Accelerated Coding Lab: More data manipulation

Ari Anisfeld

09-04-2022

Contents

For this assignment we'll use the world inequality database to investigate wealth shares of different income brackets in various countries. Download the data here. The original source data comes from https://wid.world/ which has updated it's data products since we made this lab!

Warm-up

1. Without running the code, predict what the output will be. Then, see if you were right by running the code in the console.

```
x <- c(1.1, 2.2, 3.3, 4.4, 5.5)

x
x[]
x[5]
x[c(3, 1)]
x[c(4,4,4)]
x[3:5]
x[-1]
x[-c(3,1)]
x[c(TRUE,TRUE,FALSE,FALSE,TRUE)]
x[x>3]
```

2. Without running the code, predict what the output will be. Then, see if you were right by running the code in the console. (You can `glimpse(midwest)` to recall what the data looks like. How many rows and columns are there?)

Two of these will cause errors. Why?

```
midwest[1:4]
midwest[c(1,2,3,4)]
midwest[c(13,7)]
midwest[38]
midwest[38,]
midwest[,38]
midwest[1:5, ]
midwest[, c(1,2,3)]
midwest[1:437 > 433,]
midwest[50:52, c(10,20)]
```

Note: if you try to knit code that produces an error, your knitting will fail. You can use `#` to comment out such code or use eval= FALSE (e.g. `{r, eval = FALSE}`).

3. Ari didn't know that `order()` could accept multiple column names! Google "sorting by multiple columns in base r" to help him convert the following code into base R.

```
txhousing %>%
  arrange(desc(year), month, desc(sales)) %>%
  head(3)
```

```
## # A tibble: 3 x 9
##   city     year month sales     volume median listings inventory  date
##   <chr>   <int> <int> <dbl>      <dbl>  <dbl>    <dbl>     <dbl> <dbl>
## 1 Houston  2015     1  4494 1155508809 189300    18649       2.7  2015
## 2 Dallas   2015     1  3066  773952769 203300     9063       1.8  2015
## 3 Austin   2015     1  1656  512034244 237500     5567       2.2  2015
```

4. a. What does the `distinct(midwest, state)` do?
   b. What does `unique(midwest$state)`?
   c. How are they different?
   d. Which function do you think is from `dplyr` and what are some patterns that make it similar to the `dplyr` verbs?

5. Which of the following code works? Can you think of two additional strategies to get filter the data so we only have data from IA, IL, MI and WI using `filter`?[1]

```
midwest %>%
  filter(state == "IA", "IL", "MI", "WI")

midwest %>%
  filter(state == c("IA", "IL", "MI", "WI"))

midwest %>%
  filter(state == "IA"| state ==  "IL"|
         state == "MI"| state =="WI")
```

[1] *Hint* one strategy we discussed in class; for the second strategy, look at the previous problem.

Data manipulation 3 ways.

In addition to `dplyr` and base R `[`, you may occasionally see code where a partner or professor uses base R functions that have similar functionality to `dplyr`.

1. `subset()` does filtering and selecting in the same function call. Rewrite this using `dplyr` verbs.

```r
# get city, month, year, volume, and listings where the volume
# is over $1 billion in winter 2013 to 2016.
subset(txhousing, month %in% c(12, 1, 2) &
                 between(year, 2013, 2016) &
                  volume > 1e9,
       select = c(city, year, month, volume, listings))
```

2. `within` is a doppelganger of `mutate` with peculiar syntax. Convert the following code to `dplyr`

```r
within(txhousing[c("city", "year", "month", "volume", "sales", "median")], {
  mean_sales_price <- round(volume / sales)
  best_month_sales_price <- max(mean_sales_price, na.rm=TRUE)
  percent_of_best <- mean_sales_price / best_month_sales_price
})
```

3. *A recipe for unreadable code and difficult debugging.* The following code is poor quality because it's difficult for other humans (including future you) to read. What makes it hard to read?

- There are several function calls within function calls.
- The names used are not descriptive (e.g. `msp` for `mean_sales_price` might feel like a time saver, until you have to interpret the code or output and so keep returning to the original code to figure out what it means.

a. First re-write the code in baseR so the output is identical, but the code is prettier.
b. Re-write the code in `dplyr` using `%>%`.

```r
# BAD style:
ds <- subset(within(txhousing[order(-txhousing$volume),],
              {msp <- round(volume / sales)
              msp_diff_m <- abs(msp - median)}
              ),
      city == "Houston" &
      (year == 2014  | year == 2015 | year == 2013),
      select = c(city, year, month, msp, msp_diff_m, volume))

head(ds)
```

World Inequality Database

For this assignment we'll use the world inequality database to investigate wealth shares of different income brackets in various countries.

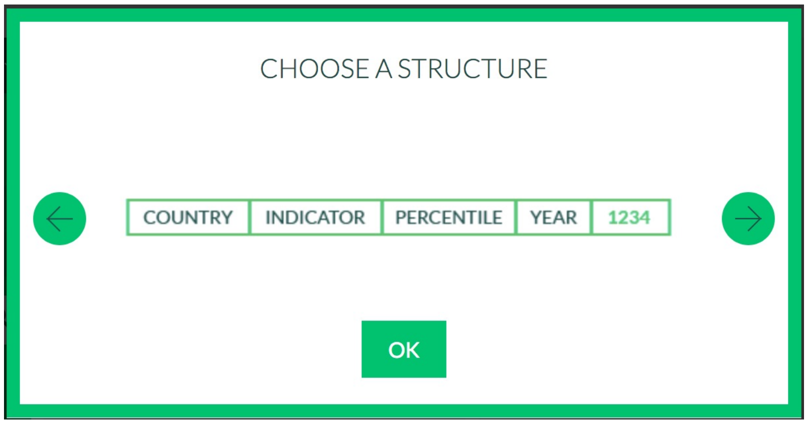
Load the data with the name `wid_data_raw`.

- Make sure you know what folder the data is in relative to your notebook.
- Pay attention to the file type.

Adding a header

What's up with the column names? Open the excel file, you'll see there are no headers!

The columns should be named like so.



You may be tempted to change the data, but we prefer to make our process reproducible! Fortunately, 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"))
```

1. Update your read_xlsx call to add `col_names`.

Remark: Now when we look at the second column. It's a mess. However, there's a tidyverse function `separate` that comes in hand here. This come from `tidyr` which we only have time for a taste of.[2]

"The goal of tidyr is to help you create tidy data. Tidy data is data where:

- Every column is variable.
- Every row is an observation.
- Every cell is a single value."[3]

Here we have multiple values in the `indicator` column. `seperate` allows us to use patterns in the data to split the data `into` distinct columns. Here, we can separate it based on where the `\n` are.[4]

Example: Let's start with a tiny example.

```
# Our data is very mess!
example <-
  tibble(indicator =
          c("stuff\nmore stuff\neven more stuff\n",
            "shweal_p99p100_z_CN\nNet personal wealth\nTop 1% | share\n"))


separate(
  data = example, # data is in the first position so this is pipe-able.
```

[2]Other high use functions are `pivot_longer()` and `pivot_wider()` for re-shaping data to make it "tidier".
[3]https://tidyr.tidyverse.org
[4]Windows users: On some Windows computer you might see `\r\n` instead of `\n`

```
  col =  indicator,
  sep = "\\n", # sep is the "seperator" for reasons beyond
               # coding lab; we need \\n instead of \n.
  into = c("col a", "col b", "col c", "empty") #  into is the new col names
)
```

```
## # A tibble: 2 x 4
##   `col a`            `col b`             `col c`         empty
##   <chr>              <chr>               <chr>           <chr>
## 1 stuff              more stuff          even more stuff ""
## 2 shweal_p99p100_z_CN Net personal wealth Top 1% | share  ""
```

Since there's are 3 `\n` to split on, we end up with *4* strings.[5] The last string is always empty. If we ignored the final string and wrote `into = c("col a", "col b", "col c")` we would get a lot of warnings, which we could also ignore.

1. Add a call to `separate` to tidy your data. `separate` takes data as it's first argument, so we can pipe our imported data into it.

```
wid_data_raw <-
  read_xlsx("world_wealth_inequality.xlsx",
            col_names = c("country", "indicator", "percentile", "year", "value")) %>%
  # here sep is includes [\\r]? so that the code works on windows or mac!
  separate(indicator, sep =  "[\\r]?\\n",
           into = c("row_tag", "type", "notes", "empty"))
```

clean-up

We want a clean reproducible code so you should just have one block of code to read 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`

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 `empty` and assign the output to the name `wid_data`.
2. Let's start to dig into the data. We have two `types` of data: "Net personal wealth" and "National income". Let's focus on "Net personal wealth" for France. Create a data set called `french_data` with the desired rows and then run the code below to visualize the data.[6]

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

[5] We see that sep is `"\\n"`. This is called a "regular expression" which allow us to flexibly match sequences of characters in text. For example, you'll see `"[\\r]?\\n"` below, which says look for `"\r\n"` or `"\n"`.

[6] We expect to see 4 lines and to get a message about several warnings.

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.

1. 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?
2. 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`.
3. Now following the same steps, explore data from the "Russian Federation".
4. 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. Filter out the `NA`s and remake the plot.
5. What year did the bottom 50 percent hold the least wealth? What year did the middle 40 percent hold the most wealth?
6. How many years does the Russian top 1 percent control more money then the 90th to 99th percentile?[7]
7. For both the Russian Federation and French data, calculate the average of the proportion of wealth owned by the top 10 percent over the period from 1995 to 2010. You'll have to choose the relevant rows and then summarize with `summarize()`.
8. Now say you want to compare France and Russia to the other countries in the database. There has to be an easier way to do this analysis without copying and pasting so much!

Introducing `group_by` you can use `group_by` to divide your data into smaller data sets determined by the `grouping` columns. Here we `group_by(country)` which tells R to treat `wid_data` as if it were made up of 8 distinct country data sets (i.e. `french_data`, `russian_data`, `indian_data` etc.) Then when we call `summarize()` it summarizes each of those data sets and puts the results into a single tibble!

```
wid_data %>%
  mutate(perc_national_wealth = value * 100) %>%
  filter(percentile == "p90p100", between(year, 1995, 2010)) %>%
  group_by(country) %>%
  summarise(top10 = round(mean(perc_national_wealth, na.rm=TRUE)))
```

```
## # A tibble: 8 x 2
##   country            top10
##   <chr>              <dbl>
## 1 China                 50
## 2 France                54
## 3 India                 56
## 4 Korea                 64
## 5 Russian Federation    63
## 6 South Africa          85
## 7 United Kingdom        51
## 8 USA                   68
```

2. What happens if you group by country and year before summarizing?

We'll return to this idea soon, but take some time to experiment with it.

[7] Suggestion: you may need to work with vectors directly.

3. The base R analog is `aggregate` here's two exampels of getting the "mean perc_national_weath"

```
wid_data_for_agg <-
  wid_data %>%
  mutate(perc_national_wealth = value * 100) %>%
  filter(percentile == "p90p100", between(year, 1995, 2010))


aggregate(wid_data_for_agg$perc_national_wealth,
          by = list(wid_data_for_agg$country),
          FUN = mean,
          na.rm = TRUE)
```

```
##              Group.1        x
## 1              China 50.37875
## 2             France 54.37562
## 3              India 55.60000
## 4              Korea 63.67333
## 5 Russian Federation 63.30000
## 6       South Africa 85.39063
## 7     United Kingdom 50.66000
## 8                USA 67.90125
```

```
aggregate(perc_national_wealth ~ country,
          FUN = mean,
          na.rm = TRUE,
          data = wid_data_for_agg)
```

```
##              country perc_national_wealth
## 1              China             50.37875
## 2             France             54.37562
## 3              India             55.60000
## 4              Korea             63.67333
## 5 Russian Federation             63.30000
## 6       South Africa             85.39063
## 7     United Kingdom             50.66000
## 8                USA             67.90125
```

Try to adjust this call to do the aggregation at the county by year level.

Challenge:

1. Repeat the `wid_data` analysis above for Korea using `[` or base R functions.