Introduction to data analysis and data transformation

Day 3

Solveig Bjørkholt

29. June

Plan for today

What we will learn today:

- Data import
- Understanding the dataframe
- Tidyverse functions
  - select
  - filter
  - rename
  - mutate
  - group_by and summarise
  - separate and unite

What does data analysis entail?

Data analysis is all about using data to explore and test assumptions, trying to find patterns that can lead us to answers. Within political science, we tend to distinguish between two major modes of methodological approach:

- Qualitative
- Quantitative

Qualitative approaches rely on case studies, document analysis and interviews. They often have few cases (for example one or two countries) and study these in detail. Being able to go in depth on specific cases, these methods have some significant advantages. However, since they are limited in scope (few cases), it is hard to say something about whether what we find is transferable to other cases. Imagine you study whether people tend to vote equally to their parents. You could interview a few people and get some detailed information on the hows and whys, but you wouldn't know whether the findings reach more generally. Using quantitative techniques, however, you could study thousands of people at the same time and see if the finding is large and consistent enough to be a general pattern.

There are three main ways of conducting quantitative analyses:

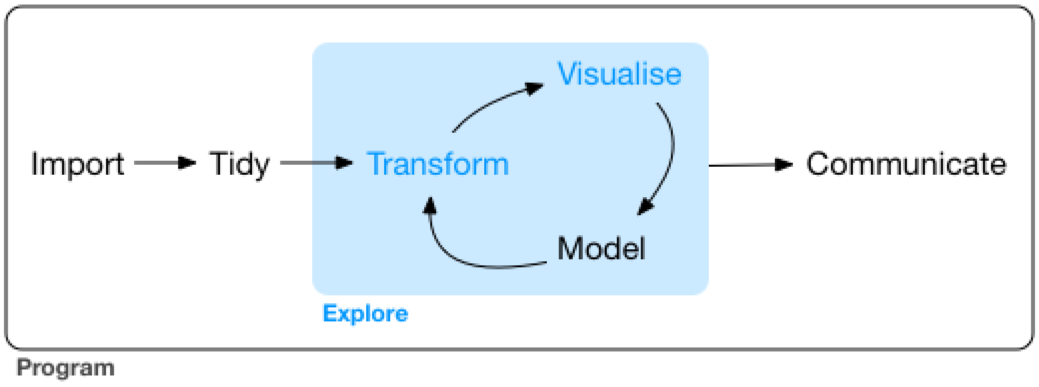
- Descriptive
- Causal
- Predictive

Descriptive analysis is all about finding sums, averages, variation and distributions in data. It involves descriptive statistics, but also visualization. Roughly, the first three weeks in this course will be dedicated to descriptive analysis.

Predictive analysis is all about trying to say something about how something works or might develop given data we do not have. As the name implies, it can often entail trying to say something about the future using data from previous years. We'll look at this during our last two weeks of machine learning.

In this course, we'll not touch upon causal analysis. These methods use data to figure out whether A leads to B – i.e. whether something is a *cause* of something else.

How these different categories link together will hopefully become clearer during the next weeks. We'll be working a lot with `R`, so it's useful to focus on the general work flow of work in `R` regardless of whether we're doing descriptive, causal or predictive analysis. This work flow is shown in the figure below.



Data import

The first stage of the work is to import data to R. We will look more at some ways of gathering data from the databases, the web or API in week 2, but for now, let's look at how to import regular data stored in `.csv`, `.RData`, `.rda` or `.rds`.

First, recall that we store things in objects when working in R. So give your data a name, add an arrow `<-`, and then write a function to read in the data. The function you need depends on the format the data is stored in.

- data**.csv** : `read_csv`
- data**.rds** : `read_rds`
- data**.RData** : `load`
- data**.rda** : `load`

End with the folder path and the name of the dataset in quotation marks, `""`.

Here, we'll work with the dataset `costofliving`, a dataset that tracks the cost of living by cities in 2022, gathered from this website. On the website, the dataset is stored in `.csv`. I show here what the code would be if the data was stored in `.rds`, `rda` and `.RData`.

The functions are from the `tidyverse` package. Therefore, load this package in using `library`. I also transform the dataset to a `tibble`, which is the name of the type of dataframe used in the `tidyverse`.

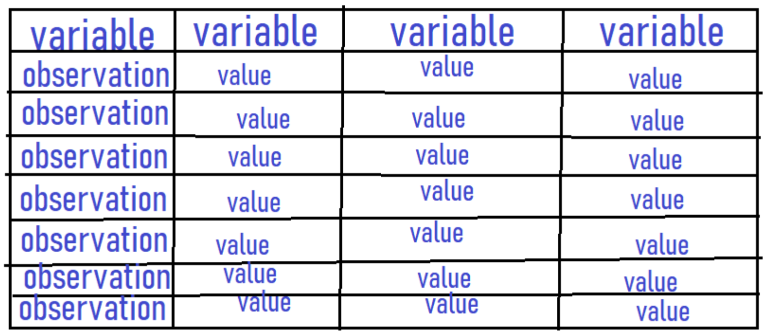
```r
library(tidyverse)

costofliving <- read_csv("../../datafolder/costofliving.csv")
costofliving <- read_rds("../../datafolder/costofliving.rds")

load("../../datafolder/costofliving.RData")

load("../../datafolder/costofliving.rda")

costofliving <- as_tibble(costofliving)
```

The dataframe

So what does the data look like when we're doing quantitative analysis? Usually, we want the dataset to be in a *dataframe*. A dataframe has three components:

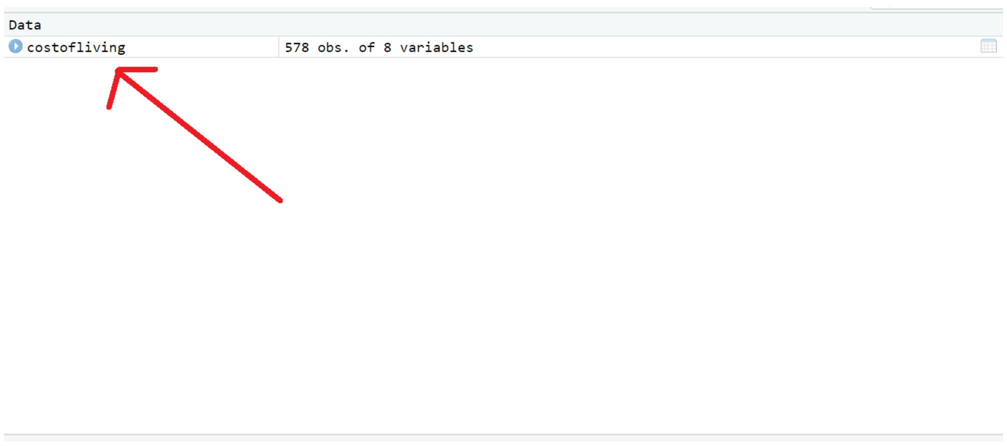
1. Rows: Observations
2. Columns: Variables
3. Cells: Values

The observations are the entities we have in our dataset, for example countries, people, time period, or likewise. Variables are the properties of our entities, for example size or type. The values are the mix between these – the specific property that this specific entity has.



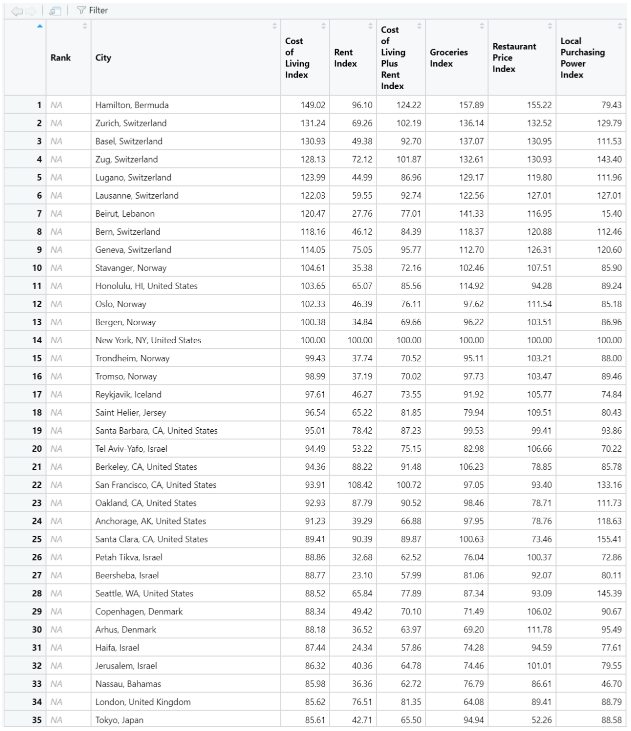
variable	variable	variable	variable
observation	value	value	value
observation	value	value	value
observation	value	value	value
observation	value	value	value
observation	value	value	value
observation	value	value	value
observation	value	value	value

To see what our dataset looks like, you can click on the name of the dataset in the upper right corner of RStudio.



```
Data
costofliving                     578 obs. of 8 variables
```

This dataset looks like below. The `City` variable denotes the observation – the city and the corresponding country. The other variables denote properties of these cities, `cost of living`, `rent index` and so on. There is also a column named `Rank` containing `NA`, which means *missing*.



	Rank	City	Cost of Living Index	Rent Index	Cost of Living Plus Rent Index	Groceries Index	Restaurant Price Index	Local Purchasing Power Index
1	NA	Hamilton, Bermuda	149.02	96.10	124.22	157.89	155.22	79.43
2	NA	Zurich, Switzerland	131.24	69.26	102.19	136.14	132.52	129.79
3	NA	Basel, Switzerland	130.93	49.38	92.70	137.07	130.95	111.53
4	NA	Zug, Switzerland	128.13	72.12	101.87	132.61	130.93	143.40
5	NA	Lugano, Switzerland	123.99	44.99	86.96	129.17	119.80	111.96
6	NA	Lausanne, Switzerland	122.03	59.55	92.74	122.56	127.01	127.01
7	NA	Beirut, Lebanon	120.47	27.76	77.01	141.33	116.95	15.40
8	NA	Bern, Switzerland	118.16	46.12	84.39	118.37	120.88	112.46
9	NA	Geneva, Switzerland	114.05	75.05	95.77	112.70	126.31	120.60
10	NA	Stavanger, Norway	104.61	35.38	72.16	102.46	107.51	85.90
11	NA	Honolulu, HI, United States	103.65	65.07	85.56	114.92	94.28	89.24
12	NA	Oslo, Norway	102.33	46.39	76.11	97.62	111.54	85.18
13	NA	Bergen, Norway	100.38	34.84	69.66	96.22	103.51	86.96
14	NA	New York, NY, United States	100.00	100.00	100.00	100.00	100.00	100.00
15	NA	Trondheim, Norway	99.43	37.74	70.52	95.11	103.21	88.00
16	NA	Tromso, Norway	98.99	37.19	70.02	97.73	103.47	89.46
17	NA	Reykjavik, Iceland	97.61	46.27	73.55	91.92	105.77	74.84
18	NA	Saint Helier, Jersey	96.54	65.22	81.85	79.94	109.51	80.43
19	NA	Santa Barbara, CA, United States	95.01	78.42	87.23	99.53	99.41	93.86
20	NA	Tel Aviv-Yafo, Israel	94.49	53.22	75.15	82.98	106.66	70.22
21	NA	Berkeley, CA, United States	94.36	88.22	91.48	106.23	78.85	85.78
22	NA	San Francisco, CA, United States	93.91	108.42	100.72	97.05	93.40	133.16
23	NA	Oakland, CA, United States	92.93	87.79	90.52	98.46	78.71	111.73
24	NA	Anchorage, AK, United States	91.23	39.29	66.88	97.95	78.76	118.63
25	NA	Santa Clara, CA, United States	89.41	90.39	89.87	100.63	73.46	155.41
26	NA	Petah Tikva, Israel	88.86	32.68	62.52	76.04	100.37	72.86
27	NA	Beersheba, Israel	88.77	23.10	57.99	81.06	92.07	80.11
28	NA	Seattle, WA, United States	88.52	65.84	77.89	87.34	93.09	145.39
29	NA	Copenhagen, Denmark	88.34	49.42	70.10	71.49	106.02	90.67
30	NA	Arhus, Denmark	88.18	36.52	63.97	69.20	111.78	95.49
31	NA	Haifa, Israel	87.44	24.34	57.86	74.28	94.59	77.61
32	NA	Jerusalem, Israel	86.32	40.36	64.78	74.46	101.01	79.55
33	NA	Nassau, Bahamas	85.98	36.36	62.72	76.79	86.61	46.70
34	NA	London, United Kingdom	85.62	76.51	81.35	64.08	89.41	88.79
35	NA	Tokyo, Japan	85.61	42.71	65.50	94.94	52.26	88.58

Another way of getting a look of the data in R is to use the function `glimpse`.

```
glimpse(costofliving)
```

```
## Rows: 578
## Columns: 7
## $ city                           <chr> "Hamilton, Bermuda", "Zurich, Switzerla~
## $ cost_of_living_index           <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, ~
```

tidyverse functions

After having imported the data into R, there is usually a bit of tidying that has to be done. We are going to learn how to do this through the tidyverse package. This package is fundamental to many operations in R. One of the main components of the tidyverse package is the %>%, also known as the pipe. The pipe allows us to access an object and apply functions to it in sequence. For example, if the object is me, the first function could be to wake_up(), then to the me that has woken up, shower(), and then to the me who has woken up and taken a shower, to drink_coffee(). This would look like below:

```
me %>%
  wake_up() %>%
  shower() %>%
  drink_coffee()
```

We'll go through some of the main functions in the tidyverse package. However, first, it can be useful to change the names of the variables to something that does not have space, as it eases the work with the variables. To do this, load the package janitor and use the function clean_names. Here, we apply the pipe to the function, which basically means; go into the object costofliving and use the function clean_names on it.

```
costofliving
```

```
## # A tibble: 578 x 7
##    city               cost_of_living_~ rent_index cost_of_living_~ groceries_index
##    <chr>                         <dbl>      <dbl>            <dbl>           <dbl>
##  1 Hamilton, Bermu~              149.        96.1            124.             158.
##  2 Zurich, Switzer~               NA         69.3            102.             136.
##  3 Basel,NA                      131.        49.4             92.7            137.
##  4 Zug, Switzerland              128.        72.1            102.             133.
##  5 Lugano, Switzer~              124.        45.0             87.0            129.
##  6 Lausanne, Switz~              122.        59.6             92.7            123.
##  7 Beirut, Lebanon               120.        27.8             77.0            141.
##  8 Bern, Switzerla~              118.        46.1             84.4            118.
##  9 Geneva, Switzer~              114.        75.0             95.8            113.
## 10 Stavanger, Norw~              105.        35.4             72.2            102.
## # ... with 568 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

```
library(janitor)

costofliving <- costofliving %>%
  clean_names()

costofliving
```

```
## # A tibble: 578 x 7
##    city               cost_of_living_~ rent_index cost_of_living_~ groceries_index
##    <chr>                         <dbl>      <dbl>            <dbl>           <dbl>
##  1 Hamilton, Bermu~              149.        96.1            124.             158.
##  2 Zurich, Switzer~               NA         69.3            102.             136.
##  3 Basel,NA                      131.        49.4             92.7            137.
##  4 Zug, Switzerland              128.        72.1            102.             133.
##  5 Lugano, Switzer~             124.       45.0              87.0              129.
##  6 Lausanne, Switz~             122.       59.6              92.7              123.
##  7 Beirut, Lebanon              120.       27.8              77.0              141.
##  8 Bern, Switzerla~             118.       46.1              84.4              118.
##  9 Geneva, Switzer~             114.       75.0              95.8              113.
## 10 Stavanger, Norw~             105.       35.4              72.2              102.
## # ... with 568 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

select

The function `select` picks out some variables from the dataset.

```
dataset %>%
  select(variable)
```

Recall that the dataset had 8 variables; `rank`, `city`, `cost_of_living_index`, `rent_index`, `cost_of_living_plus_rent_inde` `groceries_index`, `restaurant_price_index` and `local_purchasing_power_index`. If we only want a dataset with the two variables `cost_of_living_index` and `groceries_index`, we can use the `select` function.

```
costofliving %>%
  select(cost_of_living_index, groceries_index) # Picking only these two variables
```

```
## # A tibble: 578 x 2
##    cost_of_living_index groceries_index
##                   <dbl>           <dbl>
##  1                 149.            158.
##  2                  NA             136.
##  3                 131.            137.
##  4                 128.            133.
##  5                 124.            129.
##  6                 122.            123.
##  7                 120.            141.
##  8                 118.            118.
##  9                 114.            113.
## 10                 105.            102.
## # ... with 568 more rows
```

It can also be used to remove variables that we do not want in the dataset. If we want to remove more than one variable, we have to add a `c` and wrap the variables in parantheses.

```
costofliving %>%
  select(-cost_of_living_index)
```

```
## # A tibble: 578 x 6
##    city             rent_index cost_of_living_~ groceries_index restaurant_pric~
##    <chr>                 <dbl>            <dbl>           <dbl>            <dbl>
##  1 Hamilton, Bermu~       96.1             124.            158.             155.
##  2 Zurich, Switzer~       69.3             102.            136.             133.
##  3 Basel,NA               49.4              92.7           137.             131.
##  4 Zug, Switzerland       72.1            102.              133.              131.
##  5 Lugano, Switzer~       45.0             87.0             129.              120.
##  6 Lausanne, Switz~       59.6             92.7             123.              127.
##  7 Beirut, Lebanon        27.8             77.0             141.              117.
##  8 Bern, Switzerla~       46.1             84.4             118.              121.
##  9 Geneva, Switzer~       75.0             95.8             113.              126.
## 10 Stavanger, Norw~       35.4             72.2             102.              108.
## # ... with 568 more rows, and 1 more variable:
## #   local_purchasing_power_index <dbl>
```

```
costofliving %>%
  select(-c(cost_of_living_index, groceries_index))
```

```
## # A tibble: 578 x 5
##    city             rent_index cost_of_living_~ restaurant_pric~ local_purchasin~
##    <chr>                 <dbl>            <dbl>            <dbl>            <dbl>
##  1 Hamilton, Berm~        96.1            124.             155.              79.4
##  2 Zurich, Switze~        69.3            102.             133.             130.
##  3 Basel,NA               49.4             92.7            131.             112.
##  4 Zug, Switzerla~        72.1            102.             131.             143.
##  5 Lugano, Switze~        45.0             87.0            120.             112.
##  6 Lausanne, Swit~        59.6             92.7            127.             127.
##  7 Beirut, Lebanon        27.8             77.0            117.              15.4
##  8 Bern, Switzerl~        46.1             84.4            121.             112.
##  9 Geneva, Switze~        75.0             95.8            126.             121.
## 10 Stavanger, Nor~        35.4             72.2            108.              85.9
## # ... with 568 more rows
```

filter

The function `filter` picks out some observations from the dataset.

```
dataset %>%
  filter(variable_name == value)
```

Curious about which cities that have a cost of living below 100 (meaning that they are cheaper than New York)? Ask for all units that score *less than* `<` 100 on the `cost_of_living_index`.

```
costofliving %>%
  filter(cost_of_living_index < 100)
```

```
## # A tibble: 535 x 7
##    city              cost_of_living_~ rent_index cost_of_living_~ groceries_index
##    <chr>                        <dbl>      <dbl>            <dbl>           <dbl>
##  1 Tromso, Norway                99.0       37.2             70.0            97.7
##  2 Reykjavik, Icel~              97.6       46.3             73.6            91.9
##  3 Saint Helier, J~              96.5       NA               81.8            79.9
##  4 Santa Barbara, ~              95.0       78.4             87.2            99.5
##  5 Tel Aviv-Yafo, ~              94.5       53.2             75.2            83.0
##  6 Berkeley, CA                  94.4       88.2             91.5           106.
##  7 San Francisco, ~              93.9      108.             101.             97.0
##  8 Oakland, CA                   92.9       87.8              90.5            98.5
##  9 Anchorage, AK                 91.2       39.3              66.9            98.0
## 10 NA, CA                        89.4       90.4              89.9           101.
## # ... with 525 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

Or if you want all the cities with equal = or higher than > 100 on the `cost_of_living_index`, use the code below.

```
costofliving %>%
  filter(cost_of_living_index >= 100)
```

```
## # A tibble: 13 x 7
##    city              cost_of_living_~ rent_index cost_of_living_~ groceries_index
##    <chr>                        <dbl>      <dbl>            <dbl>           <dbl>
##  1 Hamilton, Bermu~              149.       96.1            124.            158.
##  2 Basel,NA                      131.       49.4             92.7           137.
##  3 Zug, Switzerland              128.       72.1            102.            133.
##  4 Lugano, Switzer~              124.       45.0             87.0           129.
##  5 Lausanne, Switz~              122.       59.6             92.7           123.
##  6 Beirut, Lebanon               120.       27.8             77.0           141.
##  7 Bern, Switzerla~              118.       46.1             84.4           118.
##  8 Geneva, Switzer~              114.       75.0             95.8           113.
##  9 Stavanger, Norw~              105.       35.4             72.2           102.
## 10 Honolulu, HI                  104.       65.1             85.6           115.
## 11 Oslo, Norway                  102.       46.4             76.1            97.6
## 12 Bergen, Norway                100.       34.8             69.7            96.2
## 13 New York, NY                  100       100              100             100
## # ... with 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

Want to know about only Oslo? Then filter out the row on the `city` variable that has the value `Oslo, Norway` (in quotation marks `""`). Notice that we use two equals signs `==`. This is to not confuse `R` into thinking that city equals Oslo, Norway.

```
costofliving %>%
  filter(city == "Oslo, Norway")
```

```
## # A tibble: 1 x 7
##   city         cost_of_living_index rent_index cost_of_living_p~ groceries_index
##   <chr>                       <dbl>      <dbl>             <dbl>           <dbl>
## 1 Oslo, Norway                 102.       46.4              76.1            97.6
## # ... with 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

Want more cities at the same time? Shift `==` with `%in%` and wrap the units into parantheses with a `c` in front. Also, remember the quotation marks `""`.

```
costofliving %>%
  filter(city %in% c("Oslo, Norway",
                     "Bergen, Norway",
                     "Trondheim, Norway"))
```

```
## # A tibble: 3 x 7
##   city              cost_of_living_~ rent_index cost_of_living_~ groceries_index
##   <chr>                        <dbl>      <dbl>            <dbl>           <dbl>
## 1 Oslo, Norway                  102.       46.4             76.1            97.6
## 2 Bergen, Norway                100.       34.8             69.7            96.2
## 3 Trondheim, Norway              NA        37.7             70.5            95.1
## # ... with 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

rename

Would you like to change the name of a variable? Use `rename`. First, write the new name of the variable, then the name of the old variable.

```
dataset %>%
  rename(new_variable_name = old_variable_name)
```

In this case, I change the name an abbreviation of the variable name, so that `cost_of_living_index` becomes `cofi`.

```
costofliving <- costofliving %>%
  rename(cofi = cost_of_living_index)

costofliving
```

```
## # A tibble: 578 x 7
##    city        cofi rent_index cost_of_living_~ groceries_index restaurant_pric~
##    <chr>      <dbl>      <dbl>            <dbl>           <dbl>            <dbl>
##  1 Hamilton,~  149.       96.1            124.             158.             155.
##  2 Zurich, S~   NA        69.3            102.             136.             133.
##  3 Basel,NA    131.       49.4             92.7            137.             131.
##  4 Zug, Swit~  128.       72.1            102.             133.             131.
##  5 Lugano, S~  124.       45.0             87.0            129.             120.
##  6 Lausanne,~  122.       59.6             92.7            123.             127.
##  7 Beirut, L~  120.       27.8             77.0            141.             117.
##  8 Bern, Swi~  118.       46.1             84.4            118.             121.
##  9 Geneva, S~  114.       75.0             95.8            113.             126.
## 10 Stavanger~  105.       35.4             72.2            102.             108.
## # ... with 568 more rows, and 1 more variable:
## #   local_purchasing_power_index <dbl>
```

separate and unite

Separate and unite allows us to split or collect two variables based on a separator.

```
dataset %>%
  separate(variable_name, into = c("new_variable_name1", "new_variable_name2"), sep = "separator")

dataset %>%
  unite(new_variable_name, c("old_variable_name1", "old_variable_name2"), sep = "separator")
```

A useful thing to do here would be to separate the `city` variable into `city` and `country`. The separator in this case is a comma `,`, but it could also have been for example a dot, a space or a word such as "and".

```
costofliving <- costofliving %>%
  separate(city, into = c("city", "country"), sep = ",")

costofliving
```

```
## # A tibble: 578 x 8
##    city      country         cofi rent_index cost_of_living_plu~ groceries_index
##    <chr>     <chr>          <dbl>      <dbl>               <dbl>           <dbl>
##  1 Hamilton  " Bermuda"      149.       96.1                124.            158.
##  2 Zurich    " Switzerland"   NA        69.3                102.            136.
##  3 Basel     "NA"            131.       49.4                 92.7           137.
##  4 Zug       " Switzerland"  128.       72.1                102.            133.
##  5 Lugano    " Switzerland"  124.       45.0                 87.0           129.
##  6 Lausanne  " Switzerland"  122.       59.6                 92.7           123.
##  7 Beirut    " Lebanon"      120.       27.8                 77.0           141.
##  8 Bern      " Switzerland"  118.       46.1                 84.4           118.
##  9 Geneva    " Switzerland"  114.       75.0                 95.8           113.
## 10 Stavanger " Norway"       105.       35.4                 72.2           102.
## # ... with 568 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

mutate

If you would like to make a new variable, `mutate` is your go-to. First, write the name of the new variable, then add an equals sign `=`, and put the operation that you need to create your new variable.

```
dataset %>%
  mutate(variable_name = operation())
```

For example, if we want a new variable with the restaurant purchasing power as a share of the local purchasing power, we can make a new variable where we divide the first by the second.

```
costofliving %>%
  mutate(share_restaurant_purchasing_power = restaurant_price_index/local_purchasing_power_index)
```

```
## # A tibble: 578 x 9
##    city      country         cofi rent_index cost_of_living_plu~ groceries_index
##    <chr>     <chr>          <dbl>      <dbl>               <dbl>           <dbl>
##  1 Hamilton  " Bermuda"      149.       96.1                124.            158.
##  2 Zurich    " Switzerland"   NA        69.3                102.            136.
##  3 Basel     "NA"            131.       49.4                 92.7           137.
##  4 Zug       " Switzerland"  128.       72.1                102.            133.
##  5 Lugano    " Switzerland"  124.       45.0                 87.0           129.
##  6 Lausanne  " Switzerland"  122.       59.6                 92.7           123.
##  7 Beirut    " Lebanon"      120.       27.8                 77.0           141.
##  8 Bern      " Switzerland"  118.       46.1                 84.4           118.
##  9 Geneva    " Switzerland"  114.       75.0                 95.8           113.
## 10 Stavanger " Norway"       105.       35.4                 72.2           102.
## # ... with 568 more rows, and 3 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>, share_restaurant_purchasing_power <dbl>
```

`ifelse`

If you want to make a variable that takes the value 1 when the country is Norway, match it with `ifelse`. The `ifelse` function works like this:

If `condiction`, then give the new variable the given value, if not, give it the other value. In this case, if the variable `country` has value "Norway", give the new variable `norway` value `1`, else give the new variable value `0`.

```
costofliving %>%
  mutate(norway = ifelse(country == "Norway", 1, 0))
```

```
## # A tibble: 578 x 9
##    city      country          cofi rent_index cost_of_living_plu~ groceries_index
##    <chr>     <chr>           <dbl>      <dbl>               <dbl>           <dbl>
##  1 Hamilton  " Bermuda"       149.       96.1               124.            158.
##  2 Zurich    " Switzerland"    NA        69.3               102.            136.
##  3 Basel     "NA"             131.       49.4                92.7           137.
##  4 Zug       " Switzerland"   128.       72.1               102.            133.
##  5 Lugano    " Switzerland"   124.       45.0                87.0           129.
##  6 Lausanne  " Switzerland"   122.       59.6                92.7           123.
##  7 Beirut    " Lebanon"       120.       27.8                77.0           141.
##  8 Bern      " Switzerland"   118.       46.1                84.4           118.
##  9 Geneva    " Switzerland"   114.       75.0                95.8           113.
## 10 Stavanger " Norway"        105.       35.4                72.2           102.
## # ... with 568 more rows, and 3 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>, norway <dbl>
```

You can also overwrite new variables with the function `mutate`. For example, if you aren't happy with the fact that the `country` variable is a "character", you can change it by using the function `as_factor` together with `mutate`. Notice in the function below that the name of the new variable is the same as the name of the old variable, `country`, thus we overwrite the old one.

```
class(costofliving$country)
```

```
## [1] "character"
```

```
costofliving <- costofliving %>%
  mutate(country = as_factor(country))

class(costofliving$country)
```

```
## [1] "factor"
```

`group_by` and `summarise`

At last, we can use the `summarise` function to make summaries of the dataset. This function works very well together with the function `group_by`.

```
dataset %>%
  group_by(variable) %>%
  summarise(new_variable_name = function(variable_name))
```

While `summarise` gives us aggregates of the data, the `group_by` specifies which units the aggregates should be by. For example, if we ask for the `sum` of the cost of living in general in the dataset, we could just use `summarise` directly.

```
costofliving %>%
  summarise(cofi = sum(cofi))
```

```
## # A tibble: 1 x 1
##    cofi
##   <dbl>
## 1    NA
```

A more useful distinction is, however, to look at the cost of living for each country. Then, we `group_by` country first, and then use `summarise` and `sum`. When using `group_by`, remember to `ungroup` afterwards.

```
costofliving %>%
  group_by(country) %>%
  summarise(cofi = sum(cofi)) %>%
  ungroup()
```

```
## # A tibble: 162 x 2
##    country          cofi
##    <fct>           <dbl>
##  1 " Bermuda"     149.
##  2 " Switzerland"  NA
##  3 "NA"            NA
##  4 " Lebanon"     120.
##  5 " Norway"       NA
##  6 " HI"          104.
##  7 " NY"           NA
##  8 " Iceland"      97.6
##  9 " Jersey"       96.5
## 10 " CA"          985.
## # ... with 152 more rows
```

Summarise also works with other descriptive functions such as:

- `min` : Minimum value
- `max` : Maximum value
- `median` : Median
- `mean` : Mean (average)
- `sd` : Standard deviation

So, for example, to find the average cost of living in lagre cities in the respective countries, use `group_by`, `summarise` and `mean`.

```
costofliving %>%
  group_by(country) %>%
  summarise(cofi = mean(cofi)) %>%
  ungroup()
```

```
## # A tibble: 162 x 2
##    country         cofi
##    <fct>          <dbl>
##  1 " Bermuda"     149.
##  2 " Switzerland"  NA
##  3 "NA"            NA
##  4 " Lebanon"     120.
##  5 " Norway"       NA
##  6 " HI"          104.
##  7 " NY"           NA
##  8 " Iceland"      97.6
##  9 " Jersey"       96.5
## 10 " CA"           82.1
## # ... with 152 more rows
```

Piping together

As you saw in the part above, pipes can be put in sequence. This is one of the great advantages of using pipes. Lots of code can be executed in a single "storyline" that says "first do this, then do this, then do this", and so on. Thus `tidyverse` is usually seen as a pedagogical tool because it's supposedly quite inuitive, and the code also becomes very readable. Consider the code below and decide for yourself whether you find it readable.

```
costofliving %>%
  select(city, country, rent_index, restaurant_price_index, groceries_index) %>%
  filter(country %in% c(" Norway", " Germany", " Spain")) %>%
  mutate(total_cost_index = rent_index + restaurant_price_index + groceries_index) %>%
  group_by(country) %>%
  mutate(max_cost = max(total_cost_index)) %>%
  ungroup() %>%
  mutate(highest_city = ifelse(total_cost_index == max_cost, 1, 0)) %>%
  group_by(highest_city) %>%
  summarise(average_rent = mean(rent_index))
```

```
## # A tibble: 3 x 2
##   highest_city average_rent
##          <dbl>        <dbl>
## 1            0         36.3
## 2            1         46.4
## 3           NA         NA
```

Dealing with `NA`

What are those `NA` things that occur in some cells in the dataset? They are called "missing values". Missing values occur when there are values in the dataset that should ideally have been there, but they're not. This could for example be because we didn't find data on that particular thing, somebody refused to give us the information we needed, or because we've done something wrong in the code, generating `NA`.

To see how much missing there is in our dataset, we can use the function `is.na` along with the function `table`. `is.na` gives the value `TRUE` if the cell has a missing value and `FALSE` otherwise, while the function `table` counts the instances in which `TRUE` and `FALSE` occurs. This means, in other words, that we have 212 missing values in our data.

```
costofliving %>%
  is.na() %>%
  table()
```

```
## .
## FALSE  TRUE
##  4476   148
```

Want to remove all rows that have one or more missing values? Use the function `na.omit`.

```
costofliving %>%
  na.omit()
```

```
## # A tibble: 444 x 8
##    city      country         cofi rent_index cost_of_living_plu~ groceries_index
##    <chr>     <fct>          <dbl>      <dbl>               <dbl>           <dbl>
##  1 Hamilton  " Bermuda"      149.       96.1               124.             158.
##  2 Basel     "NA"            131.       49.4                92.7            137.
##  3 Zug       " Switzerland"  128.       72.1               102.             133.
##  4 Lugano    " Switzerland"  124.       45.0                87.0            129.
##  5 Lausanne  " Switzerland"  122.       59.6                92.7            123.
##  6 Beirut    " Lebanon"      120.       27.8                77.0            141.
##  7 Bern      " Switzerland"  118.       46.1                84.4            118.
##  8 Geneva    " Switzerland"  114.       75.0                95.8            113.
##  9 Stavanger " Norway"       105.       35.4                72.2            102.
## 10 Honolulu  " HI"           104.       65.1                85.6            115.
## # ... with 434 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

If you would like to more carefully select which variables you'd like to remove the missing values from, use `drop_na` and choose the variables where the rows with missing values should be removed.

```
costofliving %>%
  drop_na(country, city, cofi)
```

```
## # A tibble: 548 x 8
##    city      country         cofi rent_index cost_of_living_plu~ groceries_index
##    <chr>     <fct>          <dbl>      <dbl>               <dbl>           <dbl>
##  1 Hamilton  " Bermuda"      149.       96.1               124.             158.
##  2 Basel     "NA"            131.       49.4                92.7            137.
##  3 Zug       " Switzerland"  128.       72.1               102.             133.
##  4 Lugano    " Switzerland"  124.       45.0                87.0            129.
##  5 Lausanne  " Switzerland"  122.       59.6                92.7            123.
##  6 Beirut    " Lebanon"      120.       27.8                77.0            141.
##  7 Bern      " Switzerland"  118.       46.1                84.4            118.
##  8 Geneva    " Switzerland"  114.       75.0                95.8            113.
##  9 Stavanger " Norway"       105.       35.4                72.2            102.
## 10 Honolulu  " HI"           104.       65.1                85.6            115.
## # ... with 538 more rows, and 2 more variables: restaurant_price_index <dbl>,
## #   local_purchasing_power_index <dbl>
```

Saving the dataset

When you've loaded a dataset into `R` and tidied it up a bit, you might want to save it until next time. Which format you choose depends partly on your own preferences, and party what you'd like to do further with the dataset. If you want to send it to someone else, a `.csv` is probably best. If not, I recommend `.rds` because it allows you to give a name to the object when you read it into `R`. But some people prefer `.rda` and `.RData`, and it's nice to vary a bit to see what you like best.

```r
write_csv(costofliving, file = "../../datafolder/costofliving2.csv") # To save a .csv file

saveRDS(costofliving, file = "../../datafolder/costofliving2.rds") # To save a .rds file

save(costofliving, file = "../../datafolder/costofliving2.rda") # To save a .rda file

save(costofliving, file = "../../datafolder/costofliving2.RData") # To save a .RData file
```

About the dataset

These indices are relative to New York City (NYC). Which means that for New York City, each index should be 100(%). If another city has, for example, rent index of 120, it means that on an average in that city rents are 20% more expensive than in New York City. If a city has rent index of 70, that means on average rent in that city is 30% less expensive than in New York City.

Cost of Living Index (Excl. Rent) is a relative indicator of consumer goods prices, including groceries, restaurants, transportation and utilities. Cost of Living Index does not include accommodation expenses such as rent or mortgage. If a city has a Cost of Living Index of 120, it means Numbeo has estimated it is 20% more expensive than New York (excluding rent).

Rent Index is an estimation of prices of renting apartments in the city compared to New York City. If Rent index is 80, Numbeo has estimated that price of rents in that city is on average 20% less than the price in New York.

Groceries Index is an estimation of grocery prices in the city compared to New York City. To calculate this section, Numbeo uses weights of items in the "Markets" section for each city.

Restaurants Index is a comparison of prices of meals and drinks in restaurants and bars compared to NYC.

Cost of Living Plus Rent Index is an estimation of consumer goods prices including rent comparing to New York City.

Local Purchasing Power shows relative purchasing power in buying goods and services in a given city for the average net salary in that city. If domestic purchasing power is 40, this means that the inhabitants of that city with an average salary can afford to buy on an average 60% less goods and services than New York City residents with an average salary.