“In the Name of God"

Probabilities and Statistics **CE 40181**
Sharif University of Technology

Introduction to R
Due date: Wed, Oct 4,2017

Reminders:

- **Collaboration is permitted, but you must write solutions *by yourself without* assistance.**
- **Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.**
- **Late submissions will be treated according to the course policy.**

About R:

R is an Open Source language and environment for statistical analysis, graphics representation and reporting. R is a GNU project which is similar to the S language and environment was created by **R**oss Ihaka and **R**obert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

In the last ten years, due to its vast statistical and graphical facilities as well as its expressive and easy-to-use syntax, R has become arguably one of the most popular languages used by statisticians, data analysts, researchers and marketers to retrieve, clean, analyze, visualize and present data.

In this assignment we are going to get started with R. We will first get into R environment, then we will work on some basic syntax.

During this semester you are going to present your assignments in R.

Install R

R is cross-platform so you can write a code on one platform and can easily port it to another without any issues.

You can find distributions of R for all popular platforms - Windows, Linux and Mac at https://www.r-project.org/.

In order to provide a more conventional enviroment to use, you can employ IDEs. https://www.rstudio.com/products/rstudio/download/ is the first choice of the majority of R programmers.

Install your desired version of R & Rstudio.

(https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf)

Work directory

Set your work directory:

```r
setwd("/home/elihei") # to set your directory
getwd() # to check your current direcotry
```

R packages

R packages are collections of functions and data sets developed by the community. They strengthen R by improving existing base R functionalities, or by adding new ones. Every user can develop his or her own package and share it with the community.

There are severl ways to **install** an R package:

```r
# to name but a few!
# do not forget double qoutes!

#1. install.packages() --> the most popular one
install.packages("ggplot2") # one package
install.packages(c("stringr", "data.table", "vioplot"))
# more than one package

#2. Bioconductor packages --> if install.packages does not work
source("https://bioconductor.org/biocLite.R")
biocLite(c("readxl", "reshape2")) # mostly for biological data analysis

#3. Devtools --> if previous approaches do not work
install.packages("devtools")
library(devtools)
devtools::install_github("tidyverse/dplyr") # --> github repository
```

Update, remove, or check installed package

You will have several returns to your installed package in order to update, remove or check for installed packages:

```r
#4. installed.packages()
installed.packages()

#5. remove.packages() --> to remove a specific package
remove.packages("vioplot")

#3. old.packages()--> installed packages which need to be updated
old.packages()
```

```
#4. update.packages() --> to update old packages
update.packages()

#5. .libPaths() --> to get library location
.libPaths()
```

Load packages

After installing packages you have to load them to your project:

```
# to name but a few

#1. library() --> the most popular one |
# returns error when the package has not been installed
library(ggplot2)
library(plotrix)

#2. require() --> just like library but returns no error for the lack of
#the package
require(plotrix)

#3. packagename::funciton() --> when you call the package few times
ggplot2::geom_bar()
```

Getting help with R

"?" operator works on a function name form an installed package and gives a brief documentation of it.

"??" operator works on a function name and gives a brief documentation of it.

The funciton help() provides the official documentiation of a package or a fucntion.

A vignette is a detailed tutorial of a package, helping user about all functions in the package. Use it when you know a package will work for you but you don't know the exact function.

```
?grep # ? works only on installed functions
?? geoquery # ?? works on both installed and uninstalled functions
help("geom_line") # help() works on both packages and functions
vignette("stringr") # do not forget double qoutes!' |
# works only on packages
```

Basics

Arithmatic with R

R can be used as a simple calculator having C-like sysntax:

```r
2 + 2
10 - 5
2 * 3
241 / 7
24 %% 5
```

Variable assignment

There are two type of assignments in R:

```r
# <- assignment
a <- 6
b <- 4
c <- a * b # also can be writen a * b -> c

c # this will print value of c in the console

# = assignment
a = 25
b = 4
c = 25 / 4
c

# different types of assignment
a = 2.5
b = "ali"
c = TRUE

# class of variables
class(c)
```

Almost always programmers use "<-" to assign new values to the variables.

(https://renkun.me/blog/2014/01/28/difference-between-assignment-operators-in-r.html about the difference between "=" and "<-")

Data formats

Vector

A simple sequence of elements is called **vector** in R. The elements are called components; however, the community tend to call them members, so we do.

```
# combine method

#vectors must include same types
c(1 , 2, 3)
c("ali", "reza", "taghi")
# if they consist of different types R will converts them to one
c(1, "ali" , TRUE)

# we can call an exact element
a <- c (1, 3, 5, 7)
a[2]
a[c(1,3)] # indice also could be vectors | return indice 1 and 3 of a

# note that vectors in R are 1-indexed rather than 0-indexed
a[0]
a[1]

# length of vector
length(a)

# specific ways to create a vector
days <- 1:7
steps <- seq(1, 10, 1)
# seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)))


# assign names to elements of a vector
steps.reverse <- seq(10, 1, -1) # negative steps can be used

# assign names to elements of a vector
names(days) <- c("sataurday", "sunday", "Monday",
                 "Tuesday", "Wednesday", "Thursday", "Friday")
days
days["sunday"] # elements can be called using names
# instead of indice | don't forget ""
names(days)[1] <- "shanbe" # names() itsself is a vector
days[c("Monday","Tuesday")]

# arithmatics on same-length vectors
a <- c(1, 2, 3)
```

```r
b <- c(4, 5, 6)
c <- a + b
c
sum(c)
prod(c)

# logical selection
days > 4
days == 3
days != 2
days[days > 4] # it's a little thricky!
c <= 8
c[c <= 8]
```

Matrix

A matrix is a collection of data elements arranged in a two-dimensional rectangular layout. In other words, a matrix is a rearranged vector.

```r
matrix(1:9, nrow = 3) # arrange 1 to 9 in a matrix of 3 rows |
#fill the columns sequentially
my_matix <- matrix(seq(1, 99, 2), ncol = 10)

# call elements of the matrix
my_matix[2, 3]
my_matix[1,]
my_matix[2:3, 1:5]

# name columns and rows
my.matrix <- matrix(1:100, ncol = 10)
colnames(my.matrix) <- c("one", "two", "three", "four",
                         "five", "six", "seven", "eight", "nine", "ten")
rownames(my.matrix) <- c("one", "two", "three", "four",
                         "five", "six", "seven", "eight", "nine", "ten")
my.matrix

# cumulative funcitons
rows <- rowSums(my.matrix)
cols <- colSums(my.matrix)
rows
cols

# bind two matrices
cbind(my.matrix, cols)
rbind(my.matrix, rows)
```

```r
# arithmatic on matrix
my.matrix * 2
my.matrix + 2
my.matrix * my.matrix # element-wise multiplication
my_matix %*% my.matrix # regular multiplication of two matrices

# convert matrix to vector
c(my_matix)
as.vector(my.matrix)
```

Factors

While working with data one faces a wide variety of **categorial variables**, namely types of data taking discrete and limited number of values, e.g. gender (man or woman). To address that, R provides a class of variables called **factor**.

```r
color_vector <- c("Yellow", "Red", "Red", "Blue", "Yellow", "Green")
color_vector <- as.factor(color_vector)
color_vector <- factor(color_vector)

# levels of a factor vector, that is, different types of the category
levels(color_vector)

# summarizes a vector of factors
summary(color_vector)

# ordered vector of factors
temperature <- c("warm", "hot", "cold", "mild", "hot", "cold",
                 "hot", "mild")
temp <- factor(temperature, order = T, levels = c("cold", "mild", "warm",
                                                  "hot"))
```

Dataframe

Suppose that you are collecting data about soccer players, you may record the name, the age, the height, maybe the power, the nationality, etc. of the player. Formerly, we saw matrices which could only include values of a same class. Dataframes are the most popular type of data tables in R. R provides many specific utilities for dataframes. We will see dataframes many times during this course.

```r
# data tables could be of such several formats as .csv, .xls, .dat, .txt,
#etc.
my.dataframe <- read.csv("FullData.csv") # details about Fifa 2017 players!
```

```r
# head and tale
head(my.dataframe)
tale(my.dataframe)

# structure of the dataframe
str(my.dataframe)

# creating a dataframe | exmple from datacamp.com
# same-length columns
name <- c("Mercury", "Venus", "Earth", "Mars",
          "Jupiter", "Saturn", "Uranus", "Neptune")
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
          "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant",
          "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
planets <- data.frame(name, type, diameter, rotation, rings)
# create a dataframe with its columns


View(planets) # spreadsheet-style view of the dataframe

# selecting elements
planets[3,4]
planets[,1]
planets[2:4,3:5]
planets[-1,] # excludes the first line
planets[-(2:4),-(3:5)]
planets[,"rings"] # select by column name
planets$diameter # select by column name
subset(planets, subset = rings)
# select rows by condition - a column with boolean values

# Sorting
a <- c(-1,4,2,-7,10)
order(a) # sorts a vector |
# returns order of indice in the ordered vector
a[order(a)] # sorts a vector
order(planets$rotation)
```

Lists

A list is a generic vector which may contain elements of different types, namely numbers, strings, vectors, matrices, and even functions, to name a few. They look like simple vectors at first but you will see that they are amazingly useful!

```r
a <- c(1,3,4)
b <- c("Ali", "Naghi")
c <- c(T, F, F, T, F) # boolean vector
d <- list(a, b, c)


# selecting elements from a list
d[[2]]
d[[3]][2] # selects second element of third element of d
names(d) <- c("a", "b", "c") # assing name to elements
d$a
d[["b"]]

# list to vector
e <- unlist(d)
```

Flow controls

Conditionals

If and else

If and else in R is the same as in C or Java.

```r
# if (condition) {
#     # do something
# } else {
#     # do something else
# }
x <- 1:20
if (x[3] <= 10) { #only one condition
    print("x is less than 10")
} else {
    print("x is greater than 10")
}
#ifelse is new | supports one or more conditions
ifelse(x <= 10, "x less than 10", "x greater than 10")
# any and all
x <- c(TRUE, TRUE, FALSE)
y <- c(FALSE, FALSE, FALSE)
if (any(y)){
  print("at least one element of y is true")
}else{
  print("all elements of y are false")
}
```

```
if (all(x)){
  print("all elements of x are true")
}else{
  print("at least one element of x is false")
}
```

Loops

For loops

The syntax of "for" loops are simple in r. The condition is evaluated by checking if a variable is in a sequence and if the condition is met, the following expression will be executed.

```
# for (variable in sequence) {
#     expression
# }

for (x in 1:10)
  print(x+2)

#from https://www.datacamp.com/
mymat <- matrix(nrow=30, ncol=30)
# For each row and for each column,
# assign values based on position: product of two indexes
for(i in 1:dim(mymat)[1]) {
  for(j in 1:dim(mymat)[2]) {
    mymat[i,j] = i*j
  }
}
head(mymat)
```

While loops

"While" syntax in R is the same as in C or Java.

```
# while(condtion){
#   expression
# }

#to check if 149 is a prime number
my.num <- 149
i <- 1
j <- 1
while(j != 0 && i < my.num){
  i <-  i + 1
  j <- my.num %% i
}
print(j)
```

Repeat loops

Repeat loops are like while loops with the condition been set to "True". Instead, ending of the loop is handled using "break".

```
x <- 0
repeat {
  x <- x + 1
  if (10 <= x) {
    print("the loops is over")
    break
  }
}
```

The *apply* family

The apply() family includes functions to manipulate matrices, arrays, lists and dataframes in an iterative manner. These functions allow iterating a number of times without explicitly using loops. They operate on an input list, matrix or array and apply a built-in or user-defined function with one or several optional arguments.

The called function could be:

-An aggregating function, like for example the mean, or the sum (that return a number or scalar);

-Other transforming or subsetting functions; and

-Other vectorized functions, which return more complex structures like lists, vectors, matrices and arrays.

apply

Takes an array and applies the function on a dimention:

```
# iris dataframe
summary(iris)

apply(X = iris[,1:4], MARGIN = 2, FUN = mean)
apply(iris[,1:4], 1, function(x) x**2)
```

lapply

Takes a list and applies a function on each element and returns a list:

```r
data <- list(x = 1:5, y = 6:10, z = 11:15)
lapply(data, FUN = sum)
```

sapply

Takes a list and applies a function on each element and returns a vector:

```r
data <- list(x = 1:5, y = 6:10, z = 11:15)
sapply(data, FUN = sum)
```

mapply

Takes more than one list or vector and applies a function stepwise:

```r
x <- 1:5
b <- 6:10
mapply(sum, x, b)

mapply(rep, 1:4, 4:1)
```

tapply

Splits the array based on specified data, usually factor levels and then applies the function to it:

```r
# mtcars dataframe
library(datasets)
tapply(mtcars$wt, mtcars$cyl, mean)
```

Function

```r
# func_name <- function (argument) {
#    statement
# }

pow_sum_1 <- function(x, y) {
   result <- x**y
   result2 <- x + y
   # result
   c(result ,result2)
}
pow_sum_1(y = 2, x = 8)

pow_sum_2 <- function(x, y=2) {
   result <- x**y
   result2 <- x + y
   # result
   c(result ,result2)
}
pow_sum_2(x = 8)

f <- function(a, b = 1, ...) {
    s = sum(a, ...)
    return(s ** b)
}
f(2, 5, ... = 10)
```

1. Read the file "FullData.csv" to my.dataframe.
2. Print the dimentions and names of columns of my.dataframe.
3. Sort the players by their height.
4. Print different types of position.
5. Print the names of Italian players.
6. Print the mean of numeric columns.
7. Print the names of plyers with power between 70 and 90.
8. Write a function returning the top 100 plyers in dribling.
9. Create a list containing details about Spanish palyers including the number of players, mean power, vector of different categories of positions, vector of birth days.
10. Write a function that returns the mean power of players of each nationality. (hint: use tapply funciton)
11. Print the names of most loyal players to their clubs.

Congrats! You have finished the first assignment. Hope you enjoy the assignment.