Introduction to R, Day 1 Handout

In this handout, we cover the following topics and R commands. The topics covered corresponds to Kosuke Imai's QSS Chapter 1.

Topics

- Organizing your files
- Quick Recap: RStudio Interface to R
- Using R as a Calculator
- Preparing and Working in an R Script for Analysis
- Creating and Using Objects in R
- Working with Vectors
- Basic Operations on Vectors / Basic Functions
- Reading / Loading Data into R
- Working with Data Frames: Summarizing Data
- Packages
- Environment / Workspace
- Saving Objects
- Creating Data Files
- Help Files
- The Structure of R

R Commands

- Doing arithmetic operations `+`, `-`, `*`, `^`, `sqrt()`, `log()`
- Setting and displaying the working directory using `setwd()` and `getwd()`
- Creating an object using the assignment operator `<-`
- Using `class` to obtain information about types of objects
- Using `c()` to create and combine vectors
- Using `[]` to access elements of vectors
- Basic functions/basic operations on vectors: `length()`, `min()`, `max()`, `range()`, `mean()`, `median()`, `sum()`, `abs()`, `unique()`
- Using `names()` to assign or access names of objects
- Reading/loading data using `read.csv()`, `read.table`, and `load()`
- Summarizing the data using `head()` and `summary()`
- Obtaining the dimension, number of rows and columns, of data with `dim()`, `nrow()`, and `ncol()`
- Accessing variables from data frames with `$`
- Creating new variables with `$`
- Using `[row, column]` to recover elements of a data frame
- Summarizing variable categories with `table()`
- Listing objects in the workspace with `ls()`
- Saving objects with `save()` and the workspace with `save.image()`
- Creating data files using `write.table()` and `write.csv()`
- Obtaining help for R with `help()`, `?`, and `??`

Before We Start: Organizing Your Files

- Where do you want store all your work on this workshop? This will be important when we start working with R scripts and data sets.
- I will recommend putting all the material in the same folder, and store it somewhere that you can easily find. For example, I created a folder named 'RWorkshop2021' on my desktop.
- We will turn back to this point later when we discuss setting our working directory, which is the place where we want R to look for our files.

Using R as a Calculator

- First, we'll learn how to use R as a calculator.
- We will use the console (the bottom left window) where we can directly enter R commands. You can think console like your scratch paper. Here, you can do some quick computations or try out some codes.
- Some of the key arithmetic commands are: addition `+`, subtraction `-`, multiplication `*`, division `/`, exponentiation `^`, taking the square root of a number `sqrt()`, taking the log of a number `log()`. Parentheses `( )` specify the order of operations.
- Type the following arithmetic operations in your console and and hit <enter> or <return>. You will see that the console show the output of the commands you type.

```
13 + 2
```

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

```
13 - 2
```

```
## [1] 11
```

```
13 * 2
```

```
## [1] 26
```

```
13/2
```

```
## [1] 6.5
```

```
13^2
```

```
## [1] 169
```

```
sqrt(25)
```

```
## [1] 5
```

```
log(15)
```

```
## [1] 2.70805
```

- R follows the rules of the order of operations (PEMDAS), so we use parentheses `()` to set the order of operations.

```
13 * (11 - 2)
```

```
## [1] 117
```

♠ Coding Tip

R ignores spaces, so `5+3` will return the same result as `5 + 3`. However, we add a space before and after the arithmetic operators to make it easier to read.

Preparing and Working in an R Script for Analysis

- When you want to start working on a new analysis in R, you can follow a *four step procedure:*
  1. Open an R Script
  2. Set your working directory
  3. Save your R script
  4. Start working inside your R script

Opening an R Script

- If you haven't already, you can open an R Script by clicking:

File → New File → R Script

- Text editor window (the the upper left window) is the main place where we will execute R commands in our R script.

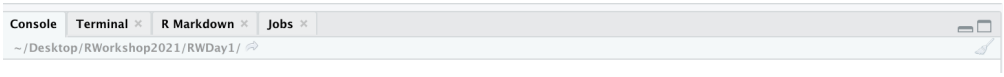
Setting working directory with 'setwd()'

- When we use R with RStudio, the first thing we always want to do is to set our working directory. In other words, we want to tell R a folder in our computer where we will save your R scripts and datasets.
- Earlier, we created a folder for the workshop – 'RWorkshop2021' – with this in mind.
- Inside this folder, we can also create specific folders for days of the workshop. For example, I created a folder called 'RWDay1' for today's workshop. I have also put all my relevants datasets to this folder.
- I suggest creating two folders inside your main workshop folder. The first folder can be a general folder for today. Once you create this folder, you can also create a dataset folder for today as well.
- Now, we are ready to set our working directory. The easiest way to do this is to do it from the Session menu. So, let's select:

Session → Set Working Directory → Choose Directory...

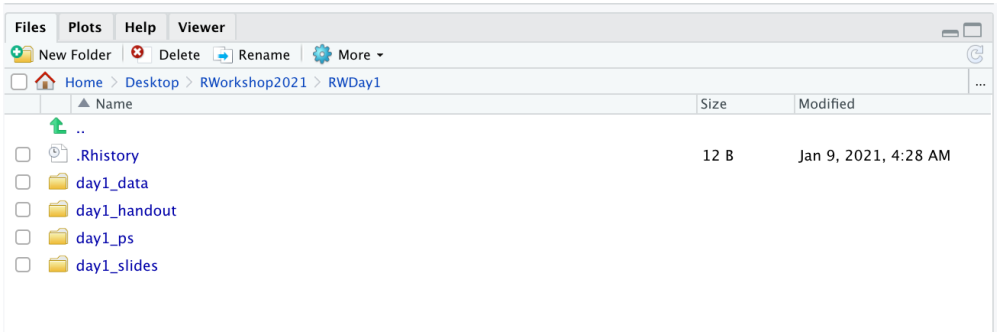
- This will tell R the folder that you will use throughout your analysis. When you're done, you'll notice some changes. These changes will give you hints to check whether you were able to set your working directory.
- First, you'll notice that the address in grey at the top of your console window has changed to show that you're in the folder that you selected for Day 1. You can tell this by looking at the path above the console.

Mine looks like this:



- Second, you will also see a change in your bottom-right window, files tab After setting your directory, update the file view and you'll see that your working directory has changed to the folder you selected. Here, you can see what's inside your folder.

Mine looks like this:



⊙ Note on Working Directory

`Working directory` is an important element in the structure of R because it's where R looks to find files. It's where our files land when we save them. In the future, you might find yourself wondering `Why can't R find this file? I can see it here!` Usually the reason is that although the file is there, the working directory is somewhere else And, where you look and where R look is different. To avoid this confusion, it's a good practice to keep the files that you can see in the bottom right window in sync with your working directory. You can always do that by pressing the grey arrow at the top (next to the path).

- Suppose that you forget or not sure where you had set your working directory... When you type `getwd()` in your console, R will tell you where you had set your current working directory.

```
getwd()
```

```
## [1] "/Users/burcukolcak/Desktop/RWorkshop2021/RWDay1"
```

- What if you want to change your working directory? You can follow the same procedure: **Session** → **Set Working Directory** → **Choose Directory...**. You also have another option. You can use the `setwd()` and manually change your desired working directory.

```
setwd("/Users/burcukolcak/Desktop/RWorkshop2021/RWDay1")
```

Saving an R Script

- Now, we are ready to save our R script to our working directory, which corresponds to our main folder. To save your script:

File → Save As

- Make sure that the folder you save your R script is the same folder you have choosen for your working directory.
- All of our R scripts will have the '.R extension'.

Preparing an R Script

- Now, before we explore new topics, we want to prepare our R script.
- The comment character `#` will play an important role when we work in an R script. This will be especially important as our code gets more complex. It allows us to annotate our R code behind commands or on a separate line. Everything that follows `#` will be ignored by R. We should annotate our own code for two main reasons: for us to remember what each part of the code does, and for others to understand what we are trying to do in our analysis.

♠ Coding Tip

It is common to use a double comment character `##` if a comment is made for a seperate line and use a single comment character `#` if a comment is made within a line after an R command. Below is an example:

```
## Double comment vs. single comment

## Commenting on a seperate line
5 + 3

5 + 3  # commenting behind a command
```

- Now, we are ready to prepare our first R script.
- First things first, let's give a title to our R script using the double comment character `##`.
- Earlier, we discussed that we always want to set our working directory first. Let's set our your working directory and display our working directory. And, annotate each command in seperate lines.

```
####################

## R Workshop Day 1

## set working directory
setwd("/Users/burcukolcak/Desktop/RWorkshop2021/RWDay1")

## check working directory
getwd()
```

- Next, let's do some basic arithmetic operations similar to what we did before. This time we will execute these commands in our R script.
- Type the following arithmetic operations in your R script and use the symbol **#** comment character for explaining what you do:

```
## Some Basic Arithmetic Operations

## Add 3 + 7
3 + 7
```

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

```
13 - 3  # subtract 13 - 3
```

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

```
## Take the square root of 100 using sqrt()
sqrt(100)
```

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

In the end, we want to see something like follows:

```
1
2  ####################
3  ## R Workshop Day 1
4
5  ## set working directory
6  setwd("/Users/burcukolcak/Desktop/RWorkshop2021/RWDay1")
7
8  ## check working directory
9  getwd()
10
11 ## Some Basic Arithmetic Operations
12
13 ## Add 3 + 7
14 3 + 7
15
16 13 - 3 # subtract 13 - 3
17
18 ## Take the square root of 100 using sqrt()
19 sqrt(100)
20
21
22
```

Executing Commands in our R script

- To execute commands on your R script:
  - Place your cursor at the end of the line of code (far right), and hit <command + return> on a Mac; <control + return> on Windows.
  - Or, place your cursor at the end of the line of code (far right), and click 'Run' in the top right corner of the upper-left window.

- After you execute each command, the result wil pop up in your console. [1]

Creating and Using Objects

- In R, we can store various information as an object with a name of your choice. Objects are very handy because they serve as "shortcuts" to some piece of information or data that we would like to save and use throughout our analysis.
- We use the assignment operator `<-` to assign a name to an object.

♠ Coding Tip

There are some specific rules we need to follow when we choose a name for our objects in R. 1. Object names cannot begin with numbers (but it can contain numbers) and no spacing is allowed. 2. In general, we should avoid special characters such as % and $ because these have specific meanings in R. 3. We should also try using intuitive and informative names for our objects. 4. Object names are case sensitive. For example, if your object name is `butterfly`, you will get an error when you type `Butterfly` to print the value of the object.

- In RStudio, once you create an object, you will see the object you created in the your Workspace/Environment (the upper right window).
- Previously, we added `3 + 7`. Let's store the result of this calculation as an object named sum37.

```
## Create an object called sum 37
sum37 <- 3 + 7
```

- By default, R will print the value of the object to the console if we just type the name of the object and run this command.

```
## display the value of sum37
sum37
```

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

- Or, you can explicitly tell R to print the value of the object by using the `print()` function.

```
## print the value of sum37
print(sum37)
```

[1] Explanation of this part taken from Prof.Katie McCabe's MLE book.

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

- Let's create another object to store information about a previous calculation we did `13-3`. Let's name this object `result`.

```
## Create an object called result
result <- 13 - 3
## display the value of result
result
```

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

- Let's use the same object name for storing information about another calculation.

```
## Create an object called result
result <- 5 * 12
## display the value of result
result
```

```
## [1] 60
```

- Oops..We lost the previous value we assigned to this object! If we assign a different value to the same object name, then R will change the value of the object we previously created. Therefore, we need to be careful about this if we plan to use the previously assigned information later. To avoid this, for example, we can instead name the first object `result` and second object `result2`

```
## Create an object called result
result <- 13 - 3
result
```

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

```
## Create an object called result2
result2 <- 5 * 12
result2
```

```
## [1] 60
```

- Previously, we noted that object names are case sensitive. Let's see what happens when we write `Result` instead of `result`.
- We can also store information about other types of objects different from numbers. For example, we can store information about characters as follows. In character strings, spacing is allowed.

```
## Create an object called statename
statename <- "New Jersey"
statename
```

```
## [1] "New Jersey"
```

- How does R recognize diffrent types of objects? In other words, how does it differentiate between a number and character strings? R recognizes different types of objects by assigning each object to a `class`. Classes are important in R because it allows R to categorize objects into classes and then consider this when performing appropriate operations.
- When we want to learn the class of an object, we will use the function `class()`.
- A number is stored as a `numeric` object whereas a character string is recognized as a `character` object.

```
class(result)  # numeric variable
```

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

```
class(statename)  # chracter variable
```

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

```
class(log)  # function
```

```
## [1] "function"
```

- There are many other classes in R, and we will revisit this topic throughout the workshop.

℧ Practice Exercises

U.S. Treasury Securities held by foreign countries increased from 3708.8 billion in January of 2010 to 4009.2 billion in June of 2010. In January, China held 889.0 billion in U.S. Treasury Securities. By the close of June, Chinese holdings decreased to 843.7 billion. In contrast, Japanese holdings increased from 765.4 billion in January to 803.6 billion in June. [2]

1. Open a new R script and give it the following title `Exercise1`.
2. Copy this question to your script and annotate it using the `##` comment character.
3. Create two seperate objects called `us01` and `us06` to store information about holdings of the US in January of 2010 and June of 2010 by. Print the values of these objects by typing their names. Annotate your steps using the `##` character.
4. Store information about holdings of the China in January of 2010 and June of 2010 by creating two seperate objects called `china01` and `china06`. Print the values of these objects by typing their names. Annotate your steps using the `##` character.
5. Store information about holdings of the Japan in January of 2010 and June of 2010 by creating two seperate objects called `japan01` and `japan06`. Print the values of these objects by typing their names. Annotate your steps using the `##` character.
6. Create three new objects for storing information about the differences between holdings of January of 2010 and June of 2010 for each country. You can call these objects `diffus`, `diffchina`, and `diffjapan` respectively. Print the values of these objects by typing their names. Annotate your steps using the `##` character.
7. Check the class of objects `diffus`, `diffchina`, and `diffjapan` using the `class()` function.

[2] Adapted from Professor Kosuke Imai's teaching materials

Working with Vectors

- A vector is simply a collection of information (numbers or character strings) stored in a specific order.
- In R, vectors are constructed by using the `c()` function, which is used to *combine* multiple values together into a vector.
- This will allow us to create a `data vector` containing multiple values with commas separating different elements of the vector we are creating. We will see that vectors would become very useful for many operations we do in R.

Creating Vectors

- To understand the logic of vectors, we are going to create a set of vectors, for some columns of the following table. This table contains data from V-Dem about the top 5 main autocratizing countries between 2009-2019 [3]

	CHANGE	LDI 2009	LDI 2019	REGIME TYPE 2009	REGIME TYPE 2019
Hungary	−0.36	0.76	0.40	Liberal Democracy	Electoral Autocracy
Turkey	−0.36	0.46	0.10	Electoral Democracy	Electoral Autocracy
Poland	−0.33	0.83	0.50	Liberal Democracy	Electoral Democracy
Serbia	−0.27	0.53	0.25	Liberal Democracy	Electoral Autocracy
Brazil	−0.25	0.76	0.51	Electoral Democracy	Electoral Democracy

- Let's create vectors for some columns by using the `c()` function. We enter the data one value at a time, each separated by a comma.

```
## Create a vector the change variable
change <- c(-0.36, -0.36, -0.33, -0.27, -0.25)
change
```

```
## [1] -0.36 -0.36 -0.33 -0.27 -0.25
```

```
## Create a vector the ldi 2009 variable
ldi2009 <- c(0.76, 0.46, 0.83, 0.53, 0.76)
ldi2009
```

```
## [1] 0.76 0.46 0.83 0.53 0.76
```

```
## Create a vector the ldi 2019 variable
ldi2019 <- c(0.4, 0.1, 0.5, 0.25, 0.51)
ldi2019
```

```
## [1] 0.40 0.10 0.50 0.25 0.51
```

[3] Source: V-Dem Democracy Report 2020

```r
## Create a vector the regime 2009 variable
regime2009 <- c("liberal democracy", "electoral democracy",
    "liberal democracy", "liberal democracy", "electoral democracy")
regime2009
```

```
## [1] "liberal democracy"   "electoral democracy" "liberal democracy"
## [4] "liberal democracy"   "electoral democracy"
```

℧ A Short Practice Exercise

Now, its your turn.

1. Create a vector the column regime 2019
2. What is the class of this vector?

- We can subtract vectors from each other. By subtracting `ldi2019` from `ldi2009`, we can calculate the change column. Let's call this change2

```r
## calculate the diff between ldi 2019 and ldi 2009
change2 <- ldi2019 - ldi2009
change2
```

```
## [1] -0.36 -0.36 -0.33 -0.28 -0.25
```

- We can also use the `c()` function to combine multiple vectors

```r
## combine ldi2009 and ldi 2019
ldi.all <- c(ldi2009, ldi2019)
ldi.all
```

```
##  [1] 0.76 0.46 0.83 0.53 0.76 0.40 0.10 0.50 0.25 0.51
```

Accessing Elements of a Vector / Indexing

- What if we want to access specific elements of a vector? We will use the the square brackets, `[ ]` to access specific elements within the vector. This is called 'indexing'.
- For example, suppose that we want the 3rd value of the vector we created for `change`. This corresponds to the country change in Poland's democracy score.

```r
change[3]  # access and print the 3rd value of the vector change
```

```
## [1] -0.33
```

- Multiple elements can be extracted via a vector of indices within square brackets.

```
change[c(3, 4)]  # access and print the 3rd and 4th value of the vector change
```

```
## [1] -0.33 -0.27
```

- Also within square brackets the dash symbol `-`, removes the corresponding element from a vector. Note that none of these operations change the original vector.

```
change[-3]  # remove the third value of the vector change
```

```
## [1] -0.36 -0.36 -0.27 -0.25
```

Basic Operations on Vectors/Basic Functions

- One way we will use R a lot is through functions, which will allow us to act on or get information about vectors and other objects.
- The following functions will be useful for doing basic operations on vectors:
- `length(x)` calculates the number of elements/entries in the `x` vector.
- `min(x)` calculates the smallest value in the `x` vector.
- `max(x)` calculates the largest value in the `x` vector .
- `mean(x)` calculates the average value in the `x` vector (that is the sum of the entries divided by the number of entries).
- `sum(x)` calculates the sum of the values in the `x` vector.
- `abs()` returns the absolute values of the `x` vector.
- `unique(x)` returns a set of unique values from the `x` vector.
- Turning back to our running example, let's use these functions to get information on our change vector.

```
length(change)
```

```
## [1] 5
```

```
min(change)
```

```
## [1] -0.36
```

```
max(change)
```

```
## [1] -0.25
```

```
mean(change)
```

```
## [1] -0.314
```

```r
sum(change)
```

```
## [1] -1.57
```

```r
abs(change)
```

```
## [1] 0.36 0.36 0.33 0.27 0.25
```

```r
unique(change)
```

```
## [1] -0.36 -0.33 -0.27 -0.25
```

Giving Names to Vectors

- We can also give names to our vectors. Recall the change vector that we have been working on.

```r
change
```

```
## [1] -0.36 -0.36 -0.33 -0.27 -0.25
```

- Although we know that each element of this vector represents a country, it is difficult to remember these. So, we might want to assign names to elements of this vector.
- Currently no names assigned to the change vector yet.

```r
names(change)
```

```
## NULL
```

- The `names()` function can access and assign names to elements of a vector.

```r
names(change) <- c("Hungary", "Turkey", "Poland", "Serbia",
    "Brazil")
change
```

```
## Hungary  Turkey  Poland  Serbia  Brazil
##   -0.36   -0.36   -0.33   -0.27   -0.25
```

♠ Coding Tip

Systematic spacing and indentation are essential when we are coding. For example, we added spaces after a comma. We also place spaces around all binary operators such as `<-`, `=`, `+`, and `-`.

Reading/Loading Data into R

- We can slowly start working with a real dataset to obtain basic information about it.
- But, first, we need to know how to read data into R.
- We will use different versions of the `UNpop` dataset. On Slack, you will find three different versions of the same dataset `UNpop.csv`, `UNpop.tsv` and `UNpop.R`. After downloading these datasets, we need put them into our dataset folder.
- Two functions can be particularly useful when we first load a dataset into R:

1. We can use the `head()` to check first five rows of this dataset.
2. We can use the `summary()` function to summarize the variables of this dataset.

These two functions will help us check whether we have loaded in the data without error.

Now, we will load each of these data files respectively.

Reading in a comma-separated `.csv` file

```
UNpop <- read.csv("day1data/UNpop.csv")
```

```
head(UNpop)
```

```
##   year world.pop
## 1 1950   2525779
## 2 1960   3026003
## 3 1970   3691173
## 4 1980   4449049
## 5 1990   5320817
## 6 2000   6127700
```

```
summary(UNpop)
```

```
##       year        world.pop
##  Min.   :1950   Min.   :2525779
##  1st Qu.:1965   1st Qu.:3358588
##  Median :1980   Median :4449049
##  Mean   :1980   Mean   :4579529
##  3rd Qu.:1995   3rd Qu.:5724258
##  Max.   :2010   Max.   :6916183
```

Reading in a tab-delimited .tsv or .txt file

```
UNpop2 <- read.table("day1data/UNpop.tsv", header = TRUE)
```

```
head(UNpop2)
```

```
##   year world.pop
## 1 1950   2525779
## 2 1960   3026003
## 3 1970   3691173
## 4 1980   4449049
## 5 1990   5320817
## 6 2000   6127700
```

```
summary(UNpop2)
```

```
##       year        world.pop
##  Min.   :1950   Min.   :2525779
##  1st Qu.:1965   1st Qu.:3358588
##  Median :1980   Median :4449049
##  Mean   :1980   Mean   :4579529
##  3rd Qu.:1995   3rd Qu.:5724258
##  Max.   :2010   Max.   :6916183
```

- For the .tsv file, we set the parameter header to TRUE to tell R that the first row of the file should be used to name each column. Let's see happens if we don't use this parameter:

```
UNpop3 <- read.table("day1data/UNpop.tsv")
```

```
head(UNpop3)
```

```
##     V1        V2
## 1 year world.pop
## 2 1950   2525779
## 3 1960   3026003
## 4 1970   3691173
## 5 1980   4449049
## 6 1990   5320817
```

Reading in an R data file

```
load("day1data/States.Rdata")
```

```
head(States)
```

```
##                 State HouseholdIncome McCainVote Region ObamaMcCain Population
## Alabama       Alabama           38160       60.4      S           M   4.525375
## Alaska         Alaska           57071       60.2      W           M   0.657755
## Arizona       Arizona           46693       53.8      W           M   5.739879
## Arkansas     Arkansas           37458       58.8      S           M   2.750000
## California California           54385       37.2      W           O  35.842038
## Colorado     Colorado           53900       44.9      W           O   4.601821
```

```
##            HighSchool   GSP College NonWhite ObamaVote
## Alabama          82.4 33264    24.6     29.4      39.6
## Alaska           90.2 59238    29.2     26.2      39.8
## Arizona          84.4 36457    30.8     31.1      46.2
## Arkansas         79.2 31215    26.7     17.8      41.2
## California       81.3 44894    31.1     53.0      62.8
## Colorado         88.3 46416    38.0     22.5      55.1
```

```
summary(States)
```

```
##     State           HouseholdIncome   McCainVote    Region  ObamaMcCain
##  Length:50          Min.   :34343   Min.   :26.60   MW:13   M:22
##  Class :character   1st Qu.:42494   1st Qu.:40.83   NE:11   O:28
##  Mode  :character   Median :47368   Median :46.80   S :13
##                     Mean   :48051   Mean   :47.76   W :13
##                     3rd Qu.:53305   3rd Qu.:56.17
##                     Max.   :66752   Max.   :65.60
##    Population        HighSchool         GSP           College
##  Min.   : 0.5059   Min.   :78.30   Min.   :27829   Min.   :20.20
##  1st Qu.: 1.7639   1st Qu.:83.80   1st Qu.:36824   1st Qu.:27.62
##  Median : 4.1699   Median :87.30   Median :39209   Median :30.60
##  Mean   : 5.8620   Mean   :86.46   Mean   :40764   Mean   :31.16
##  3rd Qu.: 6.3622   3rd Qu.:88.80   3rd Qu.:43945   3rd Qu.:34.50
##  Max.   :35.8420   Max.   :92.30   Max.   :66961   Max.   :43.40
##     NonWhite       ObamaVote
##  Min.   : 4.80   Min.   :34.40
##  1st Qu.:12.18   1st Qu.:43.83
##  Median :17.95   Median :53.20
##  Mean   :22.80   Mean   :52.24
##  3rd Qu.:32.98   3rd Qu.:59.17
##  Max.   :73.30   Max.   :73.40
```

Working with a Data Frame: Summarizing Data

- When we load datasets into R, R will store them as `data frame` objects.

```
class(UNpop)
```

```
## [1] "data.frame"
```

- A `data.frame` is an object in R that is basically like a spreadsheet which contains one column for each variable, and one row for each observation.
- Below are some basic operations we can do on data frames to obtain information about the UNpop data set:
- `names()` returns the column (variable) names of the data.

```
names(UNpop)
```

```
## [1] "year"      "world.pop"
```

- `ncol()` returns the number of columns in the data.

```
ncol(UNpop)
```

```
## [1] 2
```

- `nrow()` returns the number of rows of the data.

```
nrow(UNpop)
```

```
## [1] 7
```

- `dim()` returns a vector of the number of rows and the number of columns (the dimension of the data).

```
dim(UNpop)
```

```
## [1] 7 2
```

- `summary()` provides a summary of each variable in the data.

```
summary(UNpop)
```

```
##       year        world.pop
##  Min.   :1950   Min.   :2525779
##  1st Qu.:1965   1st Qu.:3358588
##  Median :1980   Median :4449049
##  Mean   :1980   Mean   :4579529
##  3rd Qu.:1995   3rd Qu.:5724258
##  Max.   :2010   Max.   :6916183
```

Subsetting a Data Frame I

- We'll often need to access different parts of a data frame. For example, we will want to extract information about a specific variable and find its mean. Or we will want to extract the 5th observation of a specific variable. This corresponds to subsetting a data frame in R.
- To select a particular variable from the data frame, we can use the dollar `$` operator.
- In the UNpop data frame, to select the year variable we will use the `$` operator. Here, `UNpop$year` will be a vector of just the `year` column of the `UNpop` data frame.

```
UNpop$year
```

```
## [1] 1950 1960 1970 1980 1990 2000 2010
```

Creating A New Variable with $

- We will also use the `$` sign to create a new variable. Suppose that we want to create a new variable that subtracts 5 from each year.

```
UNpop$year2 <- UNpop$year - 5
UNpop$year2
```

```
## [1] 1945 1955 1965 1975 1985 1995 2005
```

℧ A Short Practice Exercise

1. Use the `$` to select the variable `world.pop` variable from the `UNpop` data frame.
2. Calculate the mean world population over this time period

Subsetting a Data Frame II

- You can also use the square brackets `[ ]` to subset from the data frame.
- As with vectors, square brackets may be used to call specific portions of the data frame. As opposed to vectors, however, with a data frame we need to tell R whether we want to subset the rows or columns.
- To do so, we will use a comma and the bracket will have the following form: `[rows, columns]` where the expression before the comma will select the rows and the expression after the comma will select the columns. This will allow us to call specific rows and columns by either row/column numbers or row/column names
- `UNpop[,"world.pop"]` will select the `world.pop` variable from `UNpop`:

```
UNpop[, "world.pop"]
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

- `UNpop[1,]` will select the first row of `UNpop`

```
UNpop[1, ]
```

```
##   year world.pop year2
## 1 1950   2525779  1945
```

- `UNpop[c(1,2,3),]` will select the first three rows of `UNpop`

```
UNpop[c(1, 2, 3), ]
```

```
##   year world.pop year2
## 1 1950   2525779  1945
## 2 1960   3026003  1955
## 3 1970   3691173  1965
```

- `UNpop[1:3, "world.pop"]` will select the first three values of the `world.pop` variable of `UNpop`

```r
UNpop[1:3, "world.pop"]
```

```
## [1] 2525779 3026003 3691173
```

- Most of the time we will want to summarize a variable's categories. To do so, we can use the `table()` function.
- For example, suppose that we want to summarize the values of the year variable in the UNpop data frame:

```r
table(UNpop$year)
```

```
##
## 1950 1960 1970 1980 1990 2000 2010
##    1    1    1    1    1    1    1
```

Packages

- One of R's mains strengths is the community of R users who contribute to the development of R with packages.
- These packages are available through the Comprehensive R Archive Network
- Thus far, we haven't used any packages. We were able to do everything on `base R` functions. In other words, we used functions that already exist in R.
- For using other functions, we will install packages. Throughout the workshop, we will use various packages. For example, we will install a package called '`ggplot2` for data visualization in the following days.
- Suppose that we wish to load a data file produced by another statistical software such as Stata with the .dta extension. R has a package called `readstata13` which is useful when dealing with Stata data files.
- To use any package in R, if we are using that package for *the first time*, we need to do two things:

1. Install the package using `install.packages("packagename")`
2. Load the package using `library(packagename)`

⊙ Note on Packages

Package installation needs only to occur once. This means that when you want to use the `readstate13` package next time, you only need to load the `library()` in the beginning of your R script.

- Now, let's try to read a Stata file into R:

```r
library(readstata13)  # load package
UNpopdta <- read.dta13("day1data/UNpop.dta")  #a STATA data file with .dta extension
head(UNpopdta)
```

```
##   year world_pop
## 1 1950  2525.779
## 2 1960  3026.003
## 3 1970  3691.173
## 4 1980  4449.049
## 5 1990  5320.817
## 6 2000  6127.700
```

```
summary(UNpopdta)
```

```
##       year        world_pop
##  Min.   :1950   Min.   :2526
##  1st Qu.:1965   1st Qu.:3359
##  Median :1980   Median :4449
##  Mean   :1980   Mean   :4580
##  3rd Qu.:1995   3rd Qu.:5724
##  Max.   :2010   Max.   :6916
```

Environment/Workspace

- All the objects we created thus far appear in our **workspace/environment** (the upper right window).
- If we want to list the set of all objects in our current R session. We can use the function `ls()`:

```
ls()
```

```
##  [1] "change"     "change2"    "china01"    "china06"    "diffchina"
##  [6] "diffjapan"  "diffus"     "japan01"    "japan06"    "ldi.all"
## [11] "ldi2009"    "ldi2019"    "regime2009" "regime2019" "result"
## [16] "result2"    "statename"  "States"     "sum37"      "UNpop"
## [21] "UNpop2"     "UNpop3"     "UNpopdta"   "us01"       "us06"
```

The workspace will include all objects you created - variables, vectors, data frames, and a history of all commands you've run in this session.

Saving Objects

- The objects we create during an R session are only temporarily stored to the workspace. Unless you save objects from the workspace before exiting R, you will lose all objects and changes to data.
- To save any object, we can use `save(xxx, file = "yyy.RData")` function where `xxx` is the object name and `yyy.RData` is the file name. Multiple objects can be listed and they will be stored as a `single .RData file`. The extension `.RData` should always be used for the file name. Unless the full path is specified, objects will be saved to the working directory.
- For example, suppose we want to save one of the objects we created before:

```
save(change, file = "change.RData")
```

- To save the entire workspace (all objects), we can use the `save.image()` function with a `.RData` file name.

```
save.image(file = "Handout1.RData")
```

- To access the objects saved in the .RData file, simple double click the file or use the load() function as above

⊙ **Note on Saving Workspace**

When you close RStudio, you will be asked if you want to save your workspace. In general, we will say *no* to this question because we have all the ingredients in our script to replicate our analysis again. We usually save scripts, not the work space.

Creating Data Files

- If you create new variables by building on an existing dataset, you may also want to generate a text or csv files.
- You can use `write.table()` or `write.csv()` to generate data files.
- Suppose that we want to generate new data files of the UNpop data frame.

```
write.csv(UNpop, file = "UNpopnew.csv")  # generate a csv file
write.table(UNpop, file = "UNpoplast.csv")  # generate a text file
```

Help Files

- If you want to access RStudio's help functions, click on the `Help` tab in the bottom left window, and type in your question.
- If you want to access help files from the text editor window (command line), type in `?command`, e.g. `?table` to learn about the table function.
- If you do not remember the exact name of an R command, type `??tab` and R will use `fuzzy matching` to suggest some commands.

The Structure of R

So far, we have learned the following elements of R:

- The **R script**. This is where you conduct your main analysis. We use to the text editor window (the upper left window) to work inside an R Script. Your script contains all of the R commands necessary to replicate your analysis, as well as comments explaining the code.
- The **R console**. If we just want to see what a command does, or quickly calculate some quantity, we can enter your command directly into the R console.

- The **working directory**. This is the directory from where R will find your data files. The first thing we do when we start an analysis is to set and display our working directory to be sure that we and R are at the same page :)
- The **environment/workspace**. Workspace contains set of all objects in your current R session.

℧ Practice Exercises

Dataset: turnout.csv (in day1data folder)

Surveys are frequently used to measure political behavior such as voter turnout, but some researchers are concerned about the accuracy of self-reports. In particular, they worry about possible *social desirability bias* where, in postelection surveys, respondents who did not vote in an election lie about not having voted because they may feel that they should have voted. Is such a bias present in the American National Election Studies (ANES)? ANES is a nationwide survey that has been conducted for every election since 1948. ANES is based on face-to-face interviews with a nationally representative sample of adults [4]

Table below displays the names and descriptions of variables in this data set: (CORRECT THIS TABLE!)

Name	Description
year‘	election year
ANES‘	ANES estimated turnout rate
VEP‘	voting eligible population (in thousands)
VAP‘	voting age population (in thousands)
total‘	total ballots cast for highest office (in thousands)
felons‘	total ineligible felons (in thousands)
noncitizens‘	total noncitizens (in thousands)
overseas‘	total eligible overseas voters (in thousands)
osvoters‘	total ballots counted by overseas voters (in thousands)

1. Load the dataset into a data frame called `turnout`.
2. Check the first six rows of this data frame.
3. Obtain a summary of the data. What is the range of years covered in this dataset?
4. Check the dimensions of the data. How many rows and columns are there?
5. Calculate the turnout rate based on VAP (the voting age population). Note that for this dataset, we must add the total number of eligible overseas voters since the VAP variable does not include these individuals in the count. Add years as informative labels to see the turnout change over time.
6. Calculate the turnout rate based on VEP (the voting eligible population). Add years as informative labels to see the turnout change over time.
7. Calculate the difference between VEP and VAP based turnouts.
8. Compute the differences between the VAP and ANES estimates of turnout rate. How big is the difference on average? What is the range of the differences?
9. Conduct the same comparison for the VEP and ANES estimates of voter turnout.
10. (optional) Calculate the VEP turnout rate for presidential elections and midterm elections. Note that the data set excludes the year 2006. Does the bias of the ANES estimates vary across election types?
11. (practice for later) Calculate the ANES turnout rate for presidential elections and midterm elections. Note that the data set excludes the year 2006. Does the bias of the ANES estimates vary across election types?

Note: Some of the material and practice exercises in this handout are adapted from Kosuke Imai's teaching materials. His materials are publicly avaliable and provide great resources for learning and practicing R.

[4] Adapted from Kosuke Imai's QSS book, Chapter 1