Intro to Data Science - Lab 2

Week 2 – Sorting Data and Ordering a Data Frame

```
# Enter your name here: Hendi Kushta
```

Please include nice comments.

Attribution statement: (choose only one and delete the rest)

```
# 1. I did this lab assignment by myself, with help from the book and the professor.
```

1. Make a copy of the built-in iris data set like this:
   `myIris <- iris`

```
myIris <- iris
```

2. Get an explanation of the contents of the data set with the **help()** function:
   `help("iris")`

```
help("iris")
```

3. Explore **myIris** via the **str()** and **glimpse()** functions (note: you need to install and library **'tidyverse'** to use **glimpse()**). Which do you think is better? Why? Explain in a comment.

```
str(myIris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1
```

```
install.packages("tidyverse")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
library(tidyverse)
```

```
Warning message in system("timedatectl", intern = TRUE):
“running command 'timedatectl' had status 1”
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
✔ ggplot2 3.3.6     ✔ purrr   0.3.4
✔ tibble  3.1.7     ✔ dplyr   1.0.9
✔ tidyr   1.2.0     ✔ stringr 1.4.1
✔ readr   2.1.2     ✔ forcats 0.5.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

```
glimpse(myIris)
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.…
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.…
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.…
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.…
$ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s…
```

Double-click (or enter) to edit

glimpse() and str() are basically interchangeable. str() is base R while glimpse() we need to install tidyverse package. A difference might be that str() classifies string columns in factors with levels.

4. Summarize the variables in your copy of the data set, like this:
   `summary(myIris)`

```
summary(myIris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
Median :5.800   Median :3.000   Median :4.350   Median :1.300
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
setosa    :50
```

5. The **summary()** command provided the mean of each numeric variable. Choose the variable with the highest mean and list its contents. Any variable can be echoed to the console simply by typing its name. Here's an example that echoes the variable with the lowest mean:
   `myIris$Petal.Width`

```
myIris$Petal.Width
```

```
0.2 · 0.2 · 0.2 · 0.2 · 0.2 · 0.4 · 0.3 · 0.2 · 0.2 · 0.1 · 0.2 · 0.2 · 0.1 · 0.1 · 0.2 · 0.4 · 0.4 · 0.3 · 0.3 ·
0.3 · 0.2 · 0.4 · 0.2 · 0.5 · 0.2 · 0.2 · 0.4 · 0.2 · 0.2 · 0.2 · 0.2 · 0.4 · 0.1 · 0.2 · 0.2 · 0.2 · 0.2 · 0.1 ·
0.2 · 0.2 · 0.3 · 0.3 · 0.2 · 0.6 · 0.4 · 0.3 · 0.2 · 0.2 · 0.2 · 0.2 · 1.4 · 1.5 · 1.5 · 1.3 · 1.5 · 1.3 · 1.6 · 1 ·
1.3 · 1.4 · 1 · 1.5 · 1 · 1.4 · 1.3 · 1.4 · 1.5 · 1 · 1.5 · 1.1 · 1.8 · 1.3 · 1.5 · 1.2 · 1.3 · 1.4 · 1.4 · 1.7 · 1.5 ·
```

6. Now sort that attribute by calling the **sort()** function and supplying that variable. Remember to choose the variable with the highest mean.

```
sort(myIris$Sepal.Length)
```

```
4.3 · 4.4 · 4.4 · 4.4 · 4.5 · 4.6 · 4.6 · 4.6 · 4.6 · 4.7 · 4.7 · 4.8 · 4.8 · 4.8 · 4.8 · 4.8 · 4.9 · 4.9 · 4.9 ·
4.9 · 4.9 · 4.9 · 5 · 5 · 5 · 5 · 5 · 5 · 5 · 5 · 5 · 5 · 5.1 · 5.1 · 5.1 · 5.1 · 5.1 · 5.1 · 5.1 · 5.1 · 5.1 · 5.2 ·
5.2 · 5.2 · 5.2 · 5.3 · 5.4 · 5.4 · 5.4 · 5.4 · 5.4 · 5.4 · 5.5 · 5.5 · 5.5 · 5.5 · 5.5 · 5.5 · 5.5 · 5.6 · 5.6 ·
5.6 · 5.6 · 5.6 · 5.6 · 5.7 · 5.7 · 5.7 · 5.7 · 5.7 · 5.7 · 5.7 · 5.7 · 5.8 · 5.8 · 5.8 · 5.8 · 5.8 · 5.8 · 5.8 ·
```

7. Now repeat the previous command, but this time use the **order()** function, again using the variable with the highest mean.

```
order(myIris$Sepal.Length)
```

```
14 · 9 · 39 · 43 · 42 · 4 · 7 · 23 · 48 · 3 · 30 · 12 · 13 · 25 · 31 · 46 · 2 · 10 · 35 · 38 · 58 · 107 · 5 · 8 ·
26 · 27 · 36 · 41 · 44 · 50 · 61 · 94 · 1 · 18 · 20 · 22 · 24 · 40 · 45 · 47 · 99 · 28 · 29 · 33 · 60 · 49 · 6 ·
11 · 17 · 21 · 32 · 85 · 34 · 37 · 54 · 81 · 82 · 90 · 91 · 65 · 67 · 70 · 89 · 95 · 122 · 16 · 19 · 56 · 80 ·
96 · 97 · 100 · 114 · 15 · 68 · 83 · 93 · 102 · 115 · 143 · 62 · 71 · 150 · 63 · 79 · 84 · 86 · 120 · 139 ·
```

8. Write a comment in your R code explaining the difference between **sort()** and **order()**. Be prepared to explain this difference to the class.

```
# sort() will sort a vector in ascending order
```

```
# order() will return the index of each element in a vector in sorted order
```

9. Now use the **order()** command to reorder the whole data frame, and store the new dataframe in a variable called **'sortedDF'**:

```
sortedDF <- myIris[order(myIris$Sepal.Length),]
```

10. Now sort the dataframe using **arrange()**, which is part of the **tidyverse** package. This time, sort based the attribute with the lowest mean. Store the new dataframe in a variable called **'sortedDF1'**

```
sortedDF1 <- arrange(myIris, Petal.Width)
```

11. Finally, use **head()** to examine your reordered data frames and be prepared to report on the first few rows

```
head(sortedDF, 5)
```

A data.frame: 5 × 5

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
14	4.3	3.0	1.1	0.1	setosa
9	4.4	2.9	1.4	0.2	setosa
39	4.4	3.0	1.3	0.2	setosa
43	4.4	3.2	1.3	0.2	setosa
42	4.5	2.3	1.3	0.3	setosa

12. What does the following line of code do?

```
myIris [ , 1]
```

```
myIris[ , 1]
# Gets all the rows for the first column.
```

5.1 · 4.9 · 4.7 · 4.6 · 5 · 5.4 · 4.6 · 5 · 4.4 · 4.9 · 5.4 · 4.8 · 4.8 · 4.3 · 5.8 · 5.7 · 5.4 · 5.1 · 5.7 · 5.1 · 5.4 · 5.1 · 4.6 · 5.1 · 4.8 · 5 · 5 · 5.2 · 5.2 · 4.7 · 4.8 · 5.4 · 5.2 · 5.5 · 4.9 · 5 · 5.5 · 4.9 · 4.4 · 5.1 · 5 · 4.5 · 4.4 · 5 · 5.1 · 4.8 · 5.1 · 4.6 · 5.3 · 5 · 7 · 6.4 · 6.9 · 5.5 · 6.5 · 5.7 · 6.3 · 4.9 · 6.6 · 5.2 · 5 · 5.9 · 6 · 6.1 · 5.6 · 6.7 · 5.6 · 5.8 · 6.2 · 5.6 · 5.9 · 6.1 · 6.3 · 6.1 · 6.4 · 6.6 · 6.8 · 6.7 · 6 · 5.7 · 5.5 · 5.5 ·

13. What is the difference (if any) between:

    `myIris [ , "Sepal.Length"]` and

    `myIris $Sepal.Length`

```
# There are no differences. These are 2 different ways on how to print or select
# all rows for the column named Sepal.Length.
myIris[,"Sepal.Length"]
```

```
 5.1 · 4.9 · 4.7 · 4.6 · 5 · 5.4 · 4.6 · 5 · 4.4 · 4.9 · 5.4 · 4.8 · 4.8 · 4.3 · 5.8 · 5.7 · 5.4 · 5.1 · 5.7 · 5.1 ·
 5.4 · 5.1 · 4.6 · 5.1 · 4.8 · 5 · 5 · 5.2 · 5.2 · 4.7 · 4.8 · 5.4 · 5.2 · 5.5 · 4.9 · 5 · 5.5 · 4.9 · 4.4 · 5.1 · 5 ·
 4.5 · 4.4 · 5 · 5.1 · 4.8 · 5.1 · 4.6 · 5.3 · 5 · 7 · 6.4 · 6.9 · 5.5 · 6.5 · 5.7 · 6.3 · 4.9 · 6.6 · 5.2 · 5 · 5.9 ·
 6 · 6.1 · 5.6 · 6.7 · 5.6 · 5.8 · 6.2 · 5.6 · 5.9 · 6.1 · 6.3 · 6.1 · 6.4 · 6.6 · 6.8 · 6.7 · 6 · 5.7 · 5.5 · 5.5 ·
```

```
myIris $Sepal.Length
```

```
 5.1 · 4.9 · 4.7 · 4.6 · 5 · 5.4 · 4.6 · 5 · 4.4 · 4.9 · 5.4 · 4.8 · 4.8 · 4.3 · 5.8 · 5.7 · 5.4 · 5.1 · 5.7 · 5.1 ·
 5.4 · 5.1 · 4.6 · 5.1 · 4.8 · 5 · 5 · 5.2 · 5.2 · 4.7 · 4.8 · 5.4 · 5.2 · 5.5 · 4.9 · 5 · 5.5 · 4.9 · 4.4 · 5.1 · 5 ·
 4.5 · 4.4 · 5 · 5.1 · 4.8 · 5.1 · 4.6 · 5.3 · 5 · 7 · 6.4 · 6.9 · 5.5 · 6.5 · 5.7 · 6.3 · 4.9 · 6.6 · 5.2 · 5 · 5.9 ·
 6 · 6.1 · 5.6 · 6.7 · 5.6 · 5.8 · 6.2 · 5.6 · 5.9 · 6.1 · 6.3 · 6.1 · 6.4 · 6.6 · 6.8 · 6.7 · 6 · 5.7 · 5.5 · 5.5 ·
```

14. Write the R code that outputs the **'Sepal.Length'** attribute values, using the **select()** command.

```
select(myIris, Sepal.Length)
```

A data.frame: 150 × 1

Sepal.Length
<dbl>
5.1
4.9
4.7
4.6
5.0
5.4
4.6
5.0
4.4
4.9
5.4
4.8
4.8
4.3
5.8
5.7
5.4
5.1
5.7
5.1
5.4
5.1
4.6
5.1
4.8
5.0
5.0
5.2

5.2

4.7

⋮

6.9

5.6

7.7

6.3

6.7

7.2

6.2

6.1

6.4

7.2

7.4

7.9

6.4

6.3

6.1

7.7

15. Create a new column (called **'Ave.Length'**) in **myIris**, which, for each row, is the average of **Sepal.Length** and **Petal.Length**.

```
Ave.Length <- c((myIris$Sepal.Length + myIris$Petal.Length)/2)
```

6.7

```
myIris <- data.frame(myIris, Ave.Length)
```

5.8

16. What does the following line of code do:

```
which.min(myIris$Petal.Length)
```

6.7

Double-click (or enter) to edit

```
# returns the position or the index of the value where is the minimum
# petal length
```

5.9

17. Using the code from the previous step, output the row (iris observation) with the smallest petal length.

```
which.min(myIris$Petal.Length)
```

```
23
```

18. Create a new dataframe, with just the **Petal.Length** and **Petal.Width** attributes

```
df <- data.frame(myIris$Petal.Length, myIris$Petal.Width)
```

19. Create a new dataframe, using the **slice()** function, with only the first three rows in the **myIris** dataframe.

```
df1 <- slice(myIris, 1:3)
```

20. Create a new dataframe, which is a subset of **myIris**, that only includes rows where **Petal.Length is less than 1.4**, store in **shortPetalDF**

```
shortPedalDF <- myIris[c(myIris$Petal.Length < 1.4),]
```

21. How many rows are in the **shortPetalDF**?

```
# 11 rows
nrow(shortPedalDF)
```

```
11
```

22. The homework asks you to create a conditional statement with **if** and **else**. A conditional statement is part of a larger group of specialized commands that control the “flow” of a program – what command gets run and when. You can get help on if, else, and other control words. Add and run these commands:

```
help("if")
help("Control")
```

Now add and run your first conditional statement:

```
help("if")
```

```
help("Control")
```

```
myNumbers <- c(12,11,23,3,6)
```

```
if (sum(myNumbers) > 40) print("The sum is greater than 40.")
```

```
     [1] "The sum is greater than 40."
```