Assignment 3: R Basics

01/25/2022

Please answer each of these questions to the best of your abilities. Do not use loops in this assignment (points will be deducted for loop usage). If you need help, either search for help online and/or ask David and Nicole! See the Slack for office hours. Also, it might be helpful do work on the other problems if you get stuck – they will have some helpful tips. Finally, get in the habit of committing to GitHub often – doing it 2-3 exercises might be a good pace for this homework.

Here's some common mistakes you might make (from personal experience):

- Forgetting to put quotes around something that should be a string (and vice-versa).
- Missing an opening/closing parentheses. Your code will run, but you'll see a `+` in the console that suggest it's waiting for more input.
- Calling a function on a vector of the wrong type. Also, code might break if you have `NA` in your data.
- Overwriting your original data and wondering why downstream code doesn't work (this is why we suggest to make copies of your data before modifying!)
- General typos (misspellings/capitalization, misplaced punctuation, function arguments in the wrong order, etc.)

To submit, write an **R script** file called `homework3_answers.R`, formatted as follows:

```
# exercise 1.1
<code here>

# exercise 1.2
<code here>

...
# exercise 2.1
<code here>

...
```

Please put this file in your `week3_homework` folder along with the original PDF assignment file. Submit your commit to the homework Form by **8:00 am, Wednesday, February 2**. Remember to write an informative commit message (e.g. `HOMEWORK: week3 done`).

(Grading): You will get full credit if you demonstrate effort and fundamental understanding on all sections of the problems. Bonus questions will not raise your grade above a 10/10.

1 Basic R Familiarity

Exercise 1.1 (Cleaning NAs)**.** R comes with many different data sets. Looking at the `attenu` data set (loaded locally in your R):

1. Identify the rows in which there was no reported station in the `station` column. (Hint: use `is.na()`).
2. Create a copy of `attenu` called `attenu_cleaned`, where rows that are missing station information are not included.
3. Print the first 6 rows of `attenu_cleaned` (using `head()`) and its dimensions (using `dim()`).

Exercise 1.2 (Simple ifelse() to add columns)**.** Let's look at the `Theoph` data set now, which measures the dosing of a drug.

1. Make a copy of the `Theoph` data set called `Theoph_2` (again, the data set should be included in R).
2. Identify what the median treatment dose is. (Use `str()` to look see which column has dose information).
3. Then, using `ifelse()`, add a new column called `Dose_Class` to `Theoph_2`, where the dose is classified as `high` if it is **above or equal to** the median dose, and `'low` otherwise. (Use the `$` to add the column).
4. As before, print the first 6 rows and its dimensions.

Exercise 1.3 (Starbucks nutrition data)**.** Let's look at some more nutritional data in the form of Starbucks drinks.

1. Read in the Starbucks nutrition data using `read.csv()` and store it in a variable named `starbucks`. The file is called `starbucks.csv`. As with using the command line, make sure that you are in the correct directory by using `getwd()` and `setwd()`!
2. Now, let's clean up our data by filtering out the drinks without any data. Follow these steps:
   a. Determine which values contain `NA` using `is.na()`.
   b. Notice that we get a data frame of `TRUE` and `FALSE` data. How can we identify which rows are full of `NA` values? Create a boolean vector, `is_row_empty` to identify these rows. (Hint: remember that `TRUE` is 1 and `FALSE` is 0. What function sums across rows? How many `NA`s in a row do we need to say that there is no nutritional information for that drink?)
   c. Use `nrow()` or `dim()` to identify the number of rows in `starbucks`, and verify that `is_row_empty` is the same length as the number of rows. (Make sure `length(is_row_empty) == nrow(starbucks)`).
   d. With the boolean vector, create `starbucks_cleaned`, where the drinks without nutritional data are removed. (Hint: does the boolean vector select rows or columns? How do we select rows from a data frame?)
3. Now that we've cleaned everything up, let's examine the data some more. Create a plot of `Calories` (y-axis) vs. `Carb`. What do we notice about the plot? Make sure the rename the axes! (Note: carbohydrates are in grams; be sure to specify that in the name.)
4. Let's look at the drink with the highest amount of calories. Identify what is the highest number of calories by using the `max()` function. Then, use boolean indexing to identify which row has this drink with the highest number of calories (you'll get a 1 x 7 data frame). Find out the name of the drink by using either the square bracket `[]` or `$` syntax.
5. Let's highlight this point in the plot from part (4). Add a new column, `is_highest_fat`, which contains `TRUE` if the drink has the highest amount of fat, and `FALSE` otherwise. Then, reusing the code from (3), color the plot points by this column.
6. BONUS: now, color all points by a gradient based on their fat content. (Google how to do so! It's a lot easier in `ggplot`.)

Exercise 1.4 (Baseball data)**.** Let's look at some baseball data from here. The batting data is included in our GitHub folder (note: David wrote this question and he hates baseball). To analyze this data:

1. Read the data in using the `read.csv()` function.
2. Identify the number of players who scored three or more home runs in a given year. The number of homeruns in in the HR column. (note: players are duplicated, but for the purposes of this question just pretend that everyone is unique).
3. Plot the number of homeruns vs. year. (Hint: write a function using the `plot()` function to not repeat doing this analysis, as you'll see in the next steps).
4. Create a new data frame, containing players from the LA Angels. This information is in the `teamID` column, and the LA Angels code is "LAA". Then, repeat this analysis, but this time restricting your data to the LA Angels players.
5. Repeat step 4, except subset the original batting data to include players from either "ATL" or "PIT" (so, this step should be one data frame). Then, in the scatter plot, color the number of homeruns by the team.

Exercise 1.5 (Function Writing)**.** A common thing to want to do is to compare the data in two columns of a

data frame by plotting them and coloring them by some third level of data. Write a function `easy_plot(x, y, color_data)` that takes in three parameters: two numeric vectors of values for the x and y axis, and a third numeric vector to color the points by. The function does the following:

1. Identify the **median** data value of `color_data`. Then, using `ifelse()` create a new vector, `levels`, where values less than the median are `"low"` and values higher than the median are `"high"`. Then convert `color_levels` into factor by casting it as a factor as follows (you need to do this for coloring): `levels = factor(levels)`. Print this median.
2. Using `plot()` or `ggplot()`, plot `x` and `y` and color by `color_levels`. Change the points to filled-in circles using the option `pch=20`.
3. Finally, print the correlation between x and y (basically how related they are), by using the function `cor.test(x, y)`. You should see the correlation and a p-value. Note: you won't be returning anything in this function.
4. Test your function by passing the same `x` twice: to `x` and to `color_data`. How are the points colored?

Now, use `easy_plot()` to examine the cleaned Starbucks and batting data sets (pick any three variables to plot – make sure they're numeric). Do you see significant correlations between the variables you picked?

2 Extended Analysis: the Iris Data Set

For these questions, leave a comment in the script file with your answer; however, please leave the code you ran to answer the questions to show your work.

Exercise 2.1. What does the data set describe? How many observations does it contain (observations means data points)? What features does the data set contain (features means the number of variables per the data point)?

Exercise 2.2. Which features are continuous variables? (Continuous variables are always numeric, but the reverse is not necessarily true, especially when an integer is used encode a category. For example, it's common for 1 to represent category 1, 2 for category 2, etc.) Which are categorical? (Factors are categorical variables, but categorical variables are not necessarily labeled as factors such in the data set). Can you tell me the `R` data type for each column?

Exercise 2.3. Examine how each of the continuous variables look, making a histogram for each of them. Do you notice anything interesting?

BONUS: If you're feeling adventurous, add a density plot on top of the histogram.

Exercise 2.4. Let's say we want to split the data set into "narrow-sepaled" plants and "wide-sepaled" plants. To do so, let's want to add a new column to the iris data set containing this data and perform some more analysis. Follow these steps:

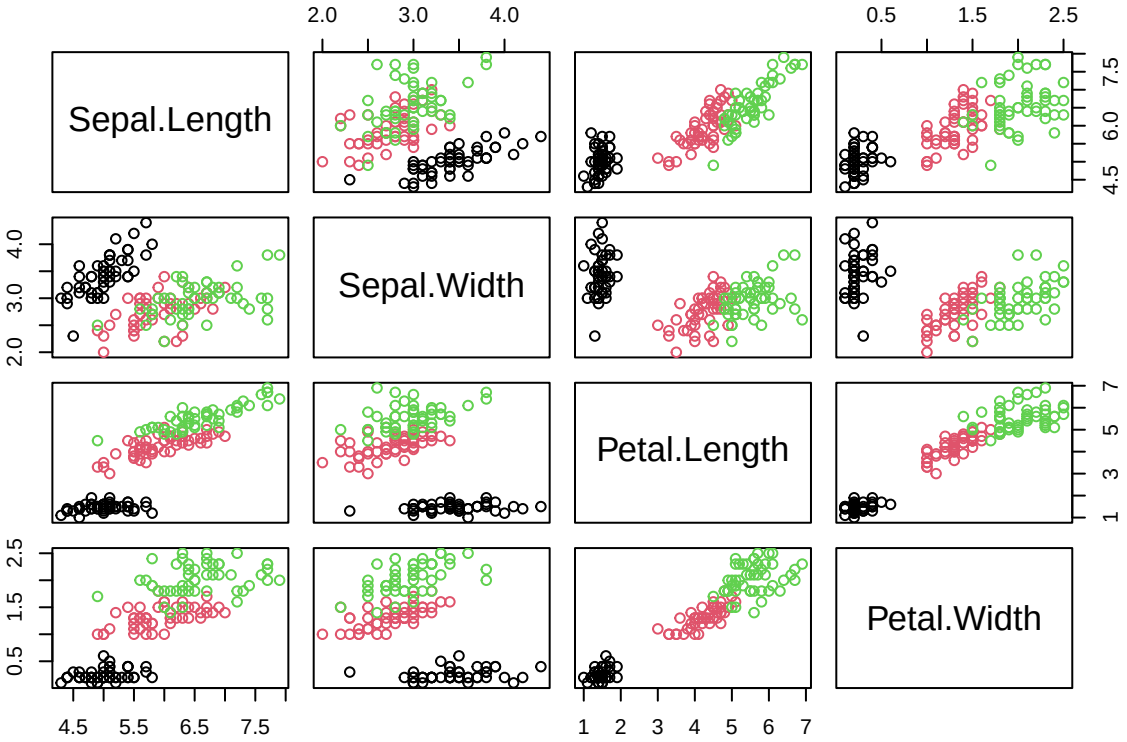
1. Find the mean sepal width in the data frame, and store it in a variable. Also, make a copy of `iris` called `iris_copy`.
2. Create a new vector using the `ifelse()` function to fill out values, comparing `Sepal.Width` to the mean value.
3. Create a new column in the `iris` data set (using the dollar sign) and assign the new vector to that column.
4. Create a boxplot plotting the `Sepal.Width` based on the new column (narrow vs. wide). To plot a boxplot by a variable, you need to use the syntax `y ~ x` instead of `x = x, y = y` as with `plot()`, where `x` and `y` are variables.

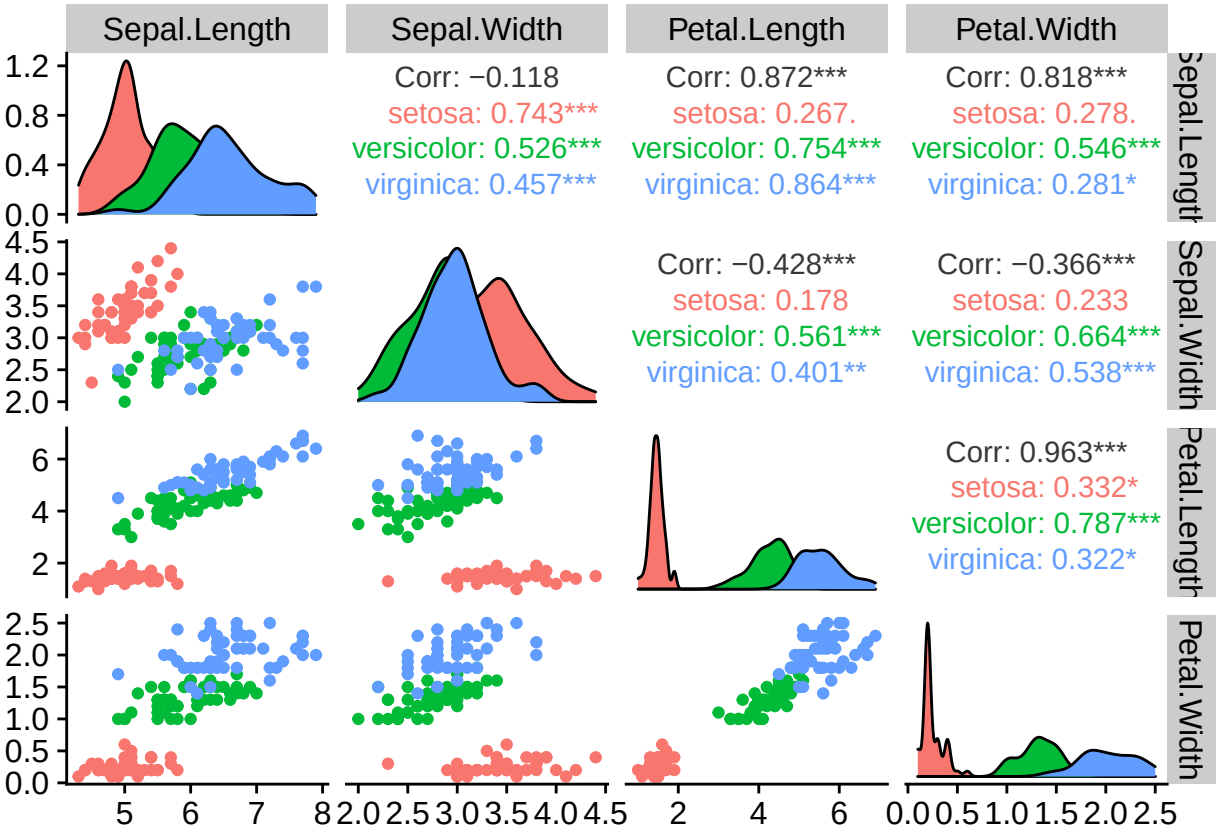
Exercise 2.5. Examine the following plots. The x and y axes of these plots are specified on the bottom and left, respectively Based on these plots, which species looks the most unique out of the three? Which species look the most similar?

Try to make something that looks somewhat similar to the first plot using the `pairs()` command. It works

very similarly to the `plot()` command. Run `?pairs` and `args(pairs)` to see what you need to pass to the function. (Hint: you'll have to remove a column to create a 4x4 plot).

BONUS: download and load the `GGally` package and use the `ggpairs()` function to recreate the second plot.





3 Prepare for Next Week

Exercise 3.1. We'll be using the `TCGAbiolinks` package next week. Install the package with `install.packages()`, then load it in with `library()`.