Introduction to R

Malachy Campbell

9/2/2020

Summary

The goal of this exercise is to allow students to become familiar with R programming and to generating reproducable research reports with R markdown. This will not be graded. For those that have some experience with R and R markdown, feel free to skip this.

Downloading and installing R

R can be downloaded from the CRAN database. I would also recommend installing R studio, which is a desktop application that makes R a bit more user friendly.

Downloading R and Rstudio These quick steps should be enough to get you started:

1. Go to the CRAN database.
2. Click download R for X, where X is your operating system.
3. Download the latest version.
4. Double click and follow the installation directions.

To download Rstudio (GUI):

1. Head over to the Rstudio webpage
2. Click Products and select Rstudio under the Open Source tab.
3. Click Rstudio desktop, download and install.

Getting started with R

R is an object-oriented langauge. This means that we can store data in objects and perform opertations on those objects. In R, `=` and `<-` can be used to store variables in the object.

For instance:

```
y = 4
y + 17
```

```
## [1] 21
```

We can store data in vectors using the `c` function, and as above we can perform operations or run functions on these vectors.

For instance:

```
y = c(2,5,6,7,3)
y + 3
```

We can return the values stored in the array by typing the object in the command line and hitting enter, or in R studio by placing the cursor at the end of the line and holding down the command key and pressing enter. Operations can be performed on the object without affecting the values stored in the object.

```
y
```

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

```
print(y)
```

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

```
mean(y)
```

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

Any manipulation is not permanent unless the original object is replaced with the manipulated object or stored in another object. Note that R is case-sensitive. Missing data is indicated as NA (by default) in R – not na, Na, or #N/A. We can use `rm()` to remove the object from memory. Any non-numeric variable must be quoted, otherwise R will think it is an object. Exceptions are T, F, TRUE or FALSE.

```
new_y <- y

y <- y + 2

print(y)
```

```
## [1] 6
```

```
print("y")
```

```
## [1] "y"
```

```
print(new_y)
```

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

```
print(Y)
```

```
## Error in print(Y): object 'Y' not found
```

```
rm(new_y)
```

There are a few types of data structures in R

- Vectors: collection of individual variables (numbers, letters, strings, factors) of the same type.
- Matrices: Two-dimensional collection of vectors of the sample type.
- Data frames: These are like excel spreadsheets. Columns must consist of the same data type.
- Lists: Lists are a collection of objects. The elements can be any of the data types above as well as indivudal variables.

Since I'm introducing a new function (`matrix`), you can access the help page for any function using `?` followed by the function name (e.g. `?matrix()`).

```
simple_numeric_vector <- c(1,2,3,5)
string_vector <- c("This", "is", "also a vector")

simple_matrix <- matrix(c(1,2,4,5), nrow = 2, ncol = 2, byrow = T)

simple_data.frame <- data.frame(ID = c("A", "B", "C"), X = c(1,2,4), Y = c(29,65,NA))

simple_list <- list(NumVect = simple_numeric_vector,
```

```
                    StrVect = string_vector,
                    Matrix = simple_matrix,
                    DF = simple_data.frame)
```

You can use the `str()` function to determine the structure of an object.

```
str(simple_numeric_vector)
```

```
##  num [1:4] 1 2 3 5
```

```
str(string_vector)
```

```
##  chr [1:3] "This" "is" "also a vector"
```

```
str(simple_matrix)
```

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

```
str(simple_data.frame)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ ID: Factor w/ 3 levels "A","B","C": 1 2 3
##  $ X : num  1 2 4
##  $ Y : num  29 65 NA
```

```
str(simple_list)
```

```
## List of 4
##  $ NumVect: num [1:4] 1 2 3 5
##  $ StrVect: chr [1:3] "This" "is" "also a vector"
##  $ Matrix : num [1:2, 1:2] 1 4 2 5
##  $ DF     :'data.frame': 3 obs. of  3 variables:
##   ..$ ID: Factor w/ 3 levels "A","B","C": 1 2 3
##   ..$ X : num [1:3] 1 2 4
##   ..$ Y : num [1:3] 29 65 NA
```

We can convert between data types (character, factor, numeric, interger).

```
as.factor(string_vector)
```

```
## [1] This          is            also a vector
## Levels: also a vector is This
```

```
as.factor(simple_numeric_vector)
```

```
## [1] 1 2 3 5
## Levels: 1 2 3 5
```

```
as.character(simple_numeric_vector)
```

```
## [1] "1" "2" "3" "5"
```

```
as.numeric(string_vector)
```

```
## Warning: NAs introduced by coercion
```

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

```
as.numeric(c("A", "B", "D", "E"))
```

```
## Warning: NAs introduced by coercion
```

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

Notice that R does not like it when we try to convert some data types to others.

Accessing the elements for these objects can be done as follows:

```
# For vectors
## Access the first element of simple_numeric_vector
simple_numeric_vector[1]
```

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

```
## Access the first and fourth elements of simple_numeric_vector
simple_numeric_vector[c(1,4)]
```

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

```
## Access the first three elements of simple_numeric_vector
simple_numeric_vector[1:3]
```

```
## [1] 1 2 3
```

```
# In a matrix or data frame (two-dimensional data types) elements can be access using '[x,y]'
## Access the first element of the matrix
simple_matrix[1,1]
```

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

```
## The second element (by row)
simple_matrix[1,2]
```

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

```
## The first column
simple_matrix[,1]
```

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

```
## The second row
simple_matrix[2,]
```

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

```
# Data frames
## The first element of the data frame
simple_data.frame[1,1]
```

```
## [1] A
## Levels: A B C
```

```
## The third row
simple_data.frame[3,]
```

```
##   ID X  Y
## 3  C 4 NA
```

```
## Second and third rows
simple_data.frame[2:3,]
```

```
##   ID X  Y
## 2  B 2 65
## 3  C 4 NA
```

```
## The first column
simple_data.frame[,1]
```

```
## [1] A B C
## Levels: A B C
```

```
## Or we can use the column names
simple_data.frame$ID
```

```
## [1] A B C
## Levels: A B C
```

```
# Lists
## The first element of the list
simple_list[[1]]
```

```
## [1] 1 2 3 5
```

```
## We can also do this by name
simple_list$NumVect
```

```
## [1] 1 2 3 5
```

We can manipulate these objects using similar operations.

```
simple_data.frame
```

```
##   ID X  Y
## 1  A 1 29
## 2  B 2 65
## 3  C 4 NA
```

```
simple_data.frame$ID <- c("D", "E", "K")

simple_data.frame
```

```
##   ID X  Y
## 1  D 1 29
## 2  E 2 65
## 3  K 4 NA
```

```
# Drop the ID column
simple_data.frame$ID <- NULL

simple_data.frame
```

```
##   X  Y
## 1 1 29
## 2 2 65
## 3 4 NA
```

```
# Vectors
simple_numeric_vector
```

```
## [1] 1 2 3 5
```

```
simple_numeric_vector[4] <- 2560
simple_numeric_vector
```

```
## [1]    1    2    3 2560
```

We can provide and manipluate the names of columns and rows in data frames, or names of elements in a list or vector.

```r
colnames(simple_data.frame)
```

```
## [1] "X" "Y"
```

```r
colnames(simple_data.frame) <- c("newX", "newY")

simple_data.frame
```

```
##   newX newY
## 1    1   29
## 2    2   65
## 3    4   NA
```

```r
row.names(simple_data.frame) <- c("a", "b", "d")

row.names(simple_data.frame)
```

```
## [1] "a" "b" "d"
```

```r
names(simple_numeric_vector)
```

```
## NULL
```

```r
names(simple_numeric_vector) <- c("D", "Q", "R", "F")

names(simple_list)
```

```
## [1] "NumVect" "StrVect" "Matrix"  "DF"
```

```r
names(simple_list)[3] <- "Hi"

names(simple_list)
```

```
## [1] "NumVect" "StrVect" "Hi"      "DF"
```

Reading and writing data

Data can be loaded from a bunch of different file formats. Most common are comma seperated files (.csv), white space delimited files (.txt) or Excel files. I would not recommend using loading Excel files into R. These are often problematic because they may contain formulas or other characters the R does not like. R objecrs can be easily saved and loaded as .Rds objects.

If the full path is not explicitly provided R will look in the default working directory.

```r
# Saving a list
saveRDS(simple_list, "~/Downloads/list.rds")

# Saving a data frame as a .csv file
write.csv(simple_data.frame, "~/Downloads/df.csv", row.names = F)

foo <- readRDS("list.rds")
```

```
## Warning in gzfile(file, "rb"): cannot open compressed file 'list.rds', probable
## reason 'No such file or directory'
```

```
## Error in gzfile(file, "rb"): cannot open the connection
```

```r
getwd()
```

```
## [1] "/Users/malachycampbell/Documents/Dropbox/Work/NRT_AdvStat/Exercises"
```

```r
foo <- readRDS("~/Downloads/list.rds")
```

The working directory can be changed using `setwd("Path/To/Working/Direct)`

Writing reproducible reports

Reproducible reports can be generated pretty easily using Rmarkdown. In fact, this document was generated using Rmarkdown (see the .Rmd file). Most of what you will need for this course is how to include code chunks and plots in the report.

The basic workflow for R markdown is to write all your code, tell R which code should be included and evaluated in the report, and to 'knit' the report together. When the report is being 'knitted' togther, all code that was supposted to be evaulated will be run (beware because this will over write files if you tell R to write a file), and if successful a pdf or html report will be generated.

Code chunks

Code chunks can be inserted by starting the code with a line ```` ```{r} ` and ending the line with ``. The three ` indicates that some code is coming, and the r " says that it should be evaluated in R.

```` ```{r, echo = T, eval = F} ```` R code that should be included in the report and evaluated/run.

```` ```{r, echo = T, eval = F} ```` R code that should be included in the report and but not run.

```` ```{r, echo = F, eval = F} ```` R code that should be note be included in the report and not evaluated.

Formatting

Formatting is relatively simple and is explained in the Rmarkdown cheatsheet.
````