Introduction to R - Young Researchers Fellowship Program

Lecture 2 - Introduction to the tidyverse and data importing

Daniel Sánchez Pazmiño

Laboratorio de Investigación para el Desarrollo del Ecuador

September 2024

The tidyverse or how modern R code is being written

Tidy data

Tidying your data means storing it in a consistent form that matches the semantics of the dataset with how it is stored (Wickham et al, 2023)

- Tidy data is a standard way of mapping the meaning of a dataset to its structure.
- A dataset is messy or tidy depending on how rows, columns, and tables are matched up with observations, variables, and types.
- In tidy data:
  - Each variable forms a column.
  - Each observation forms a row.
  - Each type of observational unit forms a table.

Who came up with this?

- Hadley Wickham introduced the concept of tidy data in his paper "Tidy Data" published in the Journal of Statistical Software in 2014.
- In the R for Data Science book (R4DS), the tidyverse is introduced as a collection of R packages designed to tidy data and work with it in a data science context.
- The tidyverse philosophy revolutionized the way R code is written and data is handled, making it more efficient and easier to understand.

The data science vs. the research perspective

- According to Hadley Wickham, *data science is an exciting discipline that allows you to transform raw data into understanding, insight, and knowledge*.
- This means we need not be afraid that the tidyverse will make us lose the ability to do research.
    - In this view, data science is not only predictive modeling.

The tidyverse steps in a data science project

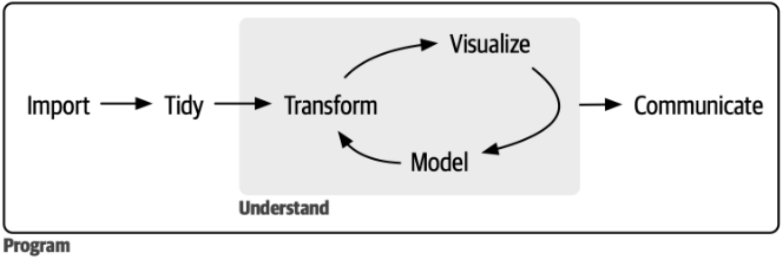

Figure 1: Tidy steps in Data

Source: R for Data Science by Wickham, Cetinkanya-Rundel & Grolemund (2023)

The tidyverse steps in a research project

1. **Import**: read data from a file or database.
2. **Tidy**: transform the data into a format that makes it easy to work with.
3. **Transform**: perform operations on the data to create new variables or summaries.

(together, transform and tidy are often referred to as wrangling - it feels like a wrestling match sometimes!)

4. **Visualize**: generate static graphics for exploratory data analysis.
5. **Model**: fit quantitative models to understand relationships between variables, complementary to visualization.
6. **Communicate**: generate reports or dashboards, or create a Shiny app. The most important step!

Where does programming fit in?

- Programming is an outer step in the process as it will be used all along the way.
- We use programming to automate the steps in the process and solve problems effectively.

The tidyverse packages

- The tidyverse is a collection of R packages that share an underlying design philosophy, grammar, and data structures.
- The packages in the tidyverse are designed to work together, and it is easier to learn them together.

The core tidyverse packages

- **ggplot2**: for data visualization.
- **dplyr**: for data manipulation.
- **tidyr**: for data tidying.
- **readr**: for data import.
- **purrr**: for functional programming.
- **tibble**: for tibbles, a modern reimagining of data frames.
- **stringr**: for strings.
- **forcats**: for factors.

Installing the tidyverse

- We install them all at once through the `tidyverse` package, which is a meta-package that installs the core tidyverse packages.

```
install.packages("tidyverse")
```

Importing data

Importing data with the tidyverse

- The `readr` package is part of the tidyverse and provides a fast and friendly way to read rectangular data.
    - "Rectangular data" is data that is organized into rows and columns.
    - This is data that appears in a `data.frame` in R, and stored in `.csv`, `.tsv`, or `.txt` files.
    - Other types of data, like spatial data, are not rectangular and are not handled by `readr`.
- The base R `read.csv()` function is also used to read data, but `readr` is faster, more user-friendly, and offers a greater range of possibilities for data loading.

The Ecuadorian used cars dataset

- Kaggle is a platform for data science competitions and datasets.
- The dataset we will use for loading is the Ecuadorian used cars dataset.
- The dataset contains information about used cars in Ecuador, such as the brand, model, year, price, and more
  - It was obtained by scraping the patiotuerca.com website.

Importing data with the Import button in RStudio

- RStudio has a built-in feature to import data from a file.
- You can click on the "Import Dataset" button in the Environment pane and select the file you want to import.
  - Loads different types of files, such as `.csv`, `.tsv`, `.txt`, `.xlsx`, and more, based on different packages
  - Loads the data into your environment as a `data.frame`, and generates the code to do so.
- Recommended for beginners as they learn the syntax for the `readr` package.

Importing data with the Import button in RStudio

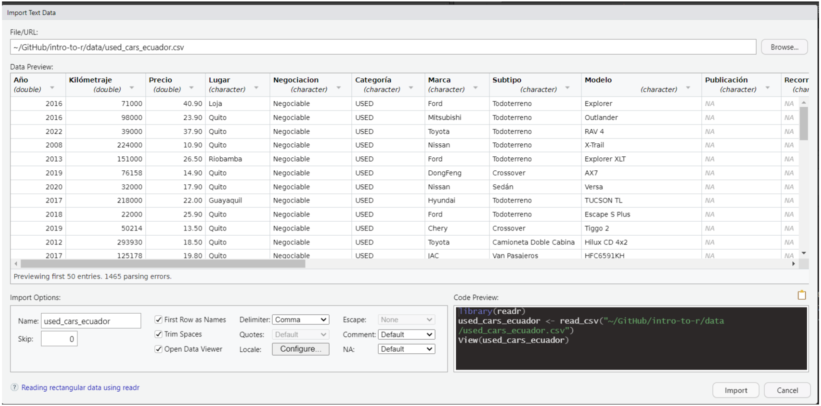

Figure 2: Import data in RStudio

Steps after importing data

- After importing the data, you should:
  - Check the structure of the data with `str()`.
  - Check the first few rows of the data with `head()`.
  - Check the last few rows of the data with `tail()`.
  - Check the summary of the data with `summary()`.
- A tidyverse function, loaded from the `dplyr` package, is `glimpse()`, which provides a more detailed view of the data.
- Further, using `janitor::clean_names()` will clean the column names.
  - This is a function from the `janitor` package, which is not part of the tidyverse but is useful for cleaning data.
  - Spanish names often have accents, which can be problematic for programming and technically invalid.
- Not using `clean_names()` would require the use of the apostrophe to refer to the columns.

Reading Microsoft Excel files

- Plenty of packages deal with Excel files in R, but the `readxl` package is part of the tidyverse and is the recommended package for reading Excel files.
    - Alternatives are the `openxlsx` and `writexl` packages.
- The `readxl` package is fast and user-friendly, and it can read `.xls` and `.xlsx` files.
- The `readxl` package has two main functions: `read_excel()` and `excel_sheets()`.

Reading Microsoft Excel files

- The `read_excel()` function reads an Excel file and returns a `tibble` or a `data.frame`.
- Will also work with the import button in RStudio.
- Might need to pre-clean the loaded data by skipping columns, rows, or specifying the range of cells to read.

Reading Microsoft Excel files - SUPERCIAS Companies' Directory

- The SUPERCIAS Companies Directory is a dataset that contains information about companies registered in Ecuador.
- Requires skipping the first 4 rows.

```
library(readxl)
library(dplyr)

# Read the Excel file

supercias <-
    read_excel("data/directorio_companias_supercias.xlsx",
               skip = 4)
```

Reading other types of files

- The `readr` package has a set of functions that are used to read different types of files:
  - `read_csv()`: reads comma-separated files.
  - `read_tsv()`: reads tab-separated files.
  - `read_delim()`: reads in files with a custom delimiter (common for `.csv` files with ; delimiters)
  - `read_fwf()`: reads in fixed-width files.
  - `read_table()`: reads in files with a custom column separator.
- For data coming from statistical software, the `haven` package is a tidyverse package that reads in `.dta`, `.sav`, and `.sas` files.
  - SPSS files might come in `.sav` or `.por` formats.
  - SAS files might come in `.sas7bdat` or `.xpt` formats.
  - Stata files might come in `.dta` format.
- More complex files, like JSON, XML, and HTML, can be read in with the `jsonlite`, `xml2`, and `rvest` packages, respectively.

Tidyverse fundamentals: tibbles and `dplyr`

The tibble: a modern data frame

- The `tibble` is a modern reimagining of the `data.frame` in R.
- It is part of the tidyverse and is used to store rectangular data.
- It is more user friendly, as when you print a `tibble`, it only shows the first 10 rows and the columns that fit on the screen.
- Other differences are that `tibble` does not convert strings to factors by default, and it does not use row names.

The tibble: a modern data frame

- We can transform a data frame into a tibble with the `as_tibble()` function.
- The `tibble` package is part of the tidyverse, so you do not need to install it separately if you've installed it.

dplyr: plying data into shape with the tidyverse

- The dplyr could be considered as the most important package in the tidyverse, used for the transform part of the tidy process.
- Provides a set of functions that perform common data manipulation operations such as filtering, selecting, mutating, summarizing, and arranging.
- dplyr brings with itself a grammar of data manipulation and a modern, different coding style based on the pipe operator %>% and tidyverse verbs.

The dplyr verbs or functions

Among others:

- `filter()`: to filter rows based on a condition.
- `select()`: to select columns.
- `mutate()`: to create new columns.
- `arrange()`: to reorder rows.
- `summarize()`: to summarize data.
- `group_by()`: to group data.
- `distinct()`: to select distinct rows.
- `rename()`: to rename columns.

The dplyr verbs or functions

- These are called "verbs" because they are functions that perform actions on a dataset.
- All follow the same general syntax:

```
verb(data, arguments)
```

where data is a data frame or tibble, and arguments are the arguments that the function takes.

Selecting columns with `select()`

- The `select()` function is used to select columns from a data frame.
- In base R, you would use the `$` operator to select columns or indexing with the `[ ]` operator.
- The `select()` function is more flexible and allows you to select columns based on their names or positions.

The used cars example

```r
library(dplyr)

# Select columns from the cars dataset
select(cars, Precio, Lugar, Negociacion)
```

```
# A tibble: 9,021 x 3
   Precio Lugar     Negociacion
    <dbl> <chr>     <chr>
 1   40.9 Loja      Negociable
 2   23.9 Quito     Negociable
 3   37.9 Quito     Negociable
 4   10.9 Quito     Negociable
 5   26.5 Riobamba  Negociable
 6   14.9 Quito     Negociable
 7   17.9 Quito     Negociable
 8   22   Guayaquil Negociable
 9   25.9 Quito     Negociable
```

Selecting columns with `select()`

- You may also use the `-` operator to exclude columns.
- The `tidyselect()` helper functions can be used to select columns based on patterns.
  - `starts_with()`
  - `ends_with()`
  - `contains()`

Filtering rows with `filter()`

- The `filter()` function is used to filter rows based on a logical condition.
- In base R, you would use the `[ ]` operator to filter rows, or `subset()`
- The `filter()` function is more flexible and allows you to filter rows based on multiple conditions.
  - Multiple conditions go within the same `filter()` function, separated by commas.
  - These would be combined as an `AND`'&' (all conditions must be met) logical operator.

The used cars example

```r
# Filter rows from the cars dataset

filter(cars, Marca == "Chevrolet", Precio < 10000)
```

```
# A tibble: 1,771 x 26
      Año Kilómetraje Precio Lugar Negociacion Categoría Marca
    <dbl>       <dbl>  <dbl> <chr> <chr>       <chr>     <chr>
 1   2014      118000   14.8 Quito Negociable  USED      Chevrole
 2   2020       98000   12.4 Quito Negociable  USED      Chevrole
 3   2019       27000    8.9 Quito Negociable  USED      Chevrole
 4   2019       27000    8.9 Quito Negociable  USED      Chevrole
 5   2022        9400   16.9 Quito Negociable  USED      Chevrole
 6   2016        9000  145   Quito Negociable  USED      Chevrole
 7   2009      296000   17.9 Quito Negociable  USED      Chevrole
 8   2013      173000   18.9 Quito Negociable  USED      Chevrole
 9   2015      235621   10.9 Quito Fijo        USED      Chevrole
10   2012      170000    9.9 Quito Fijo        USED      Chevrole
```

The pipe operator %>%

- The pipe operator %>% is used to chain operations together.
  - Takes the output of the operation on the left and passes it as the first argument to the operation on the right.
- Initially introduced in the `magrittr`[1] package, which introduced programming operators.

Figure 3: The Magrittr Hex Logo

[1] The magrittr package is named after the Belgian surrealist artist René Magritte. The original painting is called "The Treachery of Images" and features a pipe with the

The pipe operator %>%

- Think of it as an operator which facilitates the flow of composite functions in algebra.

$$f(x) = g(h(x))$$

- The pipe operator allows you to write this as:

```
x %>% h() %>% g()
```

The pipe operator %>%

- As an example, consider the following code:

```
filtered_data <- filter(cars, Marca == "Chevrolet", Precio < 100

selected_data <- select(filtered_data, Precio, Marca)

selected_data
```

```
# A tibble: 1,771 x 2
   Precio Marca
    <dbl> <chr>
 1   14.8 Chevrolet
 2   12.4 Chevrolet
 3    8.9 Chevrolet
 4    8.9 Chevrolet
 5   16.9 Chevrolet
 6  145   Chevrolet
 7   17.9 Chevrolet
```

The pipe operator %>%

- This can even be done with functions that are not part of the tidyverse.

```
cars %>%
    janitor::clean_names() %>%
    filter(marca == "Chevrolet", precio < 10000) %>%
    select(ano, precio, lugar)
```

```
# A tibble: 1,771 x 3
     ano precio lugar
   <dbl>  <dbl> <chr>
 1  2014   14.8 Quito
 2  2020   12.4 Quito
 3  2019    8.9 Quito
 4  2019    8.9 Quito
 5  2022   16.9 Quito
 6  2016  145   Quito
```

The pipe operator %>%

- The pipe operator is very useful for making code more readable and easier to follow.
- It is also useful for debugging, as you can see the output of each step in the chain.
    - "Debugging" means finding and fixing problems in your code.
    - When you don't pipe, you need to store the output of each step in a separate object, which can be cumbersome.
- All of the tidyverse packages are designed to be pipeable, even those not part of the core tidyverse!

Renaming columns with `rename()`

- The `rename()` function is used to rename columns in a data frame.
- In base R, you would use the `names()` function to rename columns.
- The `rename()` function is more flexible and allows you to rename columns based on their names.

The used cars example

```r
# Rename columns in the cars dataset
cars %>%
    janitor::clean_names() %>%
    rename(price = precio)
```

```
# A tibble: 9,021 x 26
      ano kilometraje price lugar     negociacion categoria marca
    <dbl>       <dbl> <dbl> <chr>     <chr>       <chr>     <chr>
 1   2016       71000  40.9 Loja      Negociable  USED      Ford
 2   2016       98000  23.9 Quito     Negociable  USED      Mitsu
 3   2022       39000  37.9 Quito     Negociable  USED      Toyot
 4   2008      224000  10.9 Quito     Negociable  USED      Nissa
 5   2013      151000  26.5 Riobamba  Negociable  USED      Ford
 6   2019       76158  14.9 Quito     Negociable  USED      DongF
 7   2020       32000  17.9 Quito     Negociable  USED      Nissa
 8   2017      218000  22   Guayaquil Negociable  USED      Hyund
 9   2018       22000  25.9 Quito     Negociable  USED      Ford
```

Mutating columns with `mutate()`

- The `mutate()` function is used to create new columns in a data frame.
- In base R, you would use the $ operator to create new columns, based on them being a new column or a transformation of an existing column.

The used cars example

```
# Mutate columns in the cars dataset

cars %>%
    janitor::clean_names() %>%
    mutate(precio_round = round(precio, 2))
```

```
# A tibble: 9,021 x 27
     ano kilometraje precio lugar     negociacion categoria marc
   <dbl>       <dbl>  <dbl> <chr>     <chr>       <chr>     <chr
 1  2016       71000   40.9 Loja      Negociable  USED      Ford
 2  2016       98000   23.9 Quito     Negociable  USED      Mits
 3  2022       39000   37.9 Quito     Negociable  USED      Toyo
 4  2008      224000   10.9 Quito     Negociable  USED      Niss
 5  2013      151000   26.5 Riobamba  Negociable  USED      Ford
 6  2019       76158   14.9 Quito     Negociable  USED      Dong
 7  2020       32000   17.9 Quito     Negociable  USED      Niss
 8  2017      218000   22   Guayaquil Negociable  USED      Hyun
```

Transmuting

- The `transmute()` function is used to create new columns and drop the old ones.
- It is a combination of `mutate()`, followed by `select()` and is useful when you want to create new columns and drop the old ones.

The used cars example

```
# Transmute columns in the cars dataset

cars %>%
    janitor::clean_names() %>%
    transmute(anio = ano,
              precio,
              precio_round = round(precio, 2))
```

```
# A tibble: 9,021 x 3
    anio precio precio_round
   <dbl>  <dbl>        <dbl>
 1  2016   40.9         40.9
 2  2016   23.9         23.9
 3  2022   37.9         37.9
 4  2008   10.9         10.9
 5  2013   26.5         26.5
 6  2019   14.9         14.9
```

Arrange rows with `arrange()`

- The `arrange()` function is used to reorder rows in a data frame.
- In base R, you would use the `order()` function to reorder rows.
- The `arrange()` function is more flexible and allows you to reorder rows based on multiple columns.
- By default, `arrange()` arranges rows in ascending order.
  - Use the `desc()` function to arrange rows in descending order.
  - Not pipeable, so must be the last function in the chain.

The used cars example

```
# Arrange rows in the cars dataset

cars %>%
    janitor::clean_names() %>%
    arrange(precio)

# In descending order:

cars %>%
    janitor::clean_names() %>%
    arrange(desc(precio))
```

More on pipes

Types of pipes

- The pipe operator %>% is often called the "magrittr" pipe, named after the `magrittr` package that introduced it.
  - Now, it comes with the `dplyr` package.
- There are other types of pipes in programming languages which are used for different purposes.
  - The "forward pipe" %>% is used to pass the output of one function as the first argument of the next function.
  - The "assignment pipe" %<>% is used to pass the output of one function as the first argument of the next function, but also modifies the original input object.
  - Other advanced pipes are the "tee pipe" `%T>%` and the "exposition pipe" `%$%`.
- We typically can remain with the forward pipe %>% for most of our work.

The brand new native pipe, |>

- The native pipe operator |> was introduced in R 4.1.0 (recent).
- It remains largely the same as the magrittr pipe %>%.
  - Differences emerge in the advanced use of the pipe, such as when using the `.` placeholder.
  - Hadley discusses differences and provides recommendations in this article
- The second edition of R4DS, which we are following, has now been updated to use the native pipe.
  - Hadley claims that as beginner users of the pipe, you will be unaffected by the “change”.
  - The native pipe won’t require you to load `dplyr` to use it.
  - You may turn on the native pipe in RStudio options.

So, will you choose to listen to Hadley?

Figure 4: Hadley

Tidyverse style

Tidyverse vs. base R - some brief comments

- The tidyverse often follows a certain style of programming that is different from base R.
- The tidyverse is typically easier to learn, but it is not the only way to write R code.
- Further, tidyverse code is often more readable and easier to understand than base R code.
  - The use of the pipe operator %>% makes the code more readable and easier to follow (i.e. more *expressive*)

Tidyverse style

- The tidyverse style is based on the following principles:
  - Use the pipe operator %>% to chain operations together.
  - Use the tidyverse functions for data manipulation.
  - Use the `tibble` data structure instead of the `data.frame`.
- In terms of how to work with tidyverse functions, typically we'll find the following:
  - The first argument is a data frame or tibble (i.e. functions are "pipeable").
  - Functions will use underscores to separate words rather than points (which is the base R style).

Writing code tidyverse style

- The tidyverse style guide[2] introduces a set of conventions for writing R code in the tidyverse style.
  - "Code linters" like `lintr` can be used to check your code against the style guide.
  - Install the `styler` package to automatically format your code according to the style guide, with a built-in RStudio GUI.

1. Use only valid names: lowercase, underscore for spacing, no dots, no spaces, no special characters.

```
# Good

this_is_a_good_name <- 10

# Bad

this.is.a.bad.name <- 10
```

[2] A helpful summary can be found in R4DS

Writing code tidyverse style

2. Use spaces to make code more readable! Put spaces on either side of mathematical operators apart from ^ (i.e. +, −, ==, <, etc.), and around the assignment operator (<−).
   - No spaces close to parentheses
   - Ok to add spaces to help with function alignment (generally we like to align the arguments of a function call).

```
# Good

x <- 10 + 5

# Bad

x<-10+5
```

Writing code tidyverse style

3. Pipes should always have a space before it and should typically be the last thing on a line.
   - If the function you're piping into has named arguments, each of these arguments should be on a new line.
   - Indentation is used to make the code more readable.
   - For short piping workflows, it is ok to put everything on one line (i.e. exceptions to rules apply if they make sense).

```
# Good

cars %>%
    filter(marca == "Chevrolet") %>%
    select(precio, lugar) %>%
    filter(precio < 10000,
           lugar == "Quito") %>%
    mutate(precio_round = round(precio, 2),
           precio_log = log(precio))
```

Base R vs. tidyverse

- Base R data manipulation, even though it is not as user-friendly, is still very powerful and used frequently by R developers (as opposed to "applied" users).
- This is because the tidyverse, while very human-readable, can be a bit cumbersome for production code.
    - There is often a trade-off between readability and efficiency.
- Another reason is that tidyverse code often undergoes a lot of "behind-the-scenes" work to make it more readable.
    - This might make it inefficient for some environments.
    - However, some solutions have been proposed (see the `poorman` package).

Base R vs. tidyverse

- Further, tidyverse functions are less stable than base R functions, which have been around for a long time.
  - There are many updates which are not always backwards-compatible.
  - For developers, this can be a problem, so sticking to base R might be a good idea.
- `dplyr` is not the fastest package for data manipulation.
  - Base R isn't either: Hadley suggests taking a look at `data.table` for faster data manipulation (though syntax isn't friendly).
  - Other options such as `bigmemory`, `ff`, `fst`, `tidypolars` are also available.
- In conclusion: the tidyverse is great and fit for many purposes! But as everything, not for all.

More on data importing

Cool data importing: using packages to load data from the web

- The `readr` package can read data from the web using the `read_csv()` function.
- For instance, we may read the used cars dataset from our GitHub repository using the public csv raw link.

```
# Load the cars dataset from the web

cars_web <-
   read_csv("https://raw.githubusercontent.com/laboratoriolide/
```

- The same applies for other `read_*()` functions in the `readr` package.

World Bank data

- We may use the `wbstats` package to load data from the World Bank API.
- The `wbstats` package provides a set of functions to search, download, and visualize data from the World Bank API.
  - Use `wb_search()` to search for indicators.
  - Will need the indicator_id to download the data.

Example: World Development Indicators

```
library(wbstats)

# Vector of indicators

indicators <- c(
  'inv_gdp'= 'NE.GDI.FTOT.ZS', #  Gross fixed capital formation
  'gdp_pc' = 'NY.GDP.PCAP.KD', # GDP per capita (constant 2010 U
  'gdp_g'= 'NY.GDP.PCAP.KD.ZG', # GDP per capita growth (annual
  'enrolment' = 'SE.PRM.ENRR' # Gross primary school enrolment
  )

# Download the data with the wbstats package

wb_data <- wb_data(indicator = indicators)
```

Downloading files from the web

- The `download.file()` function can be used to download files from the web.
- The `download.file()` function takes two arguments: the URL of the file and the path where you want to save the file.
- The `unzip()` function can be used to unzip files.

```
# Download a file from the web

options(timeout = 900)

download.file("https://mercadodevalores.supercias.gob.ec/reporte
              destfile = "data/downloads/supercias_downloaded.xl
```

Packages to access data

- Many other packages allow you to access data from the web, such as:
  - `rvest`: for web scraping.
  - `statcanR` and `cansim`: for accessing Statistics Canada data.
  - `WDI`: for accessing World Bank data, exclusively World Development Indicators.
  - `rgovcan`: for accessing open data from the Government of Canada.
  - `tidycensus`: for accessing US Census data[3]

[3] There is plenty on accessing US Census data. See this book and this webpage for more.

Packages to access data

- Can directly access data in CRAN packages too.
    - Stored data in CRAN packages can be accessed with the `data()` function.
    - The `datasets` package contains many datasets that are useful for learning R.
    - The `nycflights13` package contains information about all flights departing from New York City in 2013.
    - The `wooldridge` package contains datasets from the Wooldridge Econometrics textbook.

APIs

- Application Programming Interfaces (APIs) are a set of rules and protocols that allow one software application to interact with another.
- APIs are used to access data from the web, and many organizations provide APIs to access their data.

APIs

- While working with APIs is a programming skill in and of itself, there are many R packages that make it easier to work with APIs.
  - The `httr` package is a low-level package for working with web APIs.
  - The `jsonlite` package is used to work with JSON data, which is a common data format used by APIs.
  - `spotifyr` is a package that allows you to access data from the Spotify API for music charts.
  - `lastfmR` is a package that allows you to access data from the Last.fm API (though not available on CRAN).
  - `geniusR` is a package that allows you to access data from the Genius API for music lyrics.