Introduction to Data Science with RStudio: Practice Exercises

1. Vector Basics

1. Using the function `seq()`, write code to create a vector named myvector, which is a sequence of numbers from 25 to 105 at increments of 5, i.e. a sequence 25, 30, 35, . . . , 95, 100, 105.
2. Run this cell in your R(Studio) to create a vector to be used throughout the rest of this exercise.

```
myvector <- seq(25, 105, by = 5)
```

Now, divide each element of myvector by 5.

3. Use myvector you created in the previous exercise. Raise each element of myvector to the 3rd power.
4. Compute the natural logarithm of each element of myvector.
5. Subract a vector of numbers 1, 2, 3, . . . , 15 from myvector.
6. Subract a vector c(1, 2, 3) from myvector.
7. Print the 1st, 5th and 10th element of myvector.
8. Print the all elements of myvector except for the 3rd and the 14th.

2. Random Objects

1. Generate and print a vector of 10 random integers with values between 1 and 100. Once you are done, generate another vector of 10 random numbers with values between 1 and 100.
2. First, run the following code to generate the random vector rVec of letters:

```
set.seed(438567)
rVec <- sample(letters, size = 30, replace = TRUE)
rVec
```

Now, obtain a random subset of 5 elements of rVec. Note this means you are taking a random subset of a random vector.

3. Matrices

1. Create a 5 by 6 matrix named X by using the vector c(75, 51, 7, 36, 48, 44, 34, 28, 17, 18, 46, 71, 58, 2, 25, 53, 70, 94, 95, 84, 4, 72, 27, 57, 43, 49, 9, 52, 63, 87) to fill in the entries of X by column.
2. Create a matrix X2 of the same dimensions as X, but by using the same vector to fill in the entries of X2 by row.
3. Print the transpose of the matrix X.
4. Rename the column names of X to "F1", "F2", . . . , "F6" (features 1 to 6) and print X.
5. Rename the row names of X to "S1", "S2", . . . "S5" (sample 1 to 5) and print X.
6. Print the value at row 3, column 5 of the matrix X.
7. Print the second column of the matrix X. Print the third row of the matrix X. Print the submatrix of entries in rows 1, 2, and 3 and columns 4 and 5 of matrix X.

8. Set the (2, 5) entry of the matrix X to 0. Print X to check if X has been modified correctly.
9. Set all the entries in the 1st column of X to 2. Print X to check if X has been modified correctly.
10. Set the entries of the third row of X to the elements of the vector c(1,2,3,4,5,6). Print X to check if X has been modified correctly.
11. Plot the first two columns of X against each other, F2 vs F1.
12. Plot column F5 of X against column F4.

4. Lists

1. Lists are objects which can contain data of different types and sizes. Construct a list named calendar containing three elements, the first being a vector of years from 1910 to 2017, the second the vector c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec") of months of the year, and the third a vector of days from 1 to 31. Print calendar to see if it is correct.
2. If you haven't named the elements of your calendar list c("year", "month", "date") you should do it now. Print calendar to see if it is correct.
3. The elements of a list can be named at the moment of construction:

```
# Run the following commands in your R(Studio):

calendar2 <- list(year = 1910:2017,
       month = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
       "Aug", "Sep", "Oct", "Nov", "Dec"),
       date = 1:31)
print(calendar2)

# Note that the calendar and calendar2 lists have elements which contain
#  different types (numeric and character) of data and that elements have
#  different lengths.

# Run the following commands in your R(Studio):

class(calendar)
class(calendar[[1]])
class(calendar[[2]])
class(calendar[[3]])

# The functions summary() and str() provide convenient ways of printing out
#  information about lists:

summary(calendar)
str(calendar)
```

To access the elements of a list, instead of using a pair of (single) brackets [] as you would for a vector, you use a pair of double brakcets [[]] to access an individual element of a list. Use the list calendar to print out all months of the year.

4. Now, obtain the same list of 12 months, but now using the list calendar2 and using the $ operator to access its named elements:
5. Note that accessing elements of calendar with $ notation is impossible as its elements are not named, and so running the following would result in NULL (empty) values.

```
calendar$"2"
calendar$month
```

Use calendar2 list to print out the year of your birthday.

6. Use calendar2 list to print out the month of your birthday.

5. Factors

1. You are planning a vacation trip and want to book a hotel. You've found 20 hotels that are near your destination. The website booking.com gives the following quality scores (star ratings) for the 20 hotels. Run the following command in your R(Studio):

```
quality <- c(1, 3, 2, 3, 4, 3, 4, 3, 3, 1, 1, 3, 1, 3, 1, 1, 5, 5, 3, 5)
class(quality)
```

Convert the numeric vector quality into a factor. Check to see if the class of quality has changed to 'factor'

2. Print unique score values that are in the quality factor.
3. The hotel scores 1 through 5 correspond to "poor," "fair," "good," "very good," and "excellent" quality respectively. Write code that encodes scores with the more descriptive ratings, i.e. change the labels of the factor quality to the corresponding quality descriptions. Check if new labels are correct.
4. Now, print the count of each distinct level in the quality factor.

6. Data Frames

1. Below is a data frame containing data on the age, height[cm], weight[kg], gender, and country of birth of seven people. Notice how we can create a data frame in way that is very similar to how we can use the function `list()` to create a list. The difference is that each element (column) in a data frame must have the same length. You also have the option to supply row names for a data frame. Run the following code in your R(Studio):

```
df <- data.frame(Age = c(25, 30, 52, 60, 34, 43, 21),
                 Height = c(171, 156, 157, 184, 164, 192, 173),
                 Weight = c(67, 51, 49, 75, 60, 85, 80),
                 Gender = factor(c("M", "F", "F", "M", "F", "M", "M")),
                 Country = c("France", "Finland", "USA", "UK", "Italy",
                             "Israel", "Mexico"),
                 row.names = c("Alex", "Martha", "Kate", "Sean", "Francesca",
                               "Nico", "Juan"))
print(df)

# You can check that the class of df is indeed a data.frame:

class(df)

# The function str() is also a convenient utility for looking at the structure
#  of the dataset:

str(df)
```

Now, print the number of dimension of the data.frame df.

2. Construct a data.frame named df2 with seven rows corresponding to the same people as in the previous data.frame, but with a single column Employed = `c("No", "Yes", "No", "Yes", "No",`

`"Yes", "Yes")`. Print df2 to check the result.

3. Use a `data.frame()` function to combine the two data frames, df and df2, into a new data.frame called newdf. Print newdf to check if correct.
4. Print the rows of newdf corresponding to Martha and Sean.
5. Print the height of everybody from newdf.
6. Print columns Age, Gender, Country from newdf.
7. Next, let's load and inspect an R- built-in data.frame mtcars from 1974 Motor Trend US magazine (the data is included in your R console) which contains information on fuel consumption and other aspects of design and performance of 32 cars. Look at the data in your R(Studio) and print the first 6 lines of the data frame:

```r
head(mtcars)

# You can print out more lines by supplying the second argument of
#  the head() function:

head(mtcars, n = 10)

# The summary() function is very handy for data.frames:

summary(mtcars)

# Compare with the output of summary() for 'newdf' data.frame
#  which contains also factor and character values:

summary(newdf)

# Note that summary() prints the frequency of the values for factor
#  and character columns.
```

Plot miles per gallon, x = mpg, vs gross horsepower, y = hp using 'mtcars' data.frame.

7. Importing Data

If you are not sure where your current working directory is, you can always check your directory using command `getwd()`.

If you would like to set a new working directory, use the function `setwd()`. An example would be `setwd("~/Documents/Rwork")`. You can also select "Session" -> "Set Working Directory" -> "Choose Working Directory" in your navigation and choose your new working directory directly.

1. Now import the "Credit.csv" data from Data folder in Instruction file into R using the `read.csv()` function and assign it to the variable "credit".
2. Check what is the class of object credit.
3. Print first six lines of credit.
4. Note that the first column contains the copy of the rownames. This is because the original data in the 'Credit.csv' file contains a column for rownames. In R, data frames already have row names assigned separately, so you can remove this column. Read the data into R again, but this time tell R which column contains the row names. If you are unsure how to do it, I recommend checking the `read.csv()` documentation using `?read.csv`.
5. Use `head()` to again print first few lines of credit. Notice the change.

6. Download the “College.csv” data from the following website: https://www.statlearning.com/resources-first-edition You can also access the data in our folder. Import the data into your R(Studio). Then, use scan() function to read the first 10 lines of the data.
7. Read “NASA_facilities.xlsx” from our data folder. To doing so, make sure you have the package “readxl” installed in your R(Studio). Then, load it to your memory, so you can use it. Read the file into ‘nasa_fc’ variable using `read_excel()` function. You can check the function documentation by typing `?read_excel`.
8. Check the class of `'nasa_fc'`.
9. `read_excel()` output is of class `'tibble'`. Convert it to the more familiar `'data.frame'` class.
10. Double-check that the class of nasa_fc.df is now of `'data.frame'` class. Then, print the first 10 lines of nasa_fc.df.
11. Note that R generated some odd-looking column names, e.g. `"X__1"`, because the original Excel file included some merged columns. If you’d like to remove the top merged column names, use the `'skip'` argument when calling `'read_excel()'` function:

```r
nasa_fc <- read_excel("Data/NASA_Facilities.xlsx", skip = 1)
```

```
## New names:
## * `` -> ...2
```

```r
nasa_fc.df <- as.data.frame(nasa_fc)
head(nasa_fc.df)
```

Note that the first column of nasa_fc.df denotes the row index, but is incorrectly labelled `'Agency'`. Change the name of the first column to `'ID'`, and the second column one to `'Agency'` and print first few lines of the altered `data.frame`.

8. Saving Data

1. Suppose you have created a following data.frame storing information on some characters from Harry Potter:

```r
hp_info <- data.frame(first = c("Harry", "Ron", "Hermione", "Ginny"),
                      last = c("Potter", "Wesley", "Granger", "Wesley"),
                      gender = c("M", "M", "F", "F"),
                      date = c(31, 1, 19, 11),
                      month = c("Jul","March", "Sep", "Aug"))
hp_info
```

   Write (save) the hp_info data.frame to the file ‘harry_potter.txt’ in your current working directory.

2. Now check the contents of your working directory folder. Is your new file there?
3. Make sure you still have the data frames from the last exercise about NASA facilities. Save it as `"mydata.rdata"` to your working directory.
4. Check the contents of your working directory folder again, and notice it now also contains the file `'mydata.rdata'`.
5. Use `ls()` function to print out all the variables/objects generated in the session and saved in memory. Run `ls()` in your R(Studio). Now, remove all data generated in the session.
6. Check that your memory is empty. Then, load all the objects saved in `'mydata.rdata'` and check that your data frame nasa_fc.df has been restored.

7. Notice that the data frame hp_info is still not in memory. How can we reload from the file `'harry_potter.txt'`? We didn't save it as an object, so we can't use the `load()` function. We used the `write.table()` function to save it as a text file, so we can use the read.table() function.

9. Logical statements

1. Read the `'artist_data.csv'` file from our data into R. Check the file by inspecting the initial rows.
2. Check the dimension of artists_df using `dim()` function. Then, inspect the structure of artists_df using `str()` function.
3. Using square brackets, and `which()` function, extract a subset of artists who were born in the 19th century, i.e. those artists whose year of birth is from 1801 to 1900 (inclusive). Check the variable `artists19th` by inspecting its initial rows and dimensions.
4. What is the fraction of artists in artists_df born in the 19th century?
5. Observe, that in the above example the function `which()` returns a set of indices where the condition `artists_df$yearOfBirth >= 1801 & artists_df$yearOfBirth <= 1900` evaluates to `TRUE` and allows us to select the rows with artist from the 19th century. However, R also supports logical indexing and you can ommit the `which()` function:

```
artists19th_lon_par <- artists19th[artists19th$placeOfBirth
    %in% c("London, United Kingdom", "Paris, France"), ]
head(artists19th_lon_par)
dim(artists19th_lon_par)
```

Now from the set of 19th century artists select the ones whose place of birth was either London or Paris and check the variable artists19th_lon_par by inspecting its initial rows and dimensions.

6. How many artists born in the 19th century London and Paris were there in the TATE collection?
7. How many artists in artists_df have a missing year of death data?

10. XY and Image Plots

In this exercise we will plot the trends in the U.S. energy consumption by source. Firstly, read the data from https://www.eia.gov/totalenergy/data/browser/csv.cfm?tbl=T01.03

```
# Run this code in your R(Studio) to read in the data:

energy <- read.csv("https://www.eia.gov/totalenergy/data/browser/csv.cfm?tbl=T01.03",
                   stringsAsFactors = FALSE)
energy$Value <- as.numeric(energy$Value) # convert to numeric values
```

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

```
head(energy)

# In the data.frame 'energy', the column 'Description' gives the type
#  of energy source. Column 'Value' gives the actual consumption in
#  Quadrillion BTU units.

# Note that, the data.frame 'energy' contains a column 'YYYYMM' which stands
#  for year and month of the measurement. We will convert this data to a
#  date format in R:

energy$Year <- sapply(energy$YYYYMM, function(x) {substr(x, 1, 4)})
energy$Month <- sapply(energy$YYYYMM, function(x) {substr(x, 5, 6)})
```

```
# There are some entries in the data that indicate month '13', which does
#  not make sense. Since we do not know what exactly this means, we will treat
#  these entries as invalid and remove them:

energy <- energy[energy$Month != "13", ]
head(energy)
```

1. Use the `plot()` function to produce a scatter plot of the `'Coal Consumption'` over time using column `'YYYYMM'` and `'Value'`:

2. Since monthly data might be too noisy, we will aggregate the total annual consumption for each type of energy source:

```
annual_energy <- aggregate(Value~Description+Year, energy, sum)
head(annual_energy)

# Also, we will remove the records for 2017, as not a whole year has passed
#  and it would thus not be a fair comparison:

annual_energy <- annual_energy[annual_energy$Year != 2017, ]
```

Now, extract the coal consumption from `'annual_energy'` and then generate a line plot of the annual trands in its consumption.

3. You can add more lines using `lines()` function to compare consumption of different energy sources. Plot coal, natural gas, nuclear and renewable energy together using different colors for the lines.

```
# Run this code in your R(Studio) to read in the data:
unique(annual_energy$Description)

#Construct separate data.frames for different energy sources:
coal <- annual_energy[annual_energy$Description == "Coal Consumption", ]
natgas <- annual_energy[annual_energy$Description ==
                        "Natural Gas Consumption (Excluding Supplemental Gaseous Fuels)", ]
nucl <- annual_energy[annual_energy$Description == "Nuclear Electric Power Consumption", ]
renew <- annual_energy[annual_energy$Description == "Total Renewable Energy Consumption", ]
```

Now, write code to generate the line plot. Remember to set the y-axis limit to be e.g. `c(0, 30)` using `'ylim'` argument in the `plot()` function, as different data.frames have different range of values. You should also change the x-axis and the y-axis to `"Year"` and `"Consumption"` using `'xlab'` and `'ylab'` arguments.

4. Next, generate the same graph but instead of lines plot points with different markers (using `'pch'` argument) to denote different sournces of energy. Use `points()` instead of `lines()`. Title your plot using 'main' argument:

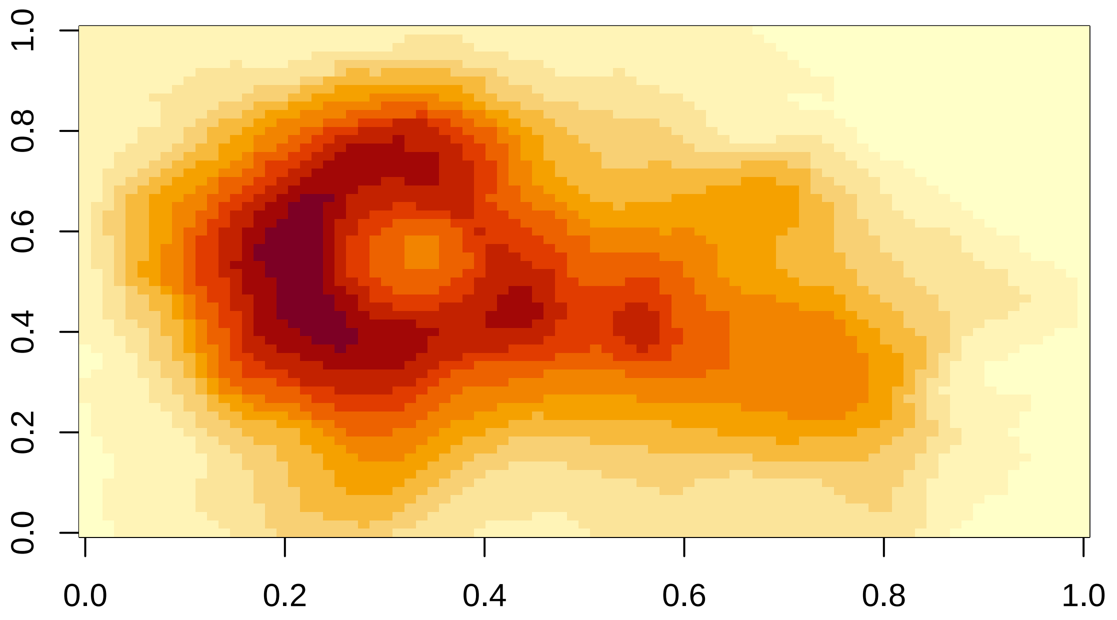
5. The built-in 'mtcars' data.frame contains information on different attributes and performance of a number of cars:

```
head(mtcars)
```

Use it to plot miles per gallon 'mpg' v. horse power 'hp' and show the car model on the plot using text() function:

6. R has a built in dataset `'volcano'` which gives topographic information for Maunga Whau on a 10m by 10m grid. Use an image function to show the data. And show contours on the plot.

```
image(volcano)
```



11. Histograms

```r
# Use data from your directory
colleges <- read.csv("Data/College.csv")

# Take a look at the data:
head(colleges)
```

1. Plot the histogram the graduation rate 'Grad.Rate' for universities in the entire dataset.
2. Plot the histogram of the graduation rate 'Grad.Rate' for private universities only.
3. Plot overlapping histograms of the graduation rate 'Grad.Rate' for private universities and non-private ones (use different colors):

12. Box Plots and Bar Plots

1. Using the College.csv data from the last exercises on Histograms, generate a box plot for the number of out-of-state students 'Outstate' for all universities in the dataset.
2. Do the same but for private and non-private separately.
3. In the following exercises you will generate a barplot for the 2016 GDP per capita for 46 countries based on OECD data from OECD.org website:

```r
link <- paste0("https://stats.oecd.org/sdmx-json/data/DP_LIVE/.GDP.TOT.USD_CAP.A/OECD?contentType=csv&d

link

# And then read in the data:
gdp <- read.csv(link)
```

Now create a barplot of gdp.

4. Now sort the countries by their GDP per capita and then plot the data.