Data transformation

This topic is covered on Jan 16.

This exercise examines how income inequality has changed over time in the U.S. We will measure inequality by the 10th, 50th, and 90th percentiles of wage and salary income from 1962 to 2022.[1] The goal is to produce a graph like this one.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(haven)
```

If you don't have **haven**, you will need to install with `install.packages("haven")` in your console.

The course website talks about where you can access data. For now, we will use simulated data.

```
cps_data <- read_dta("https://soc114.github.io/data/simulated_cps_data.dta")
```

[1] Thanks to past TA Abby Sachar for designing the base of this exercise.

Explore the data

Type `cps_data` in the console. Some columns such as `educ` have a numeric code and a label. The code is how IPUMS has stored the data. The label is what the code means. You can always find more documentation explaining the labels on the IPUMS-CPS website.

filter() to cases of interest

> In this step, you will use `filter()` to convert your `cps_data` object to a new object called `filtered`.

The `filter()` function keeps only rows in our dataset that correspond to those we want to study. The examples on the documentation page are especially helpful. The R4DS section is also helpful.

Here are two ways to use `filter()` to restrict to people working 50+ weeks per year. One way is to call the `filter()` function and hand it two arguments

- `.data = cps_data` is the dataset
- `year == 1962` is a logical condition coded `TRUE` for observations in 1962

```
filter(.data = cps_data, year == 1962)
```

The result of this call is a `tibble` with only the observations from 1962. Another way to do the same operation is with the pipe operator `|>`

```
cps_data |>
  filter(year == 1962)
```

This approach begins with the data set `cps_data`. The pipe operator `|>` hands this data set on as the first argument to the `filter()` function in the next line. As before, the second argument is the logical condition `year == 1962`.

The piping approach is often preferable because it reads like a sentence: begin with data, then filter to cases with a given condition. The pipe is also useful

The pipe operator `|>` takes what is on the first line and hands it on as the first argument to the function in the next line. This reads in a sentence: begin with the `cps_data` tibble and then `filter()` to cases with `year == 1962`. The pipe can also string together many operations, with comments allowed between them:

```r
cps_data |>
  # Restrict to 1962
  filter(year == 1962) |>
  # Restrict to ages 40-44
  filter(age >= 40 & age <= 44)
```

Your turn. Begin with the `cps_data` dataset. Filter to

- people working 50+ weeks per year (check documentation for `wkswork2`)
- valid report of `incwage` greater than 0 and less than 99999998

```r
filtered <- cps_data |>
  # Subset to cases working full year
  filter(wkswork2 == 6) |>
  # Subset to cases with valid income
  filter(incwage > 0 & incwage < 99999998)
```

> **Note**
>
> Filtering can be a dangerous business! For example, above we dropped people with missing values of income. But what if the lowest-income people refuse to answer the income question? We often have no choice but to filter to those with valid responses, but you should always read the documentation to be sure you understand who you are dropping and why.

https://www.youtube.com/embed/OE2gE_3DLf8

group_by() and summarize() for subpopulation summaries

> In this step, you will use `group_by()` and `summarize()` to convert your `mutated` object to a new object called `summarized`.

Each row in our dataset is a person. We want a dataset where each row is a year. To get there, we will group our data by year and then summarize each group by a set of summary statistics.

Introducing summarize() with the sample mean

To see how `summarize()` works, let's first summarize the sample mean income within each year. The input has one row per person. The result has one row per group. For each year, it records the sample mean income.

```
filtered |>
  group_by(year) |>
  summarize(mean_income = mean(incwage))
```

```
# A tibble: 62 x 2
    year mean_income
   <dbl>       <dbl>
 1  1962       6383.
 2  1963       5831.
 3  1964       6688.
 4  1965       6066.
 5  1966       6438.
 6  1967       6745.
 7  1968       7244.
 8  1969       8465.
 9  1970       9198.
10  1971       8490.
# i 52 more rows
```

https://www.youtube.com/embed/loOJDf1aF-w

Using summarize() with weighted quantiles

Instead of the mean, we plan to use three other summary statistics: the 10th, 50th, and 90th percentiles of income. We also want to incorporate the sampling weights provided with the Current Population Survey, in order to summarize the population instead of the sample.

We will use the `wtd.quantile` function to create weighted quantiles. This function is available in the `Hmisc` package. If you don't have that package, install it with `install.packages("Hmisc")`. Using the `Hmisc` package is tricky, because it has some functions with the same name as functions that we use in the `tidyverse`. Instead of loading the whole package, we will only load the functions we are using at the time we use them. Whenever we want to calculate a weighted quantile, we will call it with the code `packagename::functionname()` which in this case is `Hmisc::wtd.quantile()`.

The `wtd.quantile` function will take three arguments:

- `x` is the variable to be summarized
- `weights` is the variable containing sampling weights
- `probs` is the probability cutoffs for the quantiles. For the 10th, 50th, and 90th percentiles we want 0.1, 0.5, and 0.9.

The code below produces weighted quantile summaries.

```
summarized <- filtered |>
  group_by(year) |>
  summarize(
    p10 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.1),
    p50 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.5),
    p90 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.9),
    .groups = "drop"
  )
```

https://www.youtube.com/embed/IOMbo_3ynKU

pivot_longer() to reshape data

> In this step, you will use `pivot_longer()` to convert your `summarized` object to a new object called `pivoted`. We first explain why, then explain the task.

We ultimately want to make a `ggplot()` where income values are placed on the y-axis. We want to plot the 10th, 50th, and 90th percentiles along this axis, distinguished by color. We need them all in one colun! But currently, they are in three columns.

Here is the task. How our data look:

```
# A tibble: 62 x 4
   year   p10   p50    p90
  <dbl> <dbl> <dbl>  <dbl>
1  1962 1826. 4460. 11733.
2  1963 1770. 4484. 11934.
# i 60 more rows
```

Here we want our data to look:

```
# A tibble: 186 x 3
   year quantity income
  <dbl> <chr>     <dbl>
1  1962 p10       1826.
```

```
2  1962 p50       4460.
3  1962 p90      11733.
4  1963 p10       1770.
5  1963 p50       4484.
6  1963 p90      11934.
# i 180 more rows
```

This way, we can use `year` for the x-axis, `quantity` for color, and `value` for the y-axis.

Use `pivot_longer()` to change the first data frame to the second.

- Use the `cols` argument to tell it which columns will disappear
- Use the `names_to` argument to tell R that the names of those variables will be moved to a column called `quantity`
- Use the `values_to` argument to tell R that the values of those variables will be moved to a column called `income`

If you get stuck, see how we did it at the end of this page.

```
pivoted <- summarized %>%
  pivot_longer(
    cols = c("p10","p50","p90"),
    names_to = "quantity",
    values_to = "income"
  )
```

https://www.youtube.com/embed/DRuPHsX6GlM

left_join() an inflation adjustment

> In this step, you will use `left_join()` to merge in an inflation adjustment

A dollar in 1962 bought a lot more than a dollar in 2022. We will adjust for inflation using the Consumer Price Index, which tracks the cost of a standard basket of market goods. We already took this index to create a file `inflation.csv`,

```
inflation <- read_csv("https://soc114.github.io/data/inflation.csv")
```

```
Rows: 77 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (2): year, inflation_factor
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The `inflation_factor` tells us that $1 in 1962 could buy about as much as $10.10 in 2023. To take a 1962 income and report it in 2023 dollars, we should multiple it by 10.1. We need to join our data

```
# A tibble: 186 x 3
   year quantity income
  <dbl> <chr>     <dbl>
1  1962 p10       1826.
2  1962 p50       4460.
3  1962 p90      11733.
# i 183 more rows
```

together with `inflation.csv` by the linking variable `year`. Use `left_join()` to merge `inflation_factor` onto the dataset `pivoted`. Below is a hypothetical example for the structure.

```
# Hypothetical example
joined <- data_A |>
  left_join(
    data_B,
    by = join_by(key_variable_in_A_and_B)
  )
```

```
joined <- pivoted |>
  left_join(
    inflation,
    by = join_by(year)
  )
```

https://www.youtube.com/embed/EptsO1HLBs4

mutate() to adjust for inflation

In this step, you will use `mutate()` to multiple `income` by the `inflation_factor`

The mutate() function modifies columns. It can overwrite existing columns or create new columns at the right of the data set. The new variable is some transformation of the old variables.

```
# Hypothetical example
old_data |>
  mutate(new_variable = old_variable_1 + old_variable_2)
```

Use mutate() to modify `income` so that it takes the values `income * inflation_factor`.

```
mutated <- joined |>
  mutate(income = income * inflation_factor)
```

https://www.youtube.com/embed/H0vxvCYDuzU

ggplot() to visualize

Now make a `ggplot()` where

- `year` is on the x-axis
- `income` is on the y-axis
- `quantity` is denoted by color

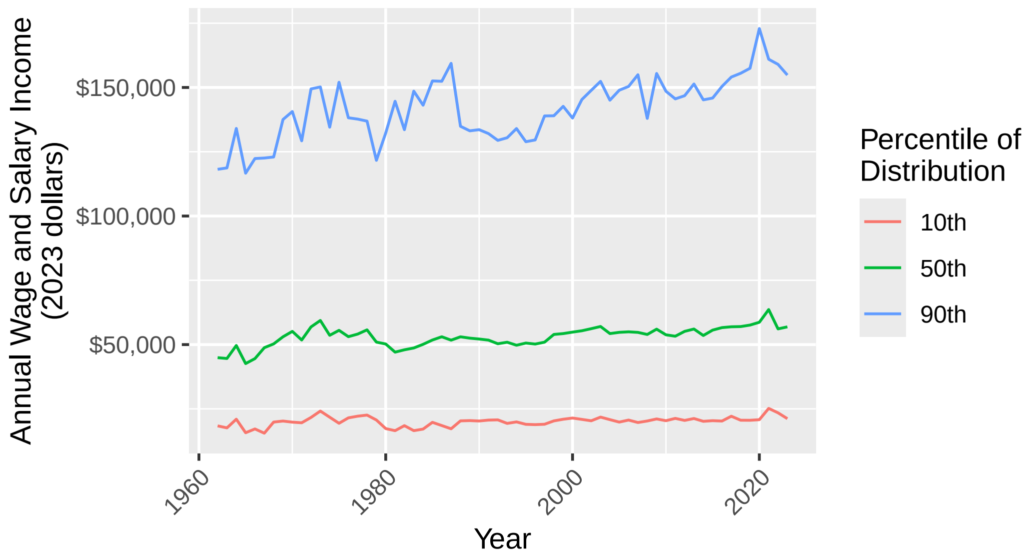
Discuss. What do you see in this plot?

All together

Putting it all together, we have a pipeline that goes from data to the plot.

```
cps_data |>
  # Subset to cases working full year
  filter(wkswork2 == 6) |>
  # Subset to cases with valid income
  filter(incwage > 0 & incwage < 99999998) |>
  # Produce summaries
  group_by(year) |>
  summarize(
    p10 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.1),
    p50 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.5),
    p90 = Hmisc::wtd.quantile(x = incwage, weights = asecwt, probs = 0.9
    ),
```

```
    .groups = "drop"
  ) |>
  pivot_longer(
    cols = c("p10","p50","p90"),
    names_to = "quantity",
    values_to = "income"
  ) |>
  # Join data for inflation adjustment
  left_join(
    read_csv("https://soc114.github.io/data/inflation.csv"),
    by = join_by(year)
  ) |>
  # Apply the inflation adjustment
  mutate(income = income * inflation_factor) |>
  # Produce a ggplot
  ggplot(aes(x = year, y = income, color = quantity)) +
  geom_line() +
  xlab("Year") +
  scale_y_continuous(name = "Annual Wage and Salary Income\n(2023 dollars)",
                     labels = scales::label_dollar()) +
  scale_color_discrete(name = "Percentile of\nDistribution",
                       labels = function(x) paste0(gsub("p","",x),"th")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



https://www.youtube.com/embed/fgQe7BaZEMQ

Want to do more?

If you have finished and want to do more, you could

- incorporate the `educ` variable in your plot. You might want to group by those who do and do not hold college degrees, perhaps using `facet_grid()`
- try `geom_histogram()` for people's incomes in a specific year
- explore IPUMS-CPS for other variables and begin your own visualization