Tutorial 0
Introduction to R

Clarke van Steenderen
Department of Zoology and Entomology
Rhodes University
South Africa

clarke.vansteenderen@ru.ac.za

- **R is a [free] programming tool developed in the 1990s for statistical analyses, data manipulation, and graphical output**
- **R Studio is an IDE (integrated development environment) which makes R more user-friendly**
- **Do not be discouraged → it take some time and practice to become used to syntax and usage** `$ # <- / % + :: * - ~ & | [,] %>%`

R hosts thousands of libraries/packages that are added by users (anyone can create a package!)

Each package contains at least one function that runs an analysis, processes data, creates a graphic, etc.

Get more information about a package or function by typing a question mark before its name in R:

```
?ggplot2
?ggplot2::geom_bar
# double colons access a function in a package
```


tibble
config
RIO
tidymodels
renv
plumber
kableExtra
tidyverse
Shiny
forecast
stringr
R6
reticulate
lubridate
devtools
covr
dplyr
data.table
test that
ggplot2
esquisse
golem

Some R syntax

- **Hashes (#) denote comments in code. Whatever follows a # is not run. Use this to annotate your code**

- **Dollar signs ($) are used to access columns in a data sheet, or an attribute of an object. E.g. If we have an Excel sheet with two columns; "`height`" and "`mass`", and we call this data "`in.data`", we can access the height column with:**

```
in.data$height # access height column
```

- **Equal signs (=) or arrows (<-) are synonymous, and mean that we are assigning a name to an object/variable**

Some R syntax

- **If we want to store the height column as a new object, we can do this:**

```
heights = in.data$height
# OR
heights <- in.data$height
```

- **Be careful with names – make them informative, and avoid the use of symbols such as @ * # % etc. in the name itself. Use underscores or full stops to replace spaces**

Some R syntax

- **Reading in data is simple: use `read.csv()` or xlsx::read.xlsx()**
- `iris = read.csv("data/iris_data.csv")`
- **This tells R to fetch the `iris_data.csv` Excel file from the data folder on your PC. Check that the file extension is correct!**
- **Many users get stuck on the file paths to their data. Check where R is currently looking for files by using the `getwd()` command (get working directory)**
  e.g. `getwd()` → `"D/MyDocuments/Rwork/2025"`
- **Set the working directory to the folder where your input files are saved either by using the `setwd()` command, or clicking on Session → Set Working Directory → Choose Directory…. In the R Studio menu**

Some R syntax

- Using `setwd()`:
- `setwd("C/MyDocuments/Honours/R")`
- Your R session is now linked to the R folder in your Honours folder, which is located in MyDocuments, in your C Drive → change according to your setup. Be sure to use / and not \
- `head(iris)# view the first 6 rows of the data`

```
> head(iris)
  Sepal.length sepal.width petal.length petal.width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Some R syntax

- **Accessing a column in `iris` can be done in numerous ways**
- **E.g. access column 2 (sepal width):**

```
iris$sepal_width
iris[ , 2] # all the rows in column 2. If this was iris[1,2],
it would mean the data in row 1, column 2
iris[[2]]
iris[["sepal_width"]]
iris[, "sepal_width"]
dplyr::select(iris, sepal_width)
dplyr::pull(iris, sepal_width)
```

- **Square brackets are used to indicate rows and/or columns, where row number/s are first, followed by column number/s**

```
iris[1:3, 2:4] # rows 1 to 3 and columns 2:4
```

```
> head(iris)
  Sepal.length sepal.width petal.length petal.width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
summary(iris$sepal.length)
```

```
> summary(iris$sepal.length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

```
mean(iris$sepal.length)
```

```
5.843333
```

- **What if we want to subset the data, so that we just have one species (e.g. *versicolor*)?**

```
versicolor.sp = iris[iris$species == "versicolor", ]
```

Or, use the `filter()` function in the `dplyr` package:

```
versicolor.sp = dplyr::filter(iris, species == "versicolor")
```

The `dplyr` package is extremely useful for data manipulation!

The pipe operator: %>% from the magrittr package

- **Shorthand for applying numerous steps/operations to an object**

```
var1 = 1 %>% # start with 1
+ 5 %>% # add 5
sqrt() %>% # square root the answer
divide_by(., 2) %>% # divide by 2
print() # print to screen

[1] 1.224745
```

- **Particularly useful when reading in and processing data frames**

for-loops

- **Very useful when working through items in a list, files in a folder, folders in folders, etc.**

```
my.values = c(100, 250, 300)
my.new.values = c() # create an empty vector

for(p in 1:length(my.values)){
    my.new.values[p] = my.values[p] + 50
}

print(my.new.values)

[1] 150 300 350
```

if (and else) statements

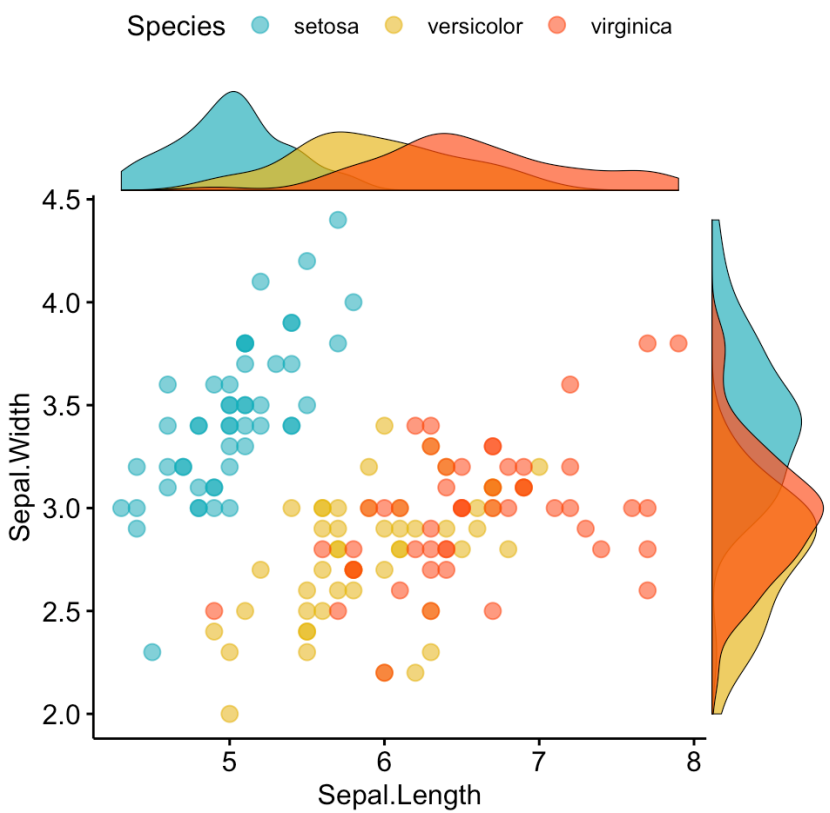
- **Perform operations based on whether conditions are or aren't met**

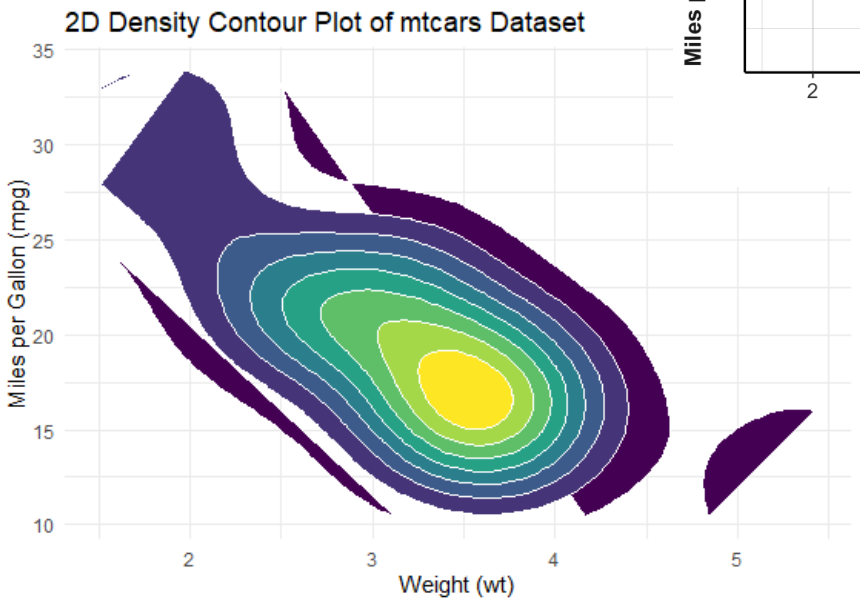
```
my.names = c("Harry", "Dumbledore", "Hagrid",
"McGonnagal", "Snape")

for(n in my.names){
  if(nchar(n) > 6) #nchar means number of characters
    print(n)
}
```

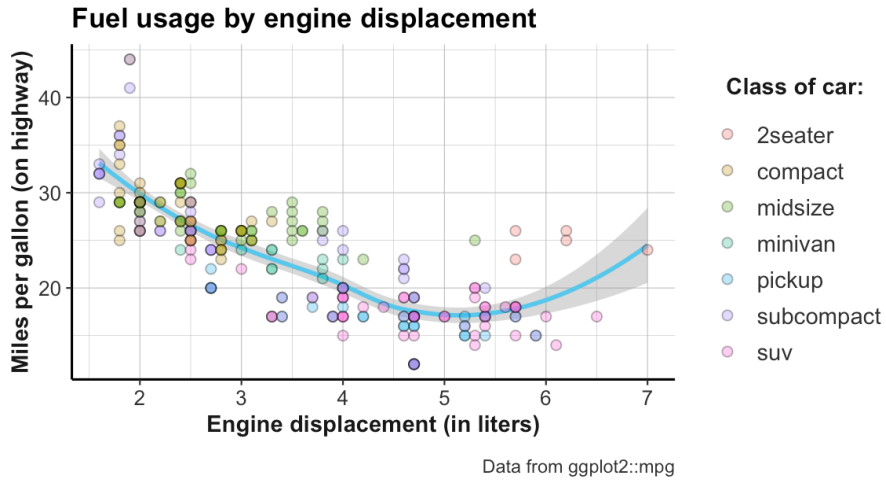
Which names do you expect to be printed?

ggplot

ggplot2 is a graphics package that can create simple to complex plots, all highly customizable







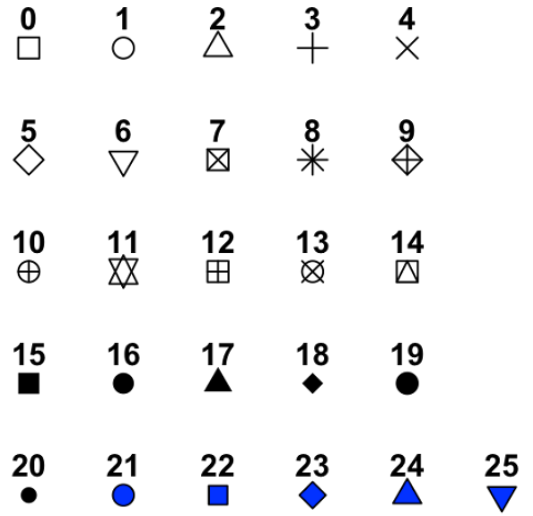
ggplot

ggplot2 works in layers, where you can keep adding additional specifications and tweaks using a plus (+)

```
ggplot2::ggplot(data = data, aes(x = x, y = y)) +
geom_point(shape = 21, fill = "lightblue") +
xlab("X-axis label") +
ylab("Y-axis label") +
theme_classic()
```

aes refers to the aesthetics of the plot

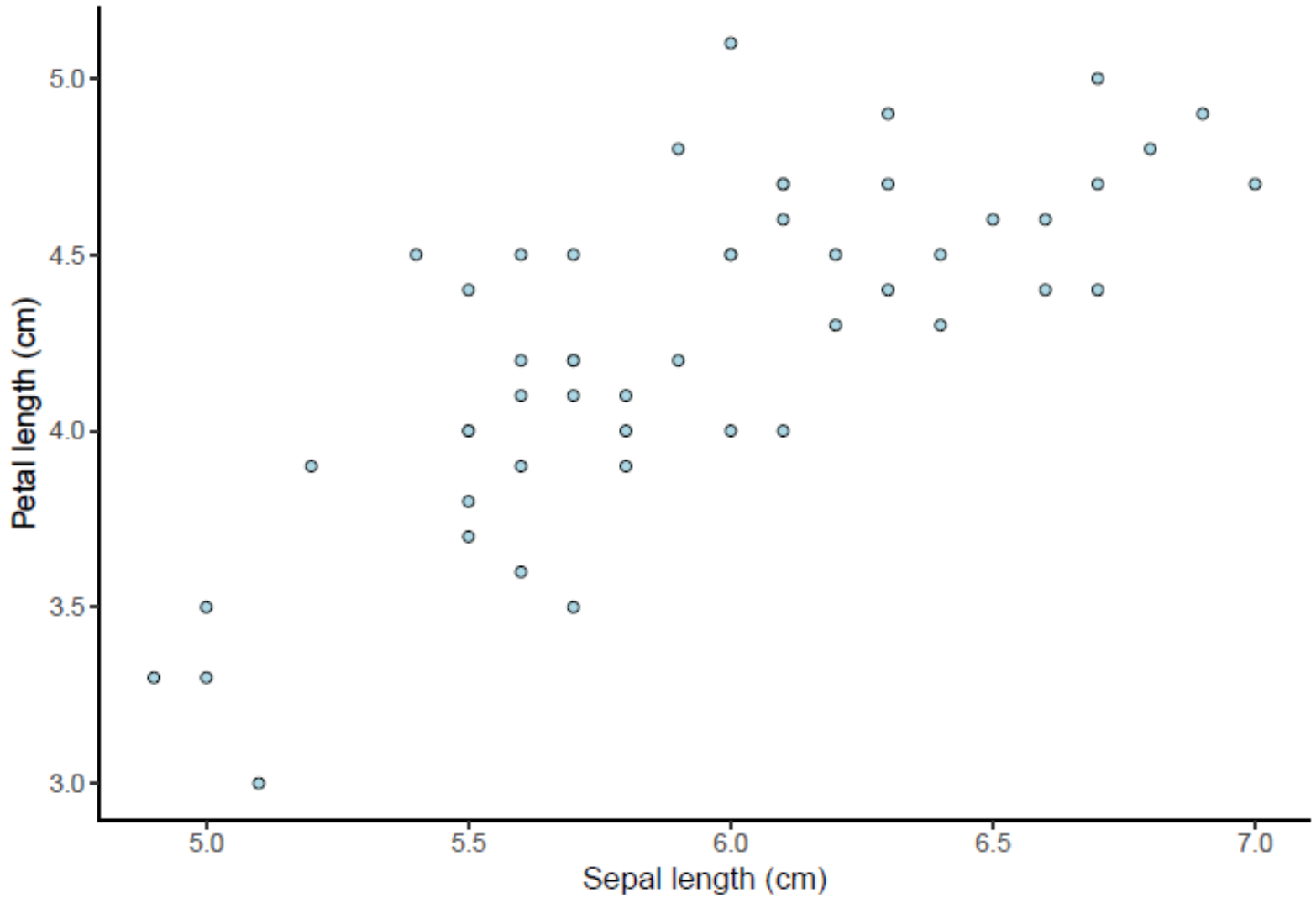
geom_point() tells R that we want a scatter plot



ggplot

To plot our *versicolor* species, we can use this code:

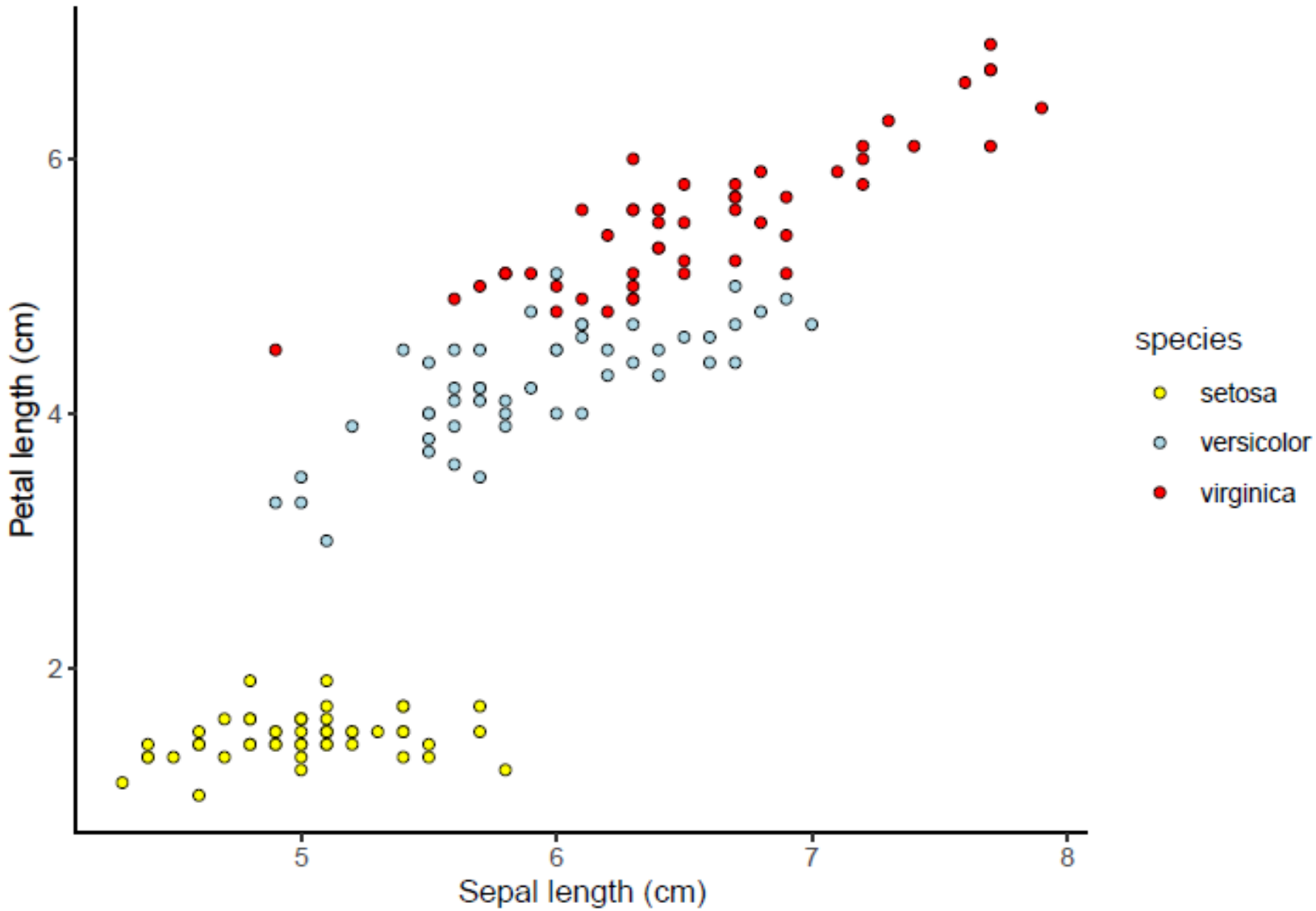
```
ggplot2::ggplot(data = versicolor.sp, aes(x =
sepal_length, y = petal_length)) +
geom_point(shape = 21, fill = "lightblue") +
xlab("Sepal length (cm)") +
ylab("Petal length (cm)") +
theme_classic()
```


Petal length (cm)
5.0
4.5
4.0
3.5
3.0
5.0
5.5
6.0
6.5
7.0
Sepal length (cm)

What if we want to plot all three species?

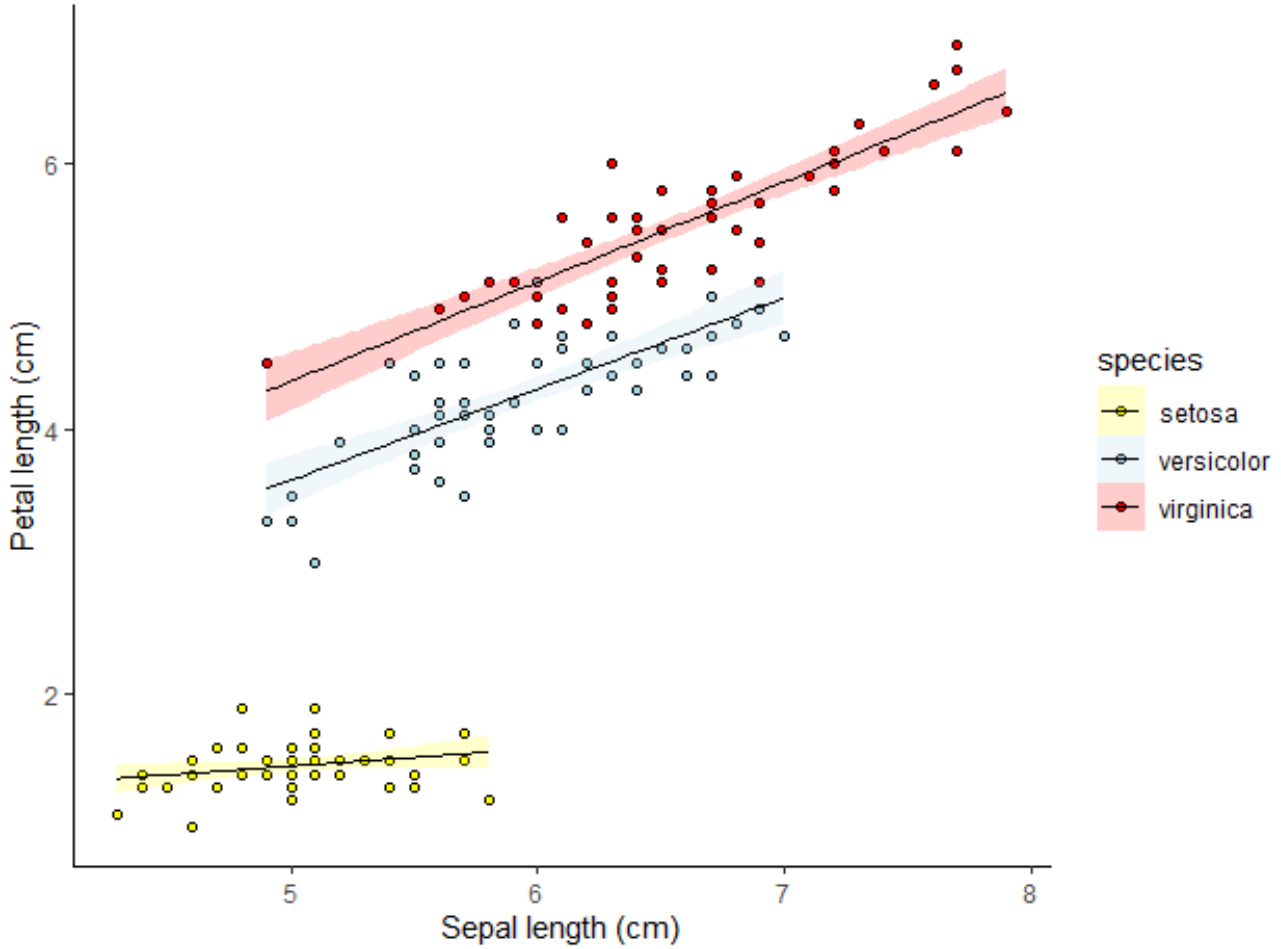
```
iris.point.plot =
ggplot2::ggplot(data = iris, aes(x = sepal_length,
y = petal_length, fill = species)) +
geom_point(shape = 21) +
scale_fill_manual(values = c("yellow",
"lightblue", "red")) +
xlab("Sepal length (cm)") +
ylab("Petal length (cm)") +
theme_classic()
```

`scale_fill_manual()` **allows you to manually specify colours → remember that they are applied to factors in alphabetical order**


Petal length (cm)
6
4
2
5
6
7
8
Sepal length (cm)
species
setosa
versicolor
virginica

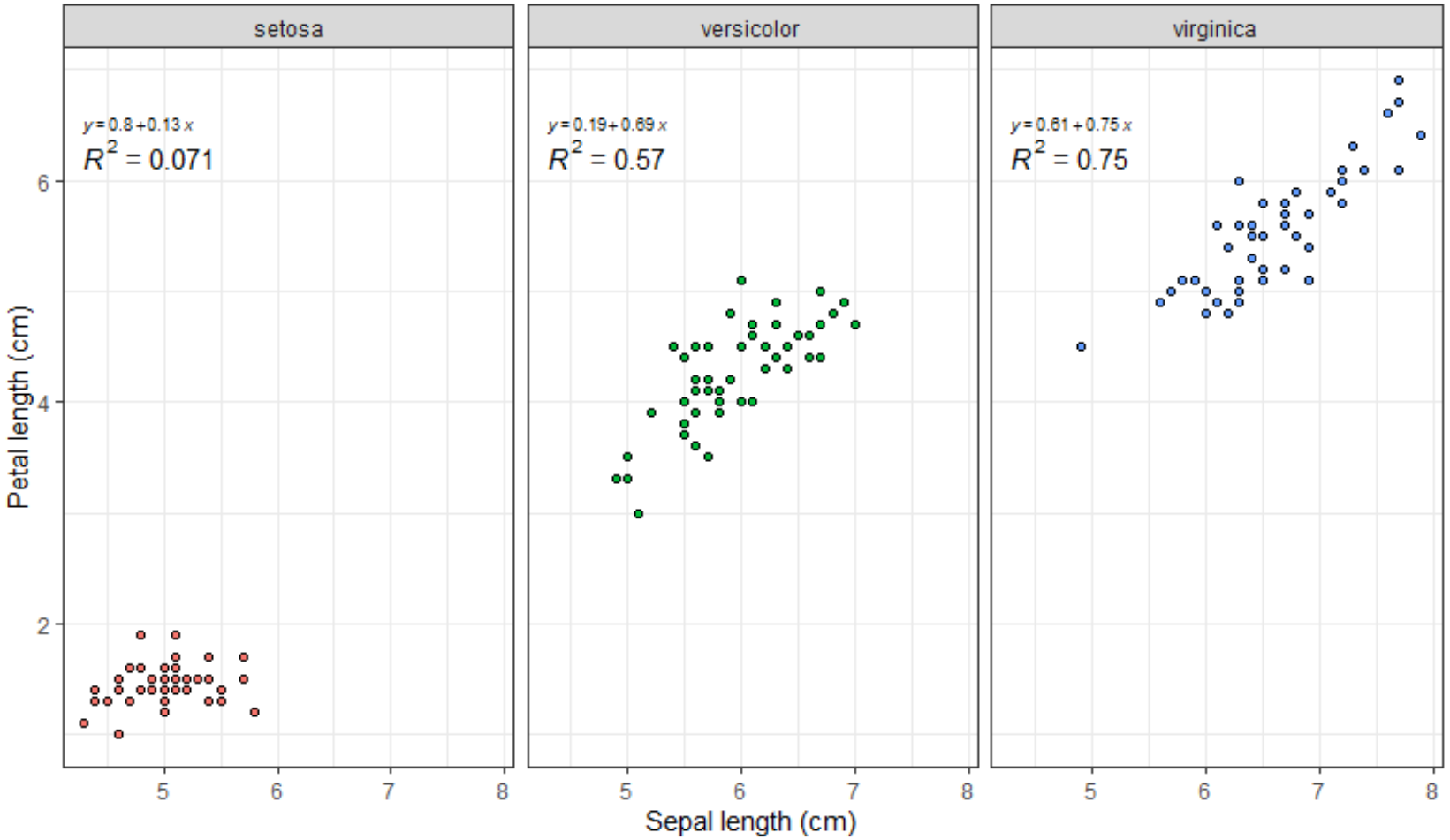
Add trendlines by adding:

```
+
stat_smooth(method = "lm", formula = y~x, geom = "smooth", alpha = 0.2,
colour = "black", linewidth = 0.4, se = TRUE)
```



Separate plots into multiple panels using:

```
+
facet_wrap(~species)
```



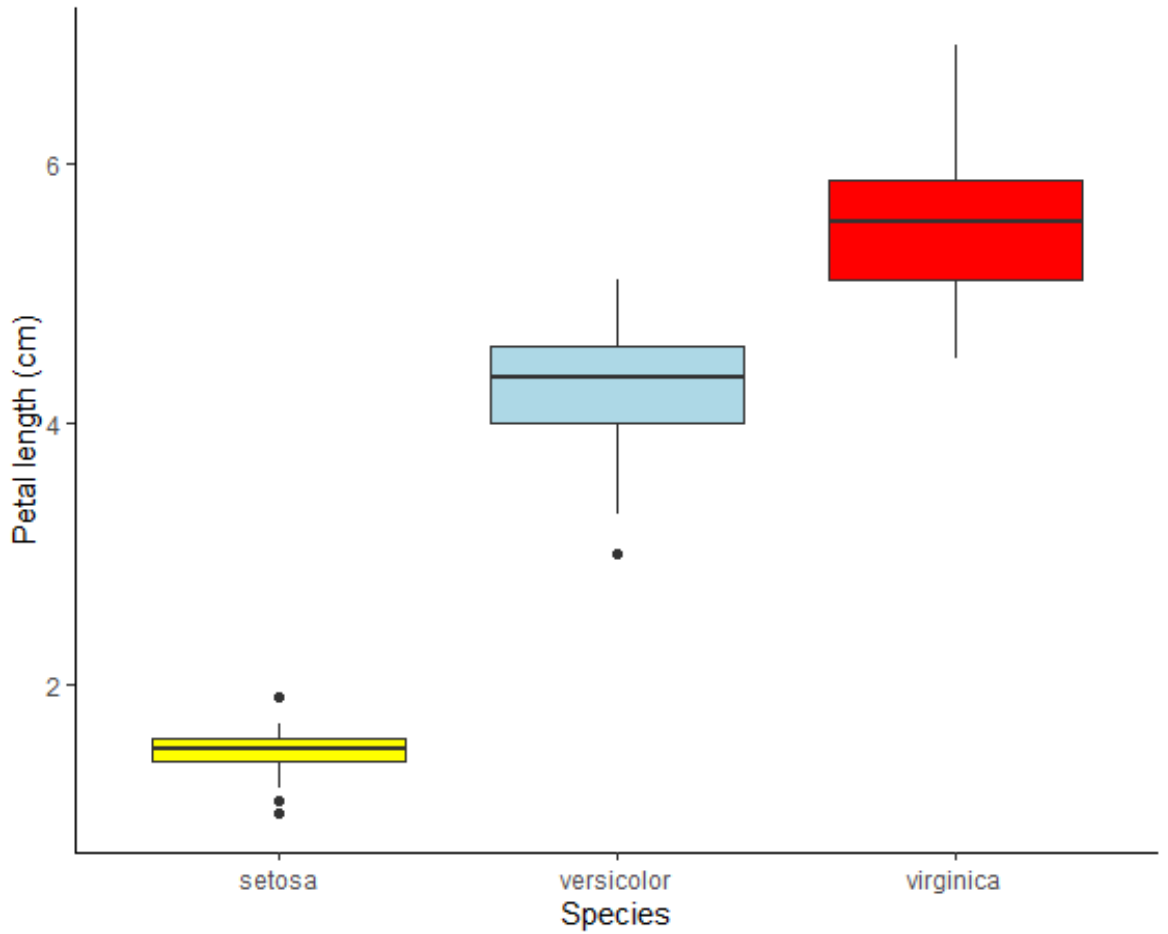
These R^2 values tell you how well the linear trend line fits the data, and falls between 0 and 1

What if we want a box plot for one of the measures?

Instead of `geom_point()`, we use `geom_boxplot()`

```
iris.boxplot =
ggplot2::ggplot(data = iris, aes(x = species,
y = petal_length,
fill = species))+
geom_boxplot() +
scale_fill_manual(values = c("yellow", "lightblue",
"red"))+
xlab("Species")+
ylab("Petal length (cm)")+
theme_classic()
```

Remember what box plots show: Q1, Q2 (median), Q3, maximum and minimum values, and outliers (dots)

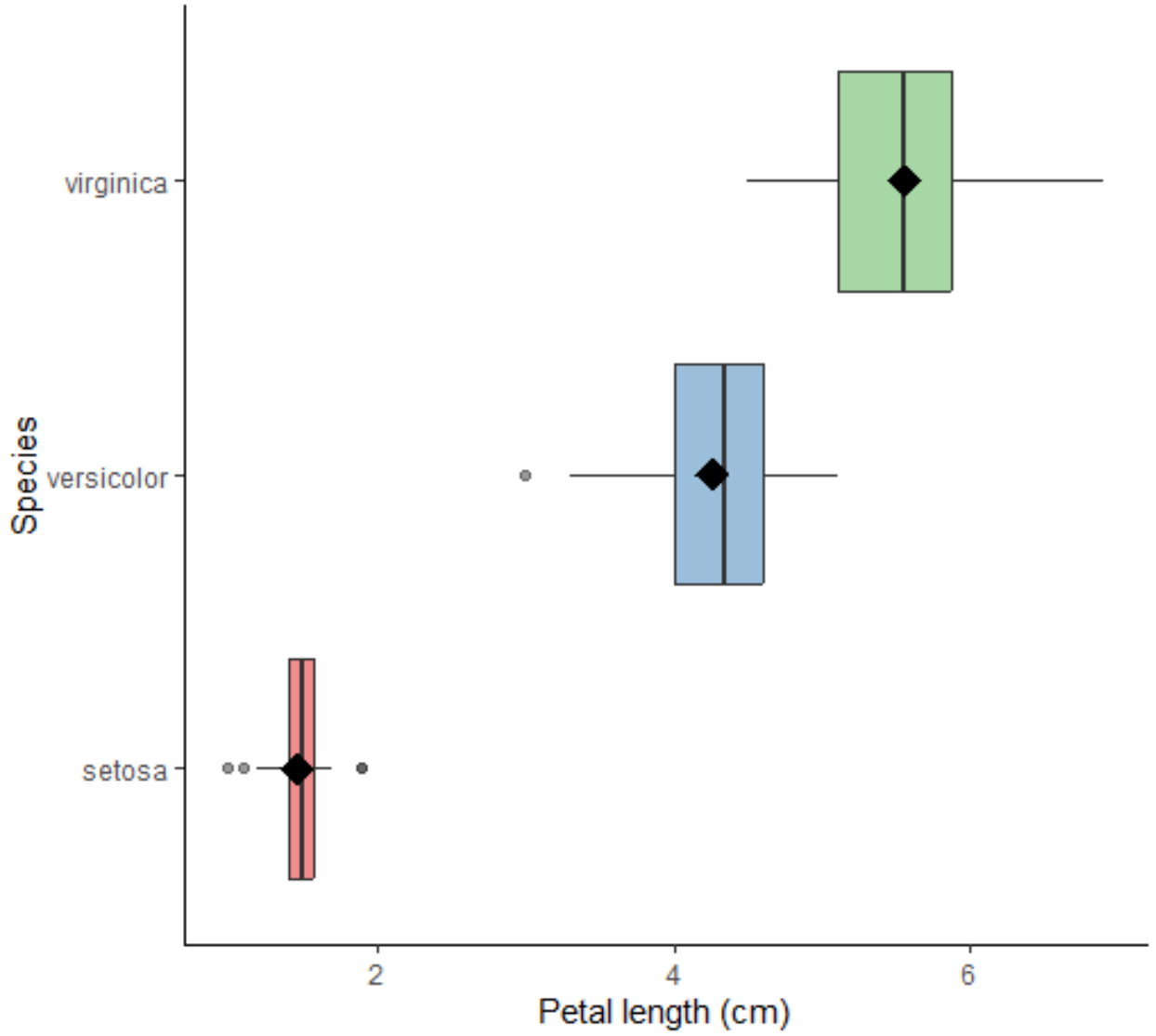


Make the plot a little fancier:

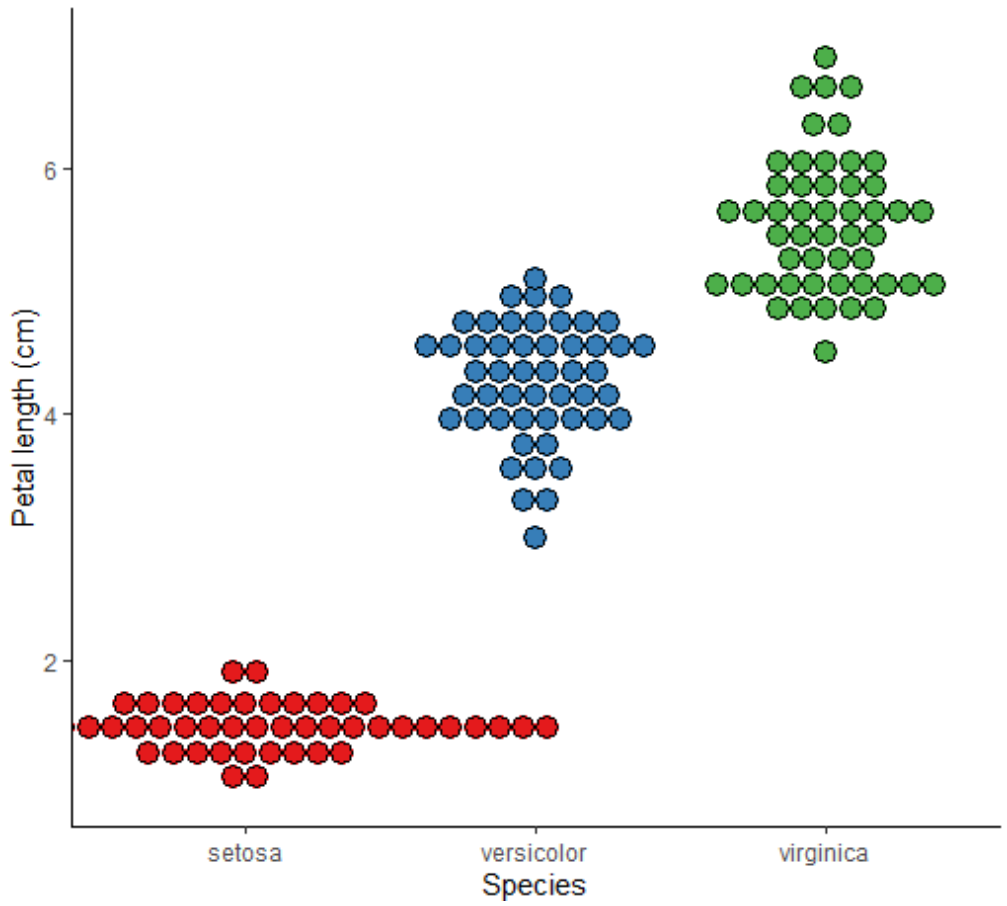
```
iris.boxplot.fancy =
ggplot2::ggplot(data = iris, aes(x = species, y =
petal_length, fill = species)) +
geom_boxplot(alpha = 0.5) +
scale_fill_brewer(palette="Set1") +
stat_summary(fun = mean, geom = "point", shape = 18, size =
5, fill="black") +
theme(legend.position="none") +
xlab("Species") +
ylab("Petal length (cm)") +
theme_classic() +
coord_flip()
```

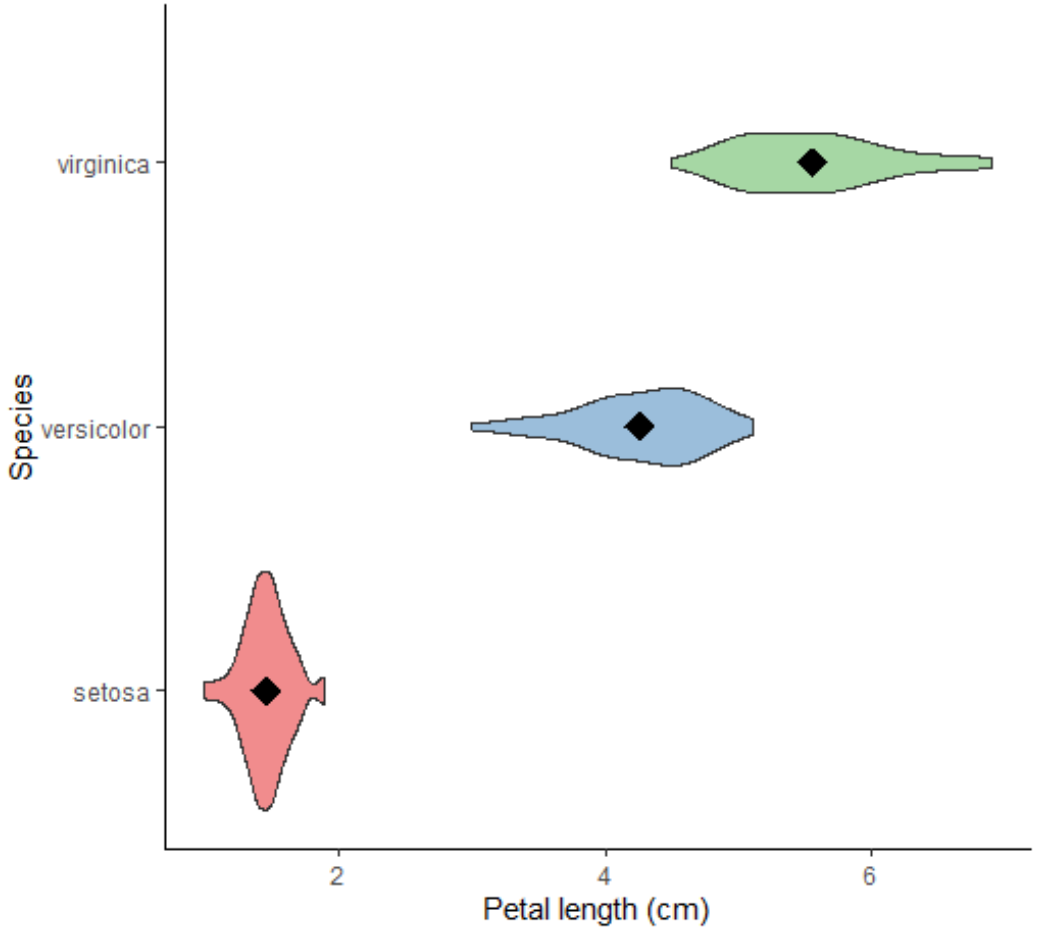
`stat_summary()` **adds the mean values to the plot**

Means are shown as the black diamonds:



Change it up!
See the notes for the code: dot plot and violin plot
This shows more clearly how the data is dispersed





Save your hard-earned plots

```
# save as PNG
dir.create("outputs") # create a folder quickly
ggplot2::ggsave(filename = "outputs/petal_length.png",
plot = iris.boxplot.fancy,
dpi = 350, width = 8, height = 4)
```

Use the `ggsave()` function in the `ggplot2` library to save plots in different formats
Change the extension as required (`.png` `.pdf` `.svg`)

`dpi` refers to dots per inch (resolution)