Lab Session 1: Read and manipulate data

Solutions

9/8/2020

Working with data and scripts (To be covered during Q&A session)

In this section you will have set-up a directory structure for coding lab and started a script that reads in your data.

```
library(tidyverse)
library(readxl)

# you should set working directory here to your data folder
# as instructed.
setwd(...your_dir...)

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

Warm-up

1. Create a new R script and add code to load the `tidyverse`.
2. Your stats 1 partner comes to you and says they can't get data to load after restarting R. You see the code:

```
install.packages("haven")
awesome_data <- read_dta("awesome_data.dta")

Error in read_dta("awesome_data.dta") :
could not find function "read_dta"
```

   a. Diagnose the problem.

   Note: If they say the code worked before, it's likely they had loaded `haven` in the console or perhaps in an earlier script. R packages will stay attached as long as the R session is live.

Solution:

Could not find function "read_dta" since the "read_dta" function is from the 'haven' package but we have yet to load the 'haven' package.

3. In general, once you have successfully used `install.packages(pkg)` for a "pkg", you won't need to do it again. Install `haven` and `readxl` using the console.
4. In your script, load `haven`. Notice that if you had to restart R right now. You could reproduce the entire warm-up by running the script. We strive for reproducibility by keeping the code we want organized in scripts or `Rmd`s.

5. It's good practice when starting a new project to clear your R environment. This helps you make sure you are not relying on data or functions you wrote in another project. After you `library()` statements add the following code `rm(list = ls())`.

Solution: Now your script looks like:

```
library(tidyverse)
library(readxl)
library(haven)
rm(list = ls())

# you should set working directory here to your data folder
# as instructed.
setwd(...your_dir...)

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

6. `rm()` is short for remove. Find the examples in `?rm` and run them in the console.

```
#run

tmp <- 1:4
## work with tmp  and cleanup
rm(tmp)

## Not run:
## remove (almost) everything in the working environment.
## You will get no warning, so don't do this unless you are really sure.
rm(list = ls())
```

Examining 'wid_data

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

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

Now when we look at the second column. It's a mess. We can seperate it based on where the `\n` are and then deal with the data later. Don't worry about this code right now.

```
wid_data_raw <-
  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.

Manipulating world inequality data with `dplyr` (20 - 25 minutes)

Now we have some data and are ready to use `select()`, `filter()`, `mutate()`, `summarize()` and `arrange()` to explore it.

1. The data comes with some redundant columns that add clutter when we examine the data. What `dplyr` verb let's you choose what 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_data`.[1]

Solutions:

The relevant 'dplyr' verb is the select function

```
wid_data <- wid_data_raw %>%
  select(-notes, everything()) %>%
  select(-row_tag)
```

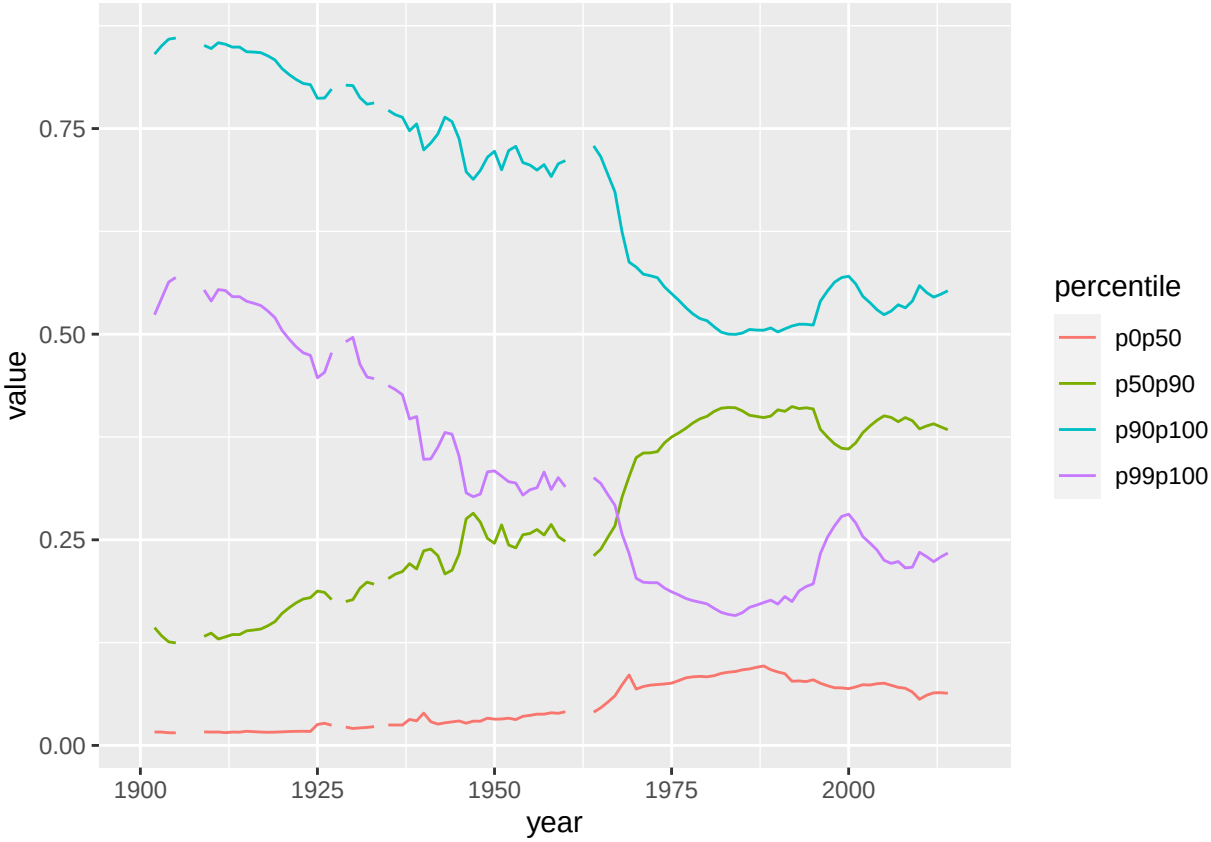
2. Let's start to dig into the data. We have two `type`s of data: "Net personal wealth" and "National income". Start by `filter()`ing the data so we only have "Net personal wealth" for France, name the resulting data `french_data` and then run the code below to visualize the data.

```
french_data <- wid_data %>%
  filter(type == "Net personal wealth", country == "France")
```

Note: When referring to words in the data, make sure they are in quotes "France", "Net personal wealth". When referring to columns, do not use quotes. We'll talk about data types in the next lecture.

```
french_data %>%
  ggplot(aes(y = value, x = year, color = percentile)) +
    geom_line()
```

[1] Hint: You can type all the column names or use the slicker `select(-notes, everything())`



Now we're getting somewhere! The plot shows the proportion of national wealth owned by different segments of French society overtime. For example in 2000, the top 1 percent owned roughly 28 percent of the wealth, while the bottom 50 percent owned about 7 percent.

3. Explain the gaps in the plot. Using `filter()`, look at `french_data` in the years between 1960 and 1970. Does what you see line up with what you guessed by looking at the graph?

Solutions:

After filtering the `french_data` to between 1960 and 1970, we can see that there are missing values in the `value` column thus explaining the gaps in the plot

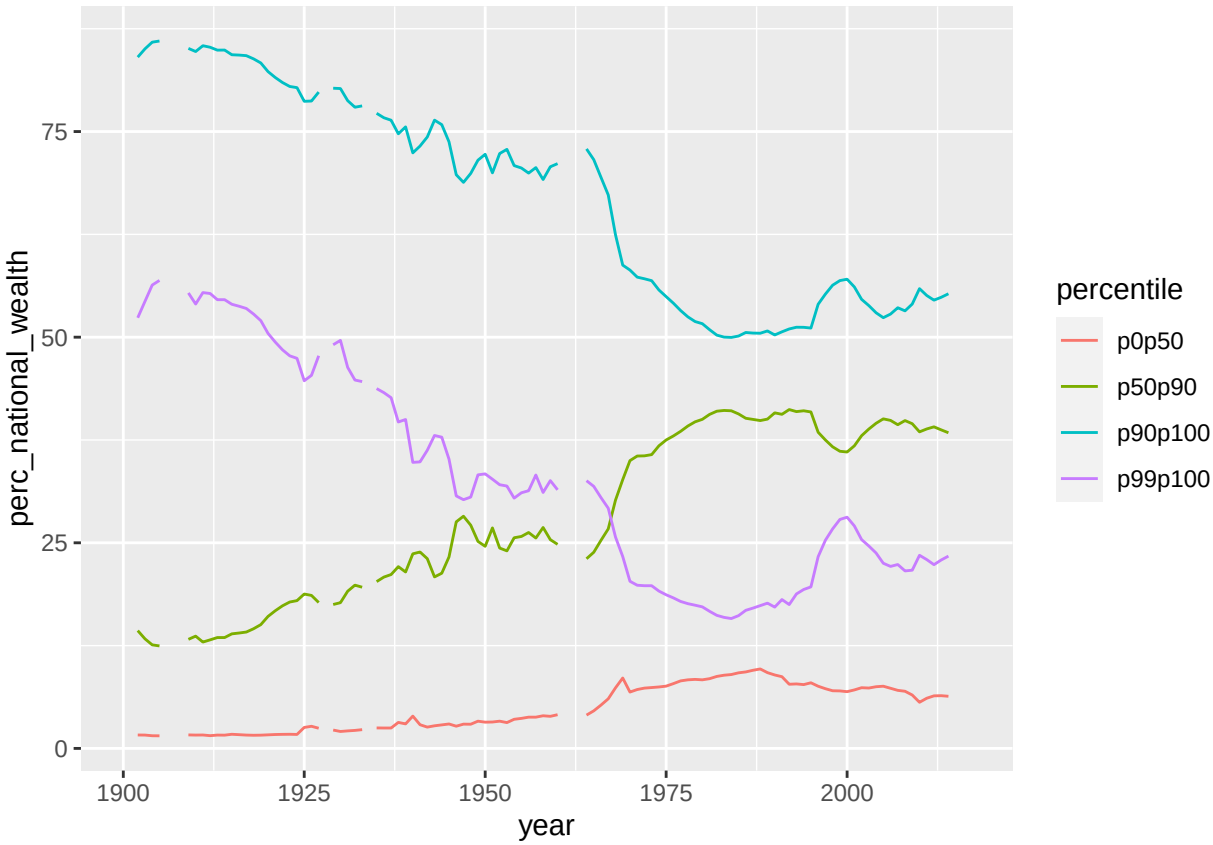
```
french_data %>%
  filter (year >= 1960 & year <= 1970)
```

4. Using `mutate()`, create a new column called `perc_national_wealth` that equals `value` multiplied by 100. Adjust the graph code so that the y axis shows `perc_national_wealth` instead of `value`.

Solutions:

```
french_data <- french_data %>%
  mutate(perc_national_wealth = value * 100)

french_data %>%
  ggplot(aes(y = perc_national_wealth, x = year, color = percentile)) +
  geom_line()
```

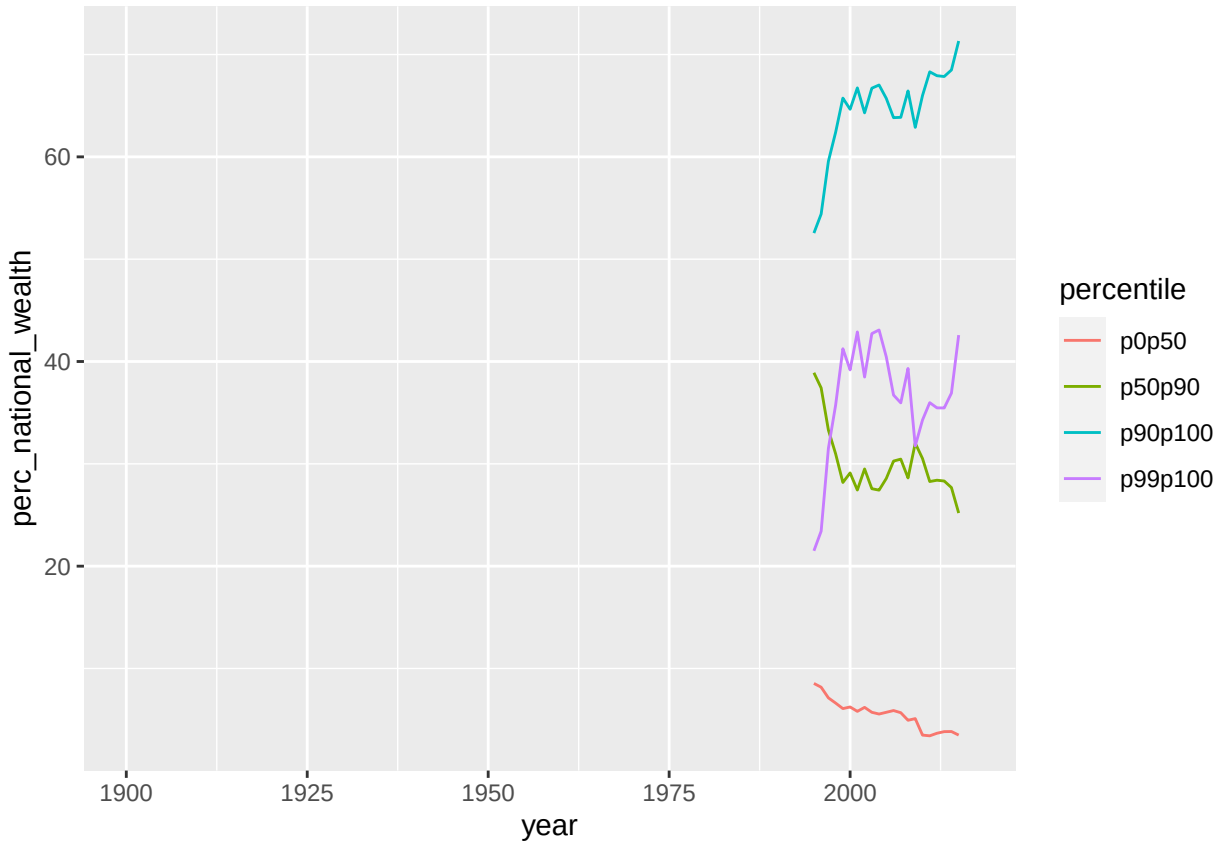


5. Now following the same steps, explore data from the "Russian Federation".

Solutions:

```
russian_data <-
  wid_data %>%
  filter(type == 'Net personal wealth', country == 'Russian Federation') %>%
  mutate(perc_national_wealth = value * 100)

russian_data %>%
  ggplot(aes(y = perc_national_wealth, x = year, color = percentile)) +
  geom_line()
```



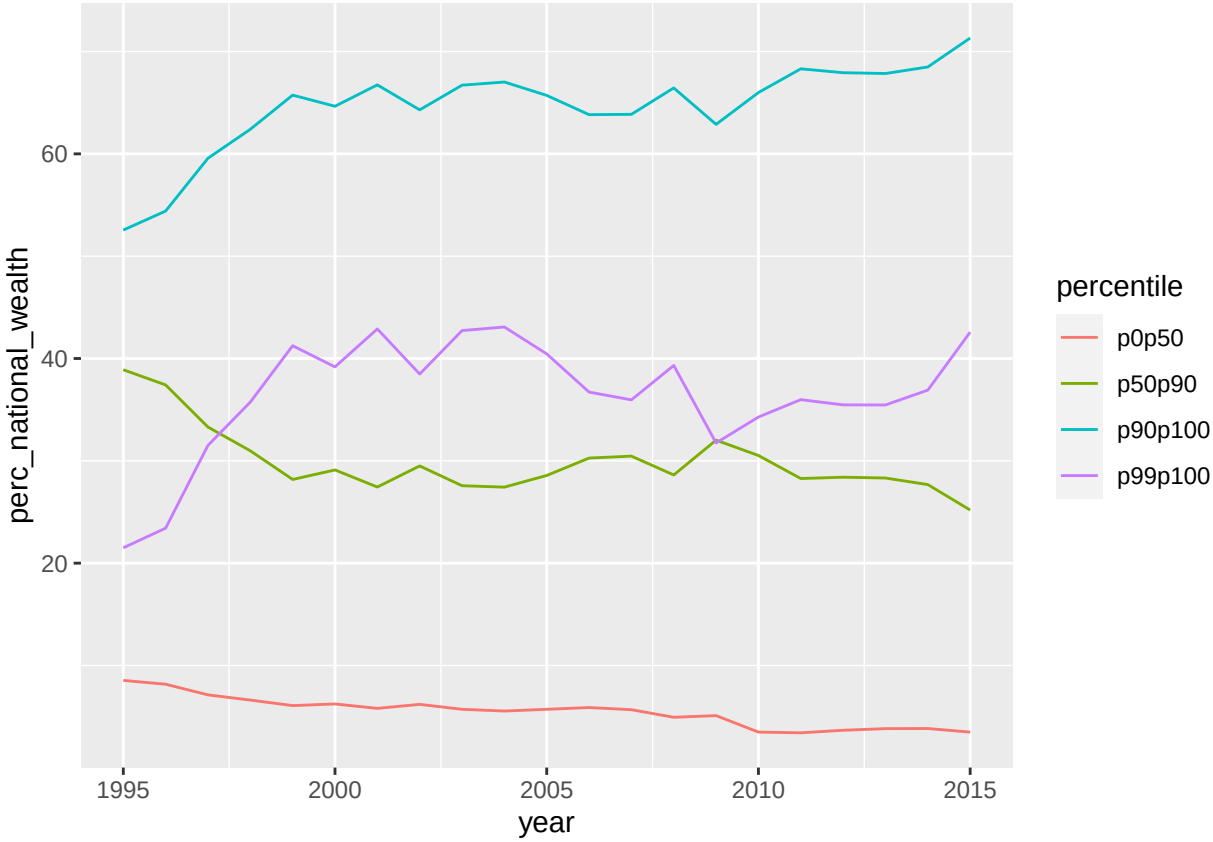
6. The data for "Russian Federation" does not start in 1900, but our y-axis does. That's because we have a bunch of `NA`s. Let's filter out the `NA`s and remake the plot. You cannot test for `NA` using `==` (Try: `NA == NA`). Instead we have a function called `is.na()`. (Try: `is.na(NA)` and `!is.na(NA)`).

Solution:

After filtering out all NA values, we plot the same graph again and see that the `russia_data` starts in the year 1995.

```
russian_data <- russian_data %>% filter(!is.na(perc_national_wealth))

russian_data %>%
  ggplot(aes(y = perc_national_wealth, x = year, color = percentile)) +
  geom_line()
```



7. Use two `dplyr` verbs to figure out what year the bottom 50 percent held the least wealth. First, choose the rows that cover the bottom 50 percent and then sort the data in descending order using `arrange()`.[2].

Solutions:

2011 was the year that the bottom 50 percent held the least wealth as it has the smallest `value`

```
russian_data %>%
  filter(percentile == 'p0p50') %>%
  arrange(value)
```

```
## # A tibble: 21 x 7
##    country      type        percentile  year  value notes      perc_national_we~
##    <chr>        <chr>       <chr>      <dbl>  <dbl> <chr>                  <dbl>
##  1 Russian Fed~ Net person~ p0p50       2011 0.0342 Bottom 50~              3.42
##  2 Russian Fed~ Net person~ p0p50       2010 0.0349 Bottom 50~              3.49
##  3 Russian Fed~ Net person~ p0p50       2015 0.0349 Bottom 50~              3.49
##  4 Russian Fed~ Net person~ p0p50       2012 0.0367 Bottom 50~              3.67
##  5 Russian Fed~ Net person~ p0p50       2013 0.0383 Bottom 50~              3.83
##  6 Russian Fed~ Net person~ p0p50       2014 0.0384 Bottom 50~              3.84
##  7 Russian Fed~ Net person~ p0p50       2008 0.0494 Bottom 50~              4.94
##  8 Russian Fed~ Net person~ p0p50       2009 0.0510 Bottom 50~              5.10
##  9 Russian Fed~ Net person~ p0p50       2004 0.0555 Bottom 50~              5.55
## 10 Russian Fed~ Net person~ p0p50       2007 0.0568 Bottom 50~              5.68
## # ... with 11 more rows
```

8. For both the Russian Federation and French data, calculate the average proportion of wealth owned by

[2]Hint: Look at the examples in `?arrange`

the top 10 percent over the period from 1995 to 2010. You'll have to filter and then summarize with `summarize()`.

Solution:

For the French data

```
french_data %>%
  filter(percentile == "p90p100") %>%
  filter(between(year, 1995, 2010)) %>%
  summarize(top10 = mean(value))
```

```
## # A tibble: 1 x 1
##   top10
##   <dbl>
## 1 0.544
```

For the Russian data

```
russian_data %>%
  filter(percentile == "p90p100") %>%
  filter(between(year, 1995, 2010)) %>%
  summarize(top10 = mean(value))
```

```
## # A tibble: 1 x 1
##   top10
##   <dbl>
## 1 0.633
```

Manipulating midwest demographic data with `dplyr`

1. Now we'll use midwestern demographic data which is here. The dataset includes county level data for a single year. We call data this type of data "cross-sectional" since it gives a point-in-time cross-section of the counties of the midwest. (The world inequality data is "timeseries" data).
2. Save `midwest.dta` in your data folder and load it into R.[3]

Solution:

```
midwest <- read_dta('midwest.dta')
```

3. Run the following code to get a sense of what the data looks like:

   ```
   glimpse(midwest)
   ```

4. I wanted a tibble called `midwest_pop` that only had county identifiers and the 9 columns from `midwest` concerned with population counts. Replicate my work to create `midwest_pop` on your own.

   Hint 1: I went to `?select` and found a *selection helper* that allowed me to select those 9 colums without typing all their names.[4]

Solutions:

```
midwest_pop <-
  midwest %>%
  select(county, state, starts_with("pop"))
```

[3] Hint: `read_dta()`

[4] Hint 2: notice that all the columns start with the same few letters.

```
names(midwest_pop)
```

```
##  [1] "county"          "state"           "poptotal"        "popdensity"
##  [5] "popwhite"        "popblack"        "popamerindian"   "popasian"
##  [9] "popother"        "popadults"       "poppovertyknown"
```

5. From `midwest_pop` calculate the area of each county.[5] What's the largest county in the midwest? How about in Illinois?

Solutions:

Largest county in the midwest: Marquette

```
midwest_pop %>%
  mutate(area = poptotal/popdensity) %>%
  arrange(desc(area)) %>%
  select(c(county, state, area))
```

```
## # A tibble: 437 x 3
##    county      state  area
##    <chr>       <chr> <dbl>
##  1 MARQUETTE   MI    110.
##  2 MARATHON    WI     94.0
##  3 BAYFIELD    WI     89.0
##  4 MARINETTE   WI     82.0
##  5 SAWYER      WI     79.0
##  6 DOUGLAS     WI     78.0
##  7 CHIPPEWA    MI     78.0
##  8 ONTONAGON   MI     78.0
##  9 SCHOOLCRAFT MI     75.0
## 10 PRICE       WI     75
## # ... with 427 more rows
```

Largest county in IL: Mclean and La Salle

```
midwest_pop %>%
  filter(state=='IL') %>%
  mutate(area = poptotal/popdensity) %>%
  arrange(desc(area)) %>%
  select(c(county, state, area))
```

```
## # A tibble: 102 x 3
##    county     state  area
##    <chr>      <chr> <dbl>
##  1 MCLEAN     IL     68.0
##  2 LA SALLE   IL     68
##  3 IROQUOIS   IL     67.0
##  4 LIVINGSTON IL     62.0
##  5 COOK       IL     58.0
##  6 CHAMPAIGN  IL     58.0
##  7 FULTON     IL     52.0
##  8 VERMILION  IL     52
##  9 ADAMS      IL     52.0
## 10 SANGAMON   IL     51.0
## # ... with 92 more rows
```

[5] Notice that `popdensity` $= \frac{poptotal}{area}$

6. From `midwest_pop` calculate percentage adults for each county. What county in the midwest has the highest proportion of adults? What's county in the midwest has the lowest proportion of adults?

Solution:

Highest proportion of adults: Keweenaw

```
midwest_pop %>%
  mutate(perc_adults = popadults/poptotal) %>%
  arrange(desc(perc_adults)) %>%
  select(c(county, state, perc_adults))
```

```
## # A tibble: 437 x 3
##    county      state perc_adults
##    <chr>       <chr>       <dbl>
##  1 KEWEENAW    MI          0.757
##  2 ROSCOMMON   MI          0.730
##  3 IRON        MI          0.728
##  4 ALCONA      MI          0.726
##  5 ADAMS       WI          0.726
##  6 VILAS       WI          0.724
##  7 IRON        WI          0.723
##  8 MONTMORENCY MI          0.703
##  9 ONTONAGON   MI          0.700
## 10 ONEIDA      WI          0.699
## # ... with 427 more rows
```

Lowest proportion of adults: Isabella

```
midwest_pop %>%
  mutate(perc_adults = popadults/poptotal) %>%
  arrange(perc_adults) %>%
  select(c(county, state, perc_adults))
```

```
## # A tibble: 437 x 3
##    county     state perc_adults
##    <chr>      <chr>       <dbl>
##  1 ISABELLA   MI          0.485
##  2 MENOMINEE  WI          0.494
##  3 ATHENS     OH          0.507
##  4 MECOSTA    MI          0.509
##  5 MONROE     IN          0.526
##  6 JACKSON    IL          0.527
##  7 TIPPECANOE IN          0.529
##  8 MCDONOUGH  IL          0.533
##  9 DE KALB    IL          0.537
## 10 HOLMES     OH          0.541
## # ... with 427 more rows
```

7. How many people live in Michigan?

Solution:

```
midwest_pop %>%
  filter(state=='MI') %>%
  summarize(totalpop = sum(poptotal))
```

```
## # A tibble: 1 x 1
##   totalpop
```

```
##       <int>
## 1   9295297
```

8. What's the total area of Illinois? What are the units?[6] If the units don't align with other sources, can this data still be useful?

Solution:

The data doesn't seem to allign with other sources from Google. It is possible that the way that the dataset calculates the population density is different from how we conventionally think about it. It is likely that the dataset used a different formula than total population/total area (for instance, total population/total inhabitable land area).

```
midwest_pop %>%
  filter(state=='IL') %>%
  mutate(area = poptotal/popdensity) %>%
  summarise(totalarea = sum(area))
```

```
## # A tibble: 1 x 1
##   totalarea
##       <dbl>
## 1     3304.
```

[6] Unless you have a great intuition about how large IL is in several metrics, this question requires googling.