Tidyverse and dataviz

Introduction

Packages

One of the main advantages that R has as a software for statistic analysis is its incremental syntax. This means that the things you can do in R are constantly updated and expanded through the creation of new **packages**, developed by researchers, users or private companies.

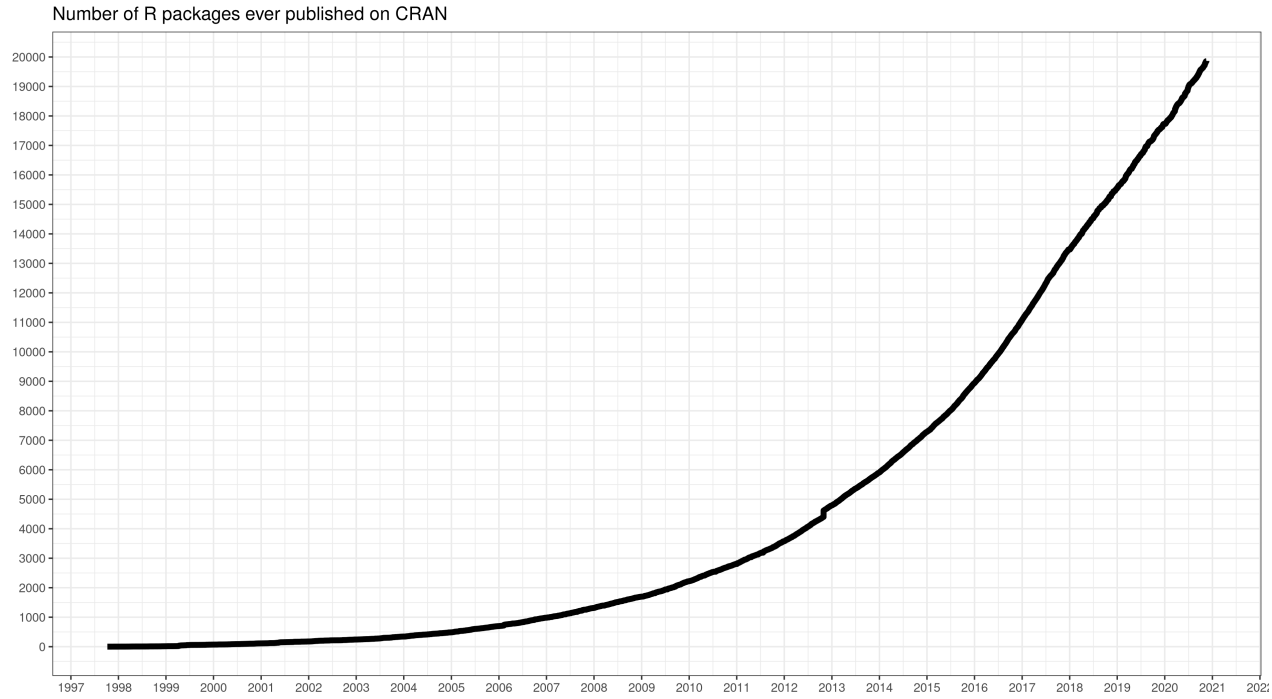


Figure 1: Source: Gergely Daróczi

These packages contain *code*, *data*, and *documentation* in a standardized collection that can be installed by users of R. Most of the time, we will install them in order to use *functions* that will do certain tasks that help us work with our data. So far, we were using functions contained in R base: such as `mean()`, `median()`, `quantile()`, etc. But as we dive deep into the data science life cycle, we might address certain challenges that require more complex, or specific, functions. We will need to import the data, tidy the data into a format that is easy to work with, explore the data, generate visualizations, carry out the analysis and communicate the insights. The tidyverse ecosystem provides a powerful tool for streamlining the workflow in a coherent manner that can be easily connected with other data science tools.

`tidyverse` + Data workflow

The tidyverse is a set of R packages designed for data science. The main idea behind tidyverse is to contain in a single installation line, a set of tools that contain the entire data analysis workflow: from the importation of data to the communication of results.

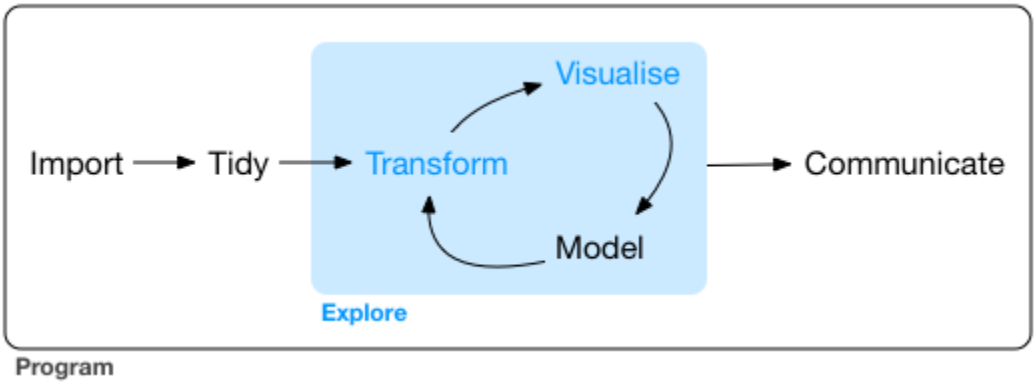
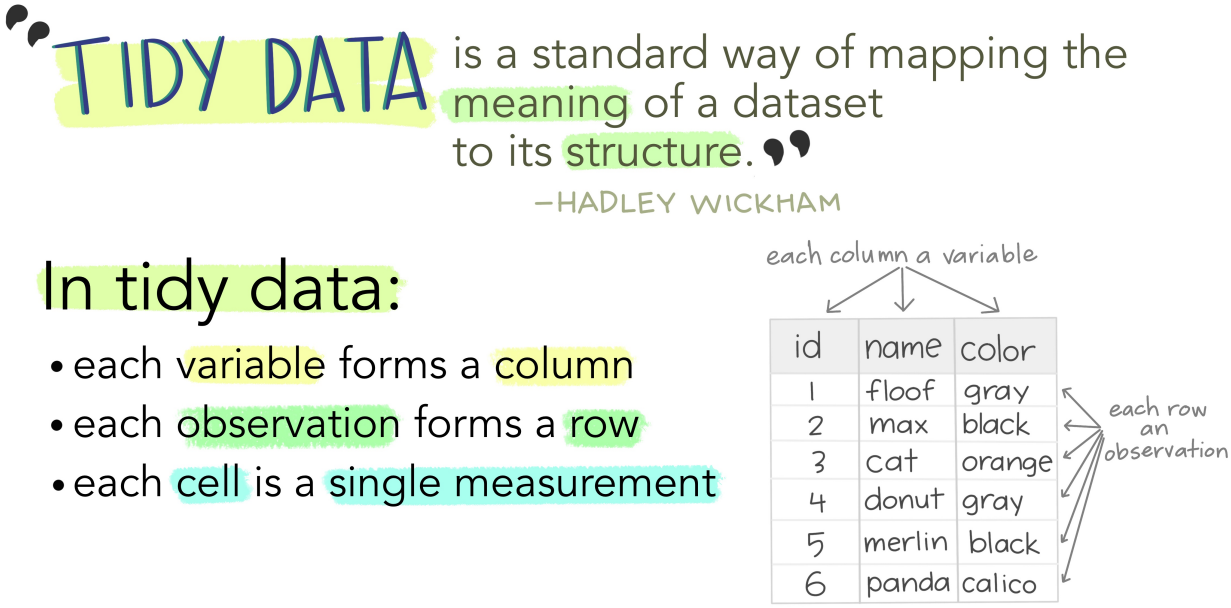


Figure 2: Source: R for Data Science

Tidy data

All packages share an underlying design philosophy, grammar, and data structures. These structures refer to the format of **tidy datasets**. But, what is tidy data? It is a way to describe data that's organized with a particular structure: a rectangular structure, where each variable has its own column and each observation has its own row (Wickham 2014a; Peng n.d.).



Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

According to (Wickham 2014b),

> "Tidy datasets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table."

Working with tidy data means that we work with information that has a consistent data structure. The main benefit behind this is that we will have to spend less time thinking how to process and clean data, because we can use existing tools instead of starting from scratch each time we work with a new dataset. In other words, we only require a small set of tools to be learned, since we can reuse them from one project to the other.

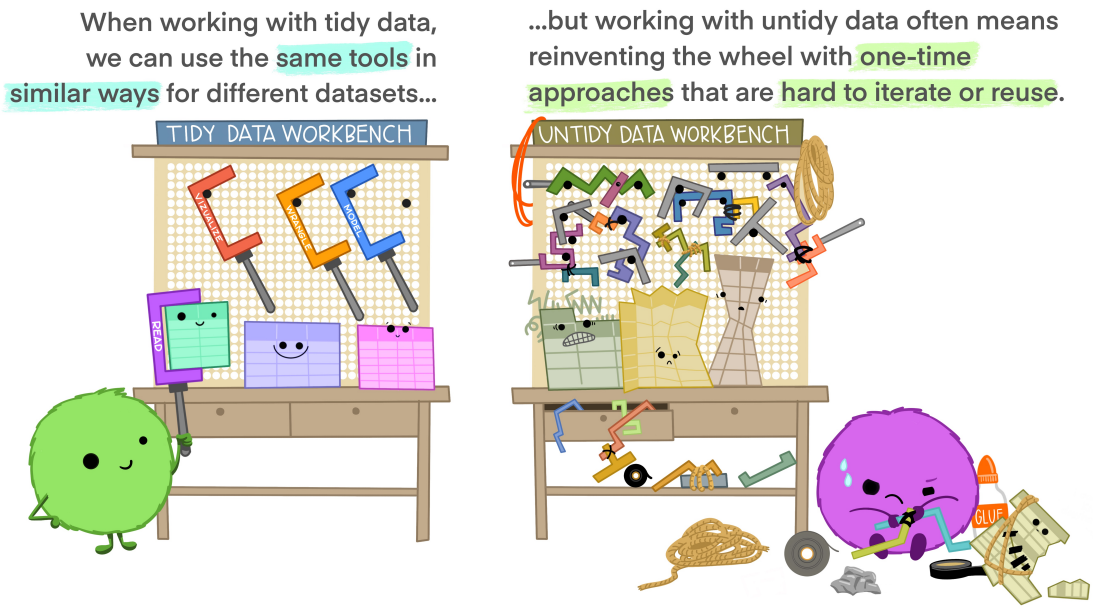


Figure 3: Julie Lowndes and Allison Horst

`tidyverse` ecosystem

We can think of the tidyverse ecosystem (Grolemund n.d.) as the set of tools we can reuse in our tidy data. It is an ecosystem because consists in a set of various packages that can be installed in one line of code (`install.packages(tidyverse)`), and each package fits into a part of the data science life cycle. This is the main reason why we prefer to use tidyverse instead of R base. It provides us with a consistent set of tools we can use for many different datasets, it is designed to keep our data tidy, and it contains all the specific tools we might need in our data science workflow.

There is a set of core tidyverse packages that are installed with the main ecosystem, which are ones you are likely to use in everyday data analyses.

- **`tibble`**: it is a package that *re-imagines the familiar R data.frame*. It is a way to store information in columns and rows, but does so in a way that addresses problems earlier in the pipeline. That is to say, it stores it in the tidy data format. The official documentation calls tibbles 'lazy and slurly', since they do less (they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist). This forces you to confront problems earlier, typically leading to cleaner, more expressive code.
- **`readr`**: this is a package we will use every time we start a new project. It helps read rectangular data into R, and it includes data in .csv and .tsv format. It is designed to flexibly parse many types of data found.
  - to read data in .xlsx or .xlx format, you should install the tidyverse-adjacent package **`readxl`**.

- `dplyr:` designed for data wrangling. It is built around five primary verbs (mutate, select, filter, summarize, and arrange) that help make the data wrangling process simpler.
- `tidyr:` it is quite similar to `dplyr`, but its main goal is to provide a set of functions to help us convert dataframes into tidy data.
- `purr:` enhances R's functional programming toolkit by providing a complete and consistent set of tools for working with functions and vectors. It makes easier for us to work with loops inside a dataframe.
- `stringr:` it is designed to help us work with data in string format.
- `forcats:` it provides functions to help us work with data in the factor format.
- `ggplot:` the main package for data visualization in the R community. It is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

While this is the main structure of the tidyverse ecosystem, there are multiple adjacent packages that we can install which fit into the tidy syntax and work well with the tidy data format. Some of them are `readxl` to read data in .xlsx or .xl format, `janitor` for cleaning data, `patchwork` to paste ggplot graphs together, `tidymodels` for machine learning… and many more.

Tidy data functions

there are some basic functions that we use to tidy data:

- `mutate`: to transform columns,
- `select`: to select certain columns,
- `filter`: to select certain rows (we can think of filter as the row-wise version of select, and select as the column-wise version of filter),
- `arrange`: to reorder values of rows.

Now, we will go over the basics of more complex data transformation functions to **reshape** the data: joins, pivots, summarizes and date processing.

For example if we have a survey on households

```
household
```

```
# A tibble: 7 x 3
  household_ID neighborhood_income n_people_household
         <int> <chr>                              <dbl>
1            1 low income                             2
2            2 low income                             1
3            3 high income                            1
4            4 high income                            1
5            5 low income                             5
6            6 high income                            4
7            7 high income                            3
```

```
household |>
  filter(n_people_household>2)
```

```
# A tibble: 3 x 3
  household_ID neighborhood_income n_people_household
         <int> <chr>                              <dbl>
1            5 low income                             5
2            6 high income                            4
3            7 high income                            3
```

```
household |>
  filter(n_people_household>2) |>
  mutate(neighborhood_income = toupper(neighborhood_income))
```

```
# A tibble: 3 x 3
  household_ID neighborhood_income n_people_household
         <int> <chr>                              <dbl>
1            5 LOW INCOME                             5
2            6 HIGH INCOME                            4
3            7 HIGH INCOME                            3
```

```
household |>
  filter(n_people_household>2) |>
  mutate(neighborhood_income = toupper(neighborhood_income)) |>
  select(neighborhood_income,n_people_household)
```

```
# A tibble: 3 x 2
  neighborhood_income n_people_household
  <chr>                            <dbl>
```

```
1 LOW INCOME                    5
2 HIGH INCOME                   4
3 HIGH INCOME                   3
```

Merges of dataframes

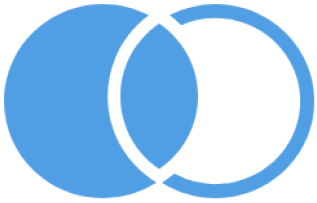
If you ever worked with SQL or multiple databases in the past, you might already be familiar with the concept of joining data. A tidy dataset should contain one type of observational unit per table. For example, if we have done a survey regarding labor conditions to individuals, but we also have sociodemographical information regarding their household, we should have each information in a different dataset. However, we will probably be interested in crossing the information regarding the individuals and their household conditions. So we need to have key IDs to be able to combine these two datasets: for example, in this case, the ID for the household.

household_ID	neighborhood_income	n_people_household
1	low income	2
2	low income	1
3	high income	1
4	high income	1
5	low income	5
6	high income	4
7	high income	3

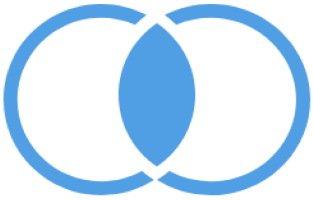
person_ID	household_ID	gender	age	income
1	1	f	37	4871
2	1	f	48	2029
3	2	m	32	3547
4	3	m	47	4650
5	4	m	36	1882
6	6	f	49	3600
7	7	f	36	4154

Note that the household ID is not the same as the person ID. Each person has a unique identifier, but so does each household. The information is consistent: when we have two individuals who share a household ID, we see in the household information that there are two people living there.

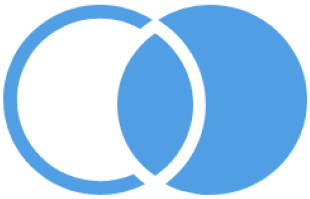
Now, how do dataframe joins work?



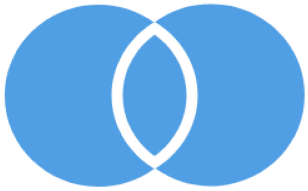
Left outer join



Inner join



Right outer join



Full outer join

- **Inner join:** this creates a new dataset that only contains the information of rows where the IDs match. For example, in our case, the data of the household 5 wouldn't appear since it doesn't join any individual data.

```
inner_join <- individual %>% inner_join(household)
inner_join
```

```
# A tibble: 7 x 7
  person_ID household_ID gender   age income neighborhood_income
      <int>        <dbl> <chr>  <dbl>  <dbl> <chr>
1         1            1 f         45   3649 low income
2         2            1 f         45   3232 low income
3         3            2 m         44   4286 low income
4         4            3 m         35   1723 high income
5         5            4 m         35   3708 high income
6         6            6 f         35   2676 high income
7         7            7 f         42   4765 high income
# i 1 more variable: n_people_household <dbl>
```

- **Left join:** it keeps all the rows of the data on the "left" and adds the columns that match the dataframe on the right. The decision of which dataframe goes where (left or right) is arbitrary and up to us, but we must keep in mind that it will be our main dataframe in the join. In our example, if we chose the individual dataset as the left, we would have the same table as the result of the inner join. But if we chose the household dataset, we would have a dataset with empty values for the ID's that don't match.

```
left_join_house <- household %>% left_join(individual)
left_join_house
```

```
# A tibble: 8 x 7
  household_ID neighborhood_income n_people_household person_ID gender   age
         <dbl> <chr>                              <dbl>     <int> <chr>  <dbl>
1            1 low income                             2         1 f         45
2            1 low income                             2         2 f         45
3            2 low income                             1         3 m         44
4            3 high income                            1         4 m         35
5            4 high income                            1         5 m         35
6            5 low income                             5        NA <NA>      NA
7            6 high income                            4         6 f         35
8            7 high income                            3         7 f         42
# i 1 more variable: income <dbl>
```

- **Full join:** it keeps all the columns and all the rows in both dataframes.

```
out_join <- household %>% full_join(individual)
out_join
```

```
# A tibble: 8 x 7
  household_ID neighborhood_income n_people_household person_ID gender   age
         <dbl> <chr>                              <dbl>     <int> <chr>  <dbl>
1            1 low income                             2         1 f         45
2            1 low income                             2         2 f         45
3            2 low income                             1         3 m         44
4            3 high income                            1         4 m         35
5            4 high income                            1         5 m         35
6            5 low income                             5        NA <NA>      NA
7            6 high income                            4         6 f         35
8            7 high income                            3         7 f         42
# i 1 more variable: income <dbl>
```

Summarizing information

Many times, we will want to summarize the information in our dataset. We can catch a glimpse of the summarized dataset with functions such as `summary()` or `str()`. However, more often, we will want to see specific information regarding groups in our dataset. For example, how many households we have in the different types of neighborhoods. To do this, it is necessary to group our data. We will not look into the total number of households, but we will group the data by the neighborhood of the households. Then, we can count each group.

```
household %>% group_by(neighborhood_income) %>%
  summarise(n=n())
```

```
# A tibble: 2 x 2
  neighborhood_income     n
  <chr>               <int>
1 high income             4
2 low income              3
```

Other times, we might be interested in getting descriptive statistics per group. For example, the median age for men and women in the individuals dataset.

```
individual %>% group_by(gender) %>%
  summarise(median_age = median(age))
```

```
# A tibble: 2 x 2
  gender median_age
  <chr>       <dbl>
1 f            43.5
2 m            35
```

Or even combine groups and subgroups. For example, the median income for women and men who live in low income or high income neighborhoods

```
individual %>%
  left_join(household) %>%
  group_by(neighborhood_income, gender) %>%
  summarise(income = median(income))
```

```
# A tibble: 4 x 3
# Groups:   neighborhood_income [2]
  neighborhood_income gender income
  <chr>               <chr>   <dbl>
1 high income         f       3720.
2 high income         m       2716.
3 low income          f       3440.
4 low income          m       4286
```

Reshaping data

Converting your data from wide-to-long or from long-to-wide data formats is referred to as **reshaping** your data.

For example, take this subset of columns from our individuals dataset.

```
individual %>% select(person_ID, gender, age, income)
```

```
# A tibble: 7 x 4
  person_ID gender   age income
      <int> <chr>  <dbl>  <dbl>
1         1 f         45   3649
2         2 f         45   3232
3         3 m         44   4286
4         4 m         35   1723
5         5 m         35   3708
6         6 f         35   2676
7         7 f         42   4765
```

This is what we call a wide format: each variable has its own column, and each row represents a single observation. Long data, on the other hand, refers to a dataset where each variable is contained in its own column. This format is often used when working with large datasets or when performing statistical analyses that require data to be presented in a more condensed format.

This is the same dataset we had previously but reshaped as long data.

```
individual %>%
  select(person_ID, gender, age, income) %>%
  pivot_longer(cols = -c(person_ID, gender), names_to = "variable", values_to = "value" )
```

```
# A tibble: 14 x 4
   person_ID gender variable value
       <int> <chr>  <chr>    <dbl>
 1         1 f      age         45
 2         1 f      income    3649
 3         2 f      age         45
 4         2 f      income    3232
 5         3 m      age         44
 6         3 m      income    4286
 7         4 m      age         35
 8         4 m      income    1723
 9         5 m      age         35
10         5 m      income    3708
11         6 f      age         35
12         6 f      income    2676
13         7 f      age         42
14         7 f      income    4765
```

As you can see, the wide data format has a separate column for each variable, whereas the long data format has a single "variable" column that indicates whether the value refers to age or income.

Data visualization

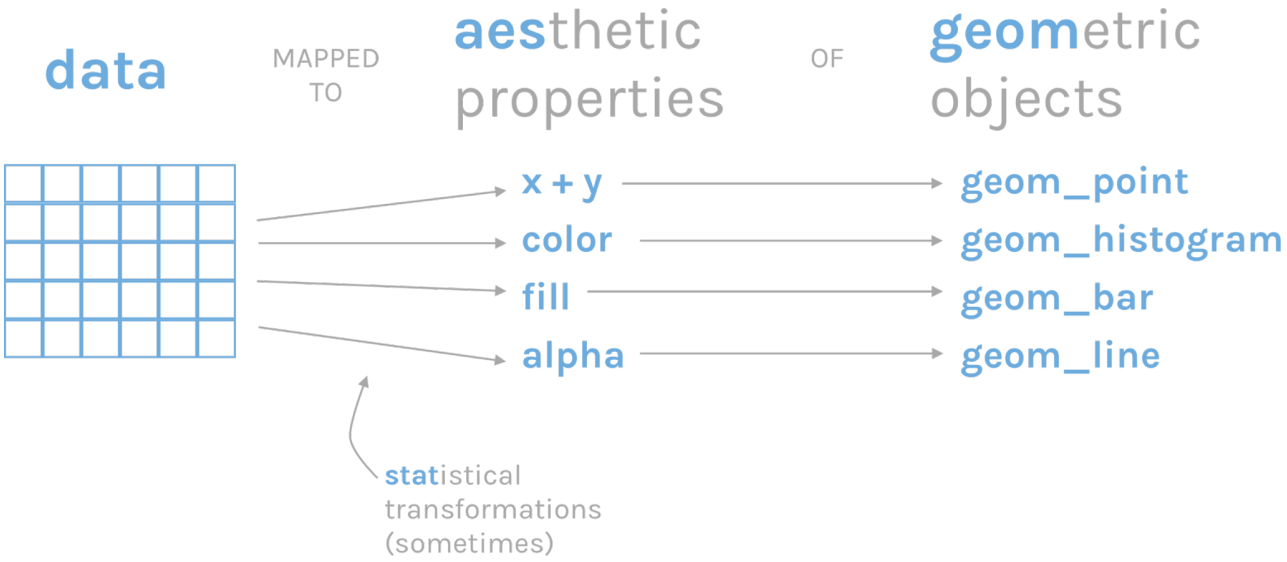
Visualization is a key part of the *data analysis workflow*. Not only does it help us explore our data and analysis results, but it also serves as the primary tool for sharing our findings with others. Whether we're presenting our work in an academic paper, journalistic article, or a presentation to colleagues, graphs are essential to attract their attention and making our work interesting. This is why it is important to learn how to make graphs that display information in an **effective** but also **aesthetically pleasing** way.

Grammar of graphics

Doing a good graphic involves data and creativity. However, there are certain common elements behind the creation of a graph that we can sum up as the **grammar of graphics** Wickham (2010). Graphics are constructed by combining independent components, such as data, aesthetics, geometries, and scales.

The package we will be using to make plots in R is called `ggplot2`, and its syntax is based on this grammar of graphics. In `ggplot2`, a plot is constructed by starting with a layer of data, and then adding additional layers to specify the aesthetics (e.g., color, shape, size) of

the plot, the type of geometry (e.g., point, line, bar) to use to display the data, and the scale (e.g., linear, log, discrete) to use for each aesthetic. Each of these components can be modified and customized in a variety of ways to create complex and informative visualizations.



This process of assigning data variables to the visual properties of a plot is called **mapping of aesthetic attributes**. In other words, it is the way to show in a visually perceptible way the difference between values. Aesthetics refer to visual properties such as color, shape, size, transparency, and position that can be used to visually represent data.

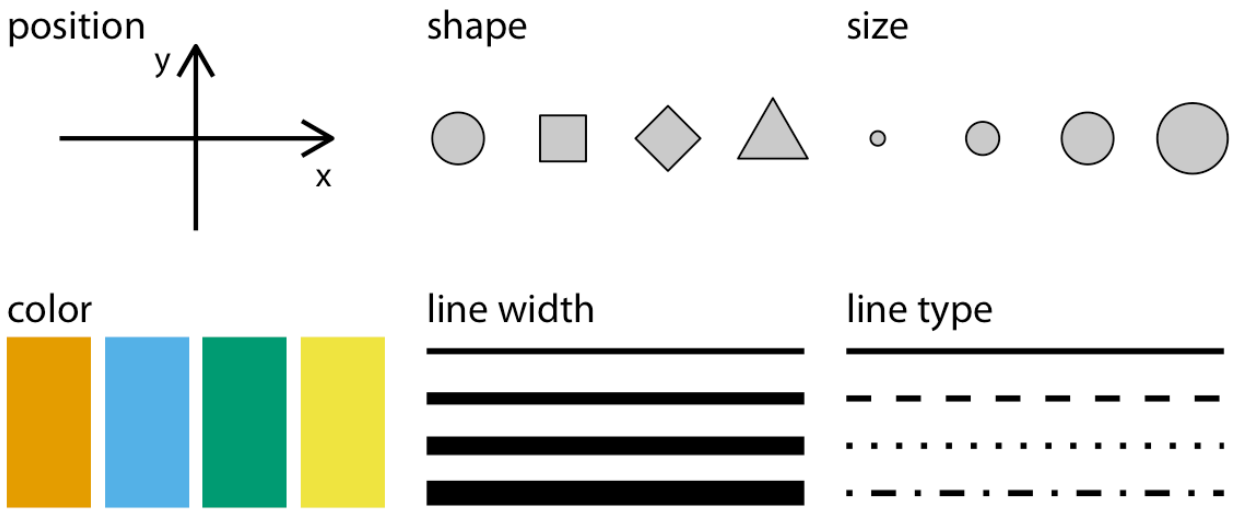
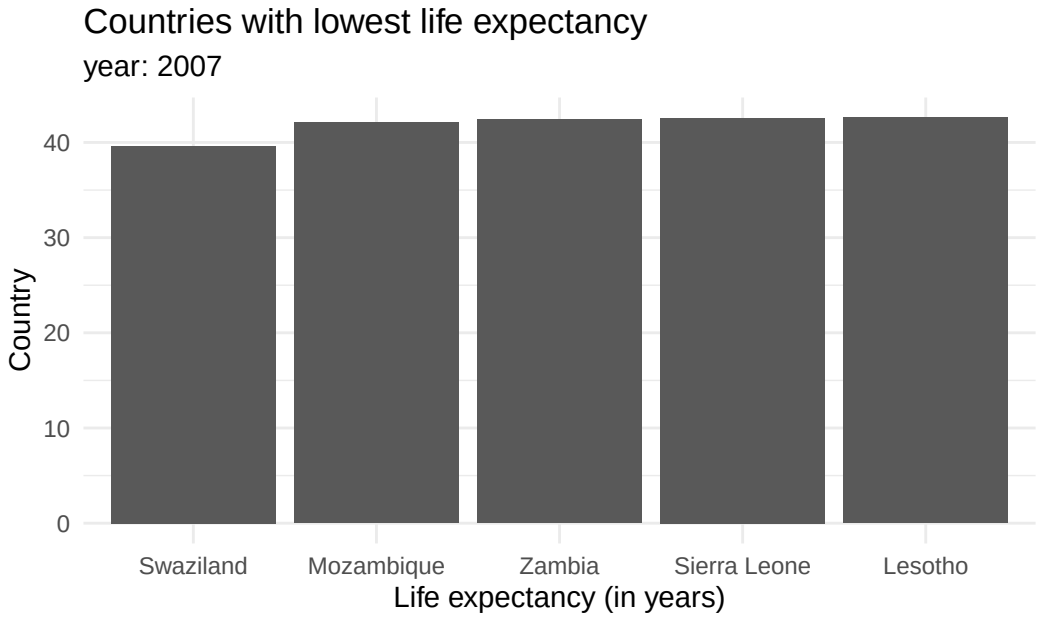


Figure 4: Commonly used aesthetics in data visualization: position, shape, size, color, line width, line type. Some of these aesthetics can represent both continuous and discrete data (position, size, line width, color) while others can usually only represent discrete data (shape, line type) (Wilke n.d.a)

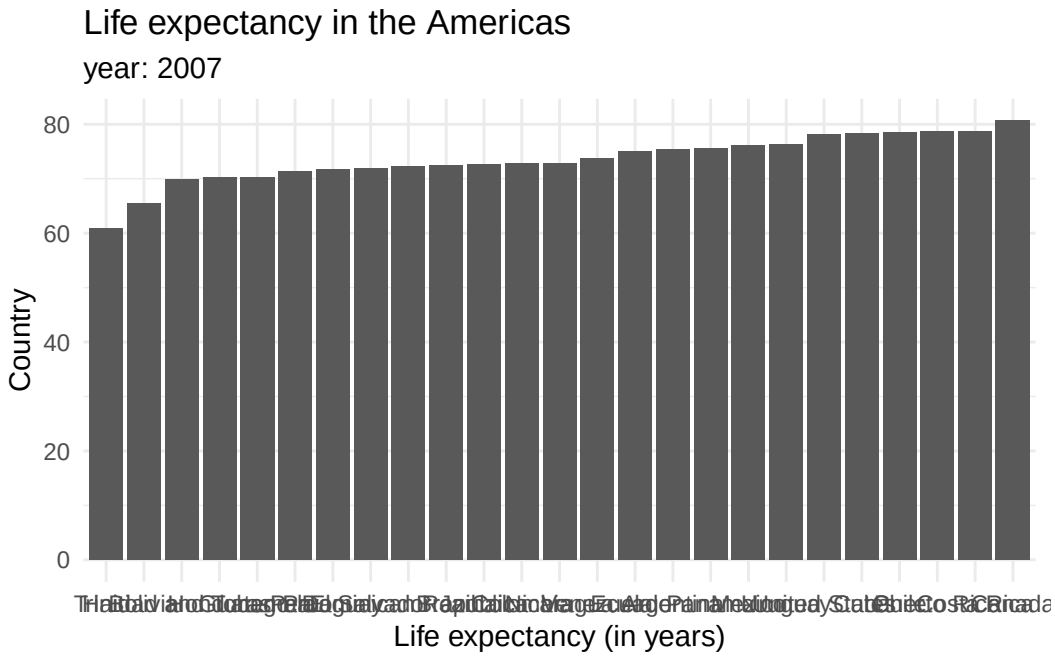
The grammar of graphics approach allows users to create a wide range of plots, from basic scatter plots and histograms to complex multi-layered visualizations, using a consistent and intuitive syntax. It is a powerful tool for data exploration and communication and has become a popular approach for data visualization in the R community.

Types of graphs

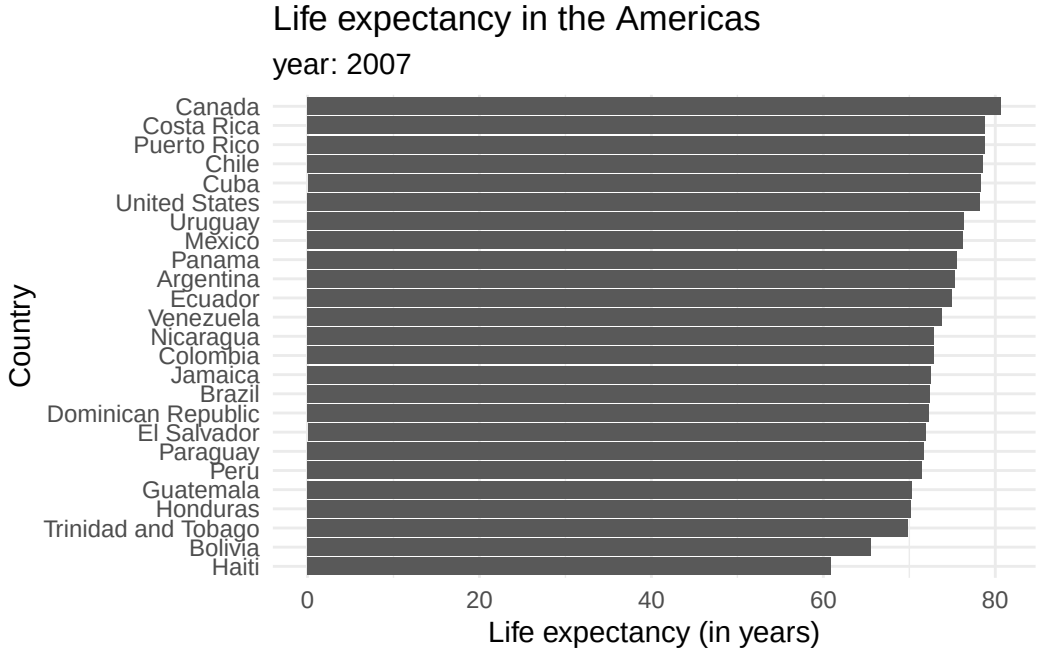
We can group plots into various clusters according to the kind of data we want to show. We will often be interested in showing the magnitude of numbers: the total number of people living in different countries, the mean salary for different groups of populations. These cases have a set of categorical values (such as countries, or demographic groups) and a quantitative value for each category. This is the same as saying we are going to show *amounts* (Wilke (n.d.b)). The most common graph to showcase amounts are **bar plots.** For example, we can display the countries with lowest life expectancy in the year 2007.



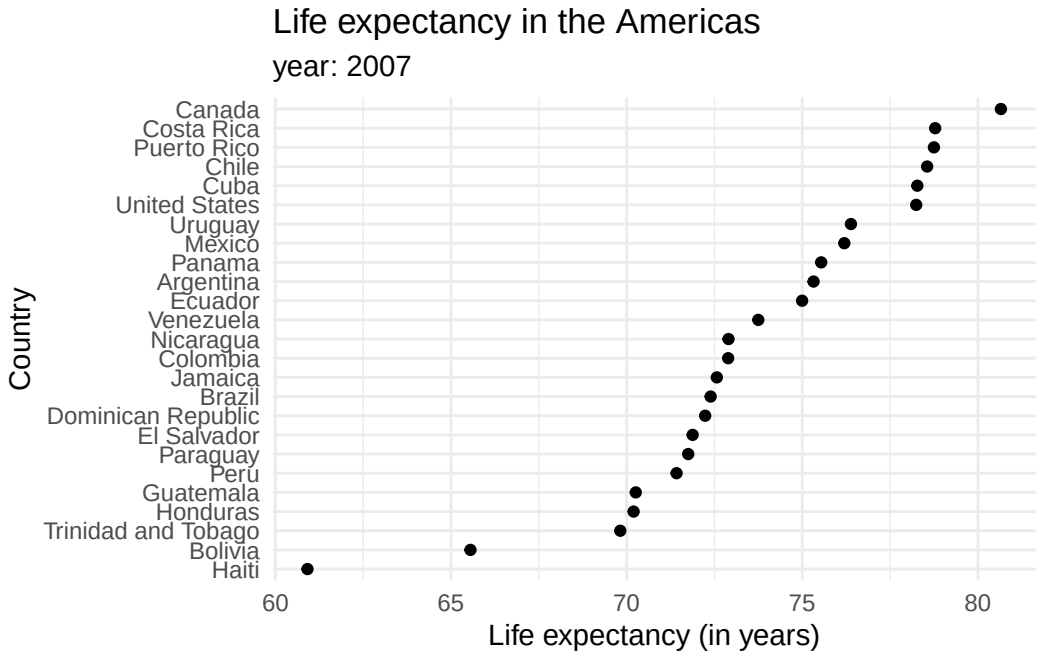
Each bar represents a categorical variable, and the length represents the quantitative value. Bars can be arranged either vertically or horizontally. By convention, when there are not many categories, vertical bars are the best option. But when one is working with many categories, the names on the horizontal axis might overlap, so displaying the values in the y axis is a better idea. Take a look at how the life expectancy ages in 2007 for the Americas would look like in a vertical axis…



and in a horizontal one.

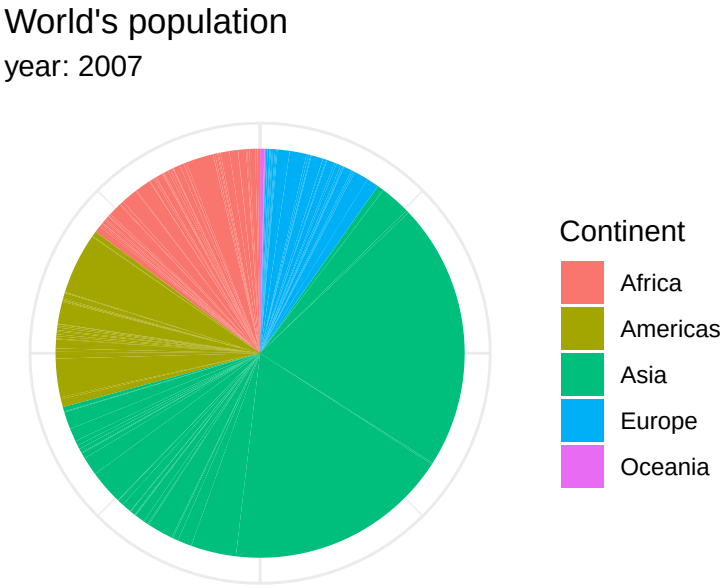


Bar plots tend to be the most common kind of visualization due to their effectiveness. Pretty much anyone can interpret them, and simple, easy-to-understand graphs are a key of data visualization. However, reusing them over and over in the same document might be repetitive and lose a reader's attention, so it is also good to keep in mind other alternatives. For example, **dot plots** provide a cleaner graph by only showing a point where the bar would end, removing the "insides" of the bars. This minimalist approach is often considered synonymous with a good plot in the data visualization community. Expanding our knowledge of different types of graphs and their uses can keep things interesting and enhance the clarity of our data representation.

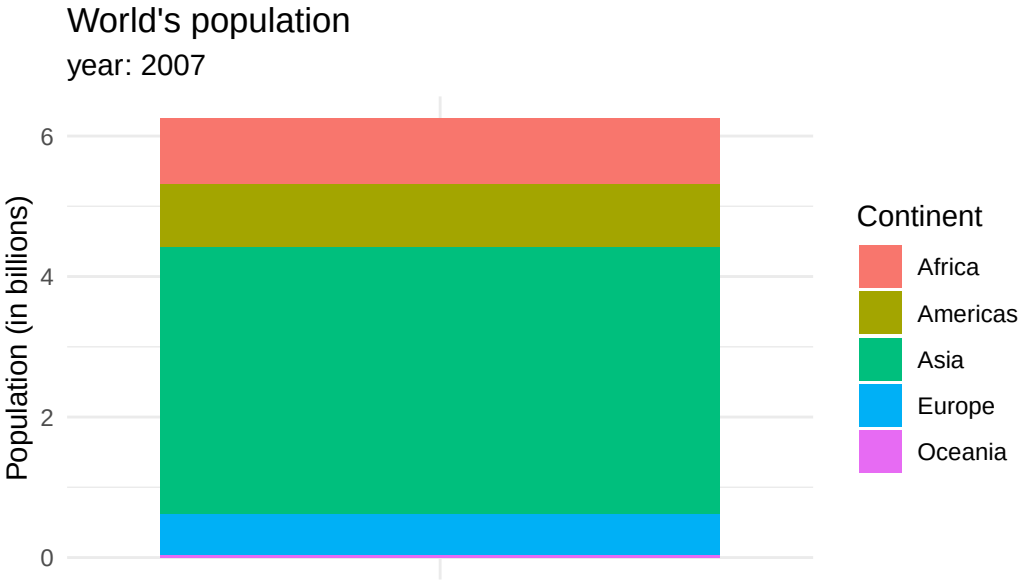


So far, we have seen examples showing values for individual groups. However, in other cases we might be interested in showing how some group, entity, or amount breaks down into individual pieces that each represent a *proportion* of the whole. Common examples include the percentage of races in a group of people (45% of the population in this neighborhood is black, 40% is white, 10% is latinx and 5% is asian) or the percentages of people voting for different political parties in an election (60% of the population voted for X candidate, while 40% voted for candidate Y).

The archetypal visualization for this kind of information is the **pie chart**, where each group is displayed as colored areas inside of a circle. However, over time, this graph has gained quite a negative reputation. While it is very popular, it has been shown that the human eye understands proportions more easily when they are displayed vertically. Take a look at the proportion of people in each continent represented within the world's total population in 2007, in a pie chart…



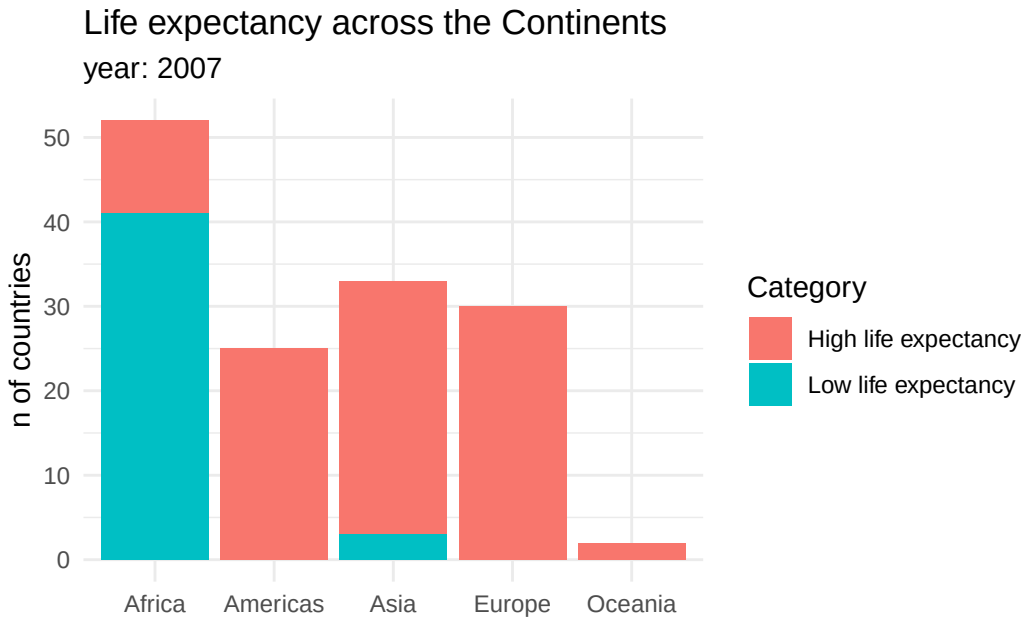
or in a stacked bar chart.



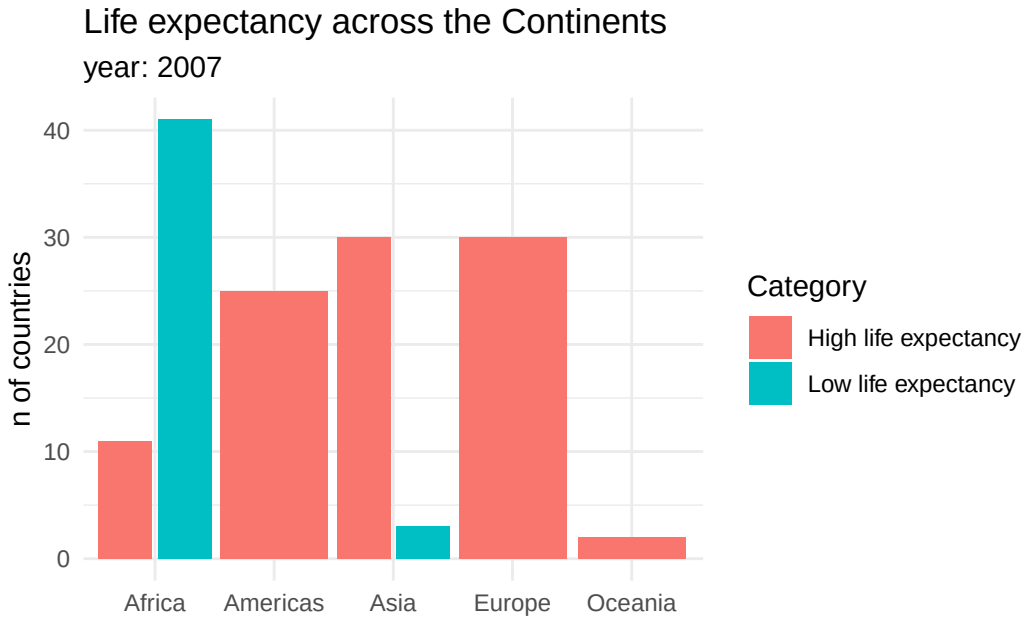
The **stacked bar chart** shows more clearly the weight each continent has on the total population of the world, and allows us to see a small line representing Oceania, which wasn't visible in the bar chart.

Visualizing proportions can be challenging, specially when the whole is broken into many different pieces. That is to say, when we want to see the values for sub-groups inside of our groups. And this is specially useful when we want to control how a variable changes by different

groups. For example, we could be interested in the amount of countries that have a high or a low life expectancy across continents. This could be done in a stacked bar chart:



Stacked bar charts are useful for showing absolute values within the larger groups. It is clear from this graph that Africa has the most countries, but also that the majority of them have a low life expectancy. Another option to show this information would be a **dodged bar chart**, where the subgroup bars are positioned next to each other rather that on top.



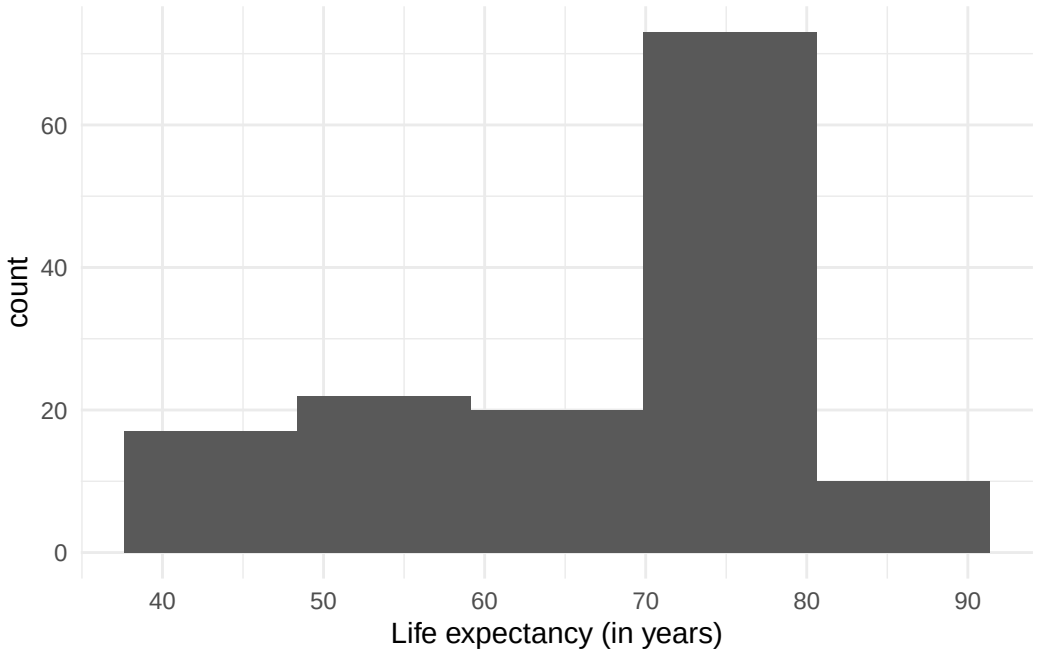
In this case, there is no clear answer to which kind of plot is better! Data visualization is a field where scientists should explore and try out different options to decide which one suits the case better.

So far, we have worked with the simplest kind of visualizations: counts or proportions. However, as we dive into the analysis, we might be interested in checking out how a particular variable is *distributed* in a dataset. That is to say, how the variable is spread: which values are more common and less common, which are extreme values, and so on.

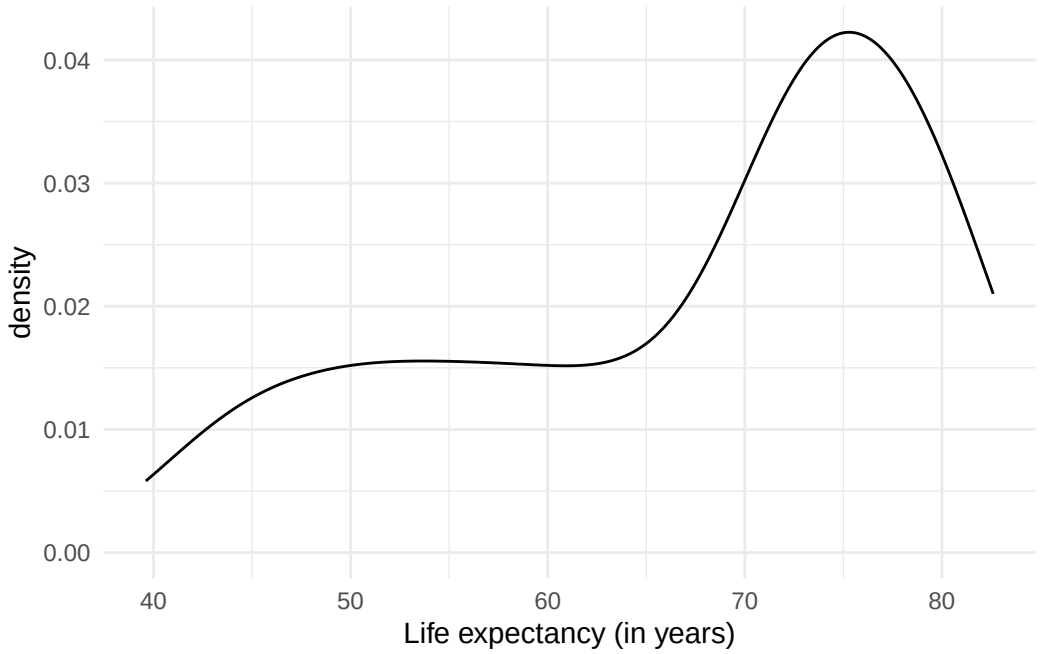
The most common visualization to show distributions is the **histogram**, which in practice is a variant of… the bar plot!

The histogram shows the range of values (from the minimum to the maximum that they reach), and how often they are observed in each range.

One thing to note about histograms is that their appearance (and therefore the message they convey) can change depending on the number of intervals used. The previous plot had divided the range of values into 30 'bins' or equal intervals (e.g. '0-10', '10-20', etc.) and counts how many observations fall into each one. We can increase the level of detail in the histogram by increasing the number of intervals, at the cost of losing generalization. Conversely, if we reduce the number of intervals, we show a more summarized distribution of the data, at the cost of losing detail.

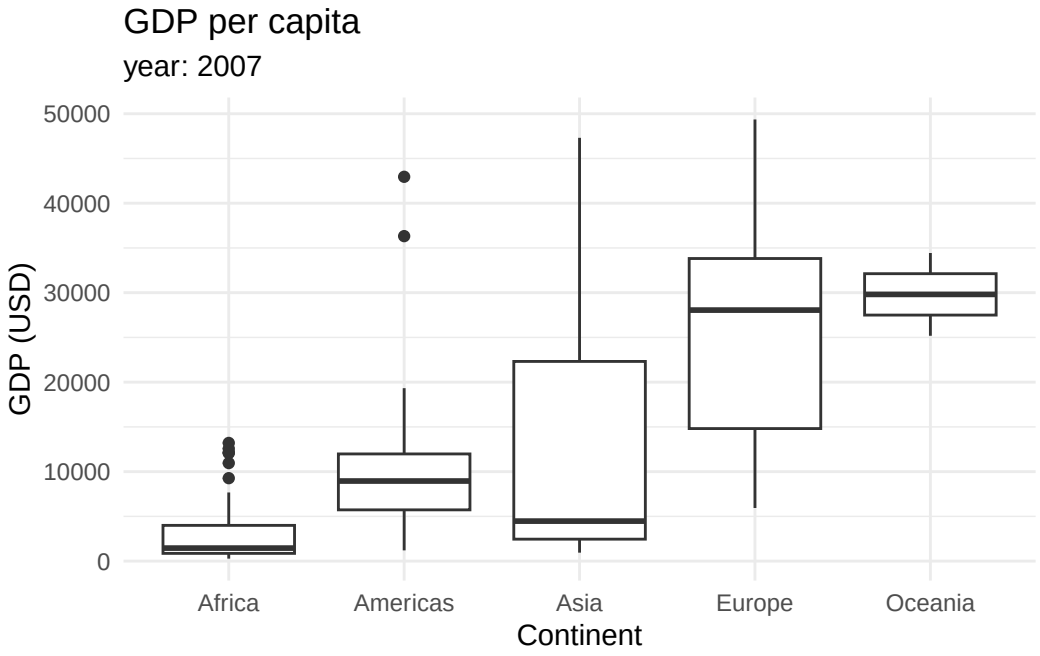


Another option to plot distributions are **density graphs**. Density plots are direct descendants of histograms. But instead of counts of observations per range of values, they show the probability distribution of the variable, i.e. how likely it is to find a particular value if we randomly selected one of the observations. Unlike histograms, which have been in use for a couple of centuries because they are relatively easy to create by hand, the (previously) laborious density plots have become popular with the advent of software and computers capable of creating them instantly.



The results of density plots are interpreted similarly to those of a histogram: we notice the range of the data and how common they are in one range compared to another.

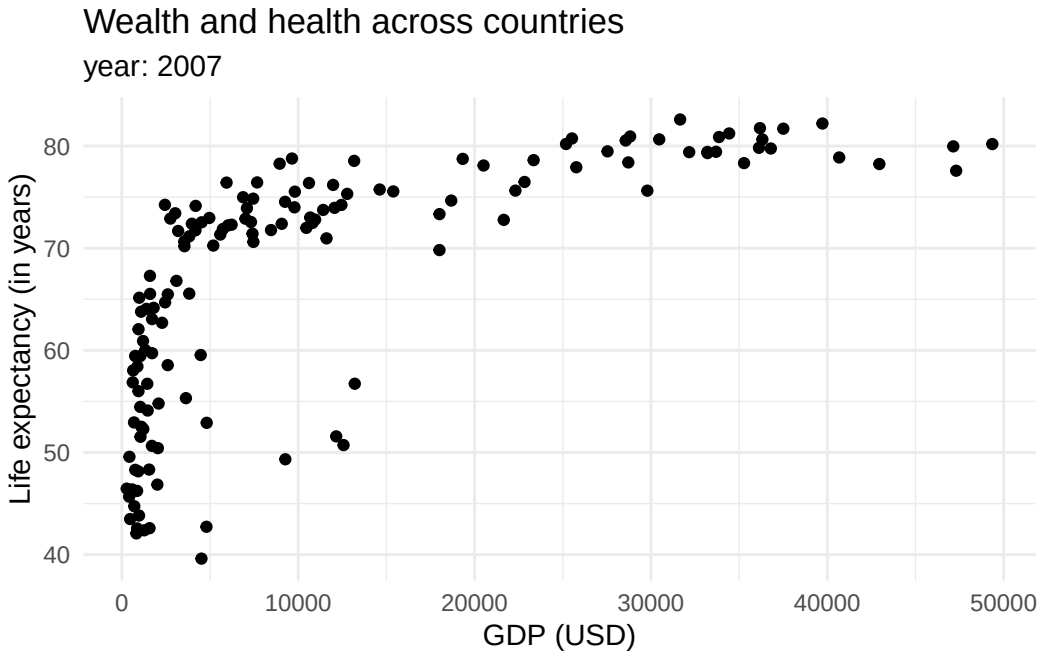
Finally, we can plot distributions as **boxplots** (also called box and whisker plot). It displays a summary of the minimum, first quartile, median, third quartile, and maximum values of the data. First, let's take a look at the following boxplot that displays the GDP per capita across continents in 2007.



This type of graphic contains a lot of information.

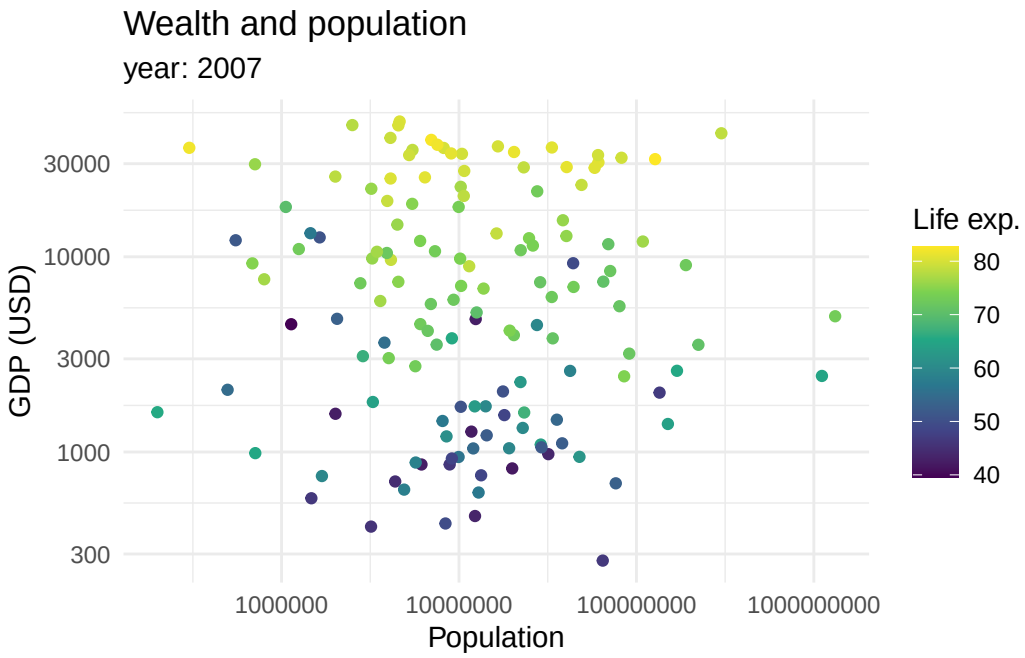
- The box in the middle represents the middle 50% of the data, with the lower edge representing the *first quartile* (25th percentile) and the upper edge representing the *third quartile* (75th percentile).
- The line inside the box represents the *median*.
- The whiskers extending from the box show the *range* of the data, typically defined as 1.5 times the *interquartile range (IQR)*, which is the distance between the first and third quartiles.
- *Outliers*, which are data points that fall outside the whiskers, are shown as individual points.

Finally, we could be interested in showing the relationship between variables (x-y). The most common way to show this is through **scatterplots**.

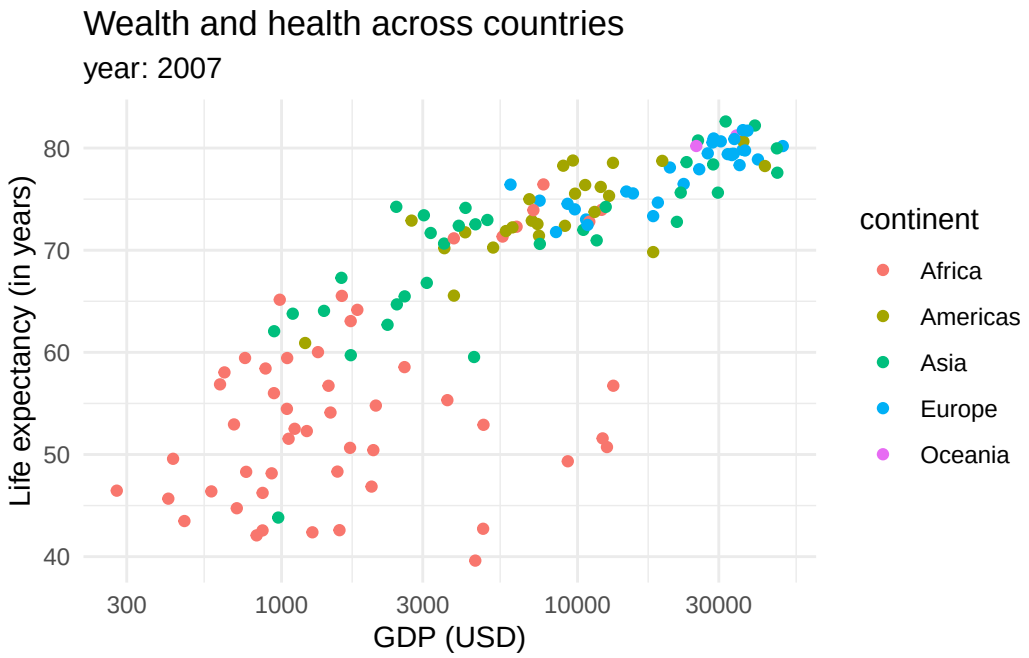


Mapping more aesthetic attributes

So far, we mostly saw how data can be mapped into an x and y axis. However, we mentioned we can also map data as *shapes, sizes* and *colors*. We briefly saw how color can be introduced to show categorical variables when we are seeing proportions of the data. However, it can also be used as a continuous variable.

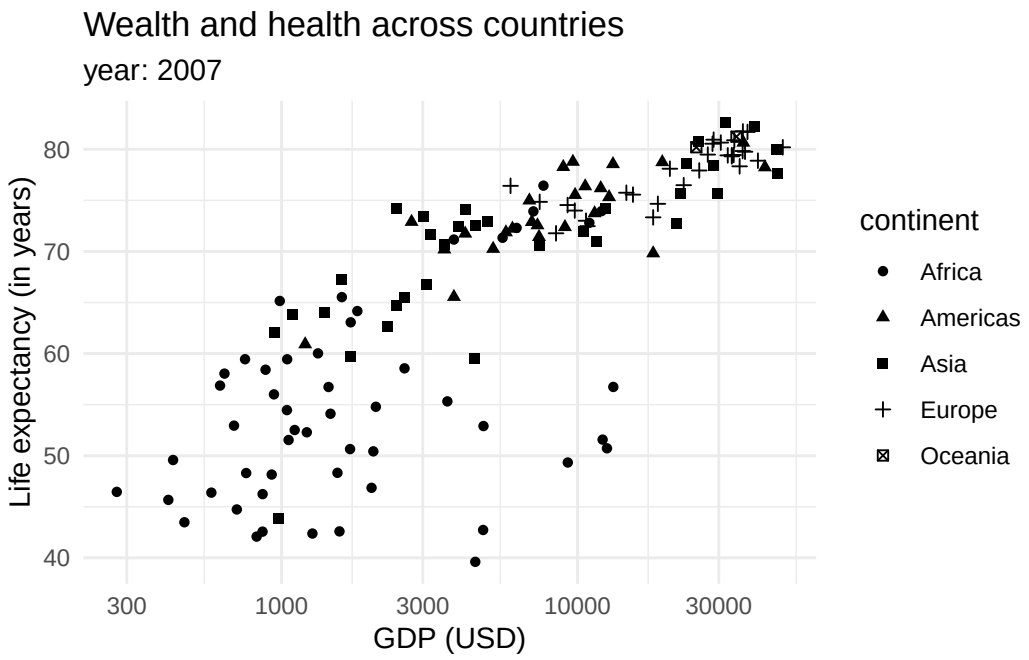


This graph shows that there is a relationship between longevity and GDP: countries with higher life expectancy are found at the top of the graph. As we continue to explore this relationship in the following plots, note that we also introduced a transformation on the X axis. We can see that the values in the scale go from 1,000,0000 to 10,000,000, and then to 100,000,000 until 1,000,000,000. This is called a **logaritmic scale**: the values are no longer evenly spaced, they increase exponentially. This is specially useful when we are working with data with a wide range of values, such as populations or income. For example, let's introduce the variable continent into our 'Wealth and health' graph with a logarithmic scale in the GDP variable.



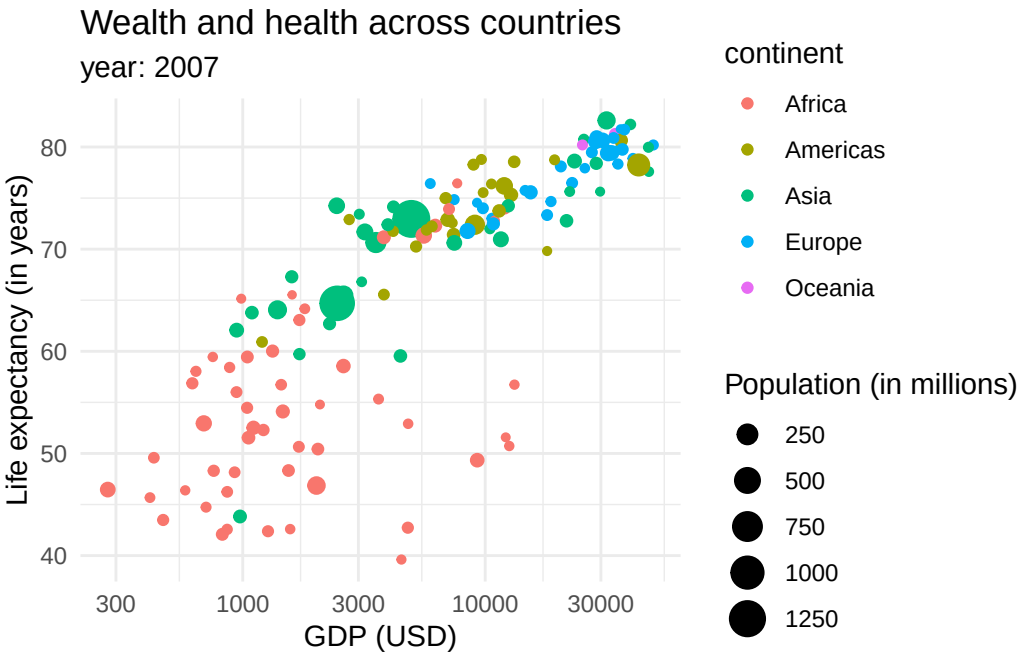
This returns us a more clear and compact graph, where we can better see the variability in the lower values for both variables, and relate them mostly to Africa.

We could also show the continents as shapes:



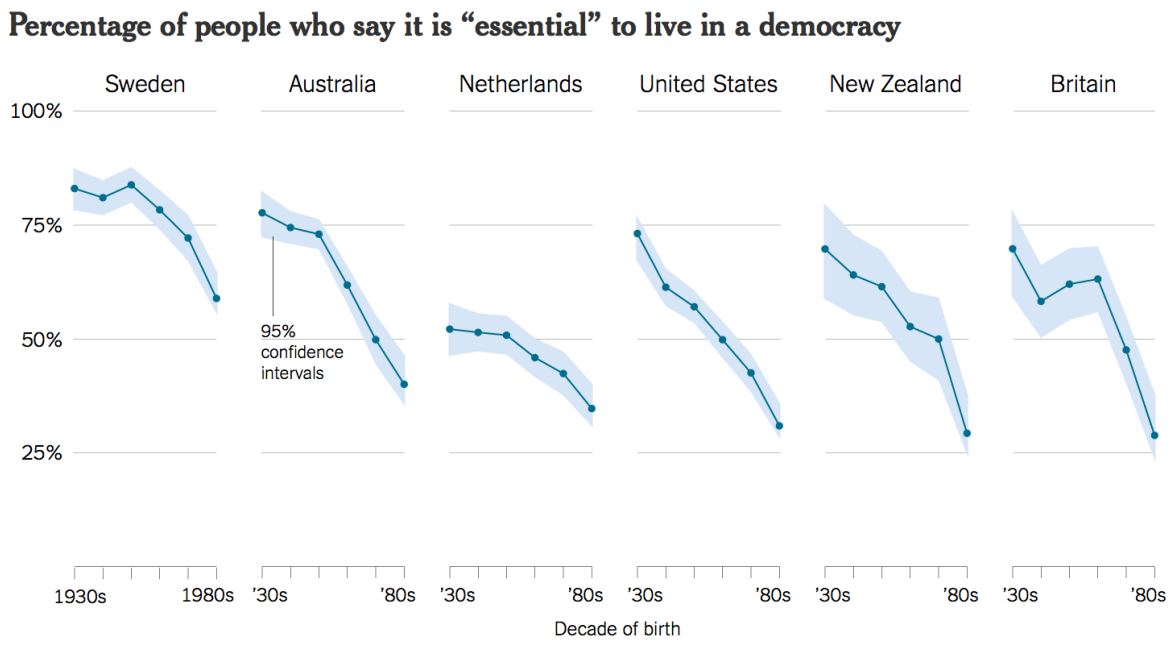
However, the color tends to be a better option to plot categorical data. We could also introduce

more data into the plot through the *size* of the dots or shapes.



Discussion

The design of a graph should always be based on the data and focused on the topic we are researching. However, we should always be careful about not lying with graphs. That is to say, it is important not to toy with our graph to show only information that reinforces our hypothesis or what we want the data to say. Kieran (2018) explains this with a clear example from the New York Times. In November 2016, this newspaper reported some research on peoples' confidence in the institutions of democracy. The headline in the *Times* ran "How Stable Are Democracies? 'Warning Signs Are Flashing Red' ", and the graph accompanying the article certainly seemed to show an alarming decline.



The graph was widely circulated on social media. However, this is an example on how one can lie with data. This plot presumable shows the the percentage of respondents who said "Yes", presumably in contrast to those who said "No". However, the survey asked respondants to rate the importance of living in a democracy on a ten point scale. The graph showed the difference across ages of people who had given of "10" (Absolutely important) only, not changes in the average score on the question. As it turns out, while there is some variation by year of birth, most people in these countries tend to rate the importance of living in a democracy very highly, even if they do not all score it as "Absolutely Important". The political scientist Erik Voeten redrew the figure based using the average response.

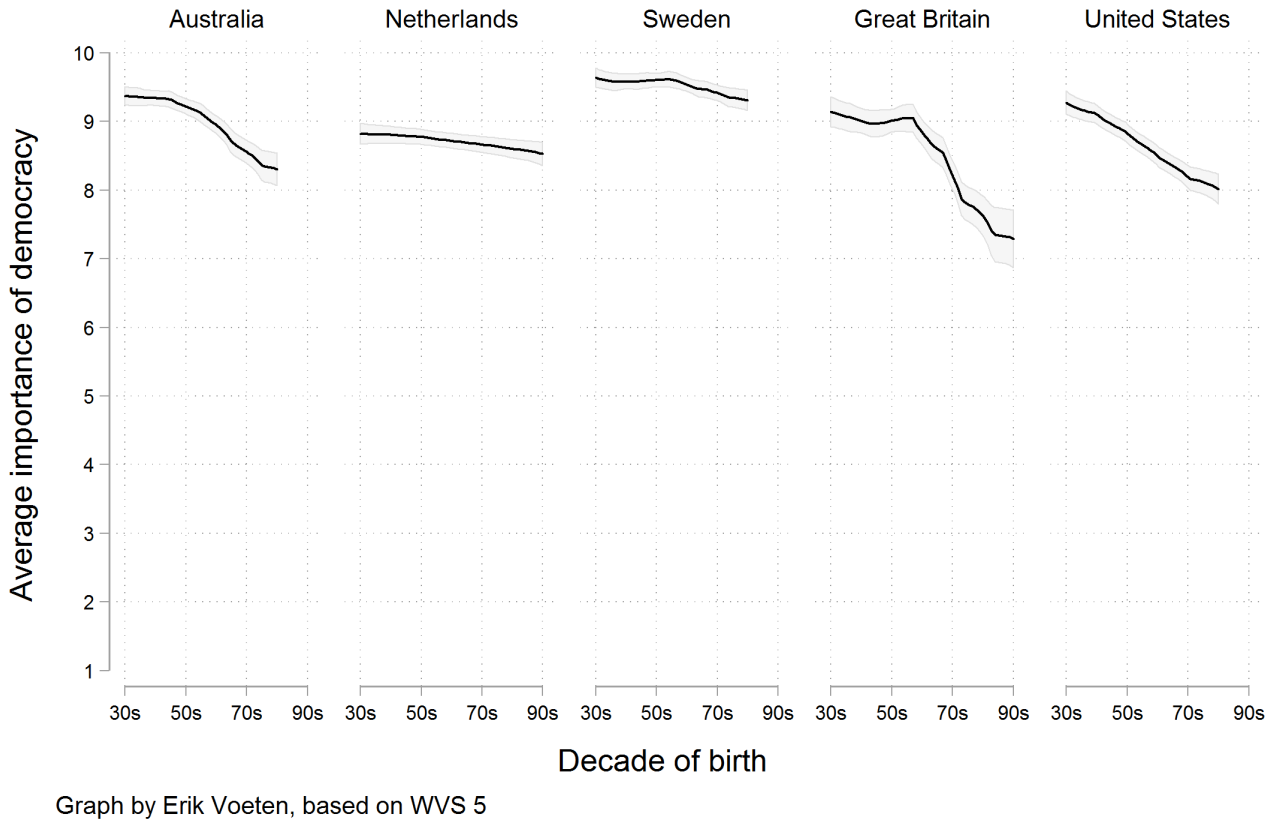


Figure 5: Erik Voeten

While we still see a decline in the average score by age cohort, on the order of between half a point to one and a half points on a ten point scale, it is not such a drastic decline as the one showed originally.

References

Grolemund, Hadley Wickham and Garrett. n.d. *Welcome | R for Data Science*. Accessed March 23, 2023. https://r4ds.had.co.nz/.

Kieran, Healy. 2018. *Data Visualization*. https://press.princeton.edu/books/hardcover/9780691181615/data-visualization.

Peng, Stephanie C. Hicks and Roger D., Shannon E. Ellis. n.d. *Tidyverse Skills for Data Science*. Accessed March 23, 2023. https://jhudatascience.org/tidyversecourse/.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1): 3–28. https://doi.org/10.1198/jcgs.2009.07098.

———. 2014a. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.

———. 2014b. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.

Wilke, Claus O. n.d.a. *Fundamentals of Data Visualization*. Accessed May 8, 2023. https://clauswilke.com/dataviz/aesthetic-mapping.html.

———. n.d.b. *Fundamentals of Data Visualization*. https://clauswilke.com/dataviz/aesthetic-mapping.html.

Wilkinson, Leland. 2005. *The Grammar of Graphics*. Statistics and Computing. New York: Springer-Verlag. https://doi.org/10.1007/0-387-28695-0.