Tutorial and Laboratories

11520G Data Capture and Preparations

Week 3

Introduction

In this tutorial will continue practising basic operations in R. In particular, you will learn about vectors and data frames in R; how to create them, access their elements and modify them in your program.

Skills Covered in this tutorial include:

- Getting Help
- Data Structures
- Obtaining specific information from data frames
- Conditionals
- Loops

Note: Do not copy-paste the commands. As you type each line, you will make mistakes and correct them, which make you think as you go along. Remember, that the objective is that you understand the commands and master the concepts, so you can reproduce their principles on your own later.

1. Getting Help

it's generally a good idea for you to try to help yourself. There is a comprehensive built-in help system in R and learning to navigate them will help you significantly in your programming efforts.

Exercise 1. At the program's command prompt, try the following commands and observe their purpose.

```
help.start()          # general help
help(plot)            # help about function plot
?plot                 # same thing
apropos("plot")       # list all R objects with a name containing the string "plot"
example(plot)         # show an example of function plot
RSiteSearch("plot ")  # searches for the string plot in online help manuals and mailing lists
```

1.1. R Help on the Internet

There are internet search sites that are specialized for R searches, including http://search.r-project.org/ (which is the site used by RSiteSearch) and https://rseek.org/.

It is also possible to use a general search site like Google, by qualifying the search with "R" or the name of an R package (or both). It can be particularly helpful to paste an error message into a search engine to find out whether others have solved a problem that you encountered.

Also, Stack Overflow is a well-organized and -formatted site for help and discussions about programming. It has excellent searchability. Topics are tagged, and "r" is a very popular tag on the site. To go directly to R-related topics, visit: http://stackoverflow.com/questions/tagged/r

2. Data Structures

R has five basic data structures: atomic vectors, matrices, arrays, lists, and data frames. These structures have specific requirements in terms of their dimension.

- One dimension: Atomic vectors and lists
- Two dimensions: Matrices and data frames
- N dimensions: Arrays

For today's lab, we will practice using vectors and data frames only.

2.1. Vectors

These are the basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex. A vector's type can be checked with the *typeof()* function. Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function *length()*.

Exercise 2. Create atomic vectors with different data types and observe their type and length.

```
int_var <- c(10L, 2L, 5L)
num_var <- c(0.4, 3.7, 2)
typeof(int_var)
length(int_var)
coe_var <- c(5L, 3.5, “A”)
typeof(coe_var)
animals <- c("mouse", "rat", "dog", "bear")
x <- seq(0, 10, 2)
y <- 2:-2;
```

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector. Note: Vector index in R starts from 1, unlike most programming languages where index start from 0.

```
x[3]          # access 3rd element
x[c(2, 4)]    # access 2nd and 4th element
x[-1]         # access all but 1st element
x[c(2, -4)]   # cannot mix positive and negative integers
x[c(2.4, 3.54)]   # real numbers are truncated to integers
```

Using character vector as index. This type of indexing is useful when dealing with named vectors. We can name each elements of a vector.

```
x <- c("first"=3, "second"=0, "third"=9)   #create vector
names(x)                    #print names of each element in the vector
x["second"]                 #access value of “second” element
x[c("first", "third")]      #access the 1st and 3rd element
```

2.2. Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame:

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

We can create a data frame using the *data.frame()* function.

Exercise 3. Create the data frame df from a group of three vectors n, s, b.

```
n = c(2, 3, 5)
s = c("aa", "bb", "cc")
b = c(TRUE, FALSE, TRUE)
df = data.frame("var1"=n, "var2"=s, "var3"=b, stringsAsFactors=FALSE)      # df is a data frame
str(df)              # check structure of df
```

Note: By default, when building a data frame, the columns that contain characters are converted into factors. If you want to keep that columns as characters, you need to pass the argument stringsAsFactors=FALSE

You can also visualize the structure of the data frame by clicking on its name in the Environment Pane. The top line of the table, called the header, contains the column names (variable names). Each horizontal line afterward denotes a data row, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a cell.

We can use either [, [[or $ operator to access columns of data frame. Practice using these operators to access data from a data frame and observe the results.

```
df[[1]]              # access data in column 1
df$var2              #access column 2 by name, returns a vector
df[2]                #access data in column 2, returns a data frame
sum(df$var1 == 5)          #returns the number of observations in variable 1 that are equal to 5
max(df$var1)               # returns the maximum value in variable 1
which(df$var1==3)          # return the element (not the value) in var1 == 3
df[c("var1", "var3")]      #access column1 and column 3 simultaneously
df[2,1]                    #access cell (2,1)
rownames(df)               #returns the name of each row
df$var4 <- c(1.5, 3.5, 5.5)  # add variable named "var4"
df <- rbind(df,list(7, "dd",FALSE,7.7))  # add a row
df$var3 <- NULL            #delete variable 3
df <- df[-1,]              #delete first row
```

As a data scientist/analyst you will be working mostly with data frames. And thus, it becomes important that you learn, understand, and practice data manipulations related to data frame.

Exercise 4. Use built-in data frames in R. You can get the list of all the datasets by using *data()* function. For this exercise, we will use a built-in data frame called **mtcars** (for a complete description of the dataset visit: https://rpubs.com/neros/61800). Have a look at the following instructions.

```
rm(list=ls())
data()                  # get the list of built-in data frames
data("mtcars")          #select mtcars data frame
head(mtcars)            #visualise only the first 6 rows
```

Now, imaging that you are required to obtain and visualize some specific information from the mtcars data frame.

- How big is the data frame? Use the function dim() or the functions nrow() and ncol().
- What is the cell value from the first row, second column of mtcars?
- Could you get the same value by using row and column names instead? Which names?
- Make a histogram of mpg(miles per gallon) in mtcars, use hist(). Which bin contains the most observations?
- Make a histogram of mpg in mtcars with 20 breaks, (hint: see the available options for the hist() function with ?hist(). Hint2: pass the argument 'breaks =').
- Make a boxplot of mpg in mtcars, use boxplot().
- Make a boxplot for each number of cylinders, use: boxplot(mtcars$mpg ~ mtcars$cyl)
- Make a scatterplot of mpg in mtcars using plot()
- Plot horsepower (hp) versus mpg using plot (hint: plot(x, y))
- Are there more automatic (0) or manual (1) transmission-type cars in the dataset? Hint: use the sum() function to sum each type of transmission in the am (automatic/manual) variable.
- Get a scatter plot of weight vs horsepower (hp) (hint: plot(x, y)). Do you see any relationship with these two variables? Explain your observations.

3. Conditionals in R

Conditionals are expressions that perform different computations or actions depending on whether a predefined boolean condition is TRUE or FALSE. Conditional statements include *if(),* the combination *if()else, else if()*, and *switch()*.

The *if* Statement. Decision making is an important part of programming. This can be achieved in R programming using the conditional if...else statement. The *if()* statement performs operations based on a simple condition:

```
if(condition) {
   true.expression
 } else {
   false.expression
}
```

<u>Exercise 5</u>. We want to print the name of a team that won using an if statement.

```
 team_A <- 3 # Number of goals scored by Team A
 team_B <- 1 # Number of goals scored by Team B
if (team_A > team_B){
  print ("Team A wins")
}
```

So we need to add a block of code that runs if our conditional expression team_A > team_B returns FALSE. We can do this by adding an *else* statement. If our comparison operator evaluates to FALSE, let's print "Team B will make the playoffs."

```
team_A <- 1 # Number of goals scored by Team A
team_B <- 3 # Number of goals scored by Team B
if (team_A > team_B){
    print ("Team A will make the playoffs")
 } else {
    print ("Team B will make the playoffs")
 }
```

4. *for* loop in R

Loops are used in programming to repeat a specific block of code. In this tutorial, a for loop is used to iterate over a vector in R programming. Syntax of for loop:

```
for (val in sequence) {
            statement
}
```

Exercise 6. We need to make a function that prints the square of the number in the following vector: numbers (1,2,3,4,5,6).

```
numbers <- c(1,2,3,4,5,6)
for(i in numbers){
  print(i^2)
}
```

Now, we need to iterate over all the elements of a vector and print each value.

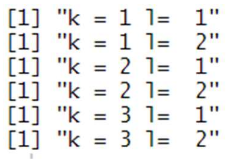
```
# Create fruit vector
fruit <- c('Apple', 'Orange', 'Passion fruit', 'Banana')
# Create the for statement
for ( i in fruit){
 print(i)
}
```

We need to count the number of elements in the vector.

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
   count = count+1
}
print(count)
```

Let’s try a nested for loop and see how the indices change over each iteration.

```
for (k in 1:3) {
   for (l in 1:2) {
         print(paste("k =", k, "l= ",l))
    }  #close first loop
}  #close second loop
```



```
[1] "k = 1 l=  1"
[1] "k = 1 l=  2"
[1] "k = 2 l=  1"
[1] "k = 2 l=  2"
[1] "k = 3 l=  1"
[1] "k = 3 l=  2"
```

Exercise 7. Let's use a *for* loop to name variables dynamically. For this exercise, we will use a *list* to store the values to be assigned to each variable.

```
my_list <- list(1:5, "ABCD", 9:5)
print(my_list)
```

In this case, the three elements in the list don't have a name. Let's create a new variable containing the values of the first element in the list. We will use the *assign* function to do this.

```
assign("variable_1", my_list[[1]])        # Apply assign function
print(variable_1)
```

An advantage of using the *assign* function is that we can create multiple variable names based on a character string. So, now let's try to create multiple variable names using a *for* loop.

```
for(i in 1:length(my_list)) {                # assign function within loop
  assign(paste0("variable_", i), my_list[[i]])
}
```

Print the variables and see the result.

NOTE: we don't create variables dynamically, instead we can store all the elements in a list. However, in some cases, it might be useful.

Solutions to exercise 4:

```
dim(mtcars)                  # dimension of data frame (rows, columns)
nrow(mtcars)                 #number of rows
ncol(mtcars)                 #number of columns
mtcars[1,2]
mtcars["Mazda RX4", "cyl"]       # using row and column names instead
hist(mtcars$mpg)                 # make a histogram of mpg(miles per gallon) in mtcars
hlst(mtcars$mpg, breaks = 20)    # make a histogram of mpg in mtcars with 20 breaks
boxplot(mtcars$mpg)              # make a boxplot of mpg in mtcars
boxplot(mtcars$mpg ~ mtcars$cyl)         # make a boxplot for each number of cylinders
plot(mtcars$mpg)                 # make a scatterplot of mpg
plot(mtcars$hp, mtcars$mpg)      # plot horsepower (hp) versus mpg
sum(mtcars$am == 1)              # get total number of manual transmission-type cars
sum(mtcars$am == 0)              # get total number of automatic transmission-type cars
plot(mtcars$wt, mtcars$hp, col = "red", xlab = "Car Weight in 1K lbs", ylab = "Horsepower" )
```