REM Tutorial 1: Introduction to R – Workbook

Prof. Dr. Kathleen Kürschner Rauck

AT 2023

Contents

Getting Started

We will be working with R and RStudio in the practical exercises. **R** is an open source programming language, widely used for statistical programming and to produce (outstanding) graphics. **RStudio** is an integrated development environment (IDE) for R, i.e., a companion to R language, which works on top of R to provide a very accessible user interface.

Open up RStudio (click **Start** and type **RStudio** or double-click the icon on the desktop).

Place your cursor in the pane *Console* to use RStudio as a calculator. To try some basic built-in functions enter the following and hit enter:

```
7 + 8
```

Don't worry about the [1] for now. Note that RStudio printed out 15, since this is the solution to the sum you typed in. In the workbooks that I will provide for you, results of the code you enter into RStudio are displayed in the format shown below:

```
## [1] 15
```

Other examples of built-in functions are *, - and / for multiplication, subtraction and division, respectively. Enter:

```
5 * 200
```

```
## [1] 1000
```

```
16 - 4
```

```
## [1] 12
```

```
3 / 8
```

```
## [1] 0.375
```

You can also assign specific values or the results of calculations to variables using the `<-` symbol, which means *assign the value on the RHS to the variable (vector) on the LHS*. It's typed with a < followed by a -. Type:

```
rent <- 1000
```

Here the value 1,000 is stored in the variable *rent*. Variables are shown in the window labelled *Environment*, in the *Workspace* pane. Analogously, you can type:

```
rent <- 5 * 200
```

Here you obtain the same result (rent is 1,000) as in the previous command line, however, the value assigned to rent is essentially the outcome from a calculation you have asked your computer to execute. To query which value it has stored in variable *rent* type:

```
rent
```

```
## [1] 1000
```

As you can see, using R language your computer has calculated and assigned the correct value to variable *rent*. Variables can also be used in subsequent calculations. For instance, to apply a 10 % discount to the rent, enter the following:

```
rent - rent * 0.1
```

```
## [1] 900
```

or introduce intermediate variables to obtain the result:

```
discount <- rent * 0.1
rent - discount
```

```
## [1] 900
```

Next, we can move on to solve some practical problems provided in the first problem set (*Tutorial 1 - Problem Set*). Doing so, we will not enter commands directly in the *Console* but develop our code in an *R Script*, which is a file type that can be generated in RStudio and saved, enabling you to re-run your code or continue working on it at a later stage.

To start working on your script, place your cursor in the *Editor* pane (note: if the *Editor* pane is empty go to **File > New File > R Script** to set up a new script file).

Organise your work space before attempting to run any code, i.e., create a working folder in a place where you can find it easily and store all datasets etc. that you may need. In this tutorial, we will be working with the **ACC_data** I uploaded on Canvas.

To set your working directory to the working folder you have just created, got to **Session > Set Working Directory > Choose Directory** and browse through the file system or type `setwd("PASTE PATH TO YOUR WORKING DIRECTORY HERE")`, but note that R requires forward slash (not backslash) to read the path, e.g.:

```
setwd("C:/Users/USERNAME/Documents/REModelling/Tutorial 1/R_WORK")
```

Click on the *Files* tab in the bottom right pane to verify the directory has been set correctly. Go to **Files > click the More icon > Go To Working Directory** (FYI: You can also set the directory directly from the bottom right pane (and generate folders in there). Go to **Files > navigate to correct directory > click the More icon > Set As Working Directory**)

Next, save your code as an R Script (this is the source file of your project) in your working directory by clicking the **save current document** icon (top left pane) or typing **Ctrl+S**. Note: to continue working at a later stage, you can open the source file directly by double-clicking on it in your working folder. (Side note: because you now have a source file saved in your folder, you could also set the working directory by going to **Session > Set Working Directory > To Source File Location**.)

Practical 1: Downloading and Plotting Data

a. Download the Swiss Market Index (financial data) from *Yahoo finance* and plot the series.

To do this we need to install and load a package called `quantmod` into R. To install a package you type `install.packages("NAME OF PACKAGE")` (needs to be only once done on your machine) and to load a package you type `library(NAME OF PACKAGE)` (needs repeating in each working session). (Note: Required package dependencies are usually automatically installed. For instance, if you are working with R for the first time, the dependencies `xts`, `zoo`, `TTR`, `curl`, and `jsonlite`, are also automatically installed in order to get the `quantmod` package to work.)

Packages contain lots of useful functions. To learn more about any package or function you can type a question mark followed by the name of the object of interest. For example, type:

```
?quantmod
```

The help file appears in the right bottom pane of R Studio (not displayed in this workbook).

We can load data via the function `getSymbols()` of the `quantmod` package.

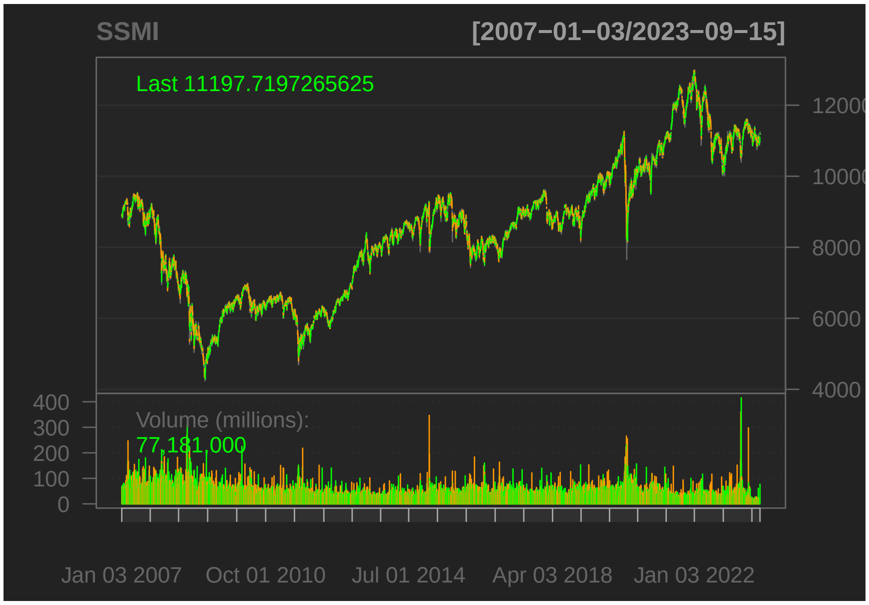
```
?getSymbols
```

The default source of this function is *Yahoo finance*, so we do not need to specify a source. However, we do need to specify the ticker (short code) used on *Yahoo finance* for the data we want to download from the web. The ticker for the *Swiss Market Index* is **^SSMI**. To download and plot the data type:

```
getSymbols("^SSMI") # downloads the series and stores it to an R object, called SSMI
```

```
## [1] "^SSMI"
```

```
chartSeries(SSMI) # plots the series
```

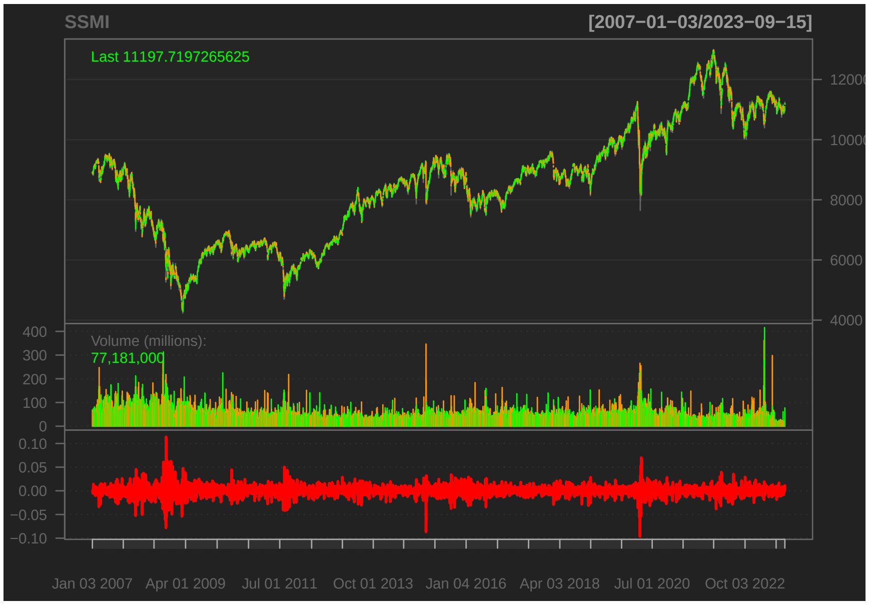


Note that you can use the symbol `#` to comment in your R script. The function `chartSeries()` plots financial time series data nicely.

b. Add the *rate of change indicator* of the series to the plot.

You can add something to the chart. For example, the *moving average convergence divergence* of the series (`addMACD()`) or its *rate of change indicator*:

```
addROC()
```



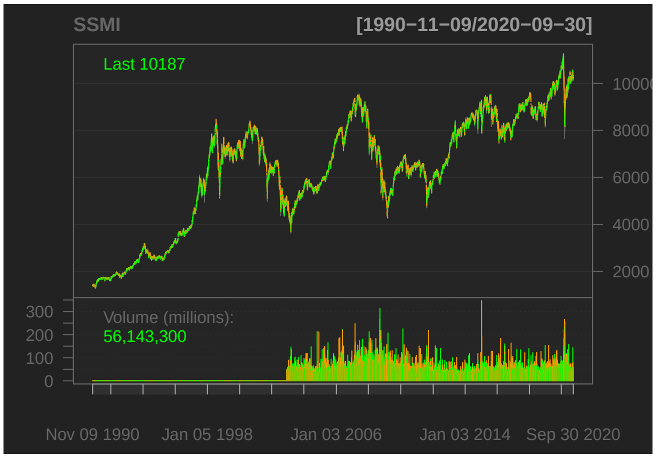
c. Download and plot (daily) financial data throughout the years 1990 and 2020.

You can also define the time period you would like to load and plot. To solve this problem, you can simply repeat step a. for a selected/extended time period (here: years 1990 through 2020).

```
getSymbols("^SSMI", from = "1990-01-01", to = "2020-10-01")
```

```
## [1] "^SSMI"
```

```
chartSeries(SSMI)
```



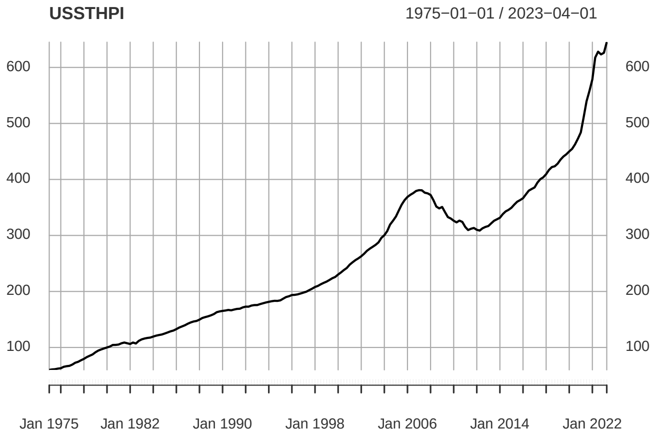
d. Download the All-Transactions House Price Index for the United States (real estate data) from *Federal Reserve Economic Data (FRED)* and plot the series.

Since we are downloading data from a different source than the default, we need to specify FRED as the source. The ticker for the data we want to load this time is **USSTHPI**.

```
getSymbols("USSTHPI", src = "FRED") # All-Transactions House Price Index for the United States
```

```
## [1] "USSTHPI"
```

```
plot(USSTHPI)
```



This time we used the `plot()` function, which is just another option to plot a series. Try `chartSeries()` as well. Then try the `plot()` function on the **SSMI** data. Can you produce a similar plot, as you did when using the `chartSeries()` function, for the **SSMI** data (hint: revisit this task after you have completed practicals 2 and 3)?

Practical 2: Data Handling and Indexing

a. Which class does the R object USSTHPI belong to?

Before we provide the solution to this problem, we revisit how we can generate an R object and learn how to determine its class. For example, we can generate a vector (variable) x, assigning a bunch of numbers to it using the `c()` function.

```
x <- c(2,4,6,1,8) # c() combines numbers
x
```

```
## [1] 2 4 6 1 8
```

Calling x displays the values of its elements. The function `class()` allows us to determine which class the R object x belongs to.

```
class(x)
```

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

We can also generate a variable y that contains characters by placing them into quotation marks:

```
y <- c("a","q","w","r","t")
class(y)
```

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

The class character is for text characters (string variables). Knowing how to determine the class of an R object is important, since commands may work differently on different classes or they may work on some classes but not on others.

The class of the R object **USSTHPI** is:

```
class(USSTHPI)
```

```
## [1] "xts" "zoo"
```

xts and **zoo** are time-series classes. Recap, in Practical 1 d.:

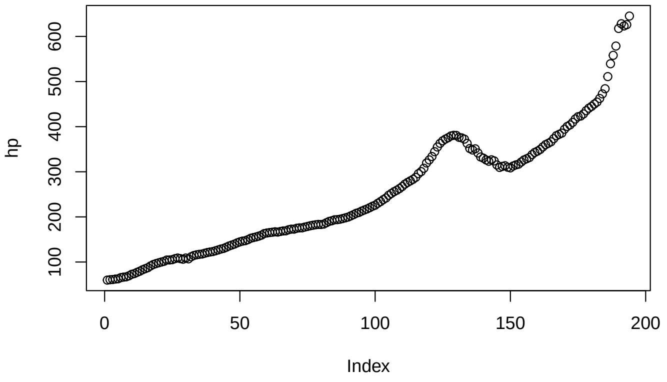
```
plot(USSTHPI)
```

created a line plot with time axis.

b. Generate an R object of class numeric, containing the same data as USSTHPI.

We translate the time-series R object **USSTHPI** to the numeric R object, called **hp** and plot it.

```
hp <- as.numeric(USSTHPI)
plot(hp)
```



See that the same command `plot()` on same data yields different results because the class of the object is different.

Note, also that *full stop* often appears in R language without a specific meaning or built-in function (cf. Stata) and there is also no problem to use it in the middle of variable names.

c. Generate a data frame containing three variables *x*, *y* & *z* with five elements each. Variable *x* is numeric, *y* is of class character and the elements of *z* should be randomly drawn from the standard normal distribution. Label the variable *y* *"Name"*, variable *x* *"ID"* and variable *z* *"Return"*.

A data frame is an R object. It is a way of storing data, where all the relevant items are stored together as a set of columns (like an internal spreadsheet).

To solve this problem, we can employ the variables, x and y, we previously generated. However, before we can create the data frame, we need to generate another variable, z, which contains five numbers that are randomly drawn from a normal distribution (with mean = 0 & std. dev. = 1). This can be done with the `rnorm()` function, specifying how many numbers you would like your computer to draw in parentheses:

```
z <- rnorm(5)
z
```

```
## [1] -0.8712102  0.5752565 -0.1245797  0.2149219  0.1628263
```

Next, using the function `data.frame()`, we can combine variables x, y and z into a data frame, called **dat1**:

```
dat1 <- data.frame(x, y, z)
```

Call **dat1** to display the content of this R object in the *Console*:

```
dat1
```

```
##   x y          z
## 1 2 a -0.8712102
## 2 4 q  0.5752565
## 3 6 w -0.1245797
## 4 1 r  0.2149219
## 5 8 t  0.1628263
```

Alternatively, you can click on the R object listed in the *Environment* window, which will show the data in a new tab. Note that this is equivalent to entering:

```
View(dat1)
```

We should also change the labels of the three variables. To accomplish this, we generate a new data frame, called **dat2**, label the three columns accordingly, and fill them with the data contained in the corresponding variables x, y and z:

```
dat2 <- data.frame(ID = x, Name = y, Return = z)
dat2
```

```
##   ID Name     Return
## 1  2    a -0.8712102
## 2  4    q  0.5752565
## 3  6    w -0.1245797
## 4  1    r  0.2149219
## 5  8    t  0.1628263
```

We can also call a single variable (e.g. Return) from the data frame using the **$** symbol, which is the *list indexing operator*, i.e., it specifies/names an element (placed after $) of an object (placed before $).

```
dat2$Return
```

```
## [1] -0.8712102  0.5752565 -0.1245797  0.2149219  0.1628263
```

d. Remove variable *Name* from the data frame & add a variable called *Month* with elements January through May.

We can also use the $ operator to specify columns that we want to delete:

```
# delete a column (i.e. variable) from data frame dat2
dat2$Name <- NULL
# see what has happened
dat2
```

```
##   ID     Return
## 1  2 -0.8712102
## 2  4  0.5752565
## 3  6 -0.1245797
## 4  1  0.2149219
## 5  8  0.1628263
```

and to create new columns:

```
# create a new column in dat2, called Month, storing the specified character strings
dat2$Month <- c("January", "February", "March", "April", "May")
# see what has happened
dat2
```

```
##   ID     Return    Month
## 1  2 -0.8712102  January
## 2  4  0.5752565 February
## 3  6 -0.1245797    March
## 4  1  0.2149219    April
## 5  8  0.1628263      May
```

e. Add a variable, called *Date* to the data frame. *Date* also contains five elements: the first is 2015M1, the second is 2015M2, . . . , the fifth is 2015M5.

We can add such a variable to the data frame using the `paste()` function, which is handy for combining a mix of characters and numbers into a character string (with or without separating symbols or blanks between them). In our example, 2015M is pasted/written to each element of column/variable/vector *Date*, followed by a numerical sequence running from 1 to 5 for elements 1 through 5, respectively. The two portions that `paste()` will combine into a character string are specified using a comma in between and followed by the option `sep = ""`, which means that the character string should be written without any blanks or symbols as separator between its components. Let's put it to practice:

```
dat2$Date <- paste("2015M",1:5, sep = "")
```

Type `View(dat2)` to see if it came through alright.

Another important feature of data indexing in R is that we can use square brackets to look at specific sections of a data frame. Take a look at some examples of positions of elements, rows and columns in our data frame **dat2** by typing the code below:

```
dat2[1,4] # element in upper right corner
```

```
## [1] "2015M1"
```

```
dat2[1,] # first row of dat2
```

```
##   ID     Return   Month   Date
## 1  2 -0.8712102 January 2015M1
```

```
dat2[,3] # third column of dat2
```

```
## [1] "January"  "February" "March"    "April"    "May"
```

```
dat2[,-c(1,2)] # dat2 without the first 2 columns
```

```
##      Month   Date
## 1  January 2015M1
## 2 February 2015M2
## 3    March 2015M3
## 4    April 2015M4
## 5      May 2015M5
```

```
dat2[2:4,2] # rows 2 to 4 of column 2
```

```
## [1]  0.5752565 -0.1245797  0.2149219
```

```
dat2[c(2,3,5),2:3] # only the rows 2, 3 and 5 of columns 2 and 3
```

```
##       Return    Month
## 2  0.5752565 February
## 3 -0.1245797    March
## 5  0.1628263      May
```

To rename a column, use the function `colnames()` and use square brackets to indicate the location of the column you want to rename in the data frame:

```
colnames(dat2)[2] <- "Return-Percentage"
```

Practical 3: Indexing

In Practical 2, we introduced the principles of indexing by means of a dummy data set (**dat2**), which we created manually using R language. Next, we want to load a ready data set into RStudio.

a. Load the dataset ACC_data.txt into RStudio, which contains information on American Campus Communities (property data). Familiarise with the data structure: Which variables (of which class) and how many property observations does it contain? Could you discard some of the variables from the data frame once you know what it is about?

Remember to check the working directory is set correctly and that you have saved the **ACC_data** in it (go to **Session > Set Working Directory > Choose Directory**)

The data is saved as a .txt file. We can use a function called `read.table()`, which reads .txt files. (Note: similarly named functions are available for data stored in other file formats, e.g., use `read.csv()` for csv files.). Run the following line of code, to load the .txt file into an R object, called **dat3**.

```
setwd("C:/Users/USERNAME/Documents/REModelling/Tutorial 1/R_WORK")

dat3 <- read.table("ACC_data.txt", header = TRUE)
# include option header = TRUE to use first row in .txt file as variable names
```

When loading data, it is always a good idea to check it came in ok. To do this, we can call the entire data set:

```
dat3 # note: the data set is relatively large (not printed in this booklet)
View(dat3) # opens the entire data frame in a separate tab
```

or preview it:

```
head(dat3) # displays only the first 6 rows of the data
```

```
##   Property.OID   Property.Type          Property.Name         City
## 1       121753 Student Housing Barrett Honors College        Tempe
## 2       125308 Student Housing      Sunnyside Commons   Morgantown
## 3       125310 Student Housing         Pirate's Place   Greenville
## 4       130727 Student Housing          Pirates Cove*   Greenville
## 5       130729 Student Housing   University Crescent*  Baton Rouge
## 6       130731 Student Housing     University Gables* Murfreesboro
##   State.Province.Country YearBuilt Size SizeUnits Latitude  Longitude
## 1           Arizona, USA      2009 1711      Beds 33.41494 -111.92748
## 2     West Virginia, USA      1925  161      Beds 39.64128  -79.95527
## 3    North Carolina, USA      1996  528      Beds 35.59764  -77.37013
## 4    North Carolina, USA      2000 1056      Beds 35.59990  -77.33379
## 5         Louisiana, USA      1999  612      Beds 30.39887  -91.17251
## 6         Tennessee, USA      2001  648      Beds 35.83154  -86.35048
```

b. Call a subset from the ACC_data that only contains properties with more than 1,000 beds.

We tell R to browse through column *Size* of **dat3** and display all columns but only for those rows, for which *Size* exceeds 1,000:

```
dat3[dat3$Size > 1000,]
```

```
##    Property.OID   Property.Type            Property.Name         City
## 1        121753 Student Housing   Barrett Honors College        Tempe
## 4        130727 Student Housing            Pirates Cove*   Greenville
## 37       130771 Student Housing     University Crossing* Philadelphia
## 41       130776 Student Housing                    Club*       Athens
## 55        73156 Student Housing    Village at Blacksburg   Blacksburg
## 64        73171 Student Housing University Village-PVAMU Prairie View
## 66        73173 Student Housing University College-PVAMU Prairie View
## 74        75154 Student Housing                  Estates  Gainesville
## 78        80365 Student Housing            Vista del Sol        Tempe
##    State.Province.Country YearBuilt Size SizeUnits Latitude  Longitude
## 1            Arizona, USA      2009 1711      Beds 33.41494 -111.92748
## 4     North Carolina, USA      2000 1056      Beds 35.59990  -77.33379
## 37      Pennsylvania, USA      1926 1016      Beds 39.95630  -75.18618
## 41           Georgia, USA      1989 1016      Beds 33.92490  -83.36932
## 55          Virginia, USA      1990 1056      Beds 37.24670  -80.42042
## 64             Texas, USA      1996 1920      Beds 30.09456  -95.99602
## 66             Texas, USA      2000 1470      Beds 30.09443  -95.98604
## 74           Florida, USA      2002 1044      Beds 29.63400  -82.37930
## 78           Arizona, USA      2008 1866      Beds 33.42200 -111.93419
```

c. Call a subset from the ACC_data that only contains properties in Florida that have been built before 2000 and assign it to a new data frame.

The new data frame is called **dat4**. We use the & operator to combine the two restrictions.

```
dat4 <- dat3[dat3$State.Province.Country == "Florida, USA" &
       dat3$YearBuilt < 2000,]
View(dat4) # check the new data frame
```

We can also describe each column in the dataset using the `summary()` function:

```
summary(dat4)
```

```
##   Property.OID    Property.Type      Property.Name          City
##  Min.   : 73165   Length:8           Length:8           Length:8
##  1st Qu.: 79453   Class :character   Class :character   Class :character
##  Median : 81225   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 85526
##  3rd Qu.: 81227
##  Max.   :130772
##  State.Province.Country   YearBuilt         Size        SizeUnits
##  Length:8               Min.   :1990   Min.   :204.0   Length:8
##  Class :character       1st Qu.:1992   1st Qu.:272.0   Class :character
##  Mode  :character       Median :1997   Median :412.0   Mode  :character
##                         Mean   :1996   Mean   :502.4
##                         3rd Qu.:1999   3rd Qu.:742.2
##                         Max.   :1999   Max.   :930.0
##     Latitude       Longitude
##  Min.   :28.58   Min.   :-84.31
##  1st Qu.:28.61   1st Qu.:-84.31
##  Median :29.63   Median :-82.34
##  Mean   :29.55   Mean   :-82.65
##  3rd Qu.:30.45   3rd Qu.:-81.21
##  Max.   :30.45   Max.   :-81.20
```

Anyhow, we will do more work on summary statistics/descriptives in *Tutorial 2*.

To do (until next class)

You are all set on R basics. Please read **Chapter 2 (Essentials of the R Language) of the R Book** (Michael J. Crawley (2007): *The R Book*, 2nd ed., John Wiley & Sons.), which is available in the **Files** section on Canvas.