Assignment 2: Coding Basics

Leo Zhang

OVERVIEW

This exercise accompanies the lessons/labs in Environmental Data Analytics on coding basics.

Directions

1. Rename this file `<FirstLast>_A02_CodingBasics.Rmd` (replacing `<FirstLast>` with your first and last name).
2. Change "Student Name" on line 3 (above) with your name.
3. Work through the steps, **creating code and output** that fulfill each instruction.
4. Be sure to **answer the questions** in this assignment document.
5. When you have completed the assignment, **Knit** the text and code into a single PDF file.
6. After Knitting, submit the completed exercise (PDF file) to Canvas.
7. Initial here to acknowledge that you did not use AI at all in completing this assignment: LZ

Basics, Part 1

1. Use R's `seq()` function to create a sequence of numbers from 100 to 333, increasing by threes. Assign this sequence a variable name.
2. Compute the *mean* of this sequence, assigning this values its own variable name.
3. Compute the *standard deviation* (`sd()`) of this sequence, assigning this values its own variable name.
4. Display the the mean minus the standard deviation as well as the mean plus the standard deviation.
5. Insert comments in your code to describe what you are doing.

```
#1. I use seq() function to create 'collection' variable. Expecting a complete sequence, I clarify that
collection <- seq(100, 334, 3)
collection
```

```
##  [1] 100 103 106 109 112 115 118 121 124 127 130 133 136 139 142 145 148 151 154
## [20] 157 160 163 166 169 172 175 178 181 184 187 190 193 196 199 202 205 208 211
## [39] 214 217 220 223 226 229 232 235 238 241 244 247 250 253 256 259 262 265 268
## [58] 271 274 277 280 283 286 289 292 295 298 301 304 307 310 313 316 319 322 325
## [77] 328 331 334
```

```r
#2. I use mean() function to create 'average' variable.
average <- mean(collection)
average
```

```
## [1] 217
```

```r
#3. I use sd() function to create 'deviate' variable.
deviate <- sd(collection)
deviate
```

```
## [1] 68.84766
```

```r
#4. Two variables, 'average' and 'deviate', undergo basic calculation.
average - deviate
```

```
## [1] 148.1523
```

```r
average + deviate
```

```
## [1] 285.8477
```

Basics, Part 2

6. Create three vectors, each with four components, consisting of (a) student names, (b) test scores, and (c) whether they are on scholarship or not (TRUE or FALSE).
7. Label each vector with a comment on what type of vector it is.
8. Combine each of the vectors into a data frame. Assign the data frame an informative name.
9. Label the columns of your data frame with informative titles.

```r
# The following vector consists of character (or string) values.
name <- c('Barbara', 'Phoebe', 'Victoria', 'Wendy')
class(name)
```

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

```r
# The following vector consists of numeric values.
score <- c(85, 96, 99, 92)
class(score)
```

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

```r
# The following vector consists of logical (or Boolean) values.
scholarship <- c(FALSE, FALSE, TRUE, TRUE)
class(scholarship)
```

```
## [1] "logical"
```

```
report <-data.frame(name, score, scholarship)
names(report) <- c('Student Name', 'Score Of Final Exam', 'Presence Of Scholarship')
report
```

```
##   Student Name Score Of Final Exam Presence Of Scholarship
## 1      Barbara                  85                   FALSE
## 2       Phoebe                  96                   FALSE
## 3     Victoria                  99                    TRUE
## 4        Wendy                  92                    TRUE
```

10. QUESTION: How is this data frame different from a matrix?

    Answer: A matrix can accommodate only one type of data. However, this restriction does not apply to a dataframe. For example, the 'report' dataframe contains character, numeric, and logical values.

Basics, Part 3

11. Create a function with one input. In this function, use `if...else` to evaluate the value of the input: if it is greater than 50, it returns the word "Pass"; otherwise it returns the word "Fail".
12. Create a second function that does the exact same thing as the previous one but uses `ifelse()` instead if `if...else`.
13. Run both functions using the value 54 as the input
14. Run both functions using the **vector** of student test scores you created as the input. (Only one will work properly...)

```
#11. Create a function using if...else
first_process <- function(x) {if(x > 50) {'Pass'} else {'Fail'}}


#12. Create a function using ifelse()
second_process <- function(x) {ifelse(x > 50, 'Pass', 'Fail')}


#13a. Run the first function with the value 54
first_explore <- first_process(54)
first_explore
```

```
## [1] "Pass"
```

```
#13b. Run the second function with the value 54
second_explore <- second_process(54)
second_explore
```

```
## [1] "Pass"
```

```r
#14a. Run the first function with the vector of test scores
#first_application <- first_process(score)
#first_application


#14b. Run the second function with the vector of test scores
second_application <- second_process(score)
second_application
```

```
## [1] "Pass" "Pass" "Pass" "Pass"
```

15. QUESTION: Which option of `if...else` vs. `ifelse` worked? Why? (Hint: search the web for "R vectorization"; it's ok here if an AI response is presented in the search response.)

    Answer: As a vectorized function, 'ifelse' works. In contrast, instead of all values in a vector, 'if ... else ...' can only handle one value.

NOTE Before knitting, you'll need to comment out the call to the function in Q14 that does not work. (A document can't knit if the code it contains causes an error!)