R Basics

1. **R**

    - R is an environment for statistical computing and graphics.

      A combination of a statistics package and a programming language.
    - R is mostly command-line driven.

      R is not the simplest software to use.

      Be very patient, work a lot with it, make lots of mistakes.

      Try to figure out what is going on and what is going wrong, and learn as you go.
    - R is completely free and runs on all popular operating systems.
    - The CRAN (`https://cloud.r-project.org`) website is something you will consult frequently for both the software, documentation and packages others have developed. You may also upload your own package for others to use.

2. **RStudio**

    - RStudio is an integrated development environment (IDE) for R. Launch Rstudio and open a new R script. Rstudio is comprised of 4 windows and the windows are arranged by default as follows.
      (a) R script window (top-left corner) : where you type in R commands.
      (b) Console window (bottom-left corner) : where commands are run.
      (c) Environment and History window (top-right corner)
        – Environment : the data sets you imported and the variables you created.
        – History : the commands you have done.

(d) File, Plot, Packages and Help window (bottom-right corner)
   - Files : navigate Windows
   - Plots : display plots and see plot history using the arrows.
   - Packages : bundles of functions.
   - Help : get more information on any specific function or command.

You can change the layout of the windows, font, appearance,

3. **Using R as a calculator**

- **The command prompt $>$ is where you dialog with R.**

Symbols	$+$	: addition
	$-$	: subtraction
	$*$	: multiplication
	$/$	: division
	ˆ	: Exponentiation

Example $> 2+5$
$> 3/5$
$> 3^5$
$> 1-3*4$
$> (1-3)*4$

4. **Built-in functions**

- Numerous mathematical and statistical functions are also provided in *R*.
- Specific names that R **recognizes**, followed by a required pair of parentheses, inside which you provide necessary function arguments.
- All text in the line following a # is ignored by *R*.

Example > Function-name(arguments)
> sqrt(40) # the square root function
> squareroot(40) # R does not recognize a function named "squareroot"
> exp(1) # the exp function: exp(x)=e^x
> log(14) # the log base-e function (natural log, or ln)

```
> Log(14) # R is case-sensitive
> abs(-0.43) #absolute value
> floor(3.4) #round down
> ceiling(3.4) #round up
```

5. **Tips**

   - **Commands can be edited** using the left and right cursor keys as well as the backspace, delete, and insert keys.
   - Use the **up-cursor key (↑)** to bring back an earlier command for reuse or modification.
   - Help is available using a **?function-name** like this one

     ```
     > ?exp
     ```

     which shows what the exp function does (and a lot more).
   - Google "R ________" for help. For example, "R standard deviation".
   - **Use R script window.** Typing in R scripts directly into the command line in the R console can be tedious, especially if you are entering many lines of script. It is easier to use a R script to "compose" your scripts.
   - **Run R commands.** You can run a certain line in R script using Ctrl+Enter (Window) or Command+Enter (Mac). If you want to run multiple command lines in R script, highlight them first and Ctrl+Enter or Command+Enter.
   - **Make comment for your R script.** You may use # to comment your R script. If you want to comment/uncomment a few lines together, use Ctrl+Shift+C (Window) or Command+Shift+C (Mac).

6. **Assignment**

   - Need to name a value so that we can refer it later.
   - Assignment is done via the $<-$ operator (shortcut: "Alt"+"-" (windows) and "Option"+"-" (mac)), then R will show the keyboard shortcut reference.), or = operator, which take the value on the right-hand side and assigns it to an R "objects" with name you provide on the left.

Example $> x < -10$ # The value 10 is assigned to an object named x.

$> x$ # call the value of x.

$> y = 2 * x$ # Assign a new object name y with twice value of x.

$> x = x + 5$ # Replace x with the value $x + 5$.

- **Acceptable names** may be made out of letters, numbers, and the period(.) symbol, but names must start with a letter.

7. **Vectors**

- We can construct a vector using the c() function.

Example
```
> x = c(5,3,6,4,6,7,1)
> x
> x[2] # Extract the 2nd element
> 2:4 # Create an integer sequence from 2 to 4
> x[2:4] # Extract the 2nd through 4th elements
> x[c(2,4,6)] # Extract the 2nd, 4th, 6th elements
> x[-2] # Extract all but the 2nd elements
> -2:-4
> x[-2:-4] # Extract all but the 2nd through 4th elements
> x[2]=10 # Assign 10 to the 2nd element
> x
> x[c(1,3,6)]=c(10,30,60) # How many elements does x have?
> x
> x=3:6
> y = 2*x # Functions apply to each element of a vector at
  the same time.
> y
> seq(from=0,to=6,by=2) # a function has multiple arguments.
> ?seq # What are the first three argument in the seq() func-
  tion?
> seq(0,6,2) # R expects a certain order for function
  arguments
> seq(6,0,2) # R could not read our mind.
> seq(to=6,from=0,by=2)
> z=seq(from=0,to=6,by=2)
```

> x*z # Entry-wise multiply

> x^z

> x%*%y # Dot (inner) product of two vectors.

> 1:100 # gives an answer that is several lines long

- R uses the notation '[n]' at the beginning of each output line to indicate which entries are shown on that line.

Practice 1

(a) Find the value of $\sqrt{32} * (-4)^3$.

(b) Assign a vector of 0.2,0.4,0.6,0.8 and 1 to object x. (There are several ways to answer this question.)

(c) Extract the 3rd and 5th elements of x.

(d) Extract all but the 2nd and 4th element of x.

(e) Assign 0 to the 1st element of x, what is the new x?

8. **Using functions on data vectors**

Example

> gpa.section.a=c(3.13, 3.55, 2.92, 2.73, 3.00, 3.18, 2.66, 3.76)

> gpa.section.b=c(3.14, 3.13, 3.25, 2.93, 3.73, 3.50, 2.70)

> all.gpa=c(gpa.section.a,gpa.section.b) # Combine (stack) data vectors

> all.gpa

> sum(all.gpa) # Returns the sum of vector elements

> length(all.gpa) # Returns the vector length (i.e., n)

> mean(all.gpa) # Returns the vector mean

> var(all.gpa) # Returns the sample variance

> sd(all.gpa) # Returns the sample standard deviation

> min(all.gpa) # Returns the minimum value

> max(all.gpa) # Returns the maximum value

> sort(all.gpa) # Returns the sorted vector elements

> summary(all.gpa) # Returns the five number summary and mean

9. **Data vectors have a "type"**

Numerical values The object named all.gpa contains *numerical* elements.

Character strings > Grade= c("freshman", "sophomore", "junior", "senior")

- *Character strings* are made with matching quotes (either double or single quotes). The numbers could be character strings

  > grade= c('1','2', '3', '4') # R is case-sensitive.

- One restriction on data vectors is that all values must be of the same type. If you try to mix data types within a vector, they will all be coerced to *character strings.*

  > Grade1= c("freshman", "sophomore", "junior", 4)

Logic *video* consists of 10 students' score for a certain video game.

  > video=c(47, 63, 58, 53, 53, 63, 53, 39, 58, 50)

- How can we let R answer "Whether a student's video game score is over 55"? In such cases, we can provide R with a *logical expression*, and R will check if and where the expression is true or false.

  > video> 55

  R recognizes video> 55 as "Is the value of each element greater than 55?"

  Logical operators

>	strictly greater than
>=	greater than or eqaul to
<	strictly less than
<=	less than or equal to
==	equal to # **Operator = is used for assignment"**
! =	not equal to

- **What happens if you apply a numerical function to a logical vector**

  The logical values **TRUE** and **FALSE** are coerced to numeric values of 1 and 0, respectively. Recall that the Bernoulli random variable assigns 1 and 0 to success and failure, respectively.

  > sum(video> 55)

  How can we interpret the output from the command sum(video> 55) ?

  > mean(video> 55)

  How can we interpret the output from the command mean(video> 55)?

- **When x is a data vector and vec is a logical vector of the same length as x, then x[vec] returns *only* the values of x for which vec's values are true.**

  ```
  > video[video> 55]
  ```

  How can we get R to calculate the mean of only those elements of scores that are above 55?

Practice 2

(a) Let R return video game score less than 52.

(b) Let R return video game score less than or equal to 53.

(c) Let R check if the mean of video game score is greater than 50.

(d) Another 9 students also played the game and their scores are as follows: c(47, 57, 47, 50, 55, 69, 26, 33, 56).
Suppose that if the score is higher than 58, then the student will receive a gift. How many of the total 19 students will receive a gift?

(e) For all the 19 students, find the variance of the video scores, for those with a video score below 53.

(f) The students' favorite flavor of ice cream information is also collected (vanilla, chocolate or strawberry). Use "V", "C" and "S" to represent the three flavors respectively. Find the proportion of students who like strawberry ice cream the most for the 19 students.

```
> flavor=c("C","V", "S", "S", "V", "V", "V", "V", "V", "V",
  "V", "V", "S", "V", "V", "V", "S", "C", "V")
```

10. **Save the code**

- **Working directory** R can read in data sets only in working directory. Make your own working directory. For example,

  (a) Create a folder entitled "STA404" (or "STA504") in your computer.

  (b) Within "STA404" (or "STA504") folder, create a folder entitled for example "week01".

  (c) "More" in the Files menu $\rightarrow$ Set As Working Directory, or Session in the menu $\rightarrow$ Set Working Direction $\rightarrow$ Choose Directory.

  Check your current working directory with the command

> getwd()

- **Workspace Management** When you open an R session, you are creating a workspace. The workspace is the collection of R objects you have created. If you quit R not saving workspace, you will lose all record of everything you did in the current session.

  To save a workspace:

  (a) You can see the objects in the current workspace in Environment window. Remove any objects you no longer want using **rm()**
  (b) In the Environment menu, click the save icon.
  (c) Choose a directory and then meaningful name the workspace of your current project i.e. **day02**, or use the command **save.image("day02.RData")**.

  To load a previously saved workspace:

  (a) Start R Studio.
  (b) In the Environment menu, click the load icon.
  (c) Find the .Rdata file you want to load, select it and open it, or use the command **load("day02.RData")**.

- When you quit R, you will be asked to save the workspace. If you save this, this will be automatically loaded the next time you start R. This is generally **not** recommended.

11. **Data frames**

- A statistical dataset in R is known as a **data frame**, and is a rectangular array of information.
- Rows of a data frame constitute different observations in the data, and the columns constitute different variables measured.
- R packages come bundled with many pre-entered datasets for exploration purpose.

  > iris

- How many variables are in the data frame?
- How many observations does each variables contain?
- The top row containing the variable names is known as the **header** in R.

- The command data() lets you see what datasets are currently in the search path.

12. **Referencing data from inside a data frame**

- Since data frames are two-dimensional, each element has a [**row,col**] index pair.
- We can access the entry in the ith row and jth column of a data frame **x** by calling **x[i,j]**

> \> dim(iris) # Check dimensions of the data frame **iris**
>
> \> names(iris) # List the name of the variables in the data frame

Important To make variables in a data frame accessible by name within the R session, first invoke the **attach()** function:

> \> attach(iris)
>
> \> iris[2,] # select the second row only
>
> \> iris[,4] # select the fourth column only
>
> \> iris[63,4] # selects the entry in the 63rd row, 4th column
>
> \> iris[c(1,7),] # selects rows 1 and 7
>
> \> iris[80:85,] # selects rows 80 through 85
>
> \> iris[,-c(1,2)] # select all data except columns 1 and 2

Practice 3 (a) Extract the 10th, 50th and 123th rows.

(b) Extract entries in the 10th, 50th and 123th rows, 2nd and 3rd column.

(c) If a data set does not have a header, how should we work with variables?

Variable extraction You may extract a variable form a data frame using the call dfname$varname

> \> Speciesvec=iris$Species

13. **Using logical reference in a data frame**

- We can select a subset of the original data frame using logical reference.

  Select a subset of data containing a puzzle score greater than 61.

  (a) What is the corresponding variable to the puzzle score?

  (b) Code the logical command.

(c) Where should we put the logical command? Row or Column? In other words, do we want to print out observations or variables?

- Recall When x is a data vector and vec is **a logical vector of the same length as x**, then x[vec] returns *only* the values of x for which vec's values are true.

  > iris[Sepal.Length>5,]
  > group1=iris[Sepal.Width>3,]

Practice 4 (a) Select a subset of data containing all observations corresponding to the Species equal to setosa, store them into any name of object you want.

(b) Compute the proportion of the setosa flowers whose Petal.Length is greater than 1.5.

- **Recommend additional resources**
  - An Introduction to R: `http://cran.r-project.org/doc/manuals/R-intro.pdf`
    - The pdf file is uploaded on our website.
  - Quick-R: `http://www.statmethods.net/`
    - Nice reference website with basic code for many tasks. Recommended.
    - Associated textbook available at `http://www.manning.com/kabacoff/`.