Chapter 1 - Introduction to R

1.1 What Is R?

R (R Core Team 2018) (is a language and environment for statistical computing and graphics. R is a free and Open Source software that provides a wide variety of statistical and highly customization graphical techniques. At its core, R provides an effective way to handle and manipulate data. It includes operators for calculations on matrices, and a collection of tools for data analysis and graphical display. R can easily be extended via *packages* (more about those in a moment).

1.2 Installing R and RStudio

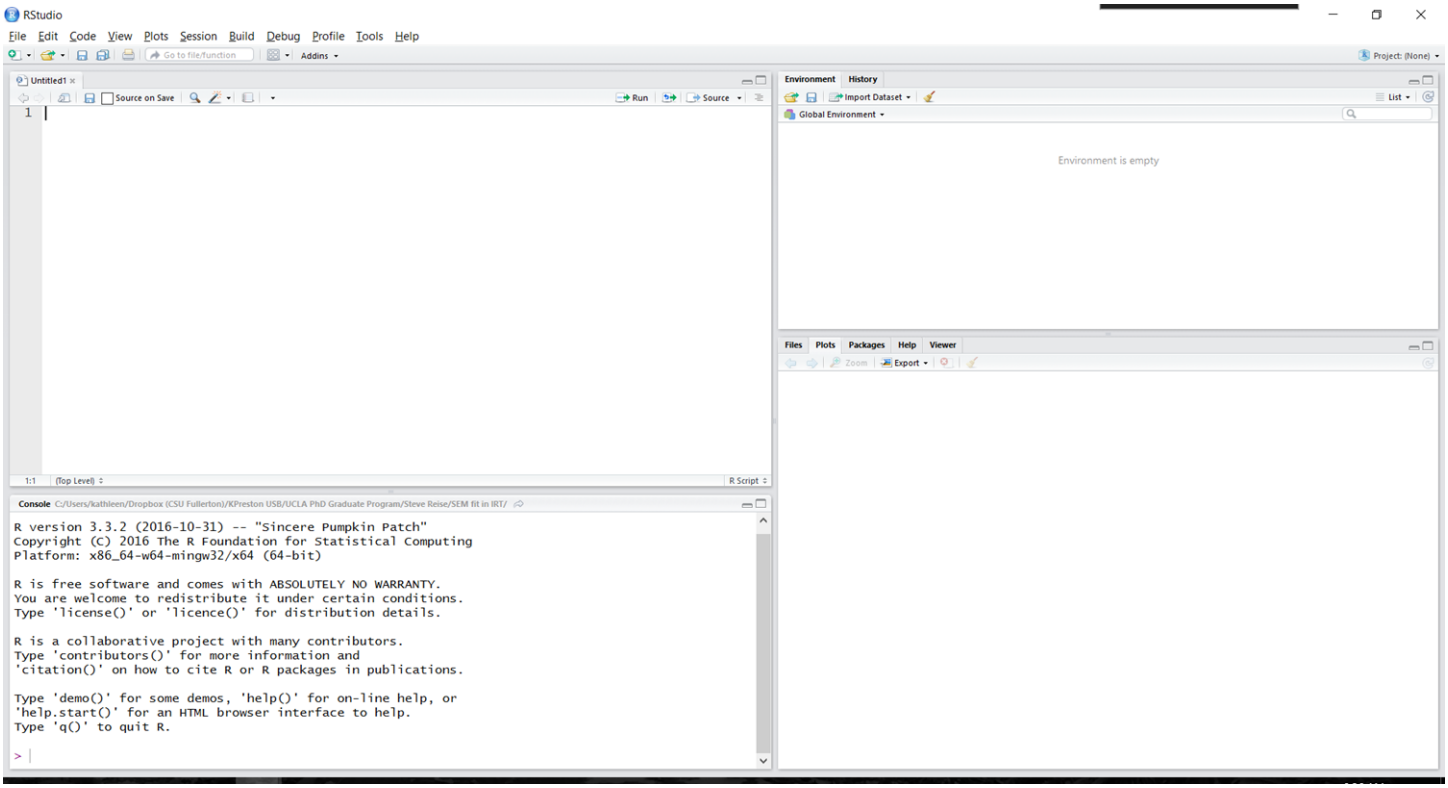
R can be downloaded from one of the Comprehensive R Archive Network (CRAN) Mirror websites found through https://cran.r-project.org/mirrors.html. When downloading the `R base`, you should choose the CRAN Mirror site close to your location.

RStudio is a free and open-source integrated environment for R. Its interface is organized so we can see the R syntax, output, and graphs all at the same time. It also offers a user-friendly `Import Dataset` feature that allows us to easily import data sets from text files (".dat," ".txt" or ".csv"), Excel (".xls" or ".xlsx"), SPSS (".sav"), SAS (".sas7bdat"), and Stata (".dta"). RStudio can be downloaded from https://www.rstudio.com/products/rstudio/download/. Make sure to download and install R before installing RStudio. When installing both R and RStudio, the program defaults can be accepted. Once you have installed RStudio, you only need to open RStudio rather than R because RStudio will open R for you. In fact, you can remove the R icon from your desktop, so you aren't tempted to open it.

1.3 Navigating RStudio

When you open RStudio, the interface will look like the screenshot displayed in Figure 1.1. The upper-left quadrant is the `R-script` window. This is the main window you will be using because you will edit your syntax here. The lower-left quadrant is the Console window, which compiles the code you write in the `R-script` window. The > operator within the Console window signifies that R is ready to accept syntax for execution. One way to think about the relationship between the `R-script` and Console windows is by referencing theater. In theater, an actor is given a script containing lines that they are to perform. The actor's performance of the script is what happens when the syntax from the `R-script` window is sent to the Console window by clicking on the "Run" button, the `R-script` is compiled and the lines are executed. Just like actors can improvise by going off-script, you are also able to type directly into the Console window without modifying the `R-script`. The upper-right quadrant is the contains the Environment and History tabs. The Environment tab imports data sets and allows us to load/save workspaces. The History tab shows the previously executed commands that RStudio automatically saves from the current R session (the workspace of your current R environment). Here, you can search and resubmit any previous commands. Finally, the lower-right quadrant contains the Files, Plots, Packages, and Help tabs. Through the Files tab, you can open any file on your computer. The Plots tab displays, you guessed it, plots. The Packages tab allows you to choose packages to install, update, or load into the current R session. The Help tab provides R documentation for all commands.

Figure 1.1 `RStudio` interface.



1.3.1 Importing a Data Set

As mentioned in Section 1.2, `RStudio` also offers a user-friendly `Import Dataset` feature that allows us to easily import data sets from text files (".dat," ".txt" or ".csv"), Excel (".xls" or ".xlsx"), SPSS (".sav"), SAS (".sas7bdat"), and Stata (".dta"). To demonstrate this feature, we will import a small tab-delimited data set called `data.dat` created from the data shown in Table 1.1. To import a ".dat" data set, we will select the "From Text (readr)" option from the "Import Dataset" menu found in the Environment tab, as shown in Figure 1.2.

Figure 1.2 Import Dataset menu.

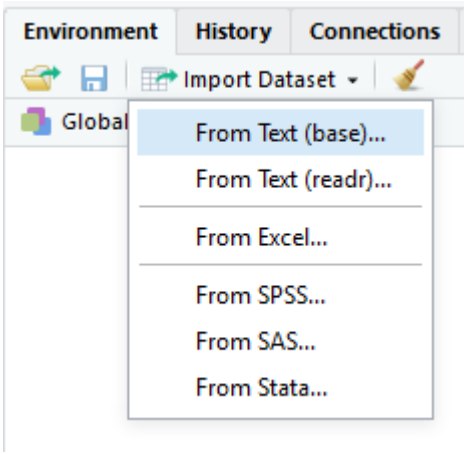
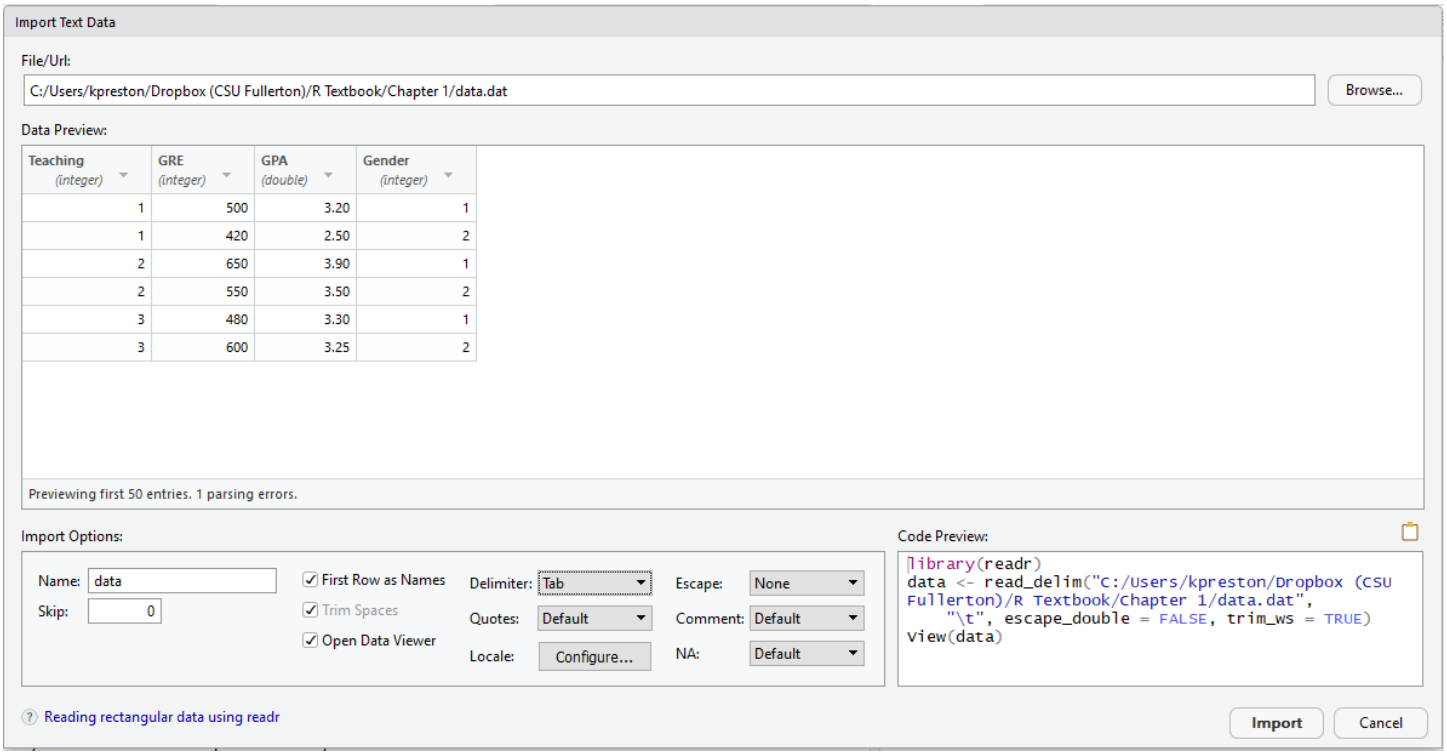


Figure 1.3 shows the new dialog window from which you will "Browse" for the location of the data set on your computer. By default, the "Delimiter" option is set to "Comma," so you will need to change it to "Tab" since this example data set is tab-delimited. Once data set looks correct int the "Data Preview" section, you can click the "Import" button and the syntax contained in the "Code Preview" section will be sent to the Console and executed.

Figure 1.3 Import Dataset dialog window.



Once the data set has been successfully imported, a new "data.dat" tab will appear in the `R-script` quadrant of `RStudio` showing the format of the imported data set. Note, that the data set displayed in this tab is "read only" and the data cannot be altered here. The Environment tab provides information about the imported data set. Specifically, the Environment tab shows the number of observations and variables contained in the imported data set. Finally, the Console quadrant shows the executed code. If you intend to use the imported data set frequently, you can copy the code from the Console and paste it into the `R-script` window so you don't have to re-import the data set from the menu system every time you start a new R session.

1.4 R Language Basics

1.4.1 Packages

Base R already comes with a number of basic data management, analysis, and graphical tools. However, sometimes you need to run an analysis or create a plot that is beyond the limits of base R. This is where Packages come in handy. Packages are collections of R functions and data sets created by R-users to improve existing functions or create new ones. R comes with a standard set of packages, which are stored in the `library`. Additional packages, can be downloaded from the `Comprehensive R Archive Network (CRAN)`. You can use the Packages tab to install new packages, update installed packages, and load installed packages into the current R session. Packages only need to be installed once, but they must be loaded every time you start an R session during which you intend to use the package. Chapter 2 provides the packages and package citations used to complete the analyses demonstrated in each chapter of the book.

1.4.2 Programming Basics

R is case sensitive. When you call functions, load packages, or create objects, you need to make sure the case matches exactly.

The # character signifies a comment in the syntax, which is not executed. Each commented line must start with a #. Commenting your syntax is an effective way to describe the analyses being conducted.

Missing data are noted in R as `NA`.

Both data and output from data analyses (as well as everything else you want to store in R) are stored in objects. Data, for instance, are assigned to and stored in objects using the <- assignment arrow or the = operator. The object name appears on the left side of the <- or = and the data appear on the right side. For example, `xyz <- 3` assigns the number 3 to the object `xyz`. To print the data or output contained within an object, we call the object by name or we can nest the syntax within `(xyz <- 3)` to assign and print the information contained within the object.

```
#assign the number 3 to object xyz
xyz <- 3
#print the value of the object xyz
xyz
```

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

```
#assign and print the value of object xyz using ( )
(xyz <- 3)
```

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

When the value of `xyz` is printed as output, we see `[1]` before the 3 to identify the location of 3 within object `xyz`. Specifically, 3 is the first element in object `xyz`.

1.4.3 Functions

Functions are where all the "work" gets done. These are special kinds of objects that, generally, accept arguments, perform some operation or analysis, and return a value. For instance, the function + does addition and returns the sum of the two specified arguments (the addends). The `sum` function also does addition, which returns the sum of all the values present in its arguments. Specifically, we can pass `1,2,3,4,5` as arguments to the `sum` function by specifying `sum(1,2,3,4,5)` and the function will return 15, which is equal to 1+2+3+4+5. Functions can have arguments which require user input, optional arguments that can be used to enhance the function's output, as well as arguments with default values already specified. The `sum` function contains an additional argument, `na.rm=FALSE`, which specifies whether missing values should be removed. If `na.rm` is `FALSE`, which is the default setting, an `NA` value in any of the arguments will cause a values of `NA` to be returned. Otherwise, if `na.rm` is `TRUE`, then `NA` values are ignored. To illustrate,

```
sum(1,2,3,4,5)
```

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

```
sum(1,2,3,4,NA)
```

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

```
sum(1,2,3,4,NA,na.rm=TRUE)
```

```
## [1] 10
```

Documentation for all functions contained in R can be found with the `help` function or from the `The R Reference Index` section of the https://cran.r-project.org/manuals.html website.

1.4.4 Navigating a Data Set

Now that we have a data set loaded into our R session, we can use a variety of functions to explore our data.

1.4.4.1 The `head` function

The `head` function prints all the columns (variables) and the top 6 rows (by default) of a data set.

```r
head(data)
```

```
##   Teaching GRE  GPA Gender
## 1        1 500 3.20      1
## 2        1 420 2.50      2
## 3        2 650 3.90      1
## 4        2 550 3.50      2
## 5        3 480 3.30      1
## 6        3 600 3.25      2
```

1.4.4.2 The `names` function

We can determine the variable names in a data set object with the `names` function.

```r
names(data)
```

```
## [1] "Teaching" "GRE"      "GPA"      "Gender"
```

1.4.4.3 The `dim` function

The `dim` function returns the number observations (rows) and variables (columns) in the data set.

```r
dim(data)
```

```
## [1] 6 4
```

1.4.4.4 The `summary` function

`summary` is a generic function used to summarize many types of `R` objects, including data sets. When a data set is passed to the `object` argument, the `summary` function returns distributional summary of variables in the data set.

```r
summary(data)
```

```
##     Teaching         GRE             GPA           Gender
##  Min.   :1.00   Min.   :420.0   Min.   :2.500   Min.   :1.0
##  1st Qu.:1.25   1st Qu.:485.0   1st Qu.:3.212   1st Qu.:1.0
##  Median :2.00   Median :525.0   Median :3.275   Median :1.5
##  Mean   :2.00   Mean   :533.3   Mean   :3.275   Mean   :1.5
##  3rd Qu.:2.75   3rd Qu.:587.5   3rd Qu.:3.450   3rd Qu.:2.0
##  Max.   :3.00   Max.   :650.0   Max.   :3.900   Max.   :2.0
```

1.4.4.5 Data Frames

Data sets are typically stored as data frames, which have a matrix structure. Observations are arranged as rows and variables, either numerical or categorical, are arranged as columns. Individual rows, columns, and cells in a data frame can be accessed through many methods of indexing. To illustrate,

```r
# Referencing within Data Frames
# single cell value
data[2, 3]
```

```
## [1] 2.5
```

```
# omitting row value implies all rows; here all rows in column 3
data[, 3]
```

```
## [1] 3.20 2.50 3.90 3.50 3.30 3.25
```

```
# omitting column values implies all columns; here all columns in row 2
data[2, ]
```

```
##   Teaching GRE GPA Gender
## 2        1 420 2.5      2
```

```
# can also use ranges - rows 2 and 3, columns 2 and 3
data[2:3, 2:3]
```

```
##   GRE GPA
## 2 420 2.5
## 3 650 3.9
```

We can also access variables directly by using their names, either with `object[,"variable"]` or `object$variable` notation.

```
data[, "GRE"]
```

```
## [1] 500 420 650 550 480 600
```

```
data$GRE
```

```
## [1] 500 420 650 550 480 600
```

1.4.4.6 Labeling Categorical Variables

For categorical variables, we can change them from numeric variables to factors, a special class that R uses for categorical variables, which also allows for value labeling. Here, using the `factor` function, we can convert the `Gender` variable of `data` to a categorical variable. The `levels` and `labels` argument specifies the coding of the variable to be females coded 1 and males coded 2. The `"Female"` and `"Male"` labels are passed to the `labels` argument in a vector using the `c` function. The `c` function combines its arguments into the form of a vector.

```
data$Gender <- factor(data$Gender, levels = 1:2, labels = c("Female","Male"))
```

Now, when we call the `summary` function, the output contains a frequency distribution table for `Gender`.

```
summary(data)
```

```
##     Teaching         GRE             GPA          Gender
##  Min.   :1.00   Min.   :420.0   Min.   :2.500   Female:3
##  1st Qu.:1.25   1st Qu.:485.0   1st Qu.:3.212   Male  :3
##  Median :2.00   Median :525.0   Median :3.275
##  Mean   :2.00   Mean   :533.3   Mean   :3.275
##  3rd Qu.:2.75   3rd Qu.:587.5   3rd Qu.:3.450
##  Max.   :3.00   Max.   :650.0   Max.   :3.900
```

1.6 Data Appropriate for Multivariate Statistics

The next section corresponds to Section 1.6 in the textbook. Chapter 2 will explain how to "use this book" and the correspondence between each section, table, and figure.

1.6.1 The Data Matrix

Rather than importing every data set, a data matrix can also be created and stored in R. Table 1.1 is an example data matrix with six participants and four variables (type of teaching technique, score on the GRE, GPA, and gender). Here we use the `data.frame` and `matrix` functions. The `matrix` function uses the arguments `(data= , nrow= , ncol= , byrow= , dimnames= )` to create a matrix from the given set of values. The `data` are passed to the `data=` argument using by listing all of the values within the `c` function. In the remaining three arguments, the data are organized by row into 6 rows and 4 columns. By passing the data matrix to the `data.frame` function before storing the data matrix in the object called `data` we will be able to specify the names of each variable. To specify the variables names, we assign a vector containing variable names `c("Teaching","GRE","GPA","Gender")` to `names(data)`. Finally, we print the data matrix by calling object `data`.

Table 1.1 A Data Matrix of Hypothetical Scores

```
data <- data.frame(matrix(c(1,500,3.20,1,
                            1,420,2.50,2,
                            2,650,3.90,1,
                            2,550,3.50,2,
                            3,480,3.30,1,
                            3,600,3.25,2),nrow=6,ncol=4,byrow=TRUE))
names(data) <- c("Teaching","GRE","GPA","Gender")
data
```

```
##   Teaching GRE  GPA Gender
## 1        1 500 3.20      1
## 2        1 420 2.50      2
## 3        2 650 3.90      1
## 4        2 550 3.50      2
## 5        3 480 3.30      1
## 6        3 600 3.25      2
```

1.6.2 The Correlation Matrix

Table 1.2 shows the correlation matrix for the `GRE`, `GPA`, and `Gender` variables of Table 1.1. To produce this correlation matrix, the desired variables are selected from the `data` object as described in Section 1.4.4.5 and passed to the `cor` function. Output from the `cor` function is then passed to the `round` function, which rounds the correlations to two decimal places by specifying `digits=2`.

Table 1.2 Correlation Matrix for Part of Hypothetical Data for Table 1.1

```
round(cor(data[,c("GRE","GPA","Gender")]),digits=2)
```

```
##          GRE   GPA Gender
## GRE     1.00  0.85  -0.13
## GPA     0.85  1.00  -0.46
## Gender -0.13 -0.46   1.00
```

1.6.3 The Variance–Covariance Matrix

The variance–covariance matrix for the data in Table 1.1 appears in Table 1.3. The syntax used to produce the correlation and variance–covariance matrices is nearly identical except for the use of the `cov` rather than the `cor` function. The resulting variance–covariance matrix is stored in the object called `Sigma`.

Table 1.3 Variance–Covariance Matrix for Part of Hypothetical Data for Table 1.1

```
(Sigma <- round(cov(data[,c("GRE","GPA","Gender")]),2))
```

```
##             GRE   GPA Gender
## GRE    7026.67 32.80  -6.00
## GPA      32.80  0.21  -0.12
## Gender   -6.00 -0.12   0.30
```

1.6.4 The Sum-of-Squares and Cross-Product Matrix

The sum-of-squares and cross-products matrix for `GRE`, `GPA`, and `Gender` in Table 1.1 appears in Table 1.4. There is not currently a function in `R` that directly produces the sum-of-squares and cross-products matrix, S, but we can solve the equation used to compute the variance–covariance matrix

$$\Sigma = \frac{1}{N-1}S$$

for S to return the sum-of-squares and cross-products matrix. Specifically, we call the `Sigma` object produced in Table 1.3 and multiply it (using the `*` operator) by the number of rows (using the `nrow` function to return the number of rows) in `data` minus 1 (using the `-` operator).

Table 1.4 Sum-of-Squares and Cross-Products Matrix for Part of Hypothetical Data of Table 1.1

```
(S <- Sigma*(nrow(data)-1))
```

```
##              GRE    GPA Gender
## GRE    35133.35 164.00  -30.0
## GPA      164.00   1.05   -0.6
## Gender   -30.00  -0.60    1.5
```

Reference

R Core Team. 2018. “R: A Language and Environment for Statistical Computing.” Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.