In-class Lab 2

ECON 425 (Justin Heflin, West Virginia University)

January 20, 2023

The purpose of this in-class lab is to continue to familiarize yourself with RStudio, more specifically today we will be learning about data frames, data files, and how to plot graphics. The lab may be completed as a group, but each student should turn in their own work. To get credit, upload your .R script to the appropriate place on eCampus (“In-Class Labs’ ’ folder).

Object	Description	Comments
vector	The basic data object in R, consisting of one or more values of a single type (for example, character, number, or integer)	Think of this as a single column or row in a spreadsheet
matrix	A multidimensional object of a single type	When you have to store numbers in many dimensions
data frame	Data frames are a special kind of named list where all components have equal length	Data frames are similar to a spreadsheet

For starters

Open up a new R script (named `ICL2_XYZ.R`, where `XYZ` are your initials)

Data Frames and Data Files

For *R* users, it is important to make the distinction between a data set (= data frame in *R* terminology) which is a collection of variables on the same observational units and a data file which can include several data sets as well as other objects.

Data Frames

A data frame is an object that collects several variables (e.g., population, income, GDP, inflation, obesity rate, homicide rate,. . .) and can be thought of as a rectangular shape with the rows representing observational units and the columns representing the variables. As such, it is similar to a matrix. For us, the most important difference to a matrix is that a data frame can contain variables of different types (like numerical, logical, string and factor), whereas matrices can only contain numerical values. In *R*, we can work with several data sets stored as data frame objects at the same time.

Like a matrix, the rows can have names. Unlike a matrix, the columns always contains names which represent the variables. We can define a data frame from scratch by using the command **data.frame** or **as.data.frame** which transform inputs of different types (like a matrix) into a data frame.

```r
# Define one x vector
year <- c(2018, 2019, 2020, 2021, 2022, 2023)

# Define a matrix of y values
Shirts <- c(775, 26, 706, 345, 372, NA)
Pants <- c(624, 97, 542, 692, NA, 369)
Shoes <- c(430, NA, 419, 942, 81, 123)
#Too often when conducting a research project there will be missing observations,
#notated by "NA"

sales_matrix <- cbind(Shirts, Pants, Shoes)

rownames(sales_matrix) <- year
sales_matrix
```

```
##      Shirts Pants Shoes
## 2018    775   624   430
## 2019     26    97    NA
## 2020    706   542   419
## 2021    345   692   942
## 2022    372    NA    81
## 2023     NA   369   123
```

```r
sales <- as.data.frame(sales_matrix)
sales
```

```
##      Shirts Pants Shoes
## 2018    775   624   430
## 2019     26    97    NA
## 2020    706   542   419
## 2021    345   692   942
## 2022    372    NA    81
## 2023     NA   369   123
```

The outputs of the matrix `sales_matrix` and the data frame `sales` look exactly the same, but they behave differently. The difference can be seen in the Workspace window (top right panel). It reports the content of `sales_matrix` to be a "num [1:6, 1:3] 775 26 706…'' whereas the content of `sales` is "6 obs. of 3 variables''.

We can address a single variable *var* of a data frame *df* using the matrix-like syntax **df[,"var"]** or by stating **df$var**. This can be used for extracting the values of a variable but also for creating new variables.

```r
#Accessing a single variable
sales$Shirts

#Generating a new variable in a data frame
sales$total <- sales$Shirts + sales$Pants + sales$Shoes

#Result
sales
#Those missing observations screwed up our sum calculation for the total column
#One option to have R ignore those missing observations is to use the "rowSums" function
sales$total_2 <- rowSums(sales[,c("Shirts", "Pants", "Shoes")], na.rm = TRUE)
```

The square brackets `[]` is how we index something, whether it be a vector or a matrix. Inside the square brackets is where is define the rows and columns. The reason why there is nothing before the comma inside the square brackets is because we want all rows to be summed for each year. After the comma is where we define the columns, in this case, shirts, pants, and shoes. The `na.rm` simply removes NA values.

```
#Results when using rowSums function
sales
```

```
##      Shirts Pants Shoes total total_2
## 2018    775   624   430  1829    1829
## 2019     26    97    NA    NA     123
## 2020    706   542   419  1667    1667
## 2021    345   692   942  1979    1979
## 2022    372    NA    81    NA     453
## 2023     NA   369   123    NA     492
```

Subsets of Data

Sometimes, we do not want to work with a whole data set but onyl with a subset. We can do so by using the command **subset(df, criterion)**, where *criterion* is a logical expression which evaluates elements to **TRUE** for the rows which are to be selected.

```
# Full data frame
sales
```

```
##      Shirts Pants Shoes total total_2
## 2018    775   624   430  1829    1829
## 2019     26    97    NA    NA     123
## 2020    706   542   419  1667    1667
## 2021    345   692   942  1979    1979
## 2022    372    NA    81    NA     453
## 2023     NA   369   123    NA     492
```

```
# Subset: all years in which sales of Shirts were >=350
subset(sales, sales$Shirts>=350)
```

```
##      Shirts Pants Shoes total total_2
## 2018    775   624   430  1829    1829
## 2020    706   542   419  1667    1667
## 2022    372    NA    81    NA     453
```

```
#Subset: all years in which sales of Shoes were <=500
subset(sales, sales$Shoes<=500)
```

```
##      Shirts Pants Shoes total total_2
## 2018    775   624   430  1829    1829
## 2020    706   542   419  1667    1667
## 2022    372    NA    81    NA     453
## 2023     NA   369   123    NA     492
```

Basic Information on a Data Set

After loading a data set into a data frame, it is often useful to get a quick overview of the variables it contains. Suppose we want information on our data frame `sales`:

- **head(df)** displays the first few rows of data
- **str(df)** lists the *structure*, i.e. the variable names, variable type (numeric, string, logical, factor,. . .) ans the first few values
- **colMeans(df)** reports the averages of all variables and **summary(df)** shows summary statistics

```r
#Using head(df) on our sales data frame
head(sales)

#Using the structure function on our sales data frame
str(sales)

#Calculating the averages of all our variables
colMeans(sales, na.rm = TRUE)
#Since we have missing observations, we need to remove them by using na.rm
```

In your script, write the values you get from calculating the averages of all the variables in our sales data frame, round to the nearest hundredth, preceded by a comment (the hashtag symbol)

Loading data

We're going to load a data set from the `wooldridge` package. The data set is called `wage1`.

```r
library(wooldridge)
wage_data <- as.data.frame(wage1)
```

To ensure the data set loaded correctly you can use the View() function to examine the data set. For example, if you wanted to double check our wage data data set, use the following code `View(wage_data)`

We called the data set `wage_data`, but you can call it whatever you want: `mydata`, `data123`, whatever.

Next, let's explore this data set a bit further by using commands we learned earlier.

```r
#Using the structure function allows us to view variable names, variable types,
#as well as the first few values
str(wage_data)
#In your R script, write the variable type (integer, numeric, character),
#for the variable "lwage", preceded by a hashtag symbol

#Allows us to extract an element from our wage_data data frame
wage_data[197,3]
#Returns the observation from the 197th row, Experience column which is
#the third column
#In your R script write the value that gets returned from the previous line of code,
#preceded by a hastag symbol

#Index by column name
wage_data["educ"]
```

```r
#Calculate the averages for all of the variables in our wage_data data frame
colMeans(wage_data)
#In your R script write the value for the average number of years of education,
#round to the nearest hundredth, preceded by a hashtag symbol

#Calculate the average of a specific variable in our wage_data
mean(wage_data$tenure)
#In your R script write the value for the average tenure in this data set,
#round to the nearest hundredth, preceded by a hashtag symbol

#Often times we want to know more than the average of a variable
#Statistics like the minimum value, maximum value, or the median
summary(wage_data)
```

Plots, Graphics

Suppose you want to visualize the entire distribution of education. You would use the following code:

```r
plot(wage_data$educ)
```

We can also dress up this graph to make it look a little cleaner so to speak by creating better labels as well as different point symbols.

```r
plot(wage_data$educ, pch = 18, cex=0.5, col="red", main = "Education Distribution",
     xlab = "Observations", ylab = "Years of Education")
#"pch" refers to the point symbol (there are 19 different options, pick a number between
#0 and 18), "cex" refers to the size of the point symbols, "col" refers to the color of
#the point symbols, "main" refers to the main title,
#"xlab" refers to the horizontal axis label, "ylab" refers to the vertical axis label
```

Exporting to a File

By default, a graph generated in one of the ways we discussed above will be displayed in its own window. The graph should populate in the Plots window (bottom right panel). This window also has an Export button which allows us to save the generated graph/plot in *different* graphics formats. To save a plot simply click the Export button, select either `Save as Image...` or `Save as PDF...` (personally, I always use the PDF option but dealer's choice). From there name your plot in the `File name` textbox. Next is to determine the location (desktop, folder, DropBox) where you would like the plot to be saved. To do that, click the `Directory` button and select where you would the plot to saved.