TA Session 8: Data Manipulation and Analysis (II) – 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'll use midwestern demographic data which is at this link. 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 "time-series" data).
3. We will also work with data sets from `recent_college_grads.dta`, which you can download here. This is a data on college majors and earnings, specifically the data behind the FiveThirtyEight story "The Economic Guide To Picking A College Major".

Warm-up

Again, we will revisit `msleep` which provides information about mammals sleeping habits.

Using `mutate()` to create new columns

`mutate()` is used to add columns with new variables to the dataset. For example, here I create a new variable called `sleep_percent` that shows what percent of the 24-hour day the mammal sleeps. I provide code with `select()` that helps you check that your work makes sense.

```
msleep %>%
  mutate(sleep_percent = sleep_total / 24) %>%
  select(name, sleep_total, sleep_percent)
```

1. Create a new variable called `sleep_nonrem` that shows the number of hours the mammal sleeps that are not in REM.
2. Create a column called `class` which is `Mammalia` for each of our mammals. (We might want to do this if we were going to join this data with data from other classes of animals such as birds `Aves`).

Using `mutate()` to create multiple new columns

`mutate()` allows you create multiple columns simultaneously and even use the new variables within the `mutate()` call. Run the code to save the output to the name `msleep_with_percents` which we'll use later.

```
msleep_with_percents <-
   msleep %>%
   mutate(sleep_percent = sleep_total / 24,
          awake_percent = 1 - sleep_percent)
```

3. Create a new variable called `percent_brain` which is the percent of body weight taken up by the brain. In the same mutate call, create a variable called `big_brain` that is TRUE if the brain takes up more than 1 percent of mass, and FALSE otherwise.

Using `summarize()` to summarize the data

`summarize()` summarizes data. As with our other functions, we can do many summaries at a time as seen here, where we calculate the median along with the mean.

```
msleep %>%
  summarize(sleep_avg = mean(sleep_total),
            sleep_median = median(sleep_total))
```

For now, we'll summarize the entire data set, but we'll see in a few lessons that we can group our data by sub-groups (e.g. `vore`-types) and get summary statistics for each group.[1]

4. Above you created `msleep_with_percents`. Use that data and create a summary of `sleep_percent` that includes mean, median, and standard deviation.

[1] For the curious, the code looks almost identical except we add `group_by(vore) %>%` before `summarize()` like so: `msleep %>% group_by(vore) %>% summarize(sleep_avg = mean(sleep_total))`

I. Manipulating Midwest Demographic Data with `dplyr`

We'll use midwestern demographic data which is at this link. 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 "time-series" data).

1. Save `midwest.dta` in your data folder and load it into R.

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

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

```
glimpse(midwest)
```

3. 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: I went to `?select` and found a selection helper that allowed me to select those 9 columns without typing all their names!

```
names(midwest_pop)
```

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

```
# replace ... with relevant code
midwest_pop <-
  midwest %>%
  select(county, state, ...)
```

4. From `midwest_pop` calculate the area of each county. What's the largest county in the midwest? How about in Illinois?
5. 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?
6. How many people live in Michigan?
7. Note that together population density and population can give you information about the area (geographic size) of a location. What's the total area of Illinois? You probably have no idea what the units are though. If you google, you'll find that it doesn't align perfectly with online sources. Given the units don't align with other sources, can this data still be useful?

II. Manipulating College Data with `dplyr`

We will explore data on college majors and earnings, specifically the data behind the FiveThirtyEight story "The Economic Guide To Picking A College Major".

We read it in with the `read_dta` function, and save the result as a new data frame called `college_recent_grads`. Because `read_dta` is a function from `haven`, we will need to load that package:

```
library(tidyverse)
library(haven)
setwd("/Users/Coding Camp/TA session 8") # change it to your working directory
college_recent_grads <- read_dta('recent_college_grads.dta')
```

To view the data, you can take a quick peek at your data frame and view its dimensions with the `glimpse` function:

```
glimpse(college_recent_grads)
```

The description of the variables, i.e. the codebook, is given below.

Variable	Description
rank	Rank by median earnings
major_code	Major code, FO1DP in ACS PUMS
major	Major description
major_category	Category of major from Carnevale et al
total	Total number of people with major
sample_size	Sample size (unweighted) of full-time, year-round ONLY (used for earnings)
men	Male graduates
women	Female graduates
sharewomen	Women as share of total
employed	Number employed (ESR == 1 or 2)
employed_full_time	Employed 35 hours or more
employed_part_time	Employed less than 35 hours
employed_full_time_yearround	Employed at least 50 weeks (WKW == 1) and at least 35 hours (WKHP >= 35)
unemployed	Number unemployed (ESR == 3)
unemployment_rate	Unemployed / (Unemployed + Employed)
median	Median earnings of full-time, year-round workers
p25th	25th percentile of earnigns
p75th	75th percentile of earnings
college_jobs	Number with job requiring a college degree
non_college_jobs	Number with job not requiring a college degree
low_wage_jobs	Number in low-wage service jobs

Which major has the lowest unemployment rate?

1. In order to answer this question, all we need to do is sort the data. We use the `arrange` function to do this, and sort it by the `unemployment_rate` variable. By default, `arrange` sorts in ascending order, which is what we want here – we're interested in the major with the lowest unemployment rate.

2. The output from part 1 should give us what we wanted, but not in an ideal form. First, the name of the major barely fits on the page. Second, some of the variables are not that useful (e.g. `major_code`, `major_category`) and some we might want front and center are not easily viewed (e.g. `unemployment_rate`). Use the `select` function to choose `rank`, `major`, and `unemployment_rate`.

```
college_recent_grads %>%
    # code for part 1
    # code for part 2
```

Ok, this is looking better, but do we really need all those decimal places in the unemployment variable? Not really!

3. Round `unemployment_rate`: We create a new variable with the `mutate` function. In this case, we're overwriting the existing `unemployment_rate` variable, by rounding it to 1 decimal place. Incomplete code is given below to guide you in the right direction, however you will need to fill in the blanks.

```
college_recent_grads <- college_recent_grads %>%
    # code for part 1
    # code for part 2
  mutate(unemployment_rate = round(___, 1))
```

4. While were making some changes, let's change `sharewomen` to numeric (it appears to be a string). Make sure to save your changes by overwriting the existing data frame!

```
college_recent_grads <- college_recent_grads %>%
  mutate(sharewomen = as.numeric(___))
```

Which major has the highest percentage of women?

To answer such a question, we need to arrange the data in descending order. For example, if earlier we were interested in the major with the highest unemployment rate, we would use the following.

5. The `desc` function specifies that we want `unemployment_rate` in descending order. Complete the code.

```
college_recent_grads %>%
  arrange(...) %>%
    # code for part 2
```

6. Using what you've learned so far, arrange the data in descending order with respect to proportion of women in a major, and display only the major, the total number of people with major, and proportion of women. Show only the top 3 majors by adding `head(3)` at the end of the pipeline.