Fall Lab 1

10/4/2021

Warm up

1. How would you install the haven package? Do it now.

```
install.packages("haven")
```

2. In the videos, you learned about head(). What if you wanted to get the tail end of your data instead?

`tail()` returns the end of the data frame.

```
tail(cars)
```

```
##    speed dist
## 45    23   54
## 46    24   70
## 47    24   92
## 48    24   93
## 49    24  120
## 50    25   85
```

3. Recall our dplyr verbs. What is the purpose of each function?

`mutate()` : Adds or changes an existing column in the data frame

`filter()` : Removes rows/observations based on a column or columns

`select()` : Keeps or removes specified columns from the data frame

`arrange()` : Sorts the data frame by a column or column

`summarize()` : Collapse the data frame into aggregated information for specified columns (sum, mean, median, etc.)

and soon we'll add:

`group_by()` : Used with mutate or summarize, to collapse the data frame by smaller groups

4. Imagine you have a data set, df with 4 variables, county, year, income, and employment. You only need the year and employment status of people whose income is below $5000. Which two dplyr commands do you need to do this? Can you write the code for this?

```
df %>%
  filter(income < 5000) %>%
  select(year, employment)
```

5. Remember the mean() function from last time? What dplyr commands would we need if we want the average income? How many rows will the resulting dataset be?

```
df %>%
  summarise(mean = mean(income))
```

The dataframe is 1 row.

Working with data and scripts

We recommend a file structure for coding lab. If you have your own preferred way of organizing code feel free to follow it.

Setting up working directory and coding environment

1. Do you have a folder on your computer for coding lab material? If not, create one and make sure you know the path to the folder.
2. We recommend creating a problem_set folder inside your coding lab folder.
3. Make folder called data inside the problem_set folder.

Putting your files in place

4. Create a new R script. Save your script in the problem_set folder. From now on, when you start a script or Rmd save it there.
5. Download the first data set from bit.ly/fall_lab_1 and put the data in your data folder.

Tell R where to find files

Local paths are like addresses on your computer.

Use getwd() to see how your computer makes addresses.

6. Add a line to your script where you setwd() to your problem set folder.

```
setwd("C:/Users/johnt/Google Drive...")
```

Working with the files

7. Finally, we are using data in an excel format. We need the package readxl to process data of this type. In the console, run install.packages("readxl").

```
install.packages("readxl")
```

8. Add code to load the tidyverse.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(readxl)
```

9. If you did everything correctly you should be able to run the following code:

```r
fed_data <- read_xlsx("data/area_report_by_year.xlsx")
```

```
## New names:
## * `` -> ...2
## * `` -> ...3
```

The path is relative to your working directory. R looks for a data folder in your working directory and then for the data file in that folder. You could also give R an absolute file path, such as: "Users/John Smith/Coding Lab/problem_sets/data/area_report_by_year.xlsx".

However, note that this absolute path wouldn't work in someone else's computer, and also wouldn't work if John decides to move his Coding Lab files elsewhere, while the relative path will work just fine as long as the working directory is set.

Analyzing Student Debt

When you open fed_data you notice there are some issues! First, we will walk you through our code we wrote to clean our data. We're including it here so you can see what our data prep looks like, but we don't expect you to know all of the functions in here yet! Then, you will analyze the data.

Data Cleaning

```r
library(tidyverse)
library(readxl)
student_loan_debt <- read_xlsx("Data/area_report_by_year.xlsx", sheet = "studentloan", skip = 3) %>%
  filter(state != "allUS") %>%
  pivot_longer(cols = -state, names_to = "year", values_to = "per_capita_student_debt") %>%
  mutate(year = str_sub(year, 4, 7),
  year = as.numeric(year))

write_csv(student_loan_debt, "student_loan_debt.csv")
```

What's going on here?

```r
library(tidyverse)
library(readxl)
```

We load the packages that have the functions we need: tidyverse and readxl.

```r
student_loan_debt <- read_xlsx("Data/area_report_by_year.xlsx",
                               sheet = "studentloan", skip = 3)
```

1. We tell our read_xlsx function to go to the data folder where we have the data, then to "area_report_by_year.xlsx", so that it can find the data. We specify the sheet in the Excel workbook we want to read, and we skip the first 3 rows in the sheet, because the data we're interested in starts on line 4.

```r
filter(state != "allUS")
```

2. We filter out rows of data that are for the entire US, leaving only rows that refer to states.

```r
pivot_longer(cols = -state, names_to = "year", values_to = "per_capita_student_debt")
```

3. We convert the data from a wide to a long format, so that year is a variable and per_capita_student_debt is also a variable. (The reason we do this is so that it is easier for functions in the "tidyverse" to process this type of data for groupwise calculations, e.g. mean debt by year, etc. Read more about tidy data in R for Data Science.)

```r
mutate(year = str_sub(year, 4, 7),
       year = as.numeric(year))
```

4. We use string manipulation to modify the existing year column, and then we convert the type of the column.

What was the original type of the year column? What is the new type of the year column?

The type was originally string, and now it is a numeric.

```r
write_csv(student_loan_debt, "student_loan_debt.csv")
```

5. We write the cleaned data to a CSV (comma-separated variables file).

Try running this code locally on your computer! Copy the code to a new script, and save it to the same folder that you've stored your downloaded data in. Make sure to set your new folder as your "working directory" correctly.

Exploratory Data Analysis

Ok, now that we've gone over how the file was created, load the cleaned data in your own R session. If you had trouble with read_xls, we have the csv with the cleaned code here

```r
# library(readr)
# library(dplyr)
student_loan_debt <- read_csv("student_loan_debt.csv")
```

To look at your data after reading it in, you can use a tidyverse function called glimpse(). This is a nicer version of a function called str(). Try running both str() and glimpse() on student_loan_debt.

```r
student_loan_debt %>%
  str()
```

```
## tibble [832 x 3] (S3: tbl_df/tbl/data.frame)
##  $ state                  : chr [1:832] "AK" "AK" "AK" "AK" ...
##  $ year                   : num [1:832] 2003 2004 2005 2006 2007 ...
##  $ per_capita_student_debt: num [1:832] 680 1730 1910 2250 2340 2530 2850 3140 3390 3680 ...
```

```r
student_loan_debt %>%
  glimpse()
```

```
## Rows: 832
## Columns: 3
## $ state                   <chr> "AK", "AK", "AK", "AK", "AK", "AK", "AK", "AK"~
## $ year                    <dbl> 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010~
## $ per_capita_student_debt <dbl> 680, 1730, 1910, 2250, 2340, 2530, 2850, 3140,~
```

Note: student_loan_debt can be long to type, so use Tab-Autocomplete. Once you start typing the variable in the function, press Tab and wait for the variable name to automatically pop up. Press Enter to fill in student_loan_debt (or click on it).

Arranging Data

We can use the arrange() function from dplyr to sort the student loan data. The syntax is arrange(data, variable). Arrange the data in student_loan_debt by per_capita_student_debt. (Can you sort the other way?)

```r
student_loan_debt %>%
  arrange(per_capita_student_debt) %>%
  head()
```

```
## # A tibble: 6 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 PR     2003                     500
## 2 PR     2004                     650
## 3 WY     2003                     670
## 4 AK     2003                     680
## 5 AR     2003                     710
## 6 SC     2003                     710
```

If we want to sort in descending order:

```r
student_loan_debt %>%
  arrange(desc(per_capita_student_debt)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 DC     2018                   13320
## 2 DC     2017                   12380
## 3 DC     2016                   12200
## 4 DC     2015                   11780
## 5 DC     2014                   11260
## 6 DC     2013                   10880
```

Hint: Look up the arrange() documentation with ?arrange to figure out how to reverse the order of the sort. The examples at the bottom of the help screen are useful, and you can run them directly in R, if it helps!

Who had the lowest per capita debt in 2003? How much was the lowest per capita debt in 2003?

```r
student_loan_debt %>%
  filter(year == 2003)%>%
  arrange(per_capita_student_debt) %>%
  head(1)
```

```
## # A tibble: 1 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 PR     2003                     500
```

How much was the highest per capita debt in 2018?

```r
student_loan_debt %>%
  filter(year == 2018)%>%
  arrange(desc(per_capita_student_debt)) %>%
  head(1)
```

```
## # A tibble: 1 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 DC     2018                   13320
```

Filtering Data

To print the state with the lowest or highest per capita debt, you can subset with base syntax, which looks something like this:

```r
student_loan_debt[row_number, column_number]
student_loan_debt[row_condition, ]$column_name
```

Or you can subset with the filter function from the tidyverse, which is a bit easier to read. The pull function does the same thing as the $ sign, which pulls a column from a data frame.

```r
filter(student_loan_debt, row_condition) %>%
  pull(column_name)
```

Note: Another function you'll run into often that works similarly to pull() is select(). To put it simply, pull() returns the data from a column, while select() can pick more than one variable and returns a tibble or data frame with all of those columns. The above code returns a single column vector, column_name. If you would have used:

```r
filter(student_loan_debt, row_condition) %>%
  select(column_name)
```

You would have gotten a data frame that contains one column, column_name. These two things might appear to be the same at first glance, but over time you'll see they're not! Digression over.

What is a "row_condition" in this case? It's just something that we want to filter on, for example:

```r
filter(student_loan_debt, per_capita_student_debt < 800) %>%
  head()
```

```
## # A tibble: 6 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 AK     2003                     680
## 2 AR     2003                     710
## 3 HI     2003                     730
## 4 NC     2003                     780
## 5 NV     2003                     730
## 6 PR     2003                     500
```

Try writing a filter statement to get all states with an average per capita student debt of 5000 or higher in the year 2008. Yes, you can combine multiple criteria - just add a comma and another filtering criteria!

Hint: Your code should look like this: filter(data, condition1, condition2).

```r
student_loan_debt %>%
  filter(per_capita_student_debt > 5000, year == 2008)
```

```
## # A tibble: 1 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 DC     2008                    7420
```

Finally, filter is great for helping us figure out where the missing values are in our data.

```r
filter(student_loan_debt, is.na(per_capita_student_debt))
```

```
## # A tibble: 2 x 3
##   state  year per_capita_student_debt
##   <chr> <dbl>                   <dbl>
## 1 PR     2017                      NA
## 2 PR     2018                      NA
```

Who is missing data in 2017 and 2018?

Puerto Rico

Grouping and Summarizing Data

You might notice that our data is a little awkward to work with right now. We have state-year data instead of just yearly data. One thing that R is great at is helping us come up with groupwise averages, minima, maxima, and more!

For example, here is code to take student_loan_debt, group it by year, and then find the maximum per capita debt by year.

```
max_debt_by_year <- student_loan_debt %>%
  group_by(year) %>%
  summarize(max_debt = max(per_capita_student_debt))

max_debt_by_year
```

```
## # A tibble: 16 x 2
##     year max_debt
##    <dbl>    <dbl>
##  1  2003     3120
##  2  2004     4350
##  3  2005     4560
##  4  2006     5900
##  5  2007     6430
##  6  2008     7420
##  7  2009     7920
##  8  2010     8700
##  9  2011     9640
## 10  2012    10670
## 11  2013    10880
## 12  2014    11260
## 13  2015    11780
## 14  2016    12200
## 15  2017       NA
## 16  2018       NA
```

Try calculating the minimum per capita debt by year. Assign it to a new dataframe called min_debt_by_year instead of max_debt_by year.

```
min_debt_by_year <- student_loan_debt %>%
  group_by(year) %>%
  summarize(min_debt = min(per_capita_student_debt))

min_debt_by_year
```

```
## # A tibble: 16 x 2
##     year min_debt
##    <dbl>    <dbl>
##  1  2003      500
##  2  2004      650
##  3  2005      720
##  4  2006      870
##  5  2007      990
##  6  2008     1070
```

```
## 7  2009     1250
## 8  2010     1420
## 9  2011     1510
## 10 2012     1770
## 11 2013     1950
## 12 2014     2100
## 13 2015     2320
## 14 2016     2620
## 15 2017       NA
## 16 2018       NA
```

How about the mean per capita debt by year? Let's call this data frame student_loan_debt_by_year, and the variable per_capita_student_debt. Write this one from scratch!

```r
student_loan_debt_by_year <- student_loan_debt %>%
  group_by(year) %>%
  summarize(per_capita_student_debt = mean(per_capita_student_debt))

student_loan_debt_by_year
```

```
## # A tibble: 16 x 2
##     year per_capita_student_debt
##    <dbl>                   <dbl>
##  1  2003                   1123.
##  2  2004                   1509.
##  3  2005                   1673.
##  4  2006                   2043.
##  5  2007                   2322.
##  6  2008                   2752.
##  7  2009                   3075.
##  8  2010                   3422.
##  9  2011                   3676.
## 10  2012                   4070.
## 11  2013                   4289.
## 12  2014                   4520.
## 13  2015                   4703.
## 14  2016                   4939.
## 15  2017                     NA
## 16  2018                     NA
```

Like with filter, you can have multiple summarize() statements separated by a comma. Combine your work from the three examples into a single block of code that returns a data frame with the min, mean and max debt levels for the US.

```r
student_loan_debt_by_year <- student_loan_debt %>%
  group_by(year) %>%
  summarize(min_debt = min(per_capita_student_debt),
            max_debt = max(per_capita_student_debt),
            per_capita_student_debt = mean(per_capita_student_debt))

student_loan_debt_by_year %>%
  head(3)
```

```
## # A tibble: 3 x 4
##    year min_debt max_debt per_capita_student_debt
##   <dbl>    <dbl>    <dbl>                   <dbl>
## 1  2003      500     3120                   1123.
## 2  2004      650     4350                   1509.
## 3  2005      720     4560                   1673.
```

What is the minimum and mean per capita debt in 2011?

Min: $1,510 Mean: $3675.96

```
student_loan_debt_by_year %>%
  filter(year == 2011)
```

```
## # A tibble: 1 x 4
##    year min_debt max_debt per_capita_student_debt
##   <dbl>    <dbl>    <dbl>                   <dbl>
## 1  2011     1510     9640                   3676.
```

Dealing with Missing Data

Notice anything strange about the years 2017 and 2018? The values were NA for everything, even though we had data for most states. This is because NAs are "sticky", which means taking the mean of a vector with NAs makes the output NA. You can get around this with the na.rm = argument in min(), max(), and mean(). Try adding it to the mean() function.

```
mean(c(NA, 1, 2, 3))
```

```
## [1] NA
```

```
mean(c(NA, 1, 2, 3),na.rm=TRUE)
```

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

Hint: Your mean function inside of summarize should look like this: mean(variable, na.rm = TRUE).

What is the mean per capita debt in 2018, excluding Puerto Rico (PR)?

```
student_loan_debt %>%
  filter(year == 2018) %>%
  summarise(per_capita_student_debt = mean(per_capita_student_debt,na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   per_capita_student_debt
##                     <dbl>
## 1                   5438.
```

```
student_loan_debt %>%
  filter(year == 2018, state != "PR") %>%
  summarise(per_capita_student_debt = mean(per_capita_student_debt))
```

```
## # A tibble: 1 x 1
##   per_capita_student_debt
##                     <dbl>
## 1                   5438.
```

Joining and Plotting Data

Our analysis is close now! One thing to note: we took an average of averages so our per capita estimate may be wrong. We saw that DC had very high debt levels. However, it has a small population compared to the states.

To tackle this, we'll use a population dataset from the same spreadsheet. We clean it with the following code, which is similar to what we did before. Here's the code used to clean the data:

```r
# library(readxl)
# library(dplyr)
population <- read_xlsx("Data/area_report_by_year.xlsx", sheet = "population", skip = 3) %>%
  filter(state != "allUS") %>%
  pivot_longer(cols = -state, names_to = "year", values_to = "population") %>%
  mutate(year = str_sub(year, 4, 7),
         year = as.numeric(year))

write_csv(population, "population.csv")
```

The cleaned dataset is population.csv. Let's load it:

```r
population <- read_csv("population.csv")
```

```
## Parsed with column specification:
## cols(
##   state = col_character(),
##   year = col_double(),
##   population = col_double()
## )
```

Let's join the population data to our debt data and then weight the data by population.

```r
joined_data <- student_loan_debt %>%
  left_join(population, by = c("state", "year"))
```

We essentially use state and year as ways to link the two dataframes to each other. This is a common functionality in databases (and in SQL), but we can do the same in R!

To reweight, follow the following steps:

1. Use mutate() to calculate the total student debt in a state. (pop x debt/person = total debt)

```r
student_loan_debt_by_year_weighted <- joined_data %>%
  mutate(total_debt = population*per_capita_student_debt)

student_loan_debt_by_year_weighted %>% head()
```

```
## # A tibble: 6 x 5
##   state  year per_capita_student_debt population total_debt
##   <chr> <dbl>                   <dbl>      <dbl>      <dbl>
## 1 AK     2003                     680     478640  325475200
## 2 AK     2004                    1730     492740  852440200
```

```
## 3 AK     2005                        1910     497340  949919400
## 4 AK     2006                        2250     502840 1131390000
## 5 AK     2007                        2340     499840 1169625600
## 6 AK     2008                        2530     497080 1257612400
```

2. Use group_by() and summarize() to calculate the total debt in the US each year and the population of the US each year. Be wary of NAs.

```
student_loan_debt_by_year_weighted <- joined_data %>%
  mutate(total_debt = population*per_capita_student_debt) %>%
  group_by(year) %>%
  summarize(total_US_debt = sum(total_debt, na.rm=TRUE),
            total_US_pop = sum(population, na.rm=TRUE))


student_loan_debt_by_year_weighted %>% head()
```

```
## # A tibble: 6 x 3
##    year total_US_debt total_US_pop
##   <dbl>         <dbl>        <dbl>
## 1  2003  252660256200    238199960
## 2  2004  345229079200    239316060
## 3  2005  391869012200    242843440
## 4  2006  481314064200    244334020
## 5  2007  546355199200    242957640
## 6  2008  638787930800    239409820
```

3. Use mutate() to calculate the per capita student debt.

```
student_loan_debt_by_year_weighted <- joined_data %>%
  mutate(total_debt = population*per_capita_student_debt) %>%
  group_by(year) %>%
  summarize(total_US_debt = sum(total_debt, na.rm=TRUE),
            total_US_pop = sum(population, na.rm=TRUE)) %>%
  mutate(per_capita_student_debt = total_US_debt/total_US_pop)

student_loan_debt_by_year_weighted %>% head()
```

```
## # A tibble: 6 x 4
##    year total_US_debt total_US_pop per_capita_student_debt
##   <dbl>         <dbl>        <dbl>                   <dbl>
## 1  2003  252660256200    238199960                   1061.
## 2  2004  345229079200    239316060                   1443.
## 3  2005  391869012200    242843440                   1614.
## 4  2006  481314064200    244334020                   1970.
## 5  2007  546355199200    242957640                   2249.
## 6  2008  638787930800    239409820                   2668.
```

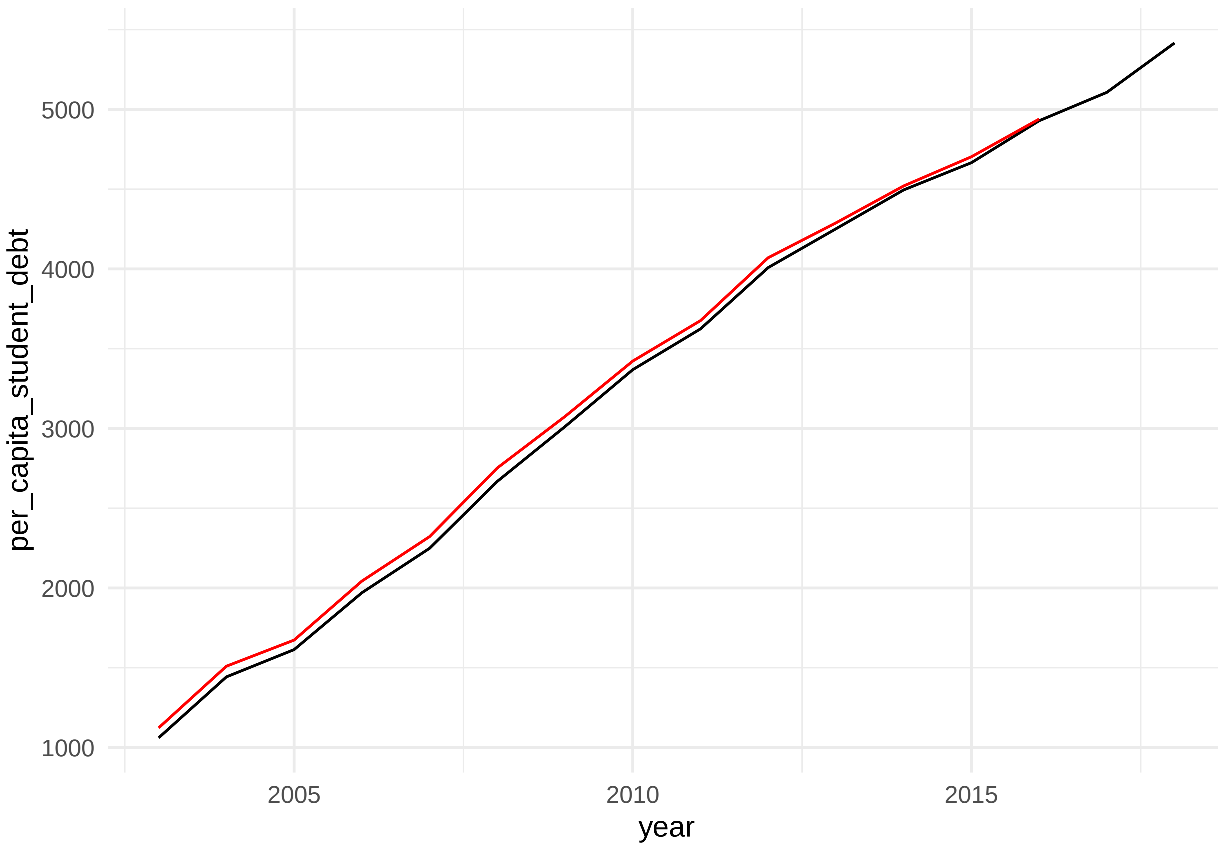
Plotting our Estimates

One of the nicest things to do in R is custom visualization. One package that is especially good for plotting and graphs is called ggplot2.

Use the following ggplot2 code to compare your original unweighted estimates in student_loan_debt_by_year to the weighted estimates in student_loan_debt_by_year_weighted. The unweighted estimates will be in red.

```
library(ggplot2)
student_loan_debt_by_year_weighted %>%
  ggplot(aes(x = year, y = per_capita_student_debt)) +
  geom_line() +
  theme_minimal() +
  geom_line(data = student_loan_debt_by_year, color = "red")
```

```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



You will see that the red line shows that the unweighted estimates of per capita debt are biased upward.Weights move the whole line downward, but by an economically small amount. Our estimates are lower than you might expect, because they include the full population. To understand how debt effects individual borrowers, we might could get additional data on the number of borrowers say population_of_borrowers and calculate the estimates as:

```
mutate(per_borrower_student_debt = total_debt/population_of_borrowers)
```

If you're interested in learning more about ggplot2, R for Data Science has a great chapter on the package and we have a "bonus" lesson on the course website.