# Appendix A: Exploring Data with R[[1]](#footnote-1)

*Getting to know RStudio*

* Console → The space where you type your command and run to see the results.

Commands in the console cannot be directly saved.

* Script → A text file that enables you to save and run R commands.
* Environment → Environment stores the data frames and values you are currently

working with on RStudio.

* Help, plot → Help of the commands, plots of the data, list of packages etc.

*Working Directories*

It is crucial to get to know where your working directory locates and set the correct working directory to the file that contains your datasets before importing data. If you are working with the wrong working directory, RStudio will not be able to locate your datasets.

Relevant commands:

* + Getting your current working directory: getwd()
  + Setting working directory: setwd(), alternatively you can go to session → set working directory and choose directory.

Always remember to hit the “run” button to see the results of your code!

*Libraries*

As a programming language, R allows us to write and share libraries (or more intuitively, packages) that manipulate data. For example, the “plyr” package provides some easy codes to aggregate data, and the “ggplot2” package has become the most commonly used one in visualizing data with R. To install a package, type install.packages (“name\_of\_the\_package”) to get RStudio download the package, and use library (name\_of\_the\_package) to load the package every time before using it (You only have to install the package once though).

*Getting help*

RStudio is extremely powerful at providing help on how to use commands and what will the proper commands be when you look for them. When trying to get help on how to use a certain command, say, setwd(), type ?setwd(). If you do not know the name of the command and would like to look it up in RStudio, use two question marks followed by the content of your intended commands such as ??standarddeviation (to get the command of calculating standard deviation) and let RStudio help you.

*Importing Data*

* + csv. → The most commonly used command is read.csv(“name\_of\_the\_file.csv”). Please note that csv. is always our preferred data format to work with. It is open source, and takes a much smaller space on your computer thus quicker and easier to proceed with.
  + txt. → The most commonly used command is read.table (“name\_of\_the\_file.txt”, header=FALSE). More detailed instructions, see the help in RStudio. Different from read.csv, read.table expects no header of the data and blank space (instead of comma) to separate the characters.
  + xlsx. → R does not have built-in commands to input an xlsx file. Instead, we install a package “xlsx” and use read.xlsx to read the file. Please note that you probably will have to update your java on your computer before successfully using this package. It is preferable to save the xlsx. into a csv. file and then import it.
  + scan directly from a website → the scan() function allows us to directly read data from a website or text file. Try ?scan() for more information on how to use this command.

**Data Structures**

In below sections we will be using RStudio’s built-in data frame, mtcars, to practice with the commands. Type mtcars in your script, hit “run”, and we will be able to see the data appearing in the console. Before moving to data structures, we need to learn some basic concepts about R.

*What is a vector*: A vector is a sequence of data elements of the same basic type. A column vector appears as a column within a data frame.

*What is a data frame*: A data frame is a list of same-length column vectors. It looks like a matrix where each column represents a measurement, and each row represents an observation. A data frame can contain vectors of different data types.

*Assigning values to an object*. For example we have our mtcars data frame and we would like to create a new object “sample\_data” that has exactly the same data as mtcars, we use the left arrow. sample\_data <- mtcars. Assigning values to an object become especially useful when you have long names of your original data sets, and a good practice for you to observe your data frame step by step alongside with your commands.

*How is the data structured? (structure)*

To take a look at the structure of data, here are several commonly used commands:

sample\_data <- mtcars

head (sample\_data) shows us the first six rows of the data frame.

tail (sample\_data) shows us the last six rows of the data frame.

str (sample\_data) presents us with a more comprehensive view of the data frame that includes the number of rows and columns of the data, and the types of the vectors.

str(sample\_data)

'data.frame': 32 obs. of 11 variables:

$ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...

$ cyl : num 6 6 4 6 8 6 8 4 4 6 ...

$ disp: num 160 160 108 258 360 ...

$ hp : num 110 110 93 110 175 105 245 62 95 123 ...

$ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...

$ wt : num 2.62 2.88 2.32 3.21 3.44 ...

$ qsec: num 16.5 17 18.6 19.4 17 ...

$ vs : num 0 0 1 1 0 1 0 1 1 1 ...

$ am : num 1 1 1 0 0 0 0 0 0 0 ...

$ gear: num 4 4 4 3 3 3 3 4 4 4 ...

$ carb: num 4 4 1 1 2 1 4 2 2 4 ...

*What type of data is it? (class)*

The class () command helps us to look at the data type, or the class of the elements.

> class (50)

## [1] "numeric"

> class (FALSE)

## [1] "logical"

> class ("SIPA")

## [1] "character"

There are nine types of data that are commonly used in R:

* + numeric
  + integer
  + complex (defined via the pure imaginary value *i*)
  + logical
  + list (generic vector containing other objects)
  + raw (hold raw bytes)
  + expression
  + factor (Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed. Both numeric and character variables can be made into factors, but a factor's levels will always be character values. You can see the possible levels for a factor through the levels command.)

*How big is the data? (dimensions)*

We use dim(), nrow() and ncol() to check the size of our data frame. In our sample data mtcars, we have 32 rows and 11 columns.

> dim(sample\_data)

## [1] 32 11

> nrow(sample\_data)

## [1] 32

> ncol(sample\_data)

## [1] 11

*What are the names of the column vectors?*

We use the colnames function, or simply the names function.

> colnames(sample\_data)

## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"

[11] "carb"

> names(sample\_data)

## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"

[11] "carb"

*How long is the period of record? (time series data)*

For time series data, it is helpful to create a “Date” or POSIXct variable to check the range of the time frame. Date class helps us to look back and forth between Date formats, while POSIXct is used to track sub-daily time records. Try ?as.Date and ?POSIXct to see how to convert integers into a Date format. . You may refer to the range() function to check the range of the time series data.

*Looking at the records (rows)*

Let’s create a small sample under our sample data.

small\_sample <- sample\_data[1:10, ] (We take the first ten rows and all columns)

> small\_sample<-sample\_data [1:10,]

> small\_sample[1,]

## mpg cyl disp hp drat wt qsec vs am gear carb

## Mazda RX4 21 6 160 110 3.9 2.62 16.46 0 1 4 4

We looked at the first row of our small sample in this case.

**Handling Real-World Data**

*Missing values*

In real world, we inevitably meet data with missing values. Some in R are shown as “NA”, “-999”, “10e-30” or blank “ ”. It is good practice to use **NA** to record missing values for the sake of consistency. NA can be any type of value. While NA stands for “Not Available”, NaN means “Not a Number”. A numeric expression that is correct in syntax but has not mathematical meaning (not a real number) will return NaN. We use is.na to detect whether there are missing values in our data frame. By using sum(is.na(data.frame)), we summarize the total number of NAs in the data frame.

> sum(is.na(sample\_data))

## [1] 0

The above function shows that we currently do not have any missing values in our data. But we also need to look at the summary of the data frame to see if the data make sense to us. The most commonly used commands are summary () and range (). Say I would like to look at the summary statistics about the disp vector in our sampe\_data data frame (here the $ sign indicates that disp is one of the vectors of the data frame sample\_data).

> summary(sample\_data$disp)

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 71.1 120.8 196.3 230.7 326.0 472.0

> range(sample\_data$disp)

## [1] 71.1 472.0

After taking a look at the summary of disp, we found that the largest number 472.0 does not make sense in real life (I am making this up here but the key point is to be able to benchmark your data through looking at summary statistics). I would like to record elements in disp that equals to 472 a missing value.

> sample\_data$disp[sample\_data$disp==472.0] <- NA

> sum(is.na(sample\_data$disp))

## [1] 1

Now we have manually introduced one missing value by taking out the value that does not make sense to us. The command na.omit will omit the entire row if there is a NA.

* Summary statistics

Except for the summary function we’ve encountered above, table is also a command used for summarizing your data. summary is used for numerical or boolean variables while table is used for character or categorical variables.

* Mean, Standard Deviation

We will use the mean and sd commands to get the mean and standard deviation of vectors.

* Subset, merge

Subsetting and merging are common techniques that we use to manipulate with data. We use the subset and merge command, respectively.

> small\_group<-subset (sample\_data,disp == 71.1)

> small\_group

mpg cyl disp hp drat wt qsec vs am gear carb

Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.9 1 1 4 1

Here the subset function takes out a small fraction of our sample data frame where column disp has the value of 71.1.

The merge function can merge two data frames by common columns or row names, or do other versions of database join operations. Let’s see a simple example here. Say we have two data frames that have one column same:

> group.A

country value

1 Cambodia A

2 Laos B

3 Thailand C

4 Myanmar D

5 Vietnam E

> group.B

country value2

1 Cambodia F

2 Laos G

3 Thailand H

4 Myanmar I

5 Vietnam J

Now we would like to merge these two data frames by the country vector that they have in common. Note that after merging the two data frames, there remains only one “country” column.

> group.C<-merge(group.A,group.B, by = "country")

> group.C

country value value2

1 Cambodia A F

2 Laos B G

3 Myanmar D I

4 Thailand C H

5 Vietnam E J

*Observing data through simple visualization*

Sometimes by graphing the data, researchers can easily get an idea of how the data spreads, if there exists a linear relationship, or spot outliers at a glance. Using hist function, you can draw a histogram of your data in r and discover potential outliers. Using the ggplot2 package, researchers can go further with visualizing the data by coming up with different forms of charts, comparing multiple charts etc.

*Writing Data*

Note that although we have manipulated with the data in RStudio, the original (imported) data files will not be changed. If you have imported an external file, for example, a csv., txt., or xlsx., you can write the current R object to an external file, which is not done automatically in R. Use the write.csv command to save your work to a separate csv. Meanwhile, you can also use the save command or file → save/save as to save your file into an R object and reopen your work in R next time.

* Resources on learning R
  + [Coursera](https://www.coursera.org)
  + [Codecademy](http://www.codecademy.com)
  + [R project for statistical computing](http://www.r-project.org)
  + [R Tutor](http://www.r-project.org)
  + [Code School](https://www.codeschool.com/?utm_source=google&utm_medium=cpc&utm_content=home_page&utm_campaign=branded&gclid=COiI-J_oxcQCFVeVvQodSwQAxA)
  + [Cookbook for R](http://www.cookbook-r.com)

1. Structure of this part is taken from the course notes of *Data Visualization* taught by Dr. Eliot Cohen at Columbia University. Github page: <https://github.com/Ecohen4/data-viz> [↑](#footnote-ref-1)