Introduction to R and Rstudio

AUTHORS
Camila Pacheco
Katrín Björnsdóttir

Our mission

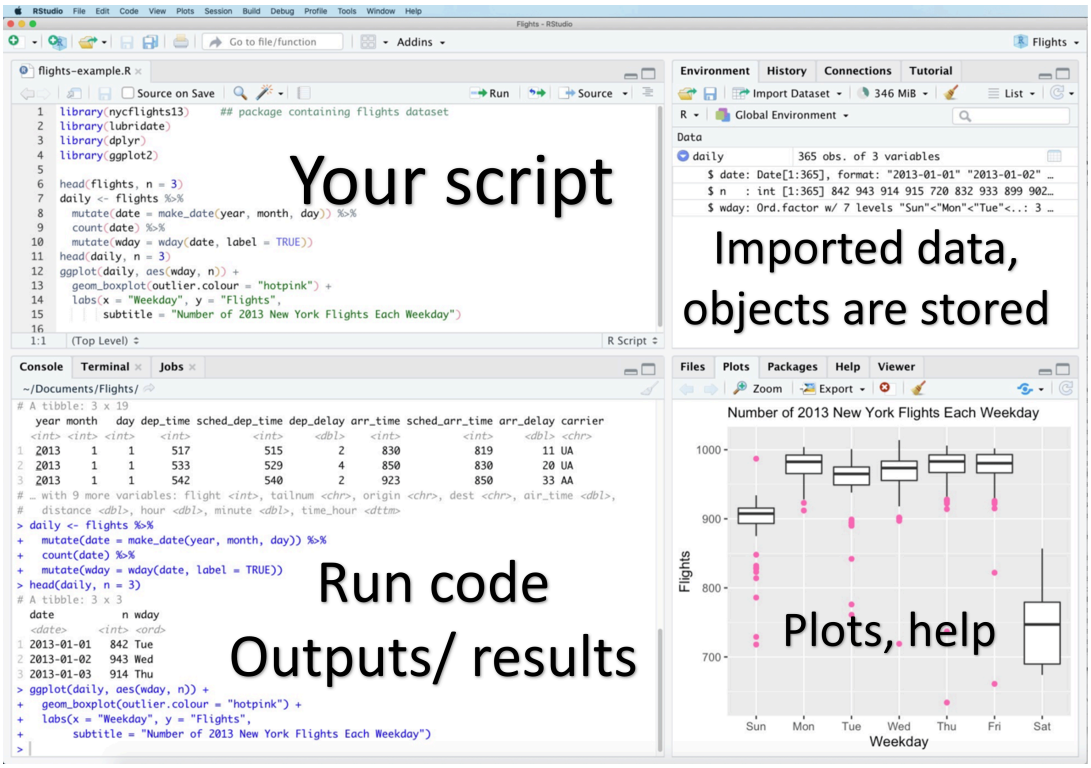
- Get you familiar with the R environment
- Teach you how to import and work with datasets
- Get you familiar with tidy data and *Tidyverse*
- Show you how to make simple plots in *ggplot*

What is R ?

- Originally developed for statistical computing and graphics
- R has evolved into a versatile programming language

What is RStudio?

RStudio is a user-friendly interface for R



R is prone to errors

- 💣 Such as typos, using the wrong letter case, forgetting a quote, bracket or comma. Such mistakes will break your code and throw an error.
- ❗ These type of errors are the most common, so always double-check your code whenever R is unhappy 😣.

R studio Projects

- It is a convenient way to organize your work in RStudio -it creates a dedicated directory (folder) on your computer where you can store all the files related to your project, including R scripts, data files, documentation, and more.
- To create a new project, go to:
  - `File` > `New Project…` > `New Directory` (or `Existing Directory`)
- If you want to create your project from an existing folder:
  - > `New Project` and choose a `Directory name` for your project.

Code versus comment

There are two types of lines: those that start with the symbol `#`, and those that do not.

```r
# This is a comment in R
# Comments are used to provide explanations or annotate the code

x <- 5  # Assigning the value 5 to the variable x
```

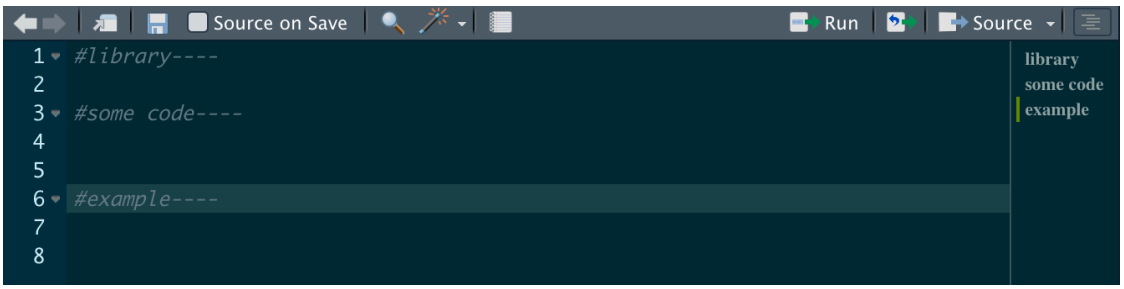
Comments to index your scrips

```r
#Library----

#Some code----

#Example----
```

Script outline in Rstudio



R package

- Package is a collection of R functions, data sets, and other resources bundled together for specific purposes.
- To install a package, type:
  - `install.packages("package-name")`
- **You only need to install packages once**
- Load the packages, type:
  - `library(package-name)`

Example

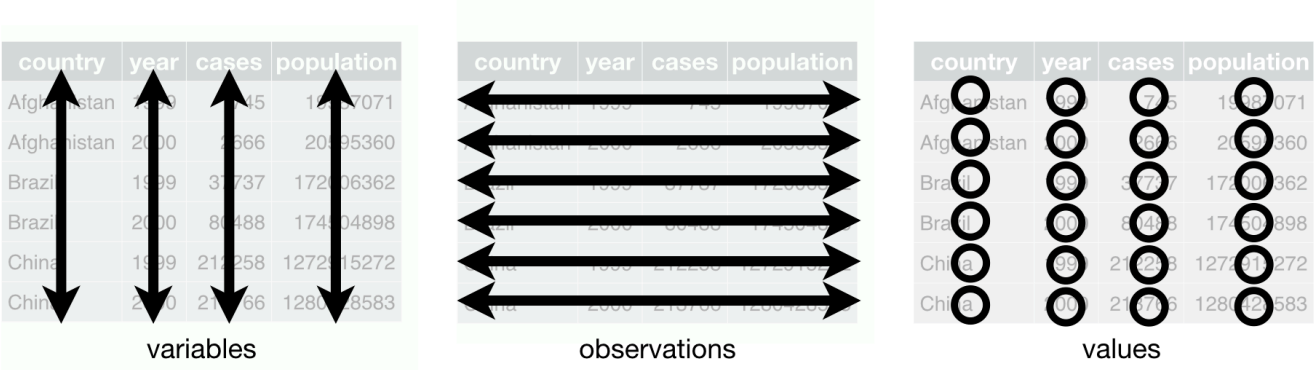
```
install.packages("dplyr")
library(dplyr)
```

Tidyverse

- It is a collection of R packages designed to make data science tasks easier and more efficient.

About tidy data

Tidy data is a standard, consistent way to organize tabular data. Briefly, tidy data follows a short series of rules:



Source: R for Data Science. Grolemund and Wickham.

Pipe operator

- One of the key features of tidyverse is the possibility to chain functions in an effective way using the pipe operator `%>%` or `|>`.
- Pipes pass the results from one function directly into the next function connected to each other via a `%>%` or `|>`, making the code easy to read and write.
- **The pipe basically means "and then".**
- **Note:** the keyboard shorcut for `|>` is `Ctrl+Shift+M` (Windows & Linux) or `Cmd+Shift+M` (Mac).

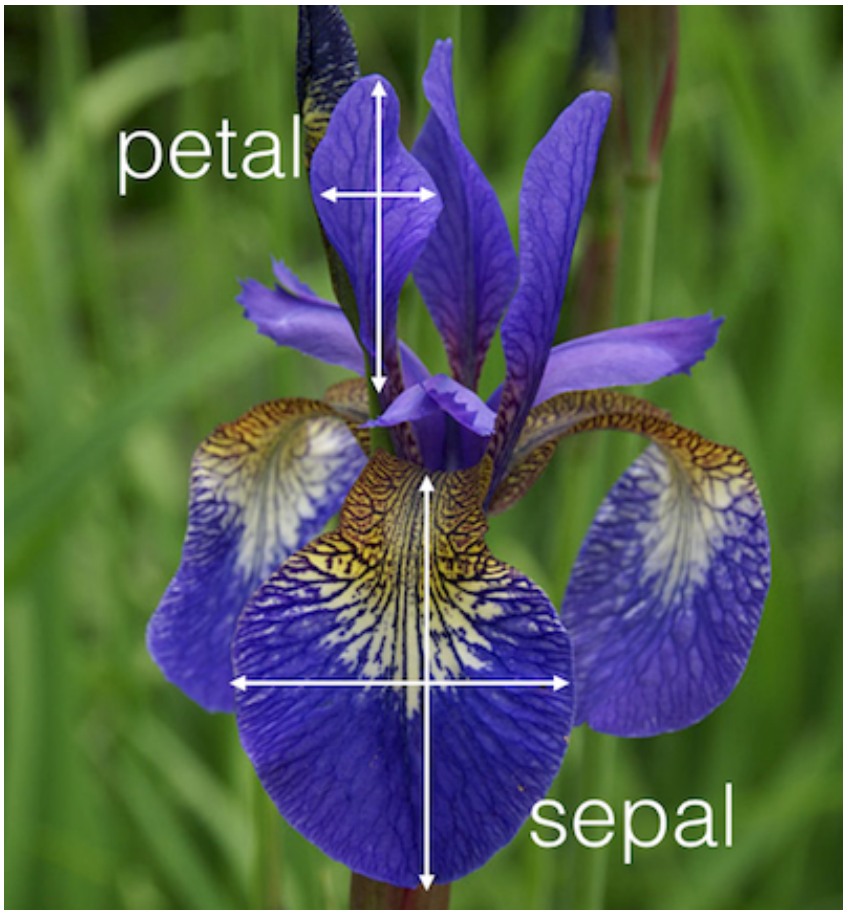
Pipe operator

First, I'll grab the coffee grounds, then I'll fill up the coffee maker with water, hit the start button, wait for it to brew, and finally pour myself a cup

```
get("coffee_groundS","23")  |>
  fill_up("coffee_maker","water")   |>
  on(start_button)  |>
  put(cup)
```

Iris dataset

The iris flower dataset was collected by Edgar Anderson, an American botanist, in the 1920s. This data was used by statistician Ronald Fisher to demonstrate statistical methods of classification.



Importing data in R

R can import data from files in many different formats. For example:

- csv files with the `readr` package
- excel files with the `readxl` package
- xlm files with the `xml2` package
- netcdf files with the `ncdf4` package
- shapefiles with the `sf` package

Formats

Function	Value Separator	Decimal Separator
read_csv()	,	.
read_csv2()	;	,
read_tsv()	tab	.

Function	Value Separator	Decimal Separator
read_delim()	custom character	.
read_table2()	space	.

Reading data

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.0     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
errors
```

```
iris <- read_csv("Data/iris.csv")
```

```
New names:
Rows: 150 Columns: 6
── Column specification
──────────────────────────────────────────────────────── Delimiter: "," chr
(1): Species dbl (5): ...1, Sepal.Length, Sepal.Width, Petal.Length,
Petal.Width
ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
Specify the column types or set `show_col_types = FALSE` to quiet this message.
• `` -> `...1`
```

```
# R basic function
iris <-read.csv("Data/iris.csv")
```

Exploring the data

Check that your data was imported without any mistakes

```
head(iris)    # Displays the first rows of a df
```

```
# A tibble: 6 × 6
   ...1 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <dbl>        <dbl>       <dbl>        <dbl>       <dbl> <chr>
1     1          5.1         3.5          1.4         0.2 setosa
2     2          4.9         3            1.4         0.2 setosa
3     3          4.7         3.2          1.3         0.2 setosa
4     4          4.6         3.1          1.5         0.2 setosa
5     5          5           3.6          1.4         0.2 setosa
6     6          5.4         3.9          1.7         0.4 setosa
```

```
tail(iris)  # Displays the last rows of a df
```

```
# A tibble: 6 × 6
   ...1 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <dbl>        <dbl>       <dbl>        <dbl>       <dbl> <chr>
1   145          6.7         3.3          5.7         2.5 virginica
2   146          6.7         3            5.2         2.3 virginica
3   147          6.3         2.5          5           1.9 virginica
4   148          6.5         3            5.2         2   virginica
5   149          6.2         3.4          5.4         2.3 virginica
6   150          5.9         3            5.1         1.8 virginica
```

Exploring the data

```
glimpse(iris) # Tells you variables types
```

```
Rows: 150
Columns: 6
$ ...1         <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17…
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.…
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.…
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.…
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.…
$ Species      <chr> "setosa", "setosa", "setosa", "setosa", "setosa", "setosa…
```

```
summary(iris) # Gives you a summary of the data
```

```
      ...1         Sepal.Length    Sepal.Width     Petal.Length
 Min.   :  1.00   Min.   :4.300   Min.   :2.000   Min.   :1.000
 1st Qu.: 38.25   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
 Median : 75.50   Median :5.800   Median :3.000   Median :4.350
 Mean   : 75.50   Mean   :5.843   Mean   :3.057   Mean   :3.758
 3rd Qu.:112.75   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100
 Max.   :150.00   Max.   :7.900   Max.   :4.400   Max.   :6.900
  Petal.Width      Species
 Min.   :0.100   Length:150
 1st Qu.:0.300   Class :character
```

```
 Median :1.300   Mode  :character
 Mean   :1.199
 3rd Qu.:1.800
 Max.   :2.500
```

Manipulating data

From wide to long

The iris data are organized is “wide” format. Let’s transform in “long” format

```
iris_long <- iris |> pivot_longer(
                      cols = -Species,
                      names_to = "trait",
                      values_to = "measurement")
head(iris_long)
```

```
# A tibble: 6 × 3
  Species trait        measurement
  <chr>   <chr>              <dbl>
1 setosa  ...1                 1
2 setosa  Sepal.Length         5.1
3 setosa  Sepal.Width          3.5
4 setosa  Petal.Length         1.4
5 setosa  Petal.Width          0.2
6 setosa  ...1                 2
```

Group_by and summarize

```r
iris_means <- iris |>
  group_by(Species) |>
  summarize(SL_mean = mean(Sepal.Length),
            SL_se = sd(Sepal.Length)/sqrt(n()))

head(iris_means)
```

```
# A tibble: 3 × 3
  Species    SL_mean  SL_se
  <chr>        <dbl>  <dbl>
1 setosa        5.01 0.0498
2 versicolor    5.94 0.0730
3 virginica     6.59 0.0899
```

Filter - subset rows

```r
iris_versicolor <- iris |>  filter( Species == "versicolor")

iris_no_versicolor <- iris |>  filter( Species != "versicolor")

iris_pl<-iris |>  filter(Petal.Length > 2)
```

Select - subset columns

```r
iris_subset<-iris |> select(Species, Petal.Width, Petal.Length)
```

Mutate

```r
iris_log <-iris |> mutate(log.Sepal.length = log(Sepal.Length))

head(iris_log)
```

```
# A tibble: 6 × 7
   ...1 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  <dbl>        <dbl>       <dbl>        <dbl>       <dbl> <chr>
1     1          5.1         3.5          1.4         0.2 setosa
2     2          4.9         3            1.4         0.2 setosa
3     3          4.7         3.2          1.3         0.2 setosa
4     4          4.6         3.1          1.5         0.2 setosa
5     5          5           3.6          1.4         0.2 setosa
```

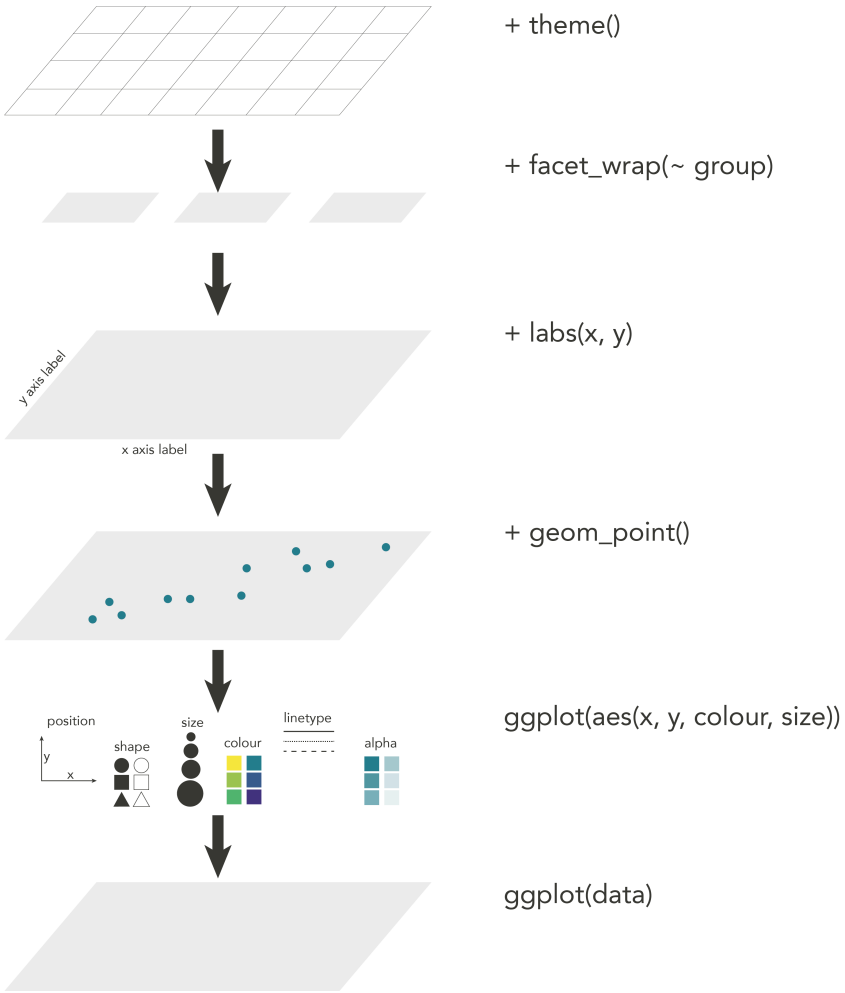
```
6     6          5.4         3.9          1.7         0.4 setosa
# ℹ 1 more variable: log.Sepal.length <dbl>
```

Plotting data

You can plot graphs using the ggplot2 package (part of the tidyverse).

Note: `ggplot` functions are chained using a `+` sign. This is because `ggplot` does not pass an object to a function but add different layers on top of each other.

ggplot layers



Source: https://biostats-r.github.io/

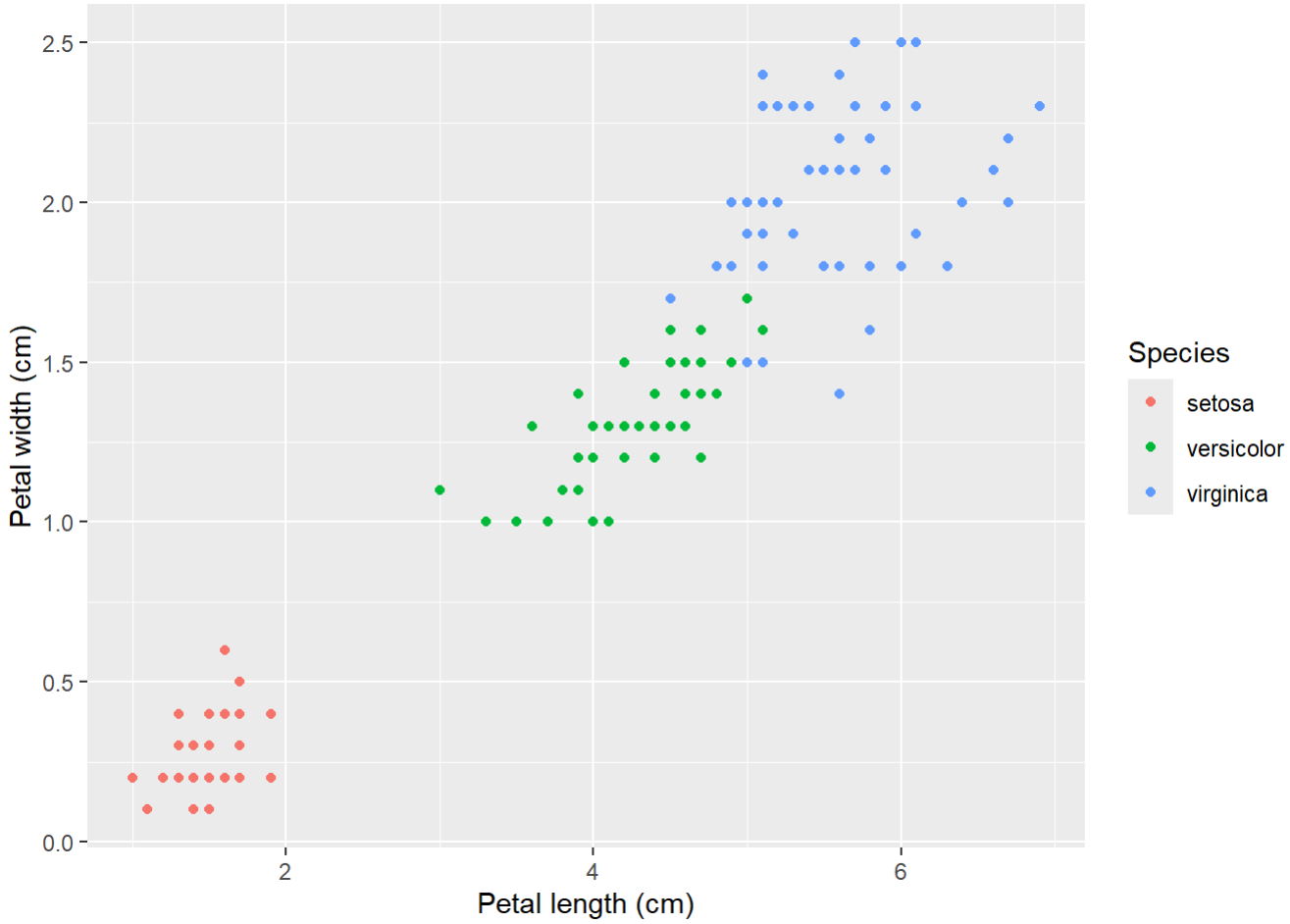
Plotting Iris

Lets say we want to plot the relationship between petal length and petal width for each species.

You can visualize such relationship by plotting petal length against petal width in a scatter plot:

```r
iris %>%
  ggplot(aes(
    x = Petal.Length,
    y = Petal.Width,
    color = Species # to assign a color to each group
  )) +
  geom_point() + # to plot a scatter plot
  labs(
    x = "Petal length (cm)",
    y = "Petal width (cm)"
  )
```

Plotting Iris



Modelling

Saving plots

You can save a plot by clicking on the `Export` button in the `Plots` window (bottom right window by default).

Save your plots as `.svg` if your text editor supports it and if you are not limited by file sizes. Otherwise, save your plots as `.png`.

```
plot |> ggsave("plot.png")
```

To go further

- Visit: Coding Club
- EDGE coding club: Every 15 days, we have a coding club on Mondays at 10:30 in the Supernova room (Natrium)
- On Monday, **May 20th**, there will be a dedicated session for the projects.