Descriptive statistics and relational data

Day 4

Solveig Bjørkholt

30. June

Plan for today

What we will learn today:

- Find descriptive statistics
- Displaying descriptive statistics
- Joining datasets

Find descriptive statistics

Descriptive statistics are useful in giving insights about patterns in your data. We are not going to delve heavily into statistics in this course, but we'll look at a few basic measures, including finding the:

- `mean`
- `median`
- `mode`
- `variance`
- `standard deviation`
- `count`

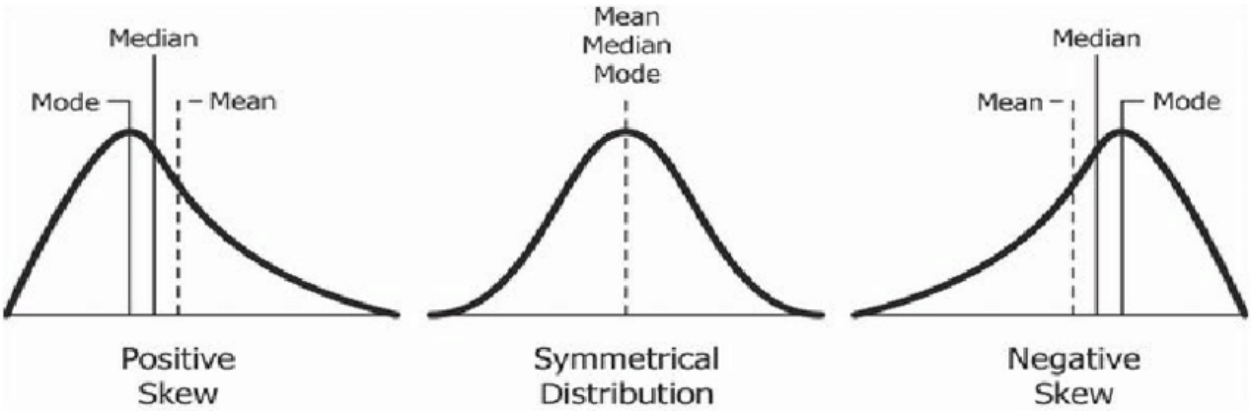
of a variable.

Let's read the dataset from yesterday in to `R`.

```
costofliving <- read_rds("../../datafolder/costofliving2.rds")
```

Measures to find central tendency

First, we'll look at measures to find central tendency. These measures reflect the center of a data distribution. What is a distribution, you might ask. Well, it's simply how our data looks with regard to how many different units have different values. For example, say our variable is how many minutes a person spends in the shower. If most people in the datset spends about 10 minutes, while a bit few spend equally 5 minutes and 15 minutes, we'd get a symmetrical distribution. However, if relatively more people spend 15 minutes, we'll get a negative skew, which means that the distribution would lean right. This is important because how the data is distributed is fundamental to a lot of techniques and models we use.

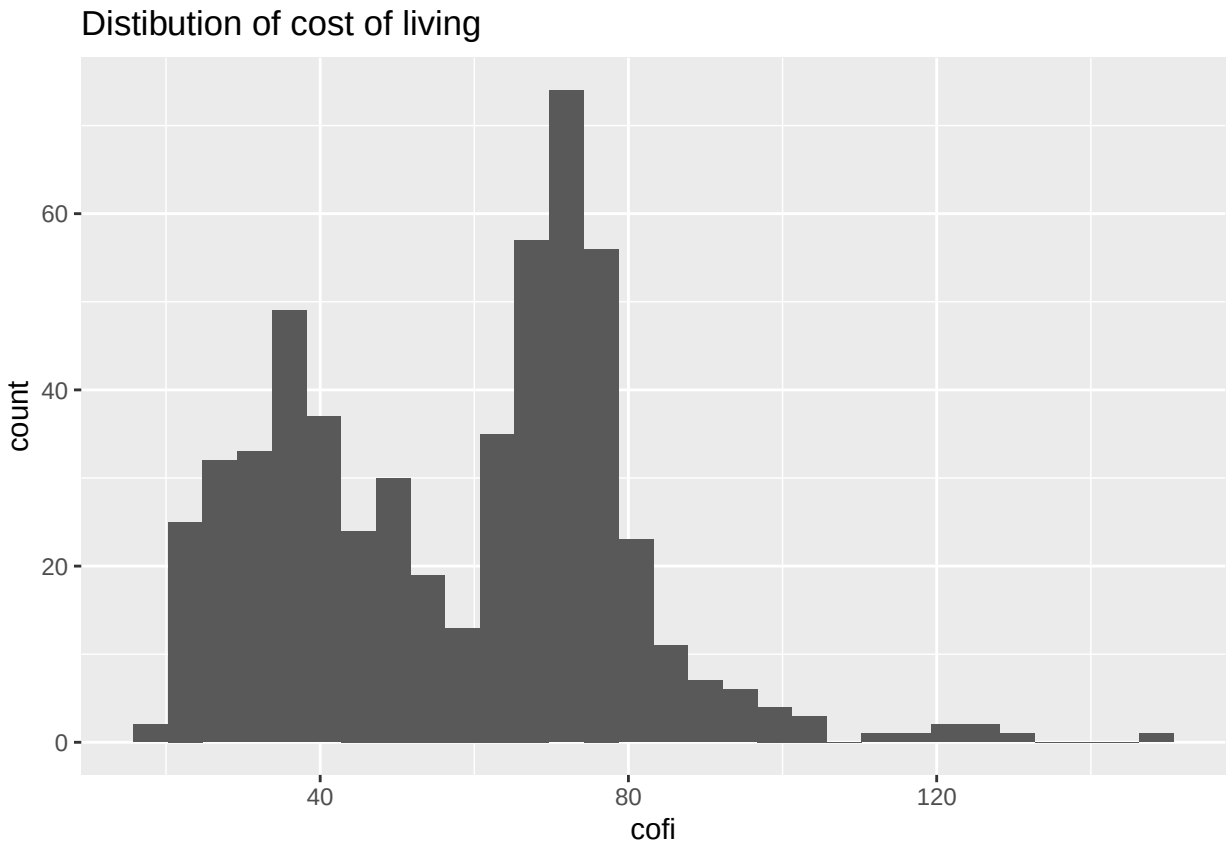


We'll look more into plotting in week 2. For now, I visualize the distribution of the `cofi` variable by making a histogram. Looks most cities here have a cost of living around 70. There's a slight negative skew to the data, meaning that most cities (thankfully) have a cost of living lower to that of New York.

```
costofliving %>%
  ggplot(aes(cofi)) +
  geom_histogram() +
  ggtitle("Distibution of cost of living")
```

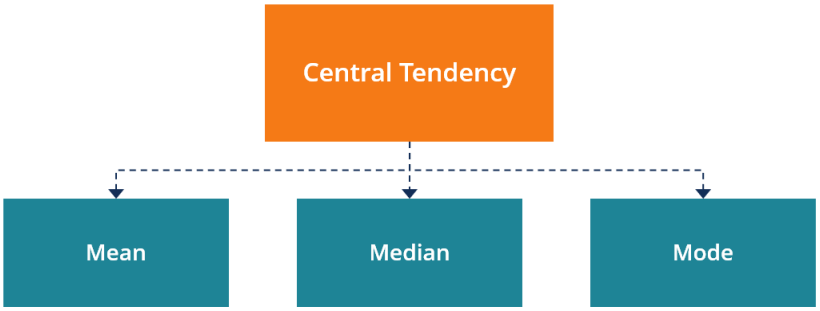
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 30 rows containing non-finite values (stat_bin).
```



Measures of central tendency, then, tells us where most of the units fall. There are three main ways of measuring central tendency:

- **Mean**: the sum of the values divided by the number of values.
- **Median**: the value that's exactly in the middle of the data when it is ordered.
- **Mode**: the number which appears most often.



Notice from the figure above that if the distribution is skewed, the mean, the median and the mode will differ. We'll find the mean, median and mode of our variable `cofi`. First, we load the `tidyverse` package.

```
library(tidyverse)
```

Then, finding the mean, we can use `summarise` together with the `mean` function. Notie that we add `na.rm = TRUE`, which means that `R` should calculate the mean without taking into account that there are missing values in the data. If there are missing values and we do not set `na.rm` to `TRUE`, the mean would become `NA`.

```
costofliving %>%
  summarise(cofi_mean = mean(cofi, na.rm = TRUE)) # Use summarise and mean on the variable, set na.rm t
```

```
## # A tibble: 1 x 1
##   cofi_mean
##       <dbl>
## 1      57.4
```

The average cost of living in all the cities in our dataset is 57.44.

To calculate the median, use the same procedure with `summarise`, but use the function `median`. Remember to add `na.rm = TRUE` to this piece of code as well.

```
costofliving %>%
  summarise(cofi_median = median(cofi, na.rm = TRUE)) # Use summarise and median on the variable, set n
```

```
## # A tibble: 1 x 1
##   cofi_median
##         <dbl>
## 1        62.4
```

The median is slightly higher than the mean, which makes sense since our data has a negative skew.

Lastly, the mode is the value that occurs most frequently in our data. It can be a good measure if you have a count variable (i.e. a variable where you for example have counted the number of times a person goes to the cinema), but in cases where you have continuous data with decimals, it's usually not that useful to see which value that is most frequent. Thus, here, we calculate the mode on the `country` variable to see which country appears most often.

```
costofliving %>%
  count(country, name = "number_of_countries") %>% # Count the number of times each country shows up, g
  slice_max(number_of_countries, n = 1) # Picking out the first maximum value of the number_of_countrie
```

```
## # A tibble: 1 x 2
##   country  number_of_countries
##   <fct>                  <int>
## 1 " India"                  45
```

Measures to find spread

Now that we know where most of the units fall on the distribution, it would also be useful to know something about the units that are not "typical" for the data. This is often referred to as the *spread* of the data, because it tells us something about all the units that fall around the central tendency.

Two measures are typically used:

- **Variance**: How much a variable varies around the mean.
- **Standard deviation**: How far on average each value lies from the mean.

To calculate the variance, use `var`. Admittedly, the *variance* measure does not tell us much except that there is variation around the mean in our data.

```
costofliving %>%
  summarise(cofi_variance = var(cofi, na.rm = TRUE)) # A positive number tells us that there is variati
```

```
## # A tibble: 1 x 1
##   cofi_variance
##           <dbl>
## 1          468.
```

To say something more substantive about the spread, we can use the *standard deviation*. To calculate this, use the `sd` function. This is a measure that occurs in the same scale as the variable, so the standard deviation is the average units around the mean.

```
costofliving %>%
  summarise(cofi_sd = sd(cofi, na.rm = TRUE)) # Units vary on average 21.63 cost of living index points
```

```
## # A tibble: 1 x 1
##   cofi_sd
##     <dbl>
## 1    21.6
```

Count

If we have a categorical variable, it does not make sense to find either the mean, median, variance or standard deviation because there are no numerical values to calculate these statistics from. In these cases, we often just count the number of times different values occur.

We already had a look at this with calculating the mode. Here, I use somewhat of the same procedure by using `group_by` and `count`. I leave out the `slice_max` function which grabs only parts of the rows, and use instead `arrange` and `desc` to get the variable in descending order.

```
costofliving %>%
  group_by(country) %>% # Find the next statistic per country
  count(name = "country_n") %>% # Count the number of instances (i.e. the number of countries per city)
  arrange(desc(country_n)) # Arrange in descending order so that the highest values come on top
```

```
## # A tibble: 162 x 2
## # Groups:   country [162]
##    country           country_n
##    <fct>                 <int>
##  1 " India"                 45
##  2 " United Kingdom"        34
##  3 "NA"                     29
##  4 " Canada"                26
##  5 " Germany"               25
##  6 " Italy"                 19
##  7 " Netherlands"           13
##  8 " Spain"                 13
##  9 " CA"                    12
## 10 " Poland"                 9
## # ... with 152 more rows
```

Alternatively, we could find the percentages. Here, I find how many cities in the dataset each country in percentage stand for. Almost 8 percent of the cities in our dataset are in India.

```
all_countries <- costofliving %>%
  count(name = "all_countries") %>%
  pull() # Fetch the value from the dataset, creating a vector

costofliving %>%
  group_by(country) %>%
  count(name = "country_n") %>%
  ungroup() %>%
  mutate(country_percentage = country_n/all_countries * 100) %>% # Make a new variable calculating the
  arrange(desc(country_percentage))
```

```
## # A tibble: 162 x 3
##    country           country_n country_percentage
##    <fct>                 <int>              <dbl>
##  1 " India"                 45               7.79
##  2 " United Kingdom"        34               5.88
##  3 "NA"                     29               5.02
##  4 " Canada"                26               4.50
##  5 " Germany"               25               4.33
##  6 " Italy"                 19               3.29
##  7 " Netherlands"           13               2.25
##  8 " Spain"                 13               2.25
##  9 " CA"                    12               2.08
## 10 " Poland"                 9               1.56
## # ... with 152 more rows
```

Displaying descriptive statistics

Sometimes, we might want to display several statistical measures at the same time. We can display the mean and the standard deviation together using `summarise`.

```
costofliving %>%
  summarise(cofi_mean = mean(cofi, na.rm = TRUE),
            cofi_sd = sd(cofi, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   cofi_mean cofi_sd
##       <dbl>   <dbl>
## 1      57.4    21.6
```

And we can combine several `tidyverse` functions such as `filter`, `select`, `group_by`, `summarise` and `rename` to get a table showing some central tendencies and variation on variables and units that are important for our analysis.

```
cofi_table <- costofliving %>%
  filter(country %in% c(" United Kingdom", " Bulgaria", " Brazil", " Japan", " India")) %>% # Picking o
  select(country, cofi) %>% # Fetching only the variables select and cofi
  na.omit() %>% # Removing missing variables from the data
  group_by(country) %>% # Finding the next summary statistics per country
  summarise(cofi_mean = mean(cofi, na.rm = TRUE), # Finding the mean of cost of living (per country)
            cofi_sd = sd(cofi, na.rm = TRUE)) %>% # Finding the standard deviation of cost of living (p
  rename("Country" = country, # Renaming the variables into something more human readable
         "Average cost of living" = cofi_mean,
         "Spread around cost of living" = cofi_sd)

cofi_table
```

```
## # A tibble: 5 x 3
##   Country           `Average cost of living` `Spread around cost of living`
##   <fct>                                <dbl>                          <dbl>
## 1 " United Kingdom"                     71.0                           4.98
## 2 " Japan"                              79.3                           5.48
## 3 " Bulgaria"                           39.2                           2.27
## 4 " Brazil"                             34.1                           2.24
## 5 " India"                              25.1                           2.28
```

If we would like to include this table in a `pdf` or `html` report, we can use the `knitr` package and the `kableExtra` package to wrap the table into a neat table-structure. The base functions in these packages is `kable` to make a pdf- or html-table, and `kable_styling` to make it a bit more beautiful. Here, I add the argument `latex_options = "HOLD_position"` because it stops the table from jumping around in the final pdf file.

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 4.1.3
```

```
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
cofi_table %>%
  kable() %>% # Make html or pdf table
  kable_styling(latex_options = "HOLD_position") # Start styling the table
```

Country	Average cost of living	Spread around cost of living
United Kingdom	71.04067	4.977239
Japan	79.31333	5.480240
Bulgaria	39.15500	2.273756
Brazil	34.09000	2.242008
India	25.09465	2.279804

There are many ways to style your table so that it becomes the way you want it to, for example as shown in the code below. If you want to learn more, have a look at this page. Here, I also round the variable to two decimals before I make a beautiful table out of them, using `mutate` and `round`.

```
cofi_table %>%
  mutate(`Average cost of living` = round(`Average cost of living`, 2), # Rounding the variable into tw
         `Spread around cost of living` = round(`Spread around cost of living`, 2)) %>%
  kable() %>%
  kable_classic(full_width = FALSE, # Make the table fill the page with theme "classic"
                html_font = "Cambria", # Give a certain font to the page
                font_size = 12, # Set font size to 12
                latex_options = "HOLD_position") %>% # Keep the table in this spot when making the pdf
  row_spec(0, color = "blue") %>% # Make the text in the top row blue
  column_spec(1, bold = TRUE, # Set the first column to bold
              border_right = TRUE) # Add a vertical line at the first column
```

Country	Average cost of living	Spread around cost of living
United Kingdom	71.04	4.98
Japan	79.31	5.48
Bulgaria	39.16	2.27
Brazil	34.09	2.24
India	25.09	2.28

Joining datasets

It is seldom the case that all the data we need exists in one dataset from the beginning. Usually, we have to put together several datasets to get all the variables we want in *one* dataset. This process is called *joining*

dataset (also known as *merging*). We'll learn about `left_join`, `right_join`, `inner_join` and `full_join` to join datasets. However, joining datasets can be quite tricky. Sometimes, the codes might not run and it can be difficult to understand why, and sometimes, the code might run and the result becomes utterly wrong. We need to have a good overview of what exactly the two datasets show. Therefore, before we do the joining, we'll have a look at the necessary steps first.

Which variables do you need?

We'll join the `costofliving` dataset with a dataset on welfare institutions. You can find the dataset here and the codebook here. Because the dataset has 400 variables to begin with, it's a good idea to use the codebook to check which variables you might need and `select` these into a smaller dataset.

- **id**: The country variable in three letters (e.g. Norway becomes NOR)
- **year**: The relevant year of the observation
- **mgini**: Measure of income inequality before taxes and redistribution[1].
- **ngini**: Measure of income inequality after taxes and redistribution[2]
- **rred**: Relative redistribution[3]
- **lowpay**: Incidence of low pay[4].

The dataset is in `.xlsx` format. To read in this type of data, we can load the package `readxl` and use the function `read_excel`. Then we can use `select` to fetch the variables we want and make a smaller dataset (often called a *subset*).

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.1.3
```

```
welfare <- read_excel("../../datafolder/CWS-data-2020.xlsx")

welfare_subset <- welfare %>%
  select(id, year, mgini, ngini, rred, lowpay)
```

What will be the units in your final dataset?

Aggregation level

When we join datasets, what we want to to **add variables to the units we already have**. So, in this case, we would like to add variables on redistribution to the cost of living in different cities. But there's a problem. Even though *city* is our smallest units in the `costofliving` dataset, it is not present in the `welfare_subset` dataset. Here, the smallest units are countries.

```
glimpse(costofliving)
```

```
## Rows: 578
## Columns: 8
```

[1]Market (Pre-Tax-and-Transfer) GINI Coefficient. Household income, whole population.
[2]Net (Post-Tax-and-Transfer) GINI Coefficient Household income, whole population.
[3]Relative Redistribution; market-income inequality minus net-income inequality, divided by market-income inequality. Household income, whole population.
[4]Defined as the percentage of workers earning less than two thirds of the median wage

```
## $ city                            <chr> "Hamilton", "Zurich", "Basel", "Zug", "~
## $ country                         <fct>  Bermuda,  Switzerland, NA,  Switzerlan~
## $ cofi                            <dbl> 149.02, NA, 130.93, 128.13, 123.99, 122~
## $ rent_index                      <dbl> 96.10, 69.26, 49.38, 72.12, 44.99, 59.5~
## $ cost_of_living_plus_rent_index <dbl> 124.22, 102.19, 92.70, 101.87, 86.96, 9~
## $ groceries_index                 <dbl> 157.89, 136.14, 137.07, 132.61, 129.17,~
## $ restaurant_price_index          <dbl> 155.22, 132.52, 130.95, 130.93, 119.80,~
## $ local_purchasing_power_index    <dbl> 79.43, 129.79, 111.53, 143.40, 111.96, ~
```

```
glimpse(welfare_subset)
```

```
## Rows: 1,299
## Columns: 6
## $ id     <chr> "AUL", "AUL", "AUL", "AUL", "AUL", "AUL", "AUL", "AUL", "AUL", ~
## $ year   <dbl> 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 197~
## $ mgini  <dbl> NA, NA, NA, NA, NA, NA, NA, 40.6, 40.6, 40.6, 40.5, 40.5, 40.4,~
## $ ngini  <dbl> NA, NA, NA, NA, NA, NA, NA, 28.5, 28.4, 28.4, 28.3, 28.1, 28.0,~
## $ rred   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 31.~
## $ lowpay <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 11.~
```

Since we cannot break the country variable in the `welfare_subset` dataset down to cities (because we simply do not know which part of the variables belong to which cities), we have to go the other way and aggregate the variables in the `costofliving` dataset up to country level. We can do this through `summarise` and `mean`, as shown below. I use `mean` instead of `sum` because it doesn't make that much sense to add up index values. Now, we find the average cost of living in each country compared to New York, based on their biggest cities.

```
costofliving_agg <- costofliving %>%
  group_by(country) %>% # Find the following statistic per country
  summarise(cofi = mean(cofi, na.rm = TRUE), # The average cost of living based on the biggest cities i
            rent_index = mean(rent_index, na.rm = TRUE),
            groceries_index = mean(groceries_index, na.rm = TRUE),
            restaurant_price_index = mean(restaurant_price_index, na.rm = TRUE),
            local_purchasing_power_index = mean(local_purchasing_power_index, na.rm = TRUE))
```

Alternatively, if you want to be a bit more advanced and do it with less code, you can use `summarise_at` and make a function.

```
costofliving %>%
  group_by(country) %>%
  summarise_at(vars("cofi", "rent_index", "groceries_index", "restaurant_price_index", "local_purchasing
               function(x){mean(x, na.rm = TRUE)}) # This function is equivalent to the one above, but
```

```
## # A tibble: 162 x 6
##    country     cofi rent_index groceries_index restaurant_pric~ local_purchasin~
##    <fct>      <dbl>      <dbl>           <dbl>            <dbl>            <dbl>
##  1 " Bermuda" 149.        96.1           158.             155.              79.4
##  2 " Switzer~ 121.        61.2           125.             126.             124.
##  3 "NA"        61.7       30.0            60.5             57.4             83.7
##  4 " Lebanon" 120.        27.8           141.             117.              15.4
##  5 " Norway"  102.        38.3            97.8            106.              87.1
##  6 " HI"      104.        65.1           115.              94.3             89.2
##  7 " NY"       80.5       49.0            78.6             78.3            106.
```

```
##  8 " Iceland"  97.6       46.3            91.9            106.             74.8
##  9 " Jersey"   96.5      NaN              79.9            110.             80.4
## 10 " CA"       82.1       74.6            85.6             79.8           123.
## # ... with 152 more rows
```

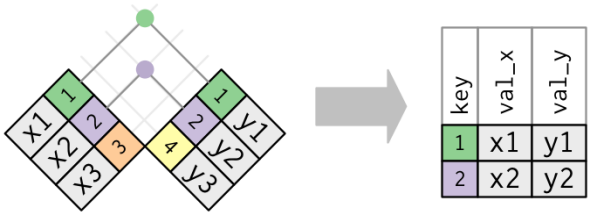
Time aspect

Another thing to consider is the time aspect. Our `welfare_subset` data has data from 1960 to 2018. Our `costofliving` dataset has only data from 2022. In theory, we cannot really merge the datasets then, because we do not know the cost of living in any of the years of the welfare data, and vice versa. However, doing a rough approximation, we can pick a late year in the `welfare_subset` dataset with sufficiently complete values and assume that the variables do not change enough from year to year, so that within a ten-year period, it should correspond more or less.

```
welfare_subset <- welfare_subset %>%
  filter(year == 2015) %>% # Filtering out the value on year that is 2015
  select(-year) # Removing the variable year since it now only has one value, 2015
```

Do the units have the same name?

The next thing to tackle is the so-called *key* of the datasets. The *key* is the variable or variables in the two datasets that are the same. Often, it is for example an id of a person, a country name and/or a time period such as a date or a year. In our case, there's only one shared key between the datasets – country.



The *key* needs to have the same values. Here, we bump into another issue. The variables indicating the countries in the `welfare_subset` and the `costofliving` datasets struggle with two problems to be a shared key: (1) the variables are called different things (id in `welfare_subset` and country in `costofliving`), and (2) they have different labels for the same values (e.g. Australia is AUS in `welfare_subset` but Australia in `costofliving`).

To change this, we could use `case_when` together with `mutate` and recode every value.

```
welfare_subset %>%
  mutate(country = case_when( # Make a new variable called country
    id == "AUS" ~ "Australia", # When the variable id is "AUS", give the country variable value "Austra
    id == "BEL" ~ "Belgium",
    id == "CAN" ~ "Canada"
  ))
```

```
## # A tibble: 22 x 6
##    id    mgini ngini  rred lowpay country
##    <chr> <dbl> <dbl> <dbl>  <dbl> <chr>
##  1 AUL    NA    NA    NA    15.5  <NA>
##  2 AUS    48.3  27.8  42.4  15.9  Australia
##  3 BEL    48.7  25.7  47.2   4.6  Belgium
```

```
##  4 CAN    46    30.9  32.8  22.2  Canada
##  5 DEN    49.1  26.3  46.4   8.24 <NA>
##  6 FIN    48.1  25.5  47     7.77 <NA>
##  7 FRA    49.2  29.5  40    NA    <NA>
##  8 FRG    52.3  29.3  44    19.4  <NA>
##  9 GRE    51.6  33.7  34.7  15.9  <NA>
## 10 IRE    51.2  29.8  41.8  24    <NA>
## # ... with 12 more rows
```

This is a bit time consuming, so we'll use a shortcut here through the `countrycode` package. This package gives a standardized framework to recode country variables into new values (as you might understand, different values on country variables has been a pain for many joining problems).

```
library(countrycode)
```

```
## Warning: package 'countrycode' was built under R version 4.1.3
```

```
welfare_subset <- welfare_subset %>%
  mutate(country = countrycode(sourcevar = id, # Use the id variable to find the country name
                               origin = "iso3c", # The inital variable has three letters for countrie
                               destination = "country.name")) # Recode these into full country names
```

```
## Warning in countrycode_convert(sourcevar = sourcevar, origin = origin, destination = dest, : Some va
```

The warning message tells us that `R` was unable to translate all the values. Usually, we might want to look further into this and perhaps manually recode these variables (e.g. using `case_when`). However, since I won't use this data in any analysis today, I'll simply proceed to the joining.

One last thing that need to be done, is to remove whitespace from the `country` variable in `costofliving`. We'll look more into these things in week 3.

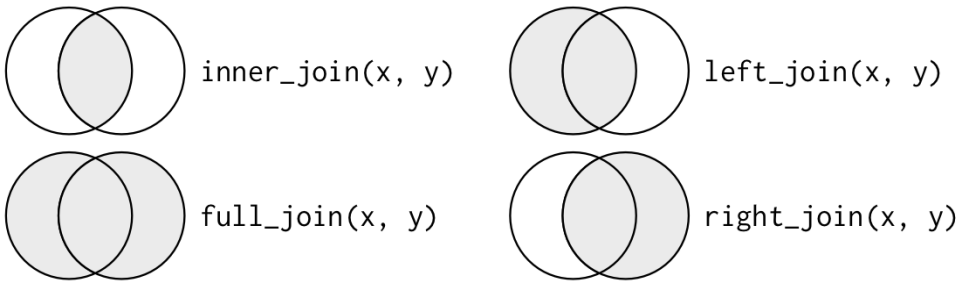
```
costofliving_agg <- costofliving_agg %>%
  mutate(country = str_squish(country))
```

What kind of join do you need?

There are four main types of joins in `R`:

- `inner_join`
- `left_join`
- `right_join`
- `full_join`

Which one to use depends on which rows it is important to keep onwards. If you only want the units that match perfectly on the key(s) of the two datasets, use `inner_join`. If you want to keep all units in the left dataset, use `left_join`, and if it is more important to maintain the units in the right dataset, use `right_join`. `full_join` keeps all units.



The `left_join` function is the most used, possibly because it's more intuitive. You start with a dataset, keep all the rows, and add variables to the rows you have. Here, I use `left_join`, keeping all the units in the `costofliving_agg` dataset. The *key* is `country` – to specify this, add `by = "country"`.

```
costofliving_agg %>%
  left_join(welfare_subset, by = "country")
```

```
## # A tibble: 162 x 11
##    country     cofi rent_index groceries_index restaurant_pric~ local_purchasin~
##    <chr>      <dbl>      <dbl>           <dbl>            <dbl>            <dbl>
##  1 Bermuda    149.        96.1           158.             155.              79.4
##  2 Switzerla~ 121.        61.2           125.             126.             124.
##  3 NA          61.7       30.0            60.5             57.4             83.7
##  4 Lebanon    120.        27.8           141.             117.              15.4
##  5 Norway     102.        38.3            97.8            106.              87.1
##  6 HI         104.        65.1           115.             94.3              89.2
##  7 NY          80.5       49.0            78.6             78.3             106.
##  8 Iceland     97.6       46.3            91.9            106.              74.8
##  9 Jersey      96.5      NaN              79.9            110.              80.4
## 10 CA          82.1       74.6            85.6             79.8             123.
## # ... with 152 more rows, and 5 more variables: id <chr>, mgini <dbl>,
## #   ngini <dbl>, rred <dbl>, lowpay <dbl>
```

Many `NA` are generated from using `left_join`. That's because values on the country variable such as `NY` and `HI` are not shared between the two datasets. What we get is 162 rows with lots of missing. If we use a `right_join`, we lose many of these rows – now we only have 22 – but they mostly have full data.

```
costofliving_agg %>%
  right_join(welfare_subset, by = "country")
```

```
## # A tibble: 22 x 11
##    country     cofi rent_index groceries_index restaurant_pric~ local_purchasin~
##    <chr>      <dbl>      <dbl>           <dbl>            <dbl>            <dbl>
##  1 Norway     102.        38.3            97.8            106.              87.1
##  2 Japan       79.3       43.2            86.6             45.6             75.8
##  3 France      77.7       31.1            77.2             77.4             82.3
##  4 Australia   77.5       40.3            76.9             74.4            106.
##  5 Luxembourg  83.0       63.4            75.8             95.2            100.
##  6 Finland     76.1       23.2            68.8             84.6             92.6
##  7 New Zeala~  75.4       36.1            74.2             72.0             92.5
##  8 Sweden      75.5       31.6            68.8             77.5             94.1
##  9 Canada      71.4       37.3            70.5             69.8             93.1
## 10 Italy       67.5       23.1            60.3             68.9             59.4
```

```
## # ... with 12 more rows, and 5 more variables: id <chr>, mgini <dbl>,
## #   ngini <dbl>, rred <dbl>, lowpay <dbl>
```

Using `inner_join` keeps only the rows where the values can be matched between the two datasets. Now, we have lots of information (less missing), but only 11 rows.

```
costofliving_agg %>%
  inner_join(welfare_subset, by = "country")
```

```
## # A tibble: 11 x 11
##    country     cofi rent_index groceries_index restaurant_pric~ local_purchasin~
##    <chr>      <dbl>      <dbl>           <dbl>            <dbl>            <dbl>
##  1 Norway     102.        38.3            97.8            106.              87.1
##  2 Japan       79.3       43.2            86.6             45.6             75.8
##  3 France      77.7       31.1            77.2             77.4             82.3
##  4 Australia   77.5       40.3            76.9             74.4            106.
##  5 Luxembourg  83.0       63.4            75.8             95.2            100.
##  6 Finland     76.1       23.2            68.8             84.6             92.6
##  7 New Zeala~  75.4       36.1            74.2             72.0             92.5
##  8 Sweden      75.5       31.6            68.8             77.5             94.1
##  9 Canada      71.4       37.3            70.5             69.8             93.1
## 10 Italy       67.5       23.1            60.3             68.9             59.4
## 11 Belgium     71.9       28.8            62.1             76.6             84.9
## # ... with 5 more variables: id <chr>, mgini <dbl>, ngini <dbl>, rred <dbl>,
## #   lowpay <dbl>
```

Last, `full_join` keeps all units, matching the ones it can match and adding missing values to the rest. This dataset contains 173 rows[5].

```
costofliving_agg %>%
  full_join(welfare_subset, by = "country")
```

```
## # A tibble: 173 x 11
##    country     cofi rent_index groceries_index restaurant_pric~ local_purchasin~
##    <chr>      <dbl>      <dbl>           <dbl>            <dbl>            <dbl>
##  1 Bermuda    149.        96.1           158.             155.              79.4
##  2 Switzerla~ 121.        61.2           125.             126.             124.
##  3 NA          61.7       30.0            60.5             57.4             83.7
##  4 Lebanon    120.        27.8           141.             117.              15.4
##  5 Norway     102.        38.3            97.8            106.              87.1
##  6 HI         104.        65.1           115.              94.3             89.2
##  7 NY          80.5       49.0            78.6             78.3            106.
##  8 Iceland     97.6       46.3            91.9            106.              74.8
##  9 Jersey      96.5      NaN              79.9            110.              80.4
## 10 CA          82.1       74.6            85.6             79.8            123.
## # ... with 163 more rows, and 5 more variables: id <chr>, mgini <dbl>,
## #   ngini <dbl>, rred <dbl>, lowpay <dbl>
```

[5]This is composed of 162 rows from the `costofliving_agg` dataset plus 22 rows from the `welfare_subset` dataset, minus the 11 rows that they share

left_join()

right_join()

inner_join()

full_join()

semi_join()

anti_join()