TA Session 7: Data Manipulation and Analysis (I) – Tidyverse

Harris Coding Camp

Summer 2024

General Guidelines

You may encounter some functions we did not cover in the lectures. This will give you some practice on how to use a new function for the first time. You can try following steps:

1. Start by typing `?new_function` in your Console to open up the help page.
2. Read the help page of this `new_function`. The description might be a bit technical for now. That's OK. Pay attention to the Usage and Arguments, especially the argument `x` or `x,y` (when two arguments are required).
3. At the bottom of the help page, there are a few examples. Run the first few lines to see how it works.
4. Apply it in your questions.

It is highly likely that you will encounter error messages while doing this exercise. Here are a few steps that might help get you through it:

1. Locate which line is causing this error first.
2. Check if you have a typo in the code. Sometimes your group members can spot a typo faster than you.
3. If you enter the code without any typo, try googling the error message. Scroll through the top few links see if any of them helps.
4. Try working on the next few questions while waiting for help by TAs.

Data and background

1. Load `tidyverse`, `haven`, and `readxl` in your Rmd/script.
2. We will use the Auto data set. Check out Auto.csv for more information.
3. Download the first data set here and put the data in your data folder and set your working directory properly. The data source is the World Inequality Database where you can find data about the distribution of income and wealth in several countries over time. Check out wid.world for more information.

Warm-up

In this part, we will revisit `msleep` and redo part of the questions from TA sessions 5 and 6.

```
glimpse(msleep)
```

If you enter `?msleep` in the console, you'll find some definitions of the columns. And learn that sleep times and weights were taken from V. M. Savage and G. B. West. A quantitative, theoretical framework for understanding mammalian sleep. Proceedings of the National Academy of Sciences, 104 (3):1051-1056, 2007.

Using `arrange()` to sort data

It takes data in the first position (so you can pipe data to it easily) and columns in the next position. The following code orders data by the animal's name in alphabetical order.

```
msleep %>%
  arrange(name)
```

1. Use `arrange` to sort the data by `sleep_total`. This shows use the mammals that sleep the fewest hours. Giraffe's must drink a lot of coffee.
2. In fact, we can sort the data by multiple columns. Compare the output of the following code. How does `arrange()` handle multiple columns?

```
msleep %>%
  arrange(name, vore)

msleep %>%
  arrange(vore, name)
```

Using `select()` to pick columns

We often want to subset our data to include certain columns. This is where `select()` comes in:

```
msleep %>%
  select(name, genus)
```

The code restricts our data to `name` and `genus`.

3. Write code to select the column `name` along with the three columns about sleep.

Using `filter()` to filter data by rows

`filter()` subsets rows in the data that matches criteria.

```
msleep %>%
  filter(conservation == "domesticated")
```

`filter()` relies on comparison operators such as `==` (equals), `!=` (not equal to), `>` (greater than), `>=` (greater than or equal to) and so forth.

4. Use `filter()` to restrict the data set to carnivores. (hint: `vore == "carni"`). You should find 19 carnivores.
5. Find the five mammals that are `awake` less than or equal to 6 hours per day.
6. You could combine the two filters above to find that the thick-tailed opossum is the carnivore that sleeps more than 3/4 of the day. In particular, run `msleep %>% filter(vore == "carni", awake <= 6)`. Can you find all non-carnivores that sleep less than 6 hours per day?

I. Manipulating Auto Data with `dplyr` (ISLR Chapter 2 Q9)

This exercise involves the Auto data set. `na.omit()` removes the missing values from the data and returns a new data frame.

1. Load the Auto.csv into a variable called `auto`:

```
# load the Auto.csv into a variable called auto using read_csv() -- fill in your code after this line

# remove all rows with missing values using na.omit()
auto <- na.omit(auto)
```

2. Extract the columns `mpg`, `horsepower`, `weight`, `acceleration` and `name` from `auto`.
3. Filter all observations with `weight` between 2000 and 5000, and keep variables specified in part 2 in your data frame.
4. Is there any relationship between `acceleration` and `weight` of a car? What might explain that pattern? Use the data from part 3, run the code below and comment on the result.

```
auto %>%
  ggplot(aes(x = weight, y = acceleration)) +
    geom_point(size = 1) +
    geom_smooth(method = 'lm', formula = y~x, se = F)
```

5. (Optional) From part 4, it seems that there is a **linear** relationship between `weight` and `acceleration`. Measure the strength of the linear relationship between the two variables, and justify your answer in part 4.
6. Run the linear regression with `acceleration` as y and `weight` as x. What is the intercept of the linear regression line? The slope? Try to interpret these two terms.
7. Finally, based on part 6, try to predict the acceleration of a car with `weight = 5500`. What could be the potential problem of doing such prediction? Hint: What is the min and max of `weight` in your data set?

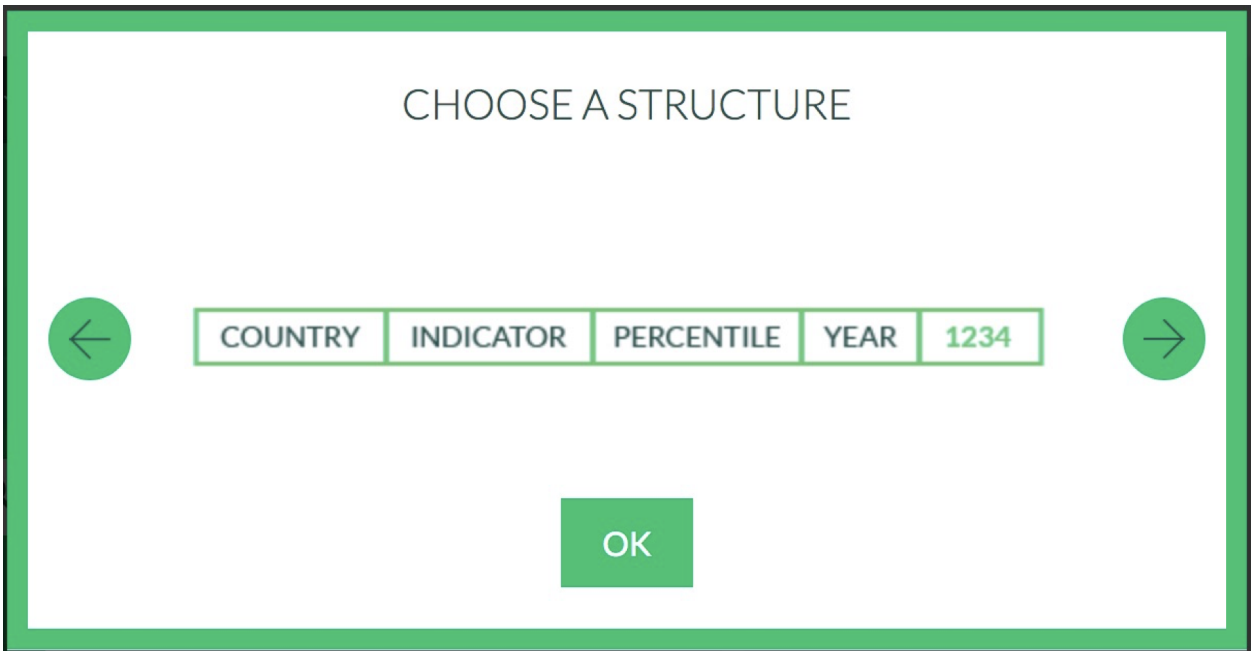
II. Manipulating World Inequality Data with `dplyr`

1. Load `wid_data` using the following code (don't forget to set your working directory appropriately):

```
wid_data <- read_xlsx("world_wealth_inequality.xlsx")
```

2. Look at the data. What is the main problem here?

3. We don't have columns headers. The World Inequality Database says the "structure" of the download is as shown in the image below.



So we can create our own header in `read_xlsx`. Calling the `read_xlsx` function using `readxl::read_xlsx()` ensures that we use the `read_xlsx()` function from the `readxl` package. Add our own header by running the code:

```
wid_data_raw <- readxl::read_xlsx("world_wealth_inequality.xlsx",
                                  col_names = c("country", "indicator",
                                                "percentile", "year", "value"))
```

4. Now let's look at the second column. It's a mess. We can separate it based on where the \n are and then deal with the data later. Don't worry about the syntax right now – just run the following code.

```
wid_data_raw <- readxl::read_xlsx("world_wealth_inequality.xlsx",
                                  col_names = c("country", "indicator",
                                                "percentile", "year",
                                                "value")) %>%
separate(indicator, sep = "\\n", into = c("row_tag", "type", "notes"))
```

Note: We want a clean reproducible script so you should just have one block of code reading the data: that last one. The other code were building blocks. If you want to keep "extra" code temporarily in your script, you can use `#` to comment out the code.

5. The data comes with some redundant columns that add clutter when we examine the data. What `dplyr` verb (e.g. `select()`, `filter()`, `mutate()`, `summarize()` and `arrange()`) lets you choose which columns to see? Remove the unwanted column `row_tag` and move `notes` to the last column position, and assign the output to the name `wid_data1`.

III. Continue on Final Project

As we said in Lecture 2, you're ready for policy school coding, if you can open a data set of interest to you and produce meaningful analysis.

1. Read through the final project description linked on Canvas.
2. What policy areas are you interested in? Are any of our suggested data sources aligned with your interests? Do you know of organizations that might collect data of interest to you? A good data set for this project will have enough observations (rows) that you need a computer to figure out what is going on! You should also be able to hypothesize about relationships between the variables (columns) in your data set.
3. Look for data! Google around etc. When you find something promising, try to load it into R.
4. Use functions like `glimpse`, `names`, `nrow`, `ncol`, `summary` etc (see the lecture notes for more) to start getting a sense of your data.
5. Generate graphs in Base R to visualize your data.

Think about what it will take to address the questions and hypotheses you made.

If you want this to be more of a challenge, we ask you to find two data sets that can be linked (e.g. both share a column like zip code); then you can use the skill we will learn in Module 8 to join data sets.