

Accelerated Coding Lab 6: More data manipulation

Harris Coding Lab

Summer 2023

Contents

Warm-up	1
World Inequality Database	2
Adding a header	3
manipulating world inequality data with <code>dplyr</code>	4
FYI: Data manipulation 3 ways.	6

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[38]

# double index: data[rows, columns]
midwest[38,]
midwest[,38]
midwest[1:5, ]
midwest[, c(1,2,3)]
midwest[1:nrow(midwest) > 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.
 - 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?
4. We're trying to get all the data from Illinois, Iowa, Michigan and Wisconsin. Which option works?

```
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")
```

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

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`.

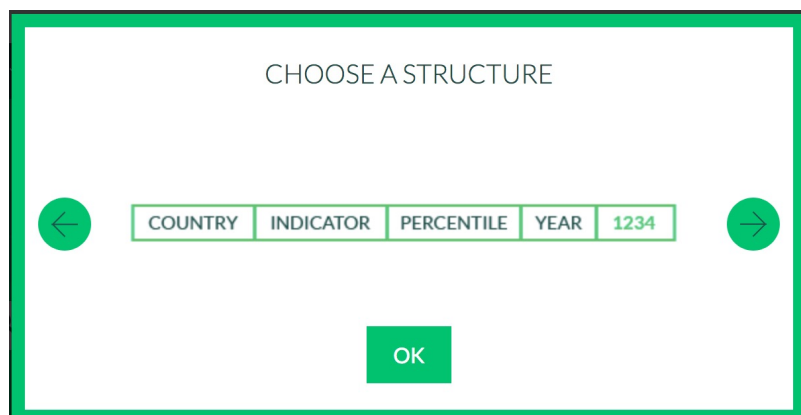
- Make sure you know what folder the data is in relative to your notebook.
- Pay attention to the file type.

¹*Hint* Sometimes it's easier to say what you don't want.

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.

“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.”²

Here we have multiple values in the `indicator` column. `separate` 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.³

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.  
  col = indicator,
```

²<https://tidyr.tidyverse.org> We saw `pivot_longer()` for re-shaping data earlier.

³Windows users: On some Windows computer you might see `\r\n` instead of `\n`

```
sep = "\\n", # sep is the "separator". we use
           # \\n instead of \n for technical reasons
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.⁴ 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 safely 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 redundant columns that add clutter when we examine the data. What `dplyr` verb let's you choose what columns to see?
2. Remove the unwanted column `row_tag` and `empty` and assign the output to the name `wid_data`.
3. 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.⁵

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

⁴We 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"`.

⁵We expect to see 4 lines on the plot 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 over time. 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. Use `filter()` to look at 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 NAs. Filter out the NAs 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?⁶
7. For both the Russian Federation and French, calculate the average of the proportion of wealth owned by the top 10 percent over the period from 1995 to 2010. 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.

⁶Suggestion: Probably easier to do this with vectors directly.

FYI: 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. What two `dplyr` verbs does `subset()` do in the same function call? Look at the examples in help.
2. There's a function called `within()`. What `dplyr` verb has the same functionality?

```
within(txhousing,
  {
    mean_sales_price <- round(volume / sales) # notice no comma
    best_month_sales_price <- max(mean_sales_price, na.rm=TRUE)
  })
```

3. The base R analog is `aggregate` here's two examples 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 `[s`.
2. Ask your own question of the data and try to answer it.