 -0.5925926
## [2,] -0.5925926  0.2592593
```

3.2.15 Arrays

```
x <- 1:4
x <- array(data = x, dim = c(2,3,2))
x
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    1
## [2,]    2    4    2
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    3    1    3
## [2,]    4    2    4
```

- Just like vectors and matrices, arrays can include only data types of the same kind.
- A 3D array is basically a combination of matrices each laid on top of other

3.2.16 Lists

A list is an R object that can contain anything.

```
x <- 1:2
y <- c("a", "b")
L <- list( numbers = x, letters = y)
```

3.2.17 Lists: indexing and subsetting

There are many ways to extract a certain element from a list.
- by index
- by the name of the element
- by dollar sign `$`

```
L[[1]] # extract the first element
```

```
## [1] 1 2
```

```
L[['numbers']] # based on element name
```

```
## [1] 1 2
```

```
L$numbers # extract the element called numbers
```

```
## [1] 1 2
```

After extracting the element, we can work on the element further:

```
L$numbers[1:3] > 2
```

```
## [1] FALSE FALSE    NA
```

3.2.18 Data Frames: creating dataframe

3.2.18.1 Data Frames: create dataframe using data.frame()

Data Frame is the R object that we will deal with most of the time in the MSc program. You can think of `data.frame` as a spreadsheet in excel.

```
df <-  data.frame(id = 1:4,
  name = c("David", "Yongdong", "Anil", "Wei"),
  wage = rnorm(n=4, mean = 10^5, sd = 10^3),
  male = c(T, T, T, T)
  )
df
```

```
##   id     name      wage male
## 1  1    David  99840.06 TRUE
## 2  2 Yongdong 100385.00 TRUE
## 3  3     Anil  99588.71 TRUE
## 4  4      Wei 100216.86 TRUE
```

3.2.18.2 Data Frames: create dataframe using read.csv()

```
#read.csv('data.csv')
```

3.2.18.3 Use R's own dataset

```
data("mtcars")
```

3.2.19 Data Frames: property

Each row stands for an observation; each column stands for a variable.

Each column should have a **unique** name.

Each column must contain the same data type, but the different columns can store different data types.
- dataframe is a special type of matrix, but allows different types across columns.

Each column must be of same length, because rows have the same length across variables.

You can verify the size of the `data.frame` using the command `dim()`; or `nrow()` and `ncol()`

```
dim(df)
```

```
## [1] 4 4
```

```
nrow(df)
```

```
## [1] 4
```

```
ncol(df)
```

```
## [1] 4
```

You can get the `data type` info using the command `str()`

```
class(df)
```

```
## [1] "data.frame"
```

```
str(df)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ id  : int  1 2 3 4
##  $ name: chr  "David" "Yongdong" "Anil" "Wei"
##  $ wage: num  99840 100385 99589 100217
##  $ male: logi  TRUE TRUE TRUE TRUE
```

```
lapply(df, class)
```

```
## $id
## [1] "integer"
##
## $name
## [1] "character"
##
## $wage
## [1] "numeric"
##
## $male
## [1] "logical"
```

3.2.20 Data Frames: inspection

```
head(df, n=3) # first n observations
```

```
##   id     name      wage male
## 1  1    David  99840.06 TRUE
## 2  2 Yongdong 100385.00 TRUE
## 3  3     Anil  99588.71 TRUE
```

```
tail(df, n=3) # last n observations
```

```
##   id     name      wage male
## 2  2 Yongdong 100385.00 TRUE
## 3  3     Anil  99588.71 TRUE
## 4  4      Wei 100216.86 TRUE
```

```
names(df)
```

```
## [1] "id"   "name" "wage" "male"
```

```
summary(df)
```

```
##        id           name                wage          male
##  Min.   :1.00   Length:4           Min.   : 99589   Mode:logical
##  1st Qu.:1.75   Class :character   1st Qu.: 99777   TRUE:4
##  Median :2.50   Mode  :character   Median :100028
##  Mean   :2.50                      Mean   :100008
##  3rd Qu.:3.25                      3rd Qu.:100259
##  Max.   :4.00                      Max.   :100385
```

```
str(df)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ id  : int  1 2 3 4
##  $ name: chr  "David" "Yongdong" "Anil" "Wei"
##  $ wage: num  99840 100385 99589 100217
##  $ male: logi  TRUE TRUE TRUE TRUE
```

3.2.21 Data Frames: subsetting

Since a dataframe is essentially a matrix, all the subsetting syntax with matrices can be applied here.

```
df$name # subset a column
```

```
## [1] "David"    "Yongdong" "Anil"     "Wei"
```

```
df[,c(2,3)] # can also subset like a matrix
```

```
##       name      wage
## 1    David  99840.06
## 2 Yongdong 100385.00
## 3     Anil  99588.71
## 4      Wei 100216.86
```

We are interesting in the cylinders and the weights of inefficient cars (lower than 15 miles per gallon).

```
poll_cars <- mtcars[mtcars$mpg<15, c("cyl", "wt")] # remember to assign the generated datafram
poll_cars
```

```
##                     cyl    wt
## Duster 360            8 3.570
## Cadillac Fleetwood    8 5.250
## Lincoln Continental   8 5.424
## Chrysler Imperial     8 5.345
## Camaro Z28            8 3.840
```

3.3 Programming Basics

3.3.1 Variables

In programming, a variable denotes an object, i.e., a variable is a name that refers to an object in the memory. Variables in R programming can be used to store any R object discussed previously, including numbers, characters, matrices, and data frames.

R is a ***dynamically programmed language**, which means that unlike other programming languages (such as C++, which is a pain to debug), we do not have to declare the data type of a variable before we can use it in our program.

For a variable to be valid, it should follow these rules

- It should contain letters, numbers, and only dot or underscore characters.
- It should not start with a number (eg:- 2iota)
- It should not start with a dot followed by a number (eg:- .2iota)
- It should not start with an underscore (eg:- _iota)
- It should not be a reserved keyword.

3.3.2 if/else

Sometimes, you want to run your code based on different conditions. For instance, if the observation is a missing value, then use the population average to impute the missing value. This is where `if/else` kicks in.

```
if (condition == TRUE) {
  action 1
} else if (condition == TRUE ){
  action 2
} else {
  action 3
}
```

Example 1:

```
a <- 15

if (a > 10) {
larger_than_10 <- TRUE
} else {
  larger_than_10 <- FALSE
}

larger_than_10
```

```
## [1] TRUE
```

Example 2:

```
x <- -5
if(x > 0){
  print("x is a non-negative number")
} else {
  print("x is a negative number")
}
```

```
## [1] "x is a negative number"
```

3.3.3 Loops

As the name suggests, in a loop the program repeats a set of instructions many times, until the stopping criteria is met.

Loop is very useful for repetitive jobs.

```
for (i in 1:10){ # i is the iterator
  # loop body: gets executed each time
  # the value of i changes with each iteration
}
```

3.3.4 Nested loops

We can also nest loops into other loops.

```
x <- cbind(1:3, 4:6) # column bind
x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
y <- cbind(7:9, 10:12) # row bind
y
```

```
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   11
## [3,]    9   12
```

```
z <- x

for (i in 1:nrow(x)) {
  for (j in 1:ncol(x)){
    z[i,j] <- x[i,j] + y[i,j]
  }
}

z
```

```
##      [,1] [,2]
## [1,]    8   14
## [2,]   10   16
## [3,]   12   18
```

3.3.5 Functions

A function takes the argument as input, run some specified actions, and then return the result to us.

Functions are very useful. When we would like to test different ideas, we can combine functions with loops: We can write a function which takes different parameters as input, and we can use a loop to go through all the possible combinations of parameters.

3.3.5.1 User-defined function syntax

Here is how to define a function in general:

```
function_name <- function(arg1 ,arg2 = default_value){
  # write the actions to be done with arg1 and arg2
  # you can have any number of arguments, with or without defaults
  return() # the last line is to return some value
}
```

Example:

```
magic <- function( x, y){
  return(x^2 + y)
}

magic(1,3)
```

```
## [1] 4
```

3.3.6 A comprehensive example

Task: write a function, which takes a vector as input, and returns the max value of the vector

```
get_max <- function(input){
  max_value <- input[1]
  for (i in 2:length(input) ) {
    if (input[i] > max_value) {
      max <- input[i]
    }
  }

  return(max)
}

get_max(c(-1,3,2))
```

```
## [1] 2
```

Exercise:

Write your own version of `which.max()` function

Chapter 4

Lecture Notes

The lecture notes will be released one week before each class. Please do not circulate without permission from the module leader. To use the lecture notes outside MSIN0094, please contact the module leader (wei.miao@ucl.ac.uk)