



# Workshop 3: Introduction to `ggplot2`

QCBS R Workshop Series

Québec Centre for Biodiversity Science



# About this workshop



# Required packages

- [grid](#)
- [gridExtra](#)
- [ggplot2](#)
- [ggsignif](#)
- [ggdendro](#)
- [maps](#)
- [mapproj](#)
- [RColorBrewer](#)
- [GGally](#)
- [patchwork](#)
- [plotly](#)

```
install.packages(c('grid', 'gridExtra', 'ggplot2', 'ggsignif', 'ggdendro', 'maps', 'mapproj', 'RColorBrewer', 'GGally', 'patchwork', 'plotly'))
```

# Learning objectives

1. Use R to create various plots

# Introduction

# Introduction

To follow along:

Code and .HTML available at <http://qcbs.ca/wiki/r/workshop3>

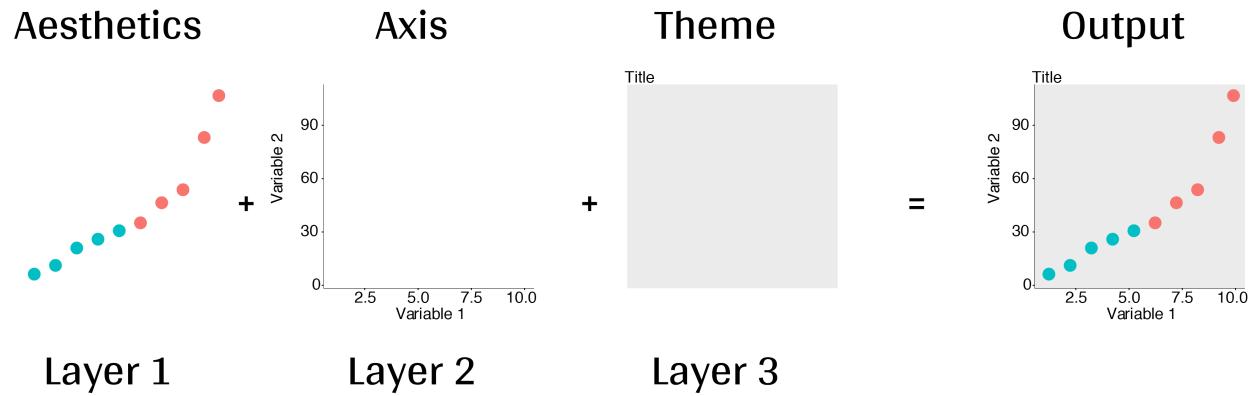
Recommendations:

1. create your own new script;
2. refer to provided code only if needed;
3. avoid copy-pasting or running the code directly from the script.

`ggplot2` is also on GitHub: <https://github.com/tidyverse/ggplot2>

# Outline

## 1. `ggplot2` mechanics



## 2. Advanced visualization

## 3. Fine-tuning

## 4. Saving plots

## 5. Conclusion

# Learning objectives

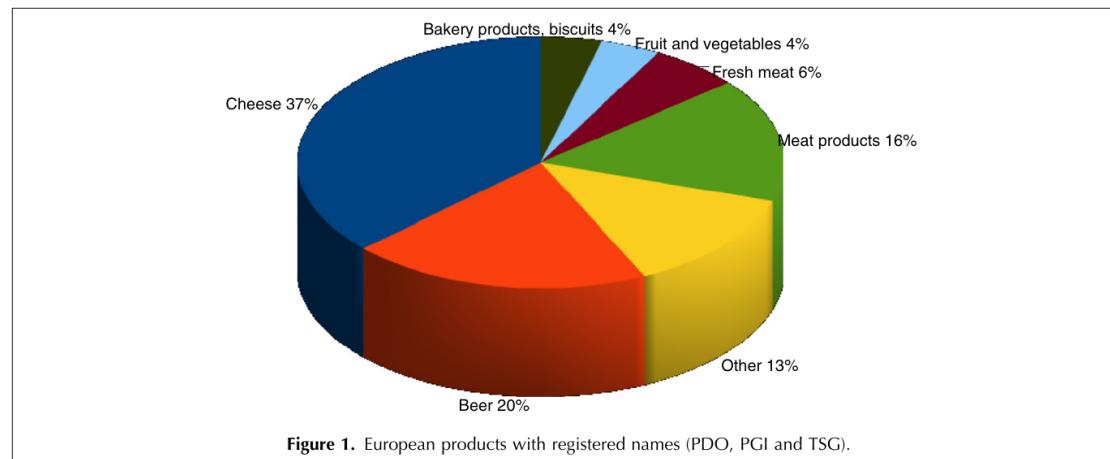
- Teach the basics of data visualization using R.
- Find packages and resources to meet your needs.
- Inspire creativity in science!
- Develop understanding of design for effective graphical communication.

# Visualization in science

# Visualization in science

Why are we using data visualization?

What makes an effective visualization?



What do you think about this one ?

# Visualization in science

1. To represent results of statistical analyses
2. To formulate hypotheses and understand summarize theory
3. To explore your own data (exploratory analysis, outlier detection)
4. To communicate and report
  - Clearly (using good design principles)
  - Precisely and accurately (a plot is worth 1000 words)
  - Effectively and efficiently

# Visualization in science

Important questions:

- What do you want to communicate?
- Who is your audience?
- What is the best way to do it?

**A rule of thumb: think simple — use less ink!**

## Additional resources

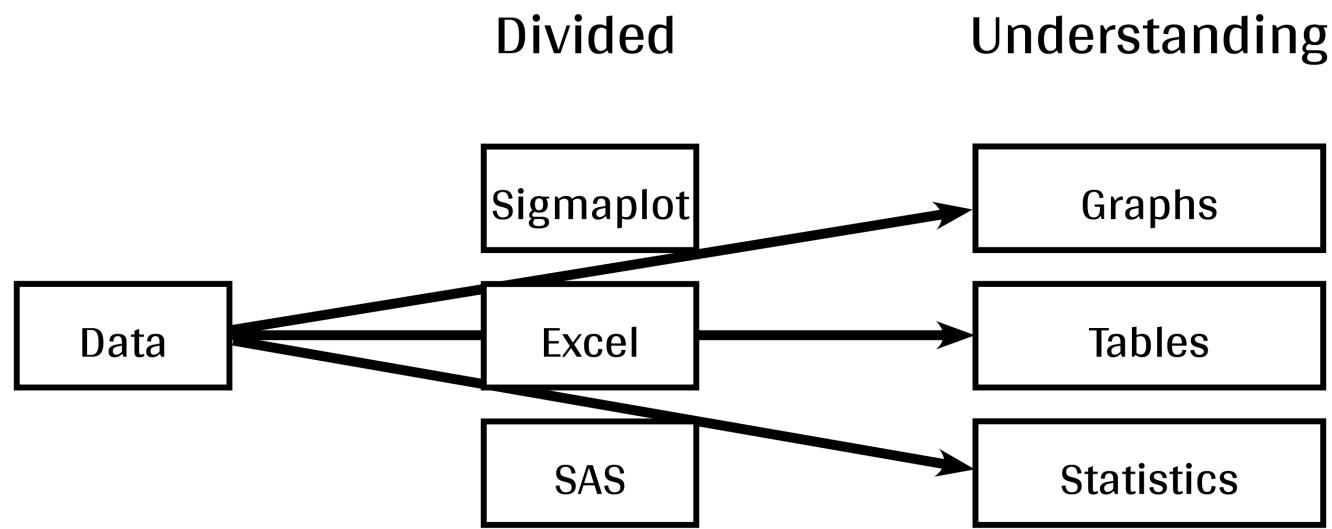
- [Fundamentals of Data Visualization](#) (ggplot)
- [A Compendium of Clean Graphs in R](#) (base plot)
- [Graphics Principles](#) (design tips)

# WARNING!

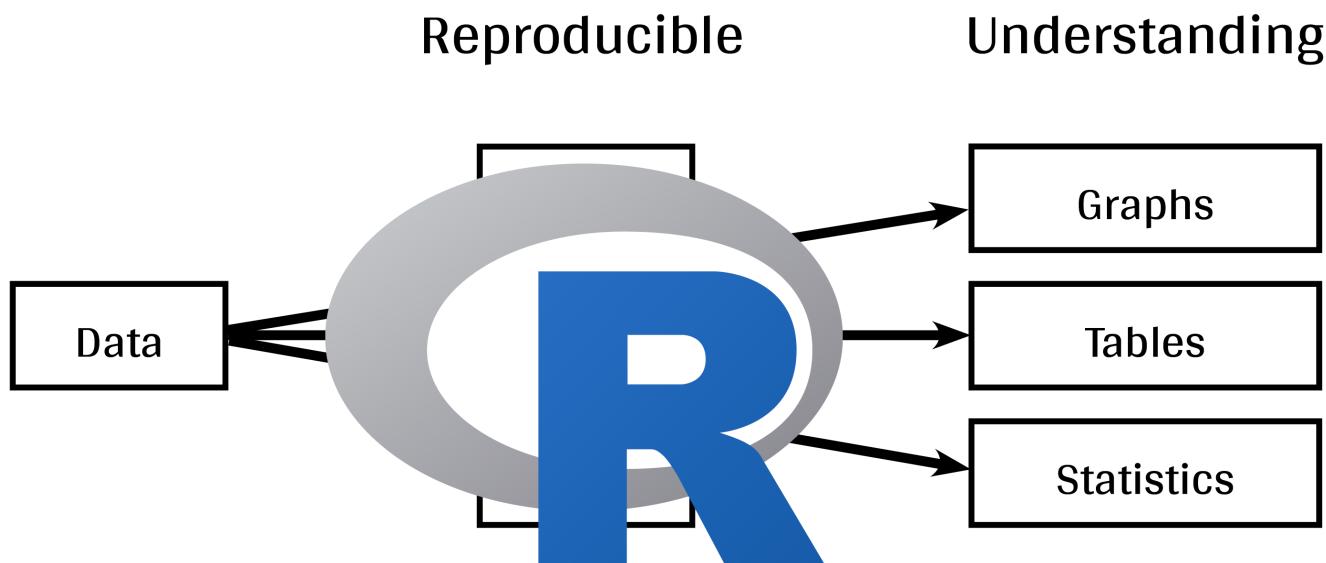
 **R** is not made for drawings.

Other drawing softwares are probably better options such as [GIMP](#) or [Inkscape](#).  
It is important to get the right tool for the right task!

# Why use R for plotting?



# Why use R for plotting? Reproducibility



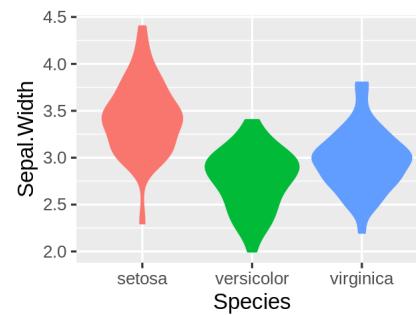
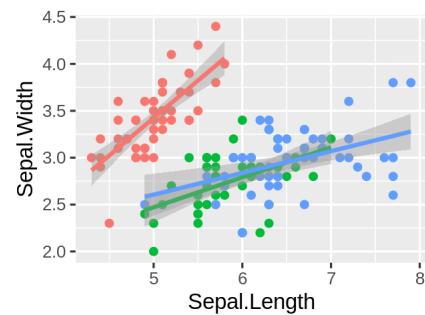
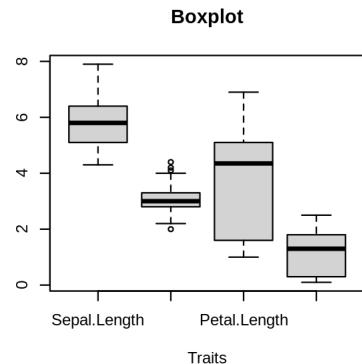
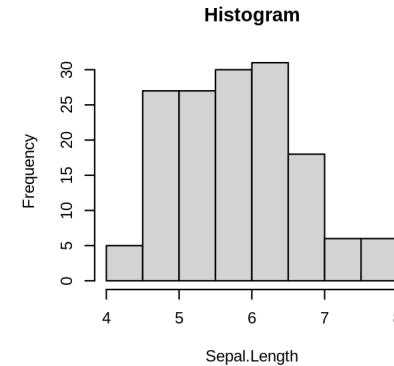
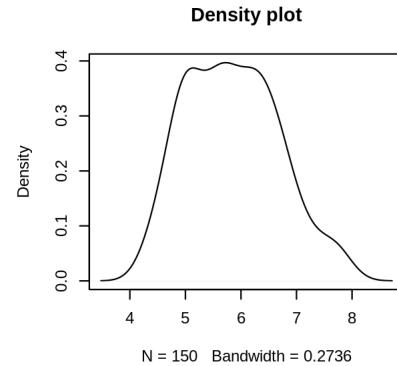
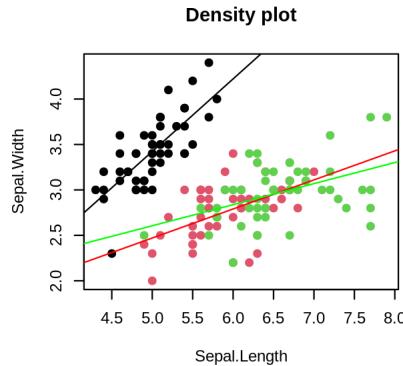
## Reproducible science comes with effort:

- comment your script
- add information in the figures (titles, labels, captions, annotations)

# Why R for graphs?

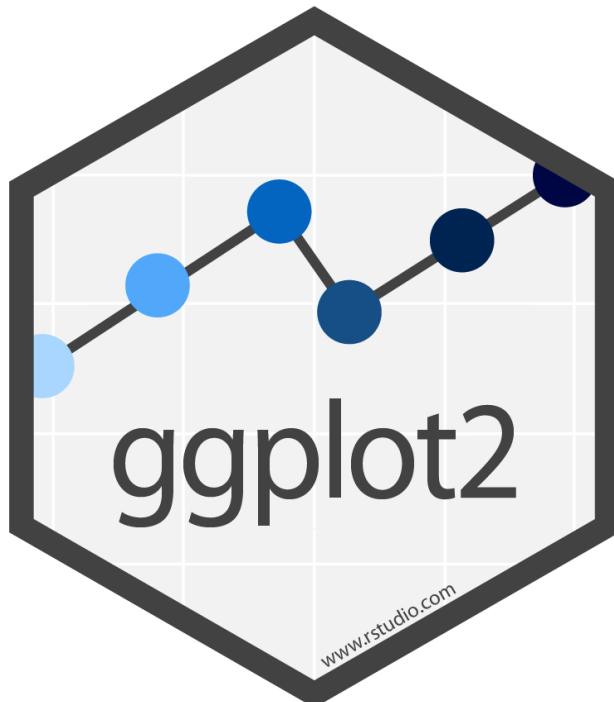
## Because of its powerful features!

In this workshop, we focus only on `ggplot2`, but multiple packages and functions can be used for great visualization (e.g., `base_R`, `plotly`, `sjPlot`, `mapview`, `igraph`).



# ggplot2 is versatile

1. `ggplot2` package lets you make *beautiful* and customizable plots;
2. it implements the grammar of graphics, an reliable system for building plots.
3. it has many extensions.

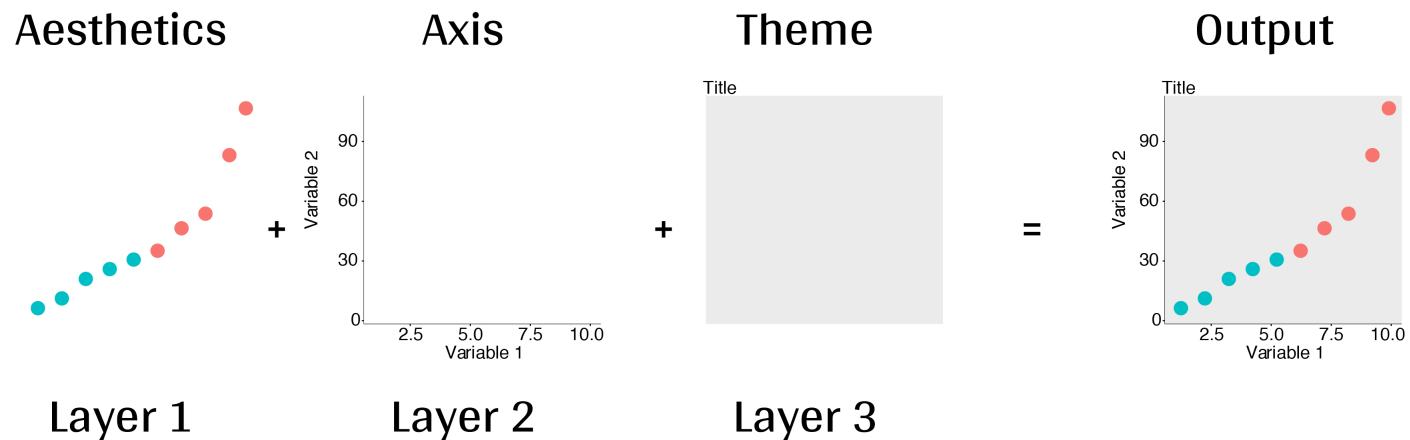


# `ggplot2` mechanics: the basics

# Grammar of Graphics (GG) basics

```
install.packages("ggplot2") # if not already installed  
library(ggplot2)
```

A graphic is made of different layers:



Using the GG system, we can build graphs step by step for customizable results.

# Grammar of Graphics (GG) basics

GG layers have specific names that you will see throughout the presentation:

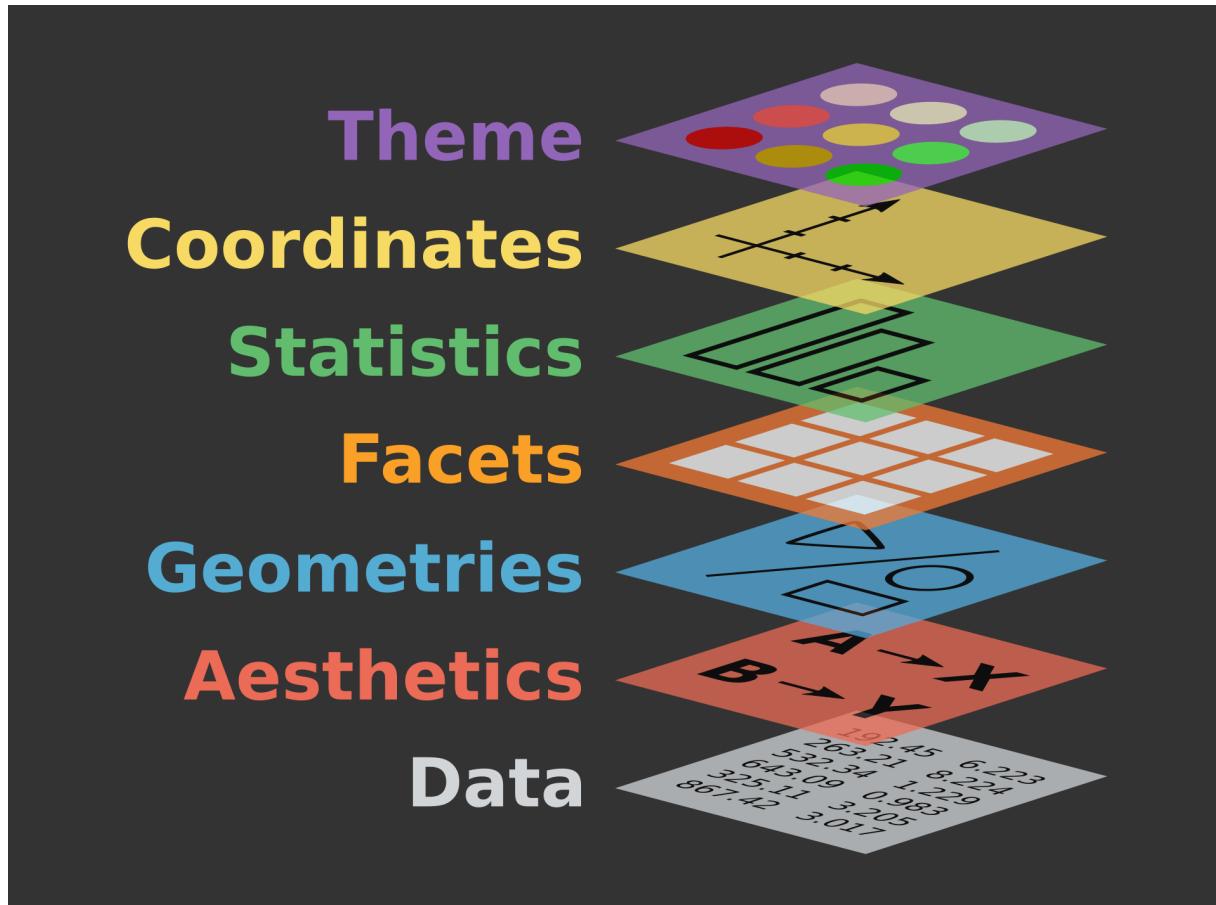


image adapted from [The Grammar of Graphics](#)

# Grammar of Graphics (GG) basics

Here are the basic requirements to draw the simplest `ggplot`:

```
ggplot(data = <DATA>) +  
  <GEOM function>(mapping=aes(<mappings>),  
    stat = <STAT>, position = <POSITION>) +  
  <COORDINATE function> +  
  <SCALE function> +  
  <THEME function> +  
  <FACET function> +...
```

Required

Not required

# Grammar of Graphics (GG)

GG layers:

- Data
  - your data, in tidy format, will provide ingredients for your plot
  - use `dplyr` techniques to prep data for optimal plotting format
  - usually, one row for every observation that you want to plot

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes), to make data visible
  - `x`, `y`: variable along the x and y axis
  - `colour`: color of geoms according to data
  - `fill`: the inside color of the geom
  - `group`: what group a geom belongs to
  - `shape`: the figure used to plot a point
  - `linetype`: the type of line used (solid, dashed, etc)
  - `size`: size scaling for an extra dimension
  - `alpha`: the transparency of the geom

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes), to make data visible
  - `x`, `y`: variable along the x and y axis
  - `colour`: color of geoms according to data
  - `fill`: the inside color of the geom
  - `group`: what group a geom belongs to
  - `shape`: the figure used to plot a point
  - `linetype`: the type of line used (solid, dashed, etc)
  - `size`: size scaling for an extra dimension
  - `alpha`: the transparency of the geom

**note: aesthetics are based on quantities in your data, but plots may have some of these qualities not based in data as well. (ie, colouring based on a data group is an aesthetic, but points will always have a colour even if not based on data.)**

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes)
- Geometric objects (geoms)
  - `geom_point()`: scatterplot
  - `geom_line()`: lines connecting points by increasing value of x
  - `geom_path()`: lines connecting points in sequence of appearance
  - `geom_boxplot()`: box and whiskers plot for categorical variables
  - `geom_bar()`: bar charts for categorical x axis
  - `geom_histogram()`: histogram for continuous x axis
  - `geom_violin()`: distribution kernel of data dispersion
  - `geom_smooth()`: function line based on data

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes)
- Geometric objects (geoms)
- Facets
  - `facet_wrap()` or `facet_grid()` for small multiples

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes)
- Geometric objects (geoms)
- Facets
- Statistics
  - similar to geoms, but computed
  - show means, counts, and other statistical summaries of data

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes)
- Geometric objects (geoms)
- Facets
- Statistics
- Coordinates - fitting data onto a page
  - `coord_cartesian` to set limits
  - `coord_polar` for circular plots
  - `coord_map` for different map projections

# Grammar of Graphics (GG)

GG layers:

- Data
- Aesthetics (aes)
- Geometric objects (geoms)
- Facets
- Statistics
- Coordinates
- Themes
  - overall visual defaults
  - fonts, colors, shapes, outlines

# How layer in ggplot works

1. Create a simple plot object:

- `plot.object <- ggplot()`

2. Add geometric layers:

- `plot.object <- plot.object + geom_*( )`

3. Add appearance layers:

- `plot.object <- plot.object + coord_*( ) + theme()`

4. Repeat step 2-3 until satisfied, then print:

- `plot.object` or `print(plot.object)`

# Prepare data for `ggplot2`

`ggplot2` requires you to prepare the data as an object of class `data.frame` or `tibble` (common in the `tidyverse`).

```
library(tibble)
class(iris) # all set!
# [1] "data.frame"

ir <- as_tibble(iris) # acceptable
class(ir)
# [1] "tbl_df"     "tbl"        "data.frame"
```

♻ Recall from the [Loading and manipulating data workshop](#):

More complex plots in `ggplot` require the long data frame format

# iris dataset

```
## ?iris
str(iris)
# 'data.frame': 150 obs. of 5 variables:
# $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
# $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
# $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
# $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
# $ Species      : Factor w/ 3 levels "setosa", "versicolor", ... 1 1 1 1 1 1 1 1 1 1
```

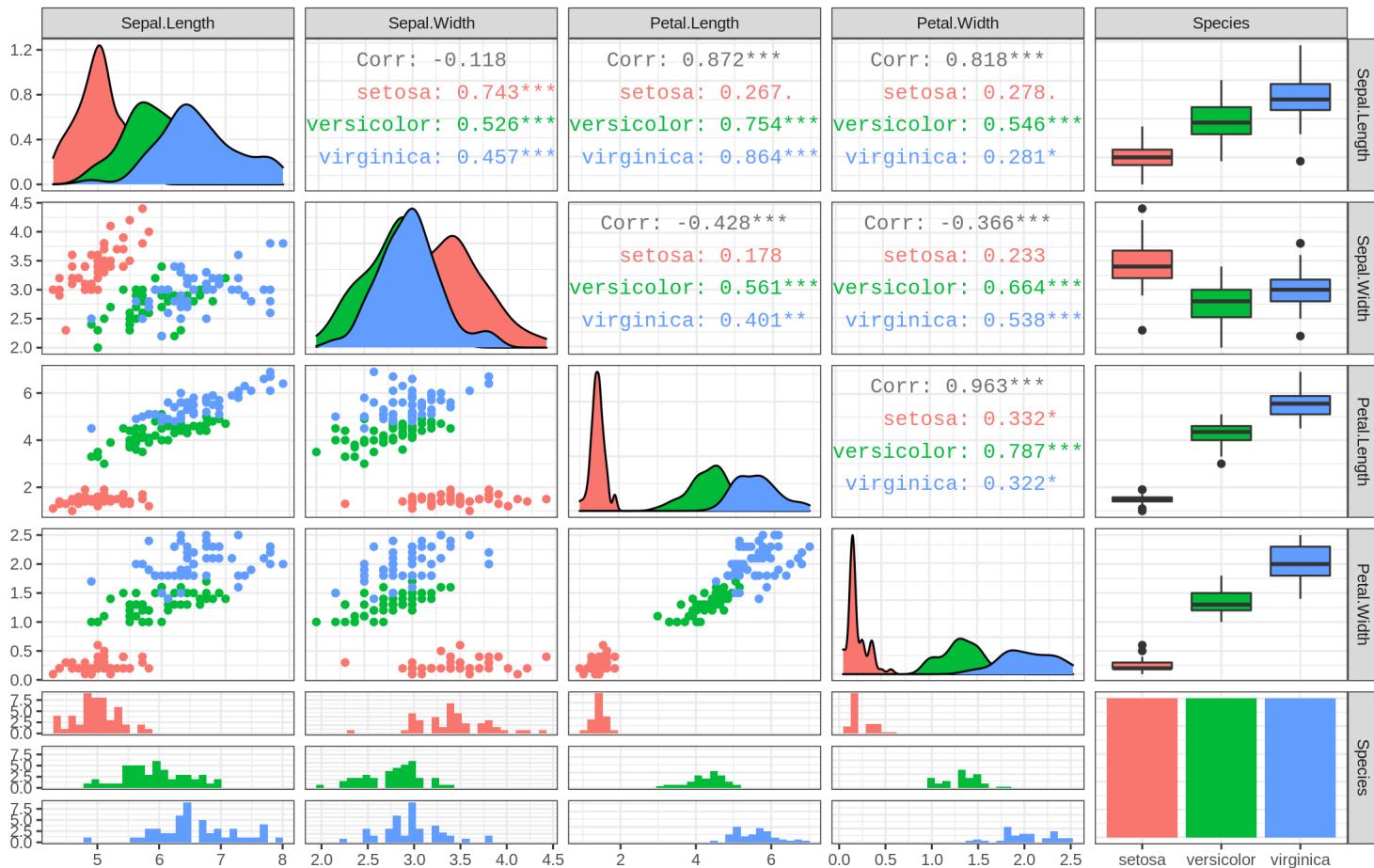
## Scientific questions

- Is there a relation between the **length** & the **width** of the iris **Sepal** ?
  - Does the size of the **Petal & Sepal** vary together ?
  - How are these measures distributed among the **3 Iris species** ?

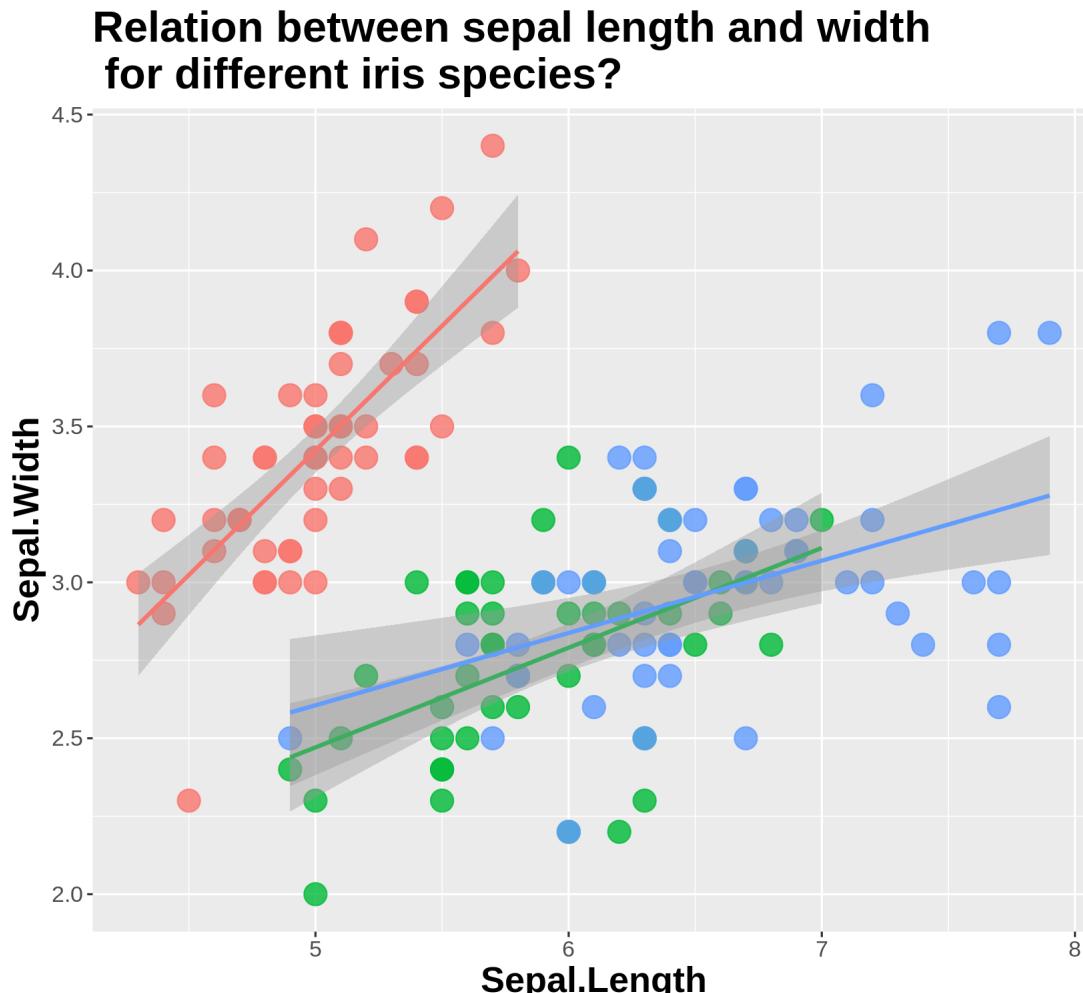
**How can we graphically address these questions with `ggplot`?**

# Exploring data structure

```
install.packages("GGally")
library(GGally)
ggpairs(iris, aes(colour=Species)) + theme_bw()
```



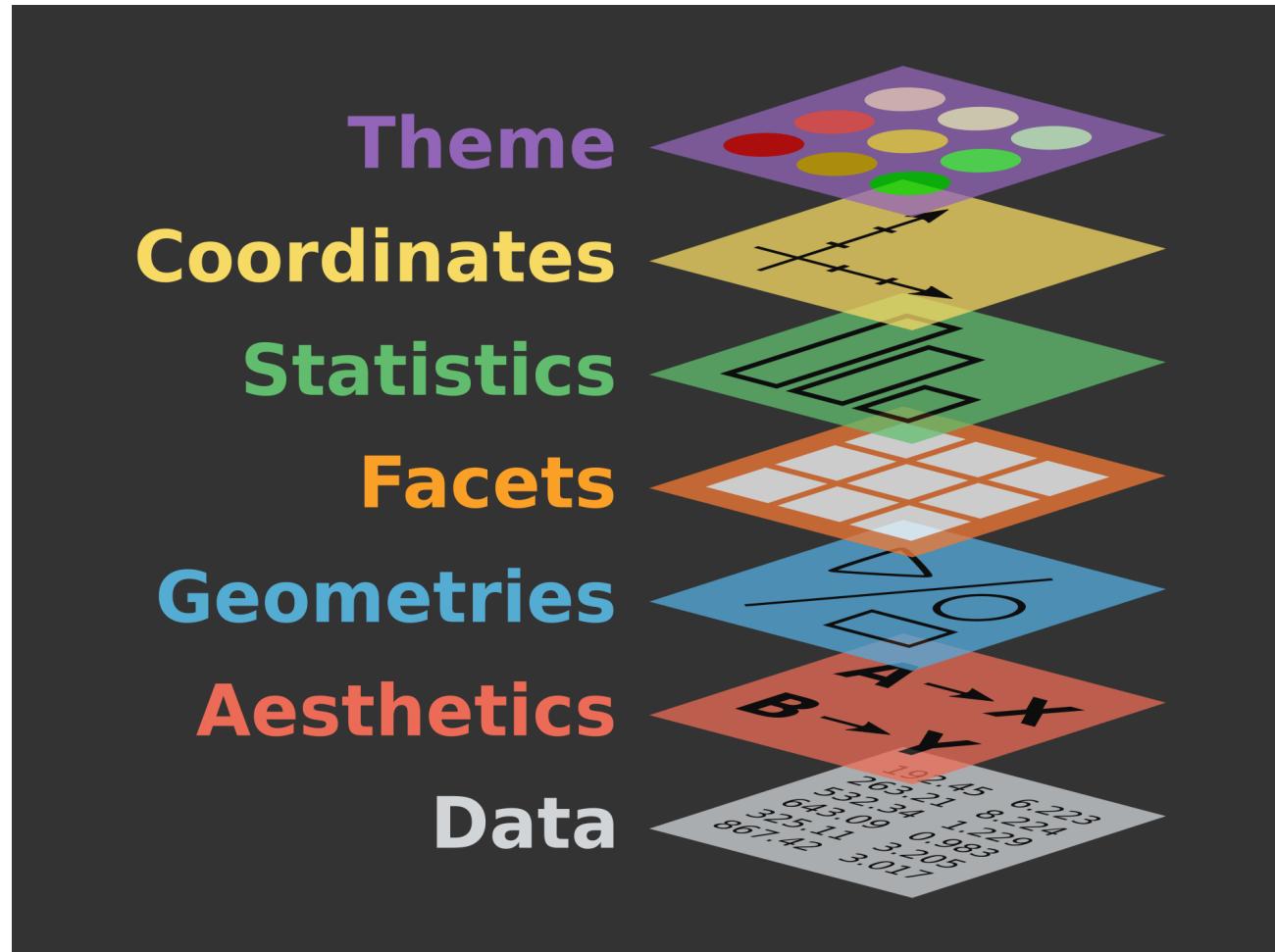
# Exploring data structure



♻ See [Loading and manipulating data workshop](#) to learn how to clean your dataset.

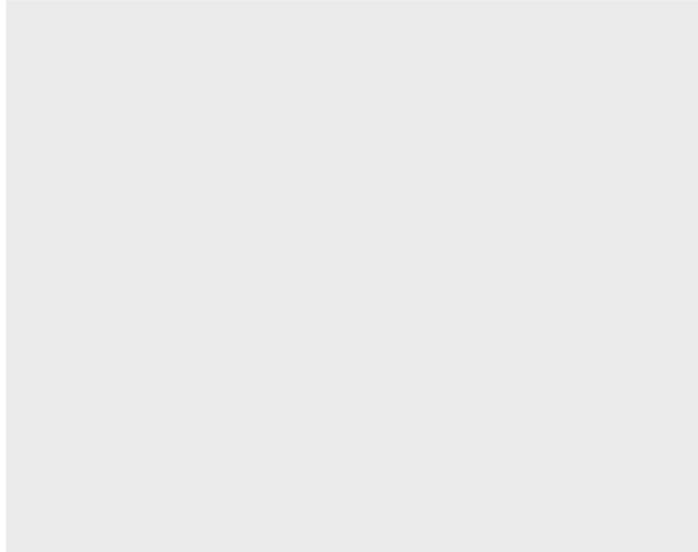
# Grammar of Graphics: *recall*

Remember: a graphic is made of different layers:



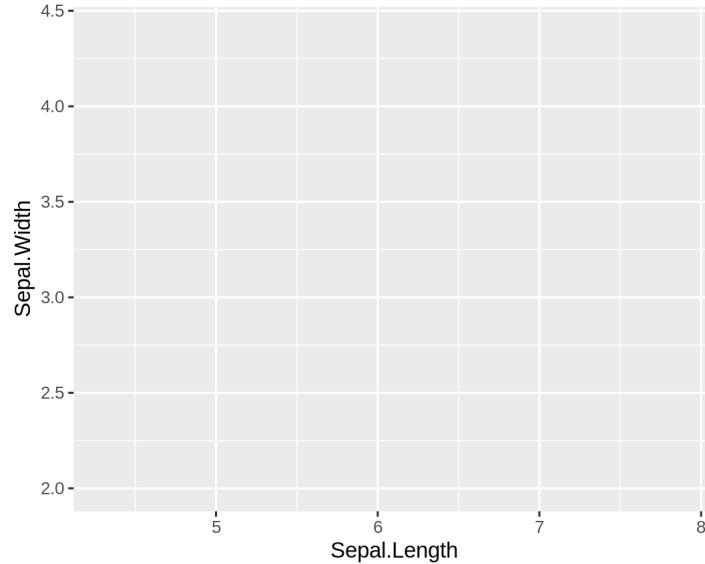
# ggplot() dynamics: data layer

```
ggplot(data = iris)
```



# ggplot() dynamics: aesthetics layer

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width))
```



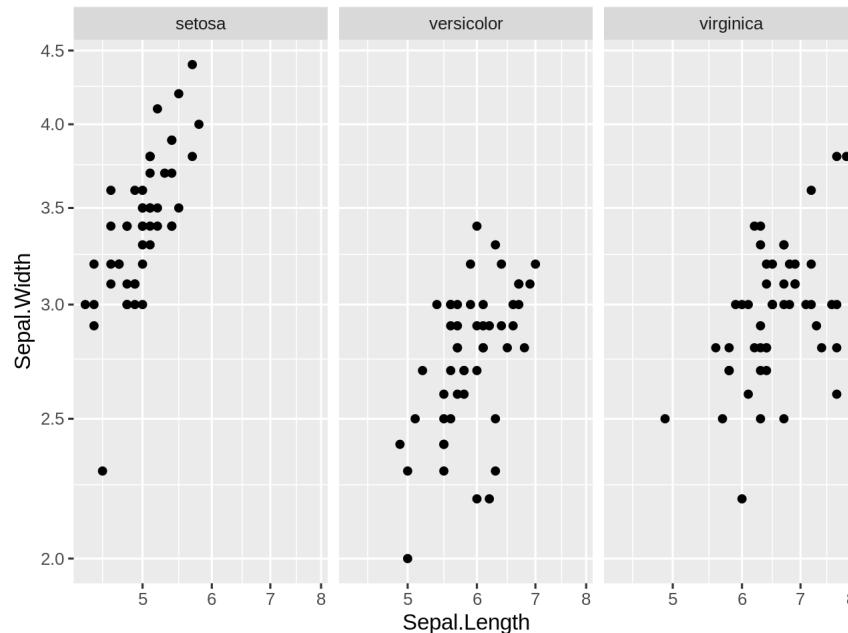
# ggplot() dynamics: geometric Layer

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point()
```



# ggplot() dynamics: extras- facet, stats, coord layers

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point() +  
  facet_wrap(~Species) +  
  coord_trans(x = "log10",  
              y = "log10")
```





# Challenge #1 (5min)

Draw your 1st ggplot!

## Question

Is there a relation between the **length** & the **width** of the iris **petal** ?

Does the *width* of the petal increase with its *length* ?

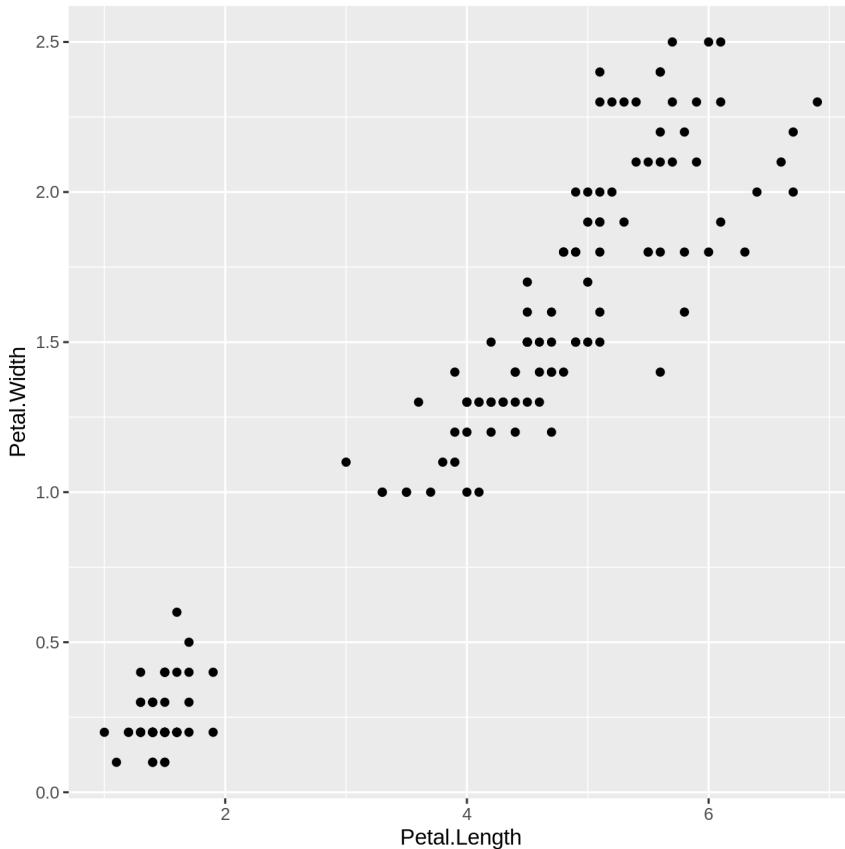
Parameters to consider when addressing this question:

<b>data</b>	<b>geom</b>	<b>x value</b>	<b>y value</b>
iris	geom_point	Petal length	Petal width



# Challenge 1#: Solution

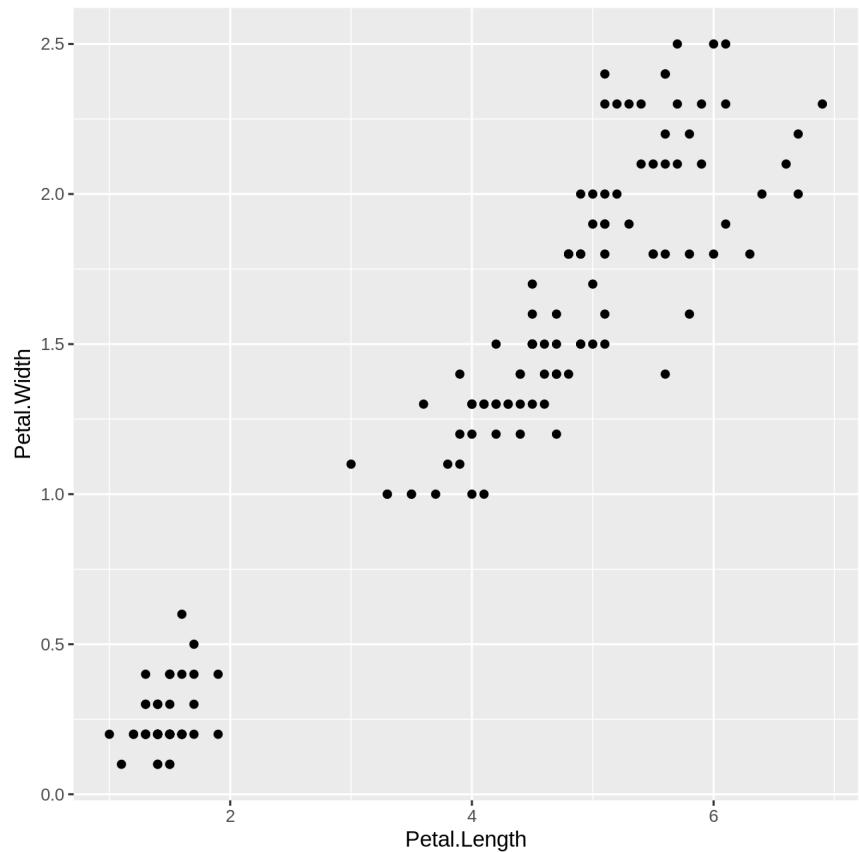
```
ggplot(data = iris, aes(x = Petal.Length,  
                      y = Petal.Width)) +  
  geom_point()
```





# Challenge 1#: Solution

```
ggplot(data = iris, aes(x = Petal.Length,  
                        y = Petal.Width)  
       + geom_point()
```

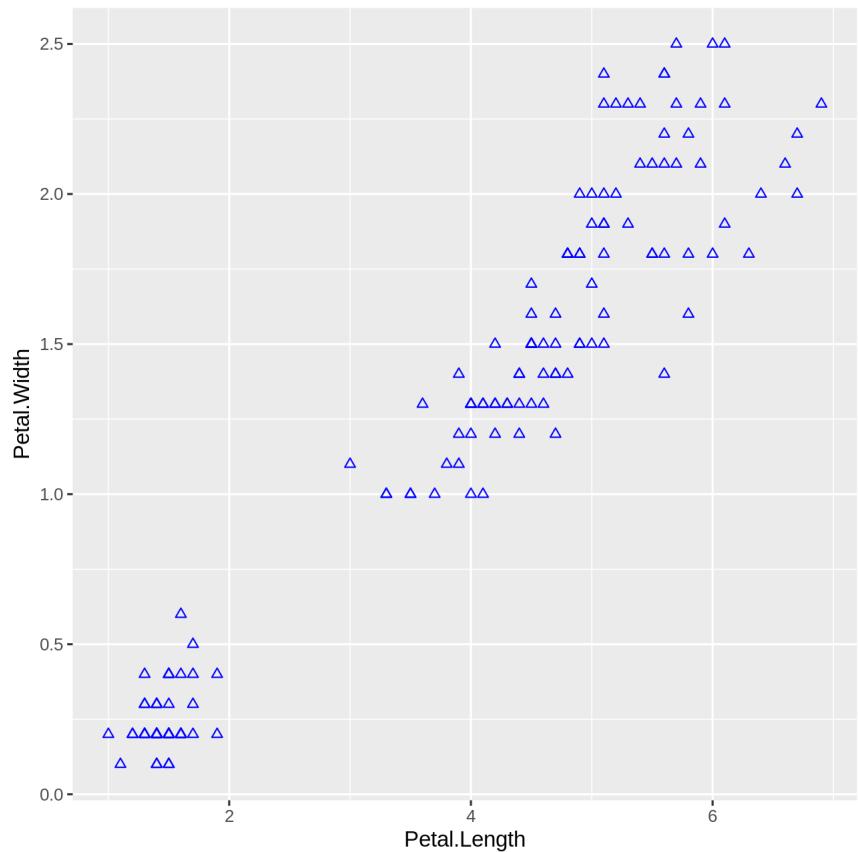


**note:** aesthetics can either be in the `ggplot()` line, and will be inherited by every geom, or in the `geom_*`() line to apply to that geom only!



# Challenge 1#: Solution

```
ggplot(data = iris, aes(x = Petal.Length,  
                        y = Petal.Width)  
       geom_point(shape = 2, color="blue")
```



**note:** aesthetics can either be in the `ggplot()` line, and will be inherited by every geom, or in the `geom_*` line to apply to that geom only!

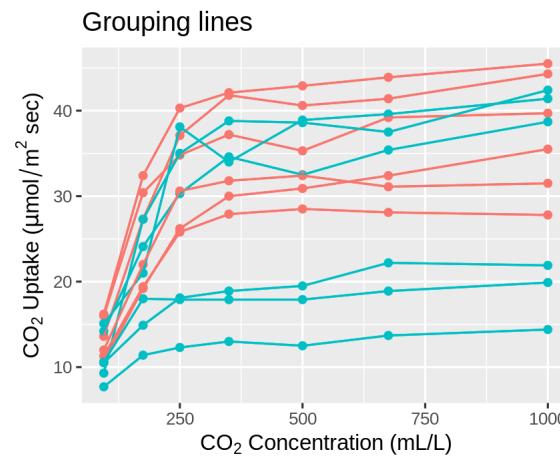
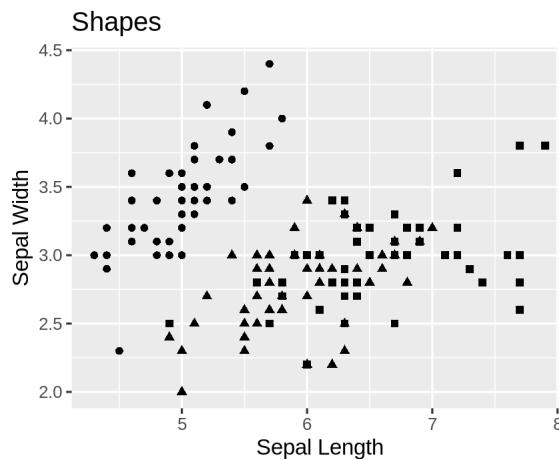
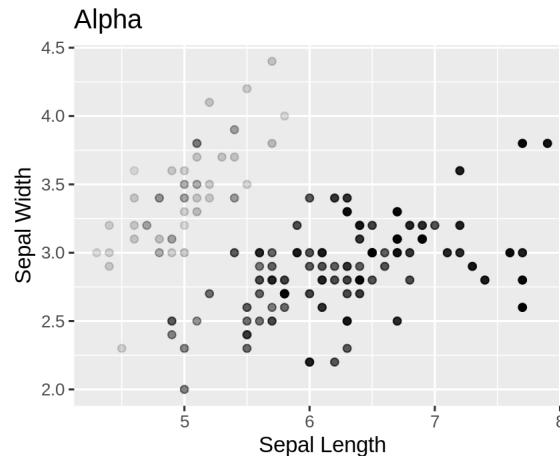
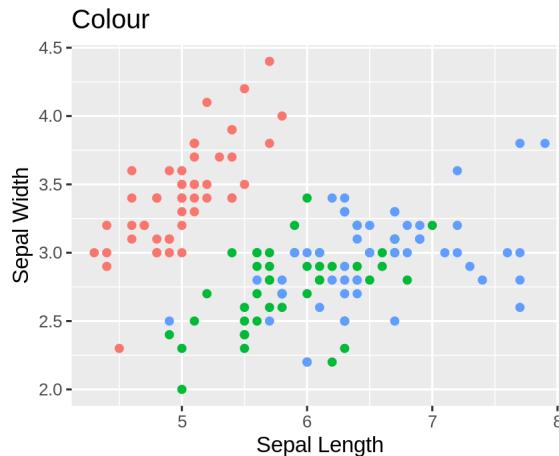
**note 2:** colour, alpha, shape, and size commands can be set outside of `aes()` values, and will be static, not data-dependent.

# Aesthetic mapping

colour, shape, size, labels and transparency

# Aesthetic

Use aesthetics (`aes()`) to distinguish classes, groups and structure

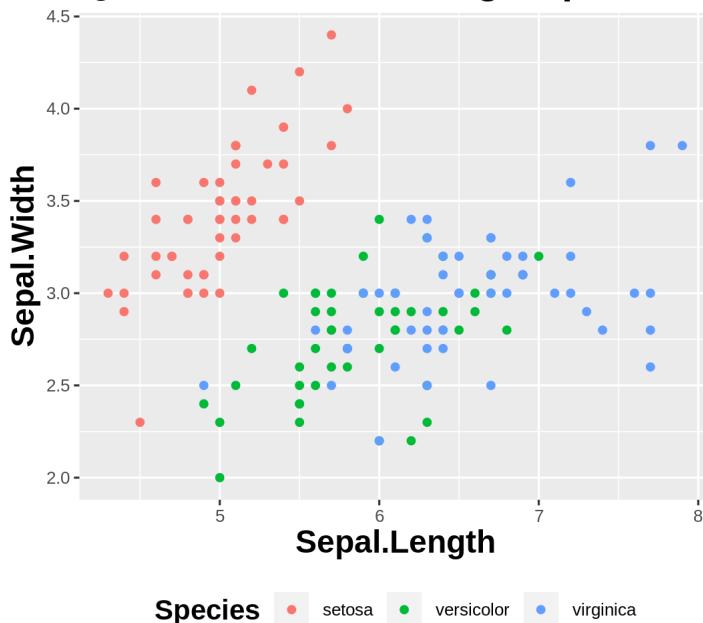


# Colouring: make your points talk

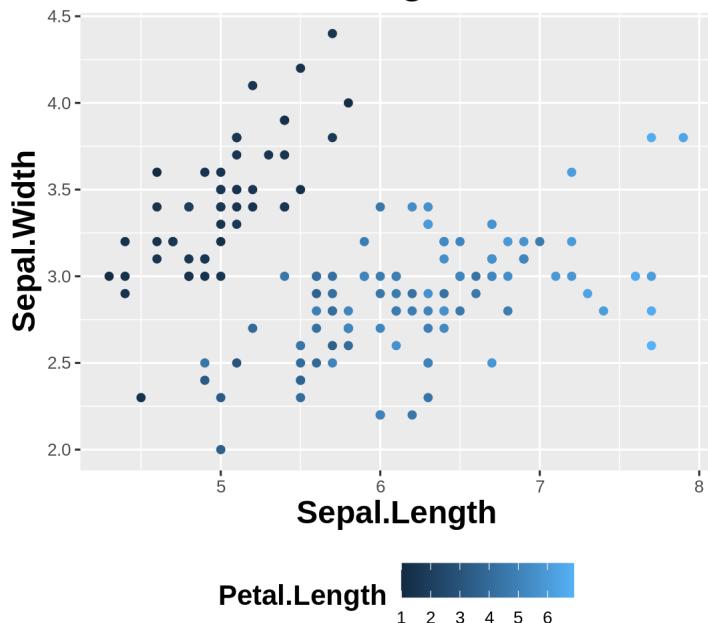
Change **colour** to

- differentiate between groups
- represent data values
- highlight specific elements

Qualitative colour for groups



Gradient colouring for values

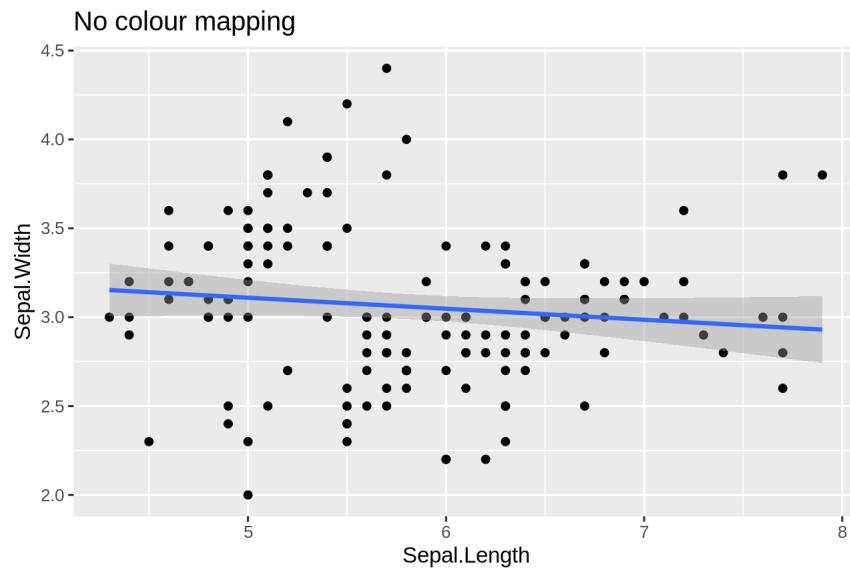


See [Fundamentals of Data Visualization](#)

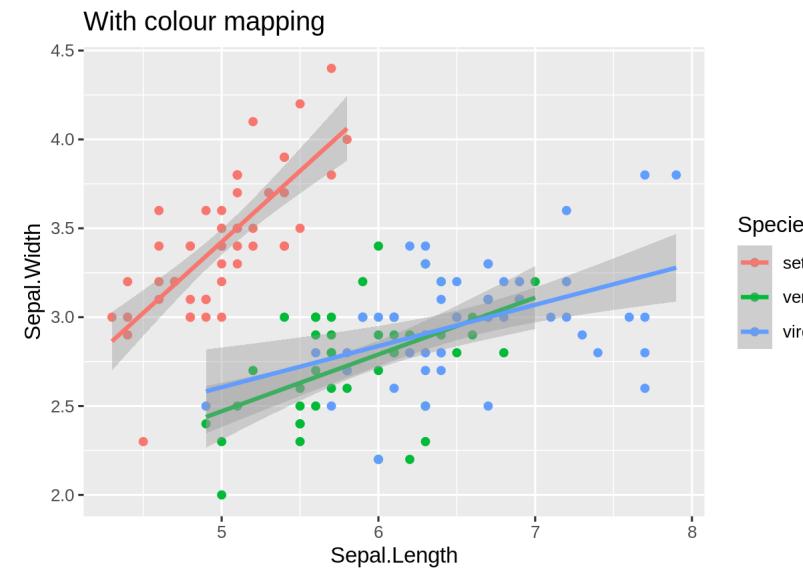
# Using `aes()` use to change colour

Do the sepal length and width vary differently across species?

```
# No colour mapping  
ggplot(data = iris, aes(x = Sepal.Length,  
                         geom_point() +  
                         geom_smooth(method=lm)+  
                         labs(title = "No colour mapping")
```

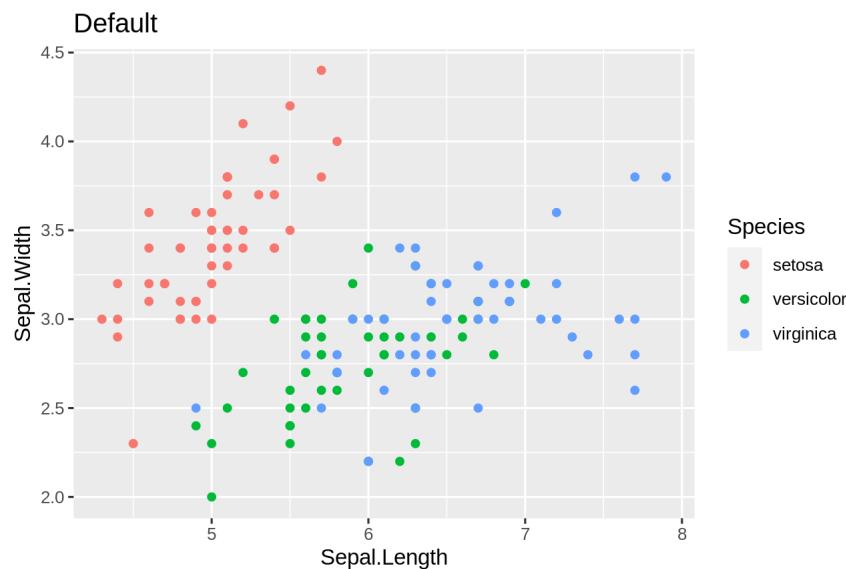


```
# With colour mapping  
ggplot(data = iris, aes(x = Sepal.Length,  
                         geom_point() +  
                         geom_smooth(method=lm) +  
                         labs(title = "With colour mapping")
```

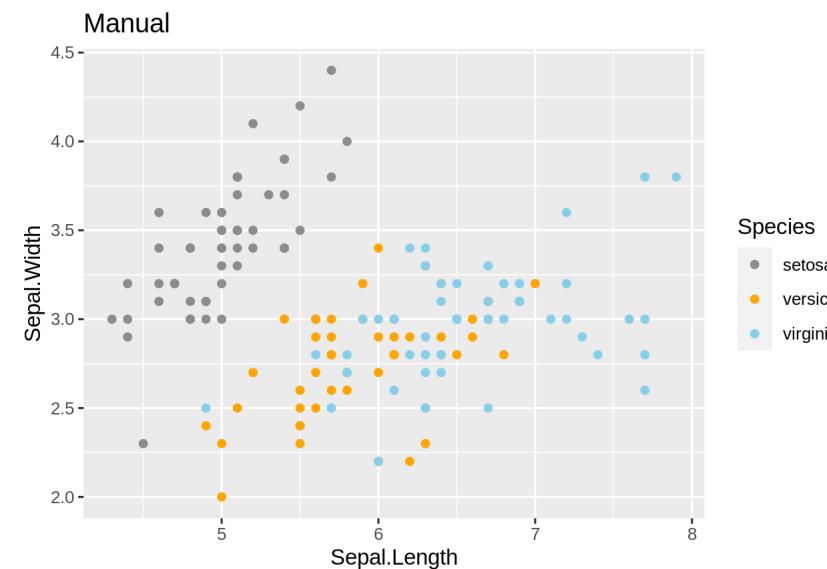


# Change colour manually

```
# Default  
pp <- ggplot(data = iris) +  
  geom_point(mapping = aes(x = Sepal.Length,  
                           y = Sepal.Width))  
pp + labs(title = "Default")
```

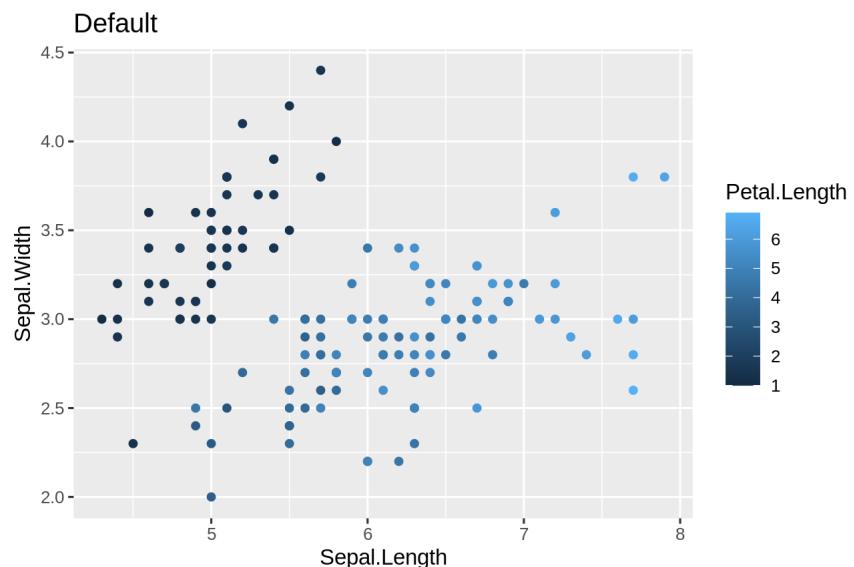


```
# Manual  
pp +  
  scale_colour_manual(values = c("grey", "orange", "lightblue"))  
pp + labs(title = "Manual")
```

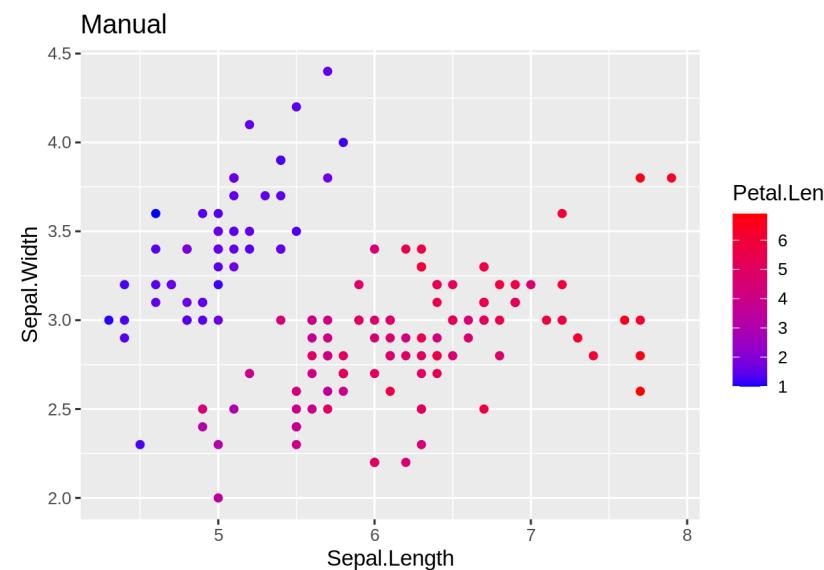


# Colour gradients

```
# Default  
pp2 <- ggplot(data = iris) +  
  geom_point(mapping = aes(x = Sepal.Length,  
                           colour = Petal.Length))  
pp2 + labs(title = "Default")
```



```
# Manual  
pp2 + scale_colour_gradient(low = "blue",  
                             high = "red")  
  labs(title = "Manual")
```



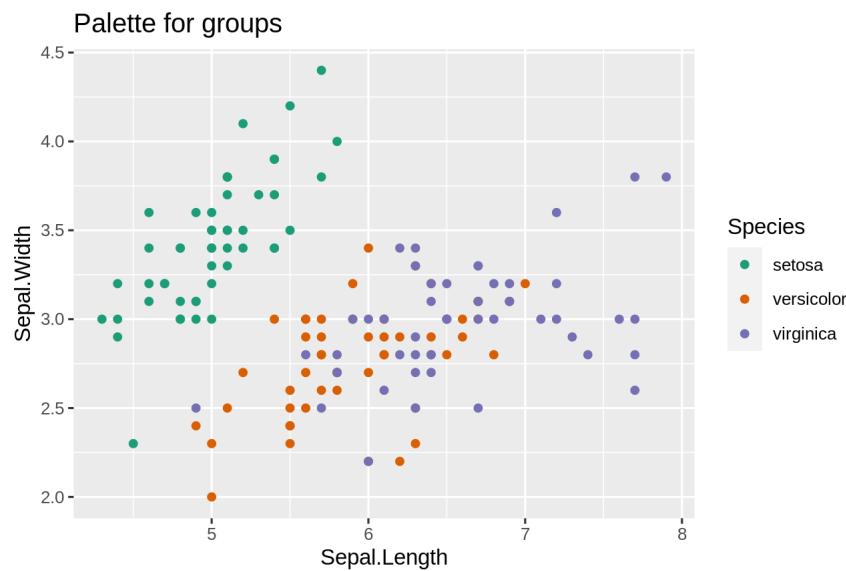
# Use a predefined colour palette

```
install.packages("RColorBrewer")
require(RColorBrewer)
display.brewer.all()
```



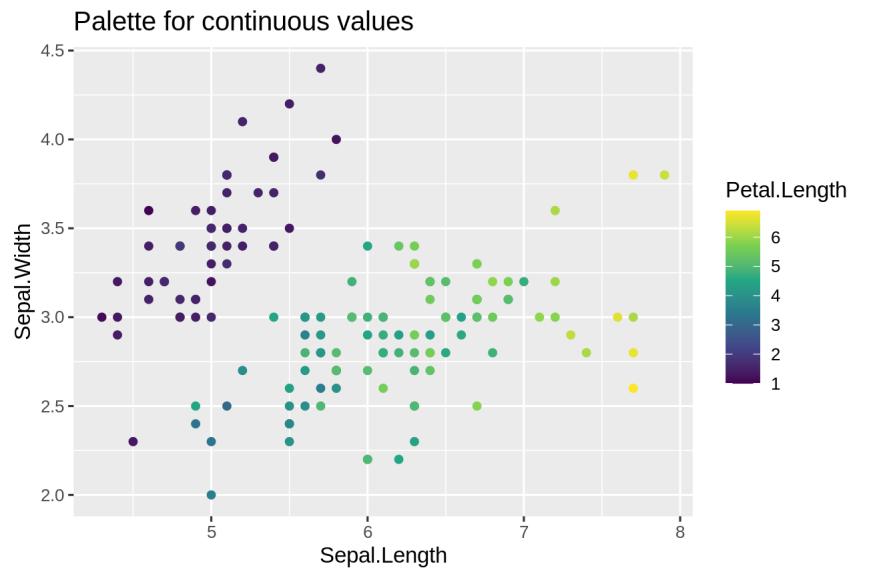
# Use a predefined colour palette

```
# Palette for groups  
pp + scale_colour_brewer(palette = "Dark2") +  
  labs(title = "Palette for groups")
```



# Use a predefined colour palette

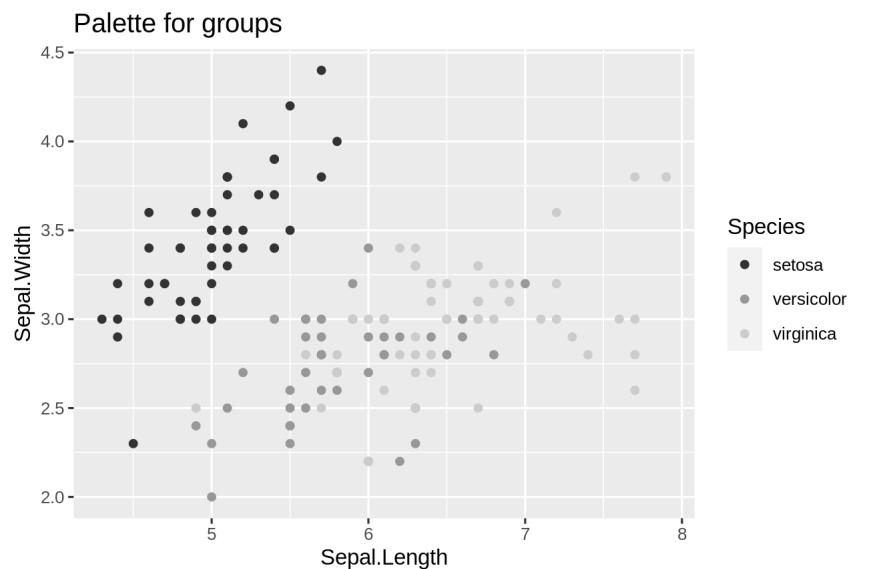
```
# Palette for continuous values  
pp2 + scale_color_viridis_c() +  
  labs(title = "Palette for continuous values")
```



# Use a predefined colour palette

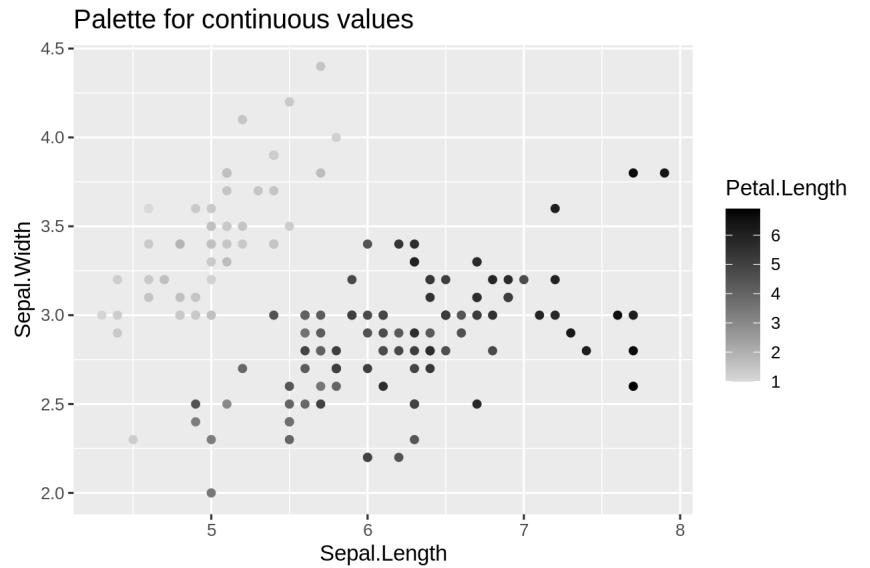
Grey-scale palette for publication purposes

```
# Palette for groups  
pp + scale_colour_grey() +  
  labs(title = "Palette for groups")
```



# Use a predefined colour palette

```
# Palette for continuous values  
pp2 + scale_colour_gradient(low = "grey85", high = "black") +  
  labs(title = "Palette for continuous values")
```



# Use colourblind-friendly palettes

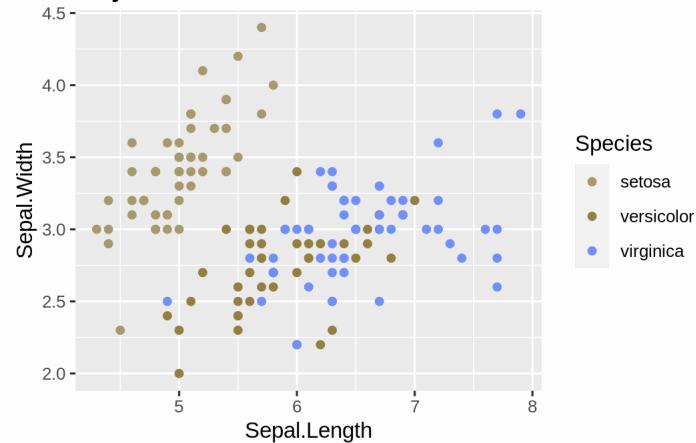
How your figure might appear under various forms of colourblindness? We can use [colorblindr](#) that is not currently on CRAN, so we install it with the packages [remotes](#).

```
remotes::install_github("clauswilke/colorblindr", quiet = TRUE)  
library(colorblindr)
```

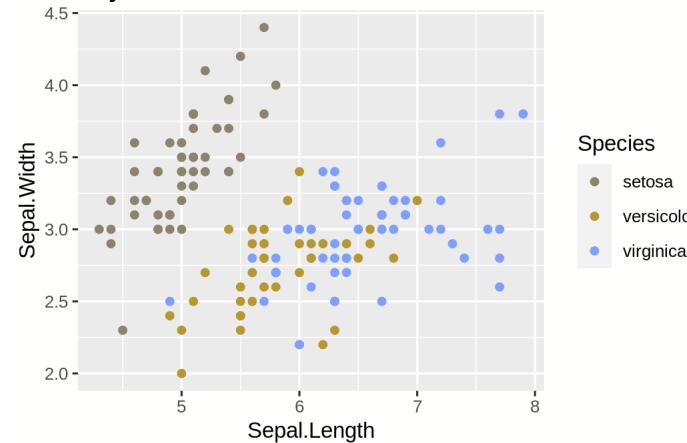
# Use colourblind-friendly palettes

cvd\_grid(pp)

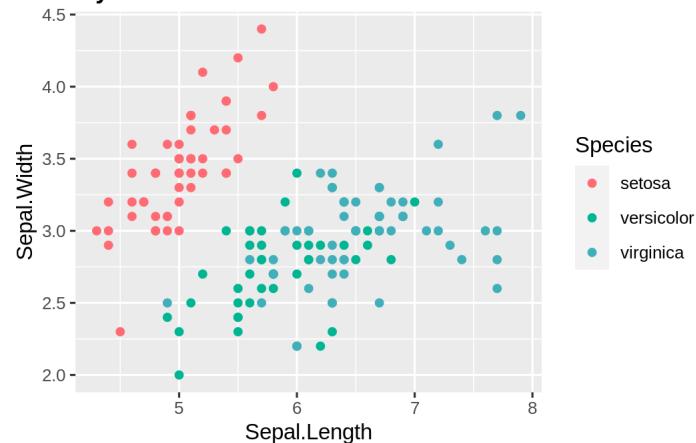
**Deutanomaly**



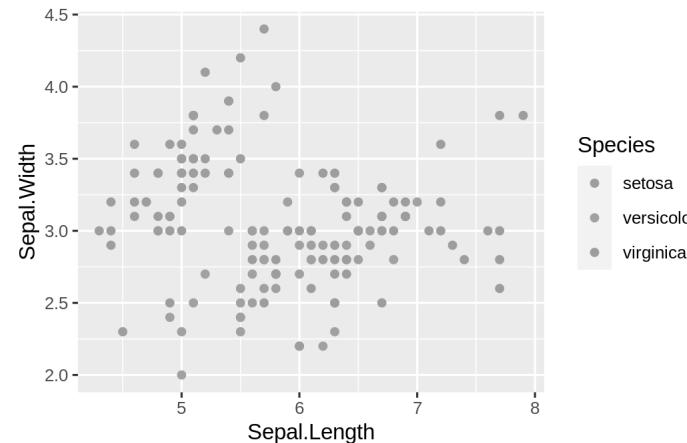
**Protanomaly**



**Tritanomaly**

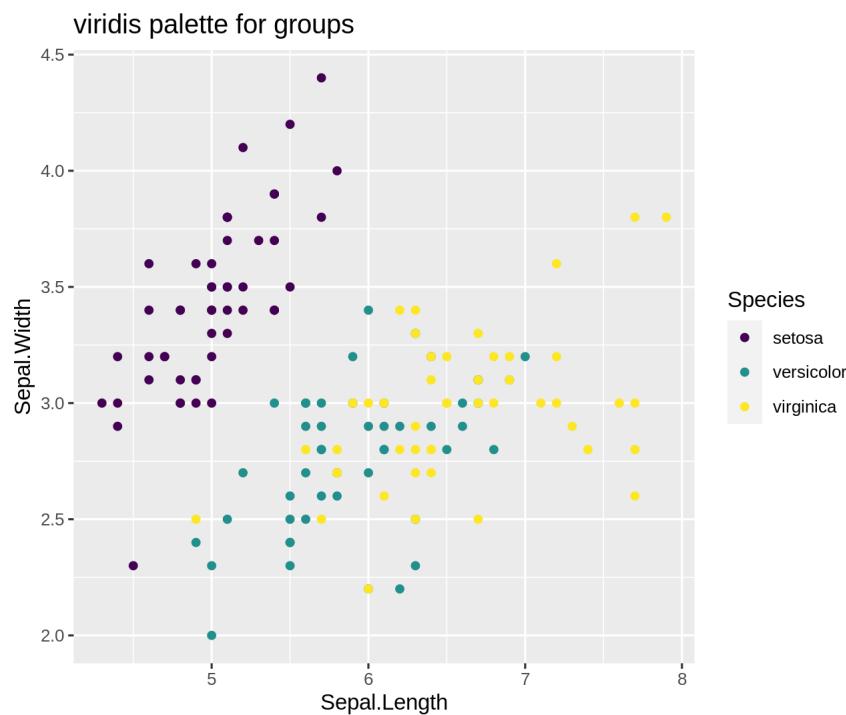


**Desaturated**

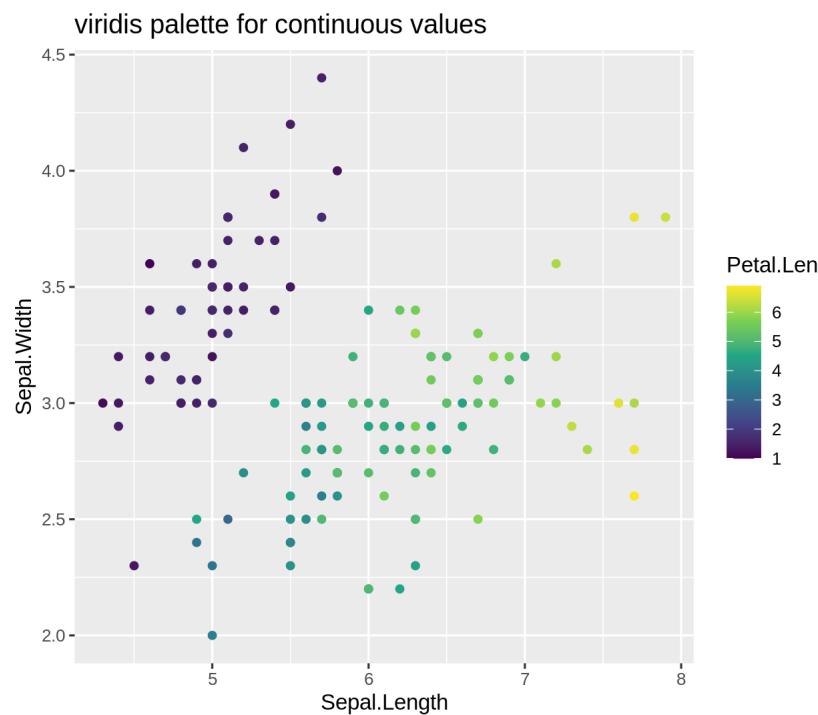


# Use colourblind-friendly palettes

```
# Palette for groups  
pp + scale_colour_viridis_d() +  
  labs(title = "viridis palette for gro
```

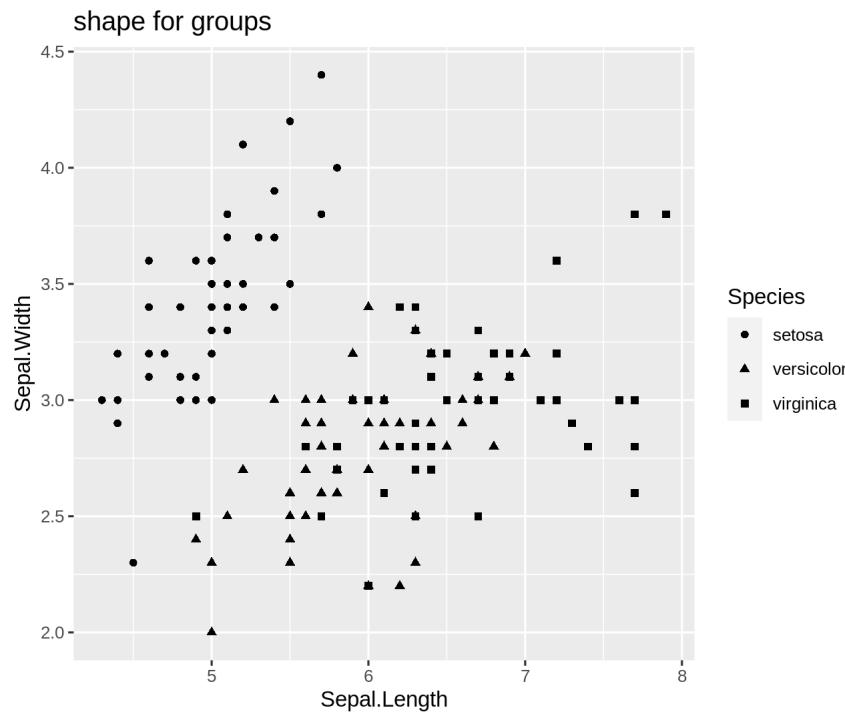


```
# Palette for continuous values  
pp2 + scale_colour_viridis_c() +  
  labs(title = "viridis palette for co
```

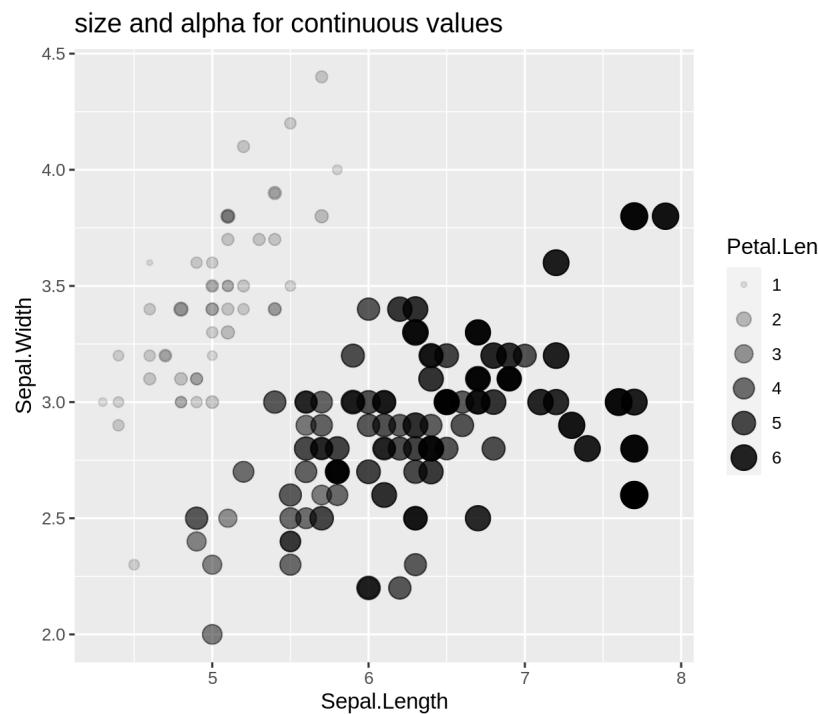


# Changing shape, size and alpha

```
# shape for groups  
ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length, y =  
    labs(title = "shape for groups"))
```



```
# size and alpha for continuous values  
ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length, y =  
    size = Petal.Length,  
    labs(title = "size and alpha for con")
```





# Challenge #2

- Produce an informative plot from built-in datasets such as `mtcars`, `CO2` or `msleep`.
- Use appropriate aesthetic mappings for different data types

Data	x	y	Aesthetics
mtcars	<i>wt</i>	<i>mpg</i>	<i>disp</i> and <i>hp</i>
CO2	<i>conc</i>	<i>uptake</i>	<i>Treatment</i> and <i>Type</i>
msleep	$\log_{10}(\text{bodywt})$	<i>awake</i>	<i>vore</i> and <i>conservation</i>
ToothGrowth	<i>dose</i>	<i>len</i>	<i>supp</i>

**⚠ Pay attention to the data types!**



# Challenge #2

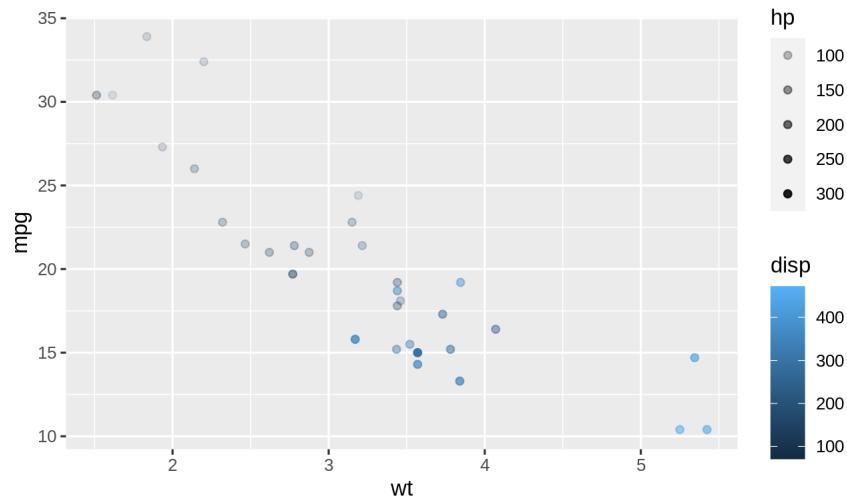
## Breakout rooms! - 15 min

make a plot to share with the group

# Challenge #2 - Solution example #1

Data	x	y	Aesthetics
mtcars	wt	mpg	disp and hp

```
data(mtcars)
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = wt, y = mpg,
                           colour = disp,
                           alpha = hp))
```

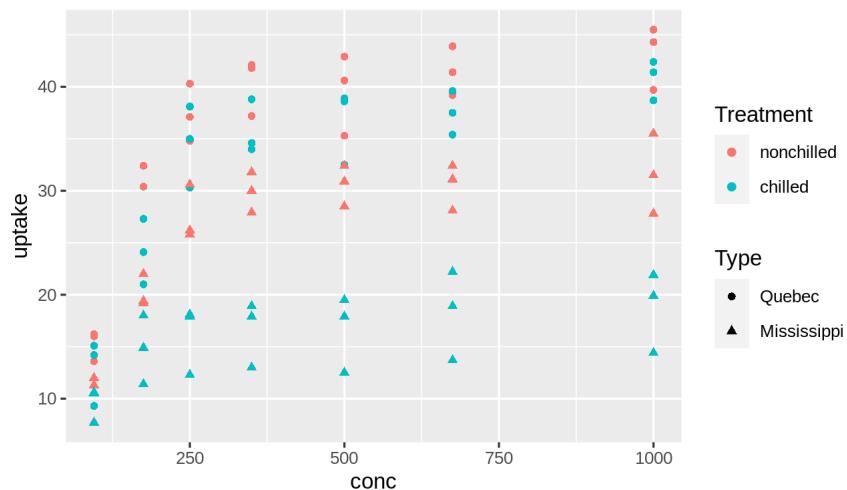


Could you use `size` instead of `alpha`? What about `shape`?

# Challenge #2 - Solution example #2

Data	x	y	Aesthetics
CO2	conc	uptake	Treatment and Type

```
data(CO2)
ggplot(data = CO2) +
  geom_point(mapping = aes(x = conc, y = uptake,
                           colour = Treatment, shape = Type))
```

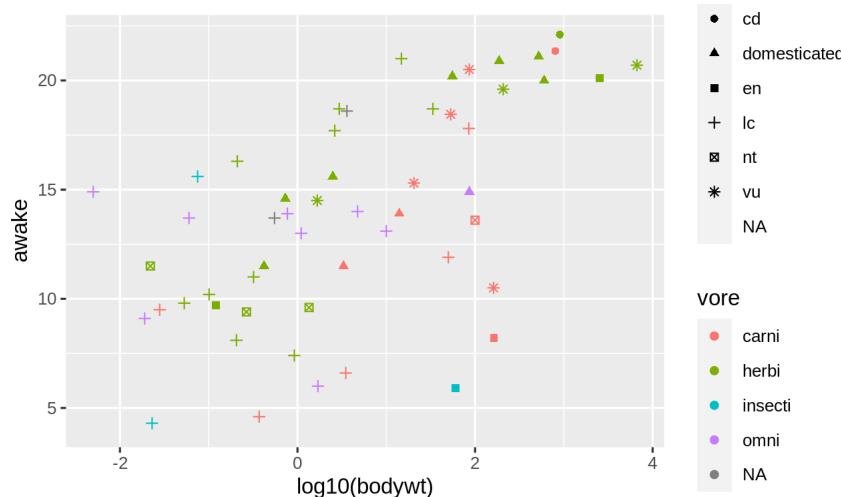


Why not use `size = Type`?

# Challenge #2 - Solution example #3

Data	x	y	Aesthetics
msleep	$\log_{10}(\text{bodywt})$	awake	vore and conservation

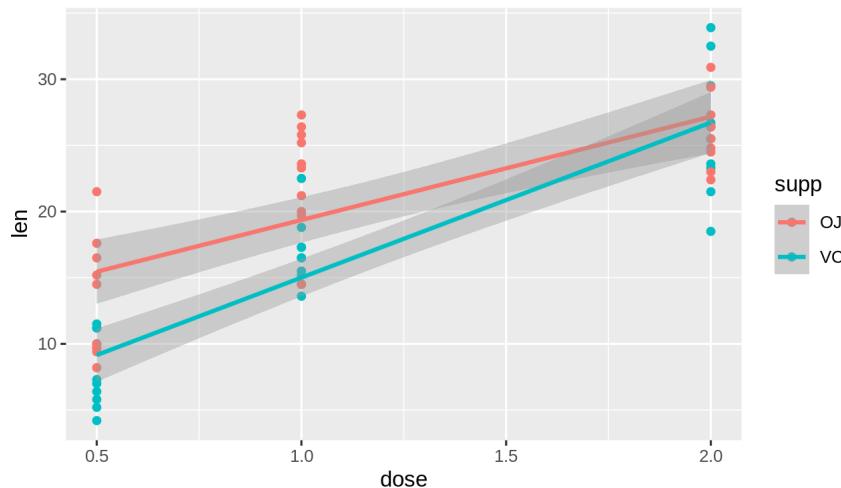
```
data(msleep)
ggplot(data = msleep) +
  geom_point(mapping = aes(x = log10(bodywt), y = awake,
                           colour = vore, shape = conservation))
```



# Challenge #2 - Solution example #4

Data	x	y	Aesthetics
ToothGrowth	dose	len	supp

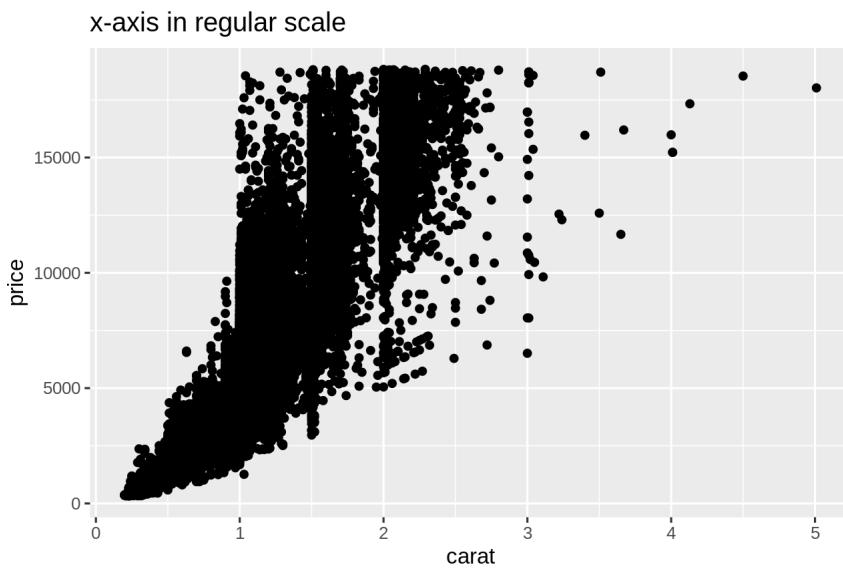
```
data(ToothGrowth)
ggplot(ToothGrowth, aes(x=dose, y=len, color=supp)) +
  geom_point() +
  geom_smooth(method = lm, formula='y~x')
```



# Changing the scale of the axes

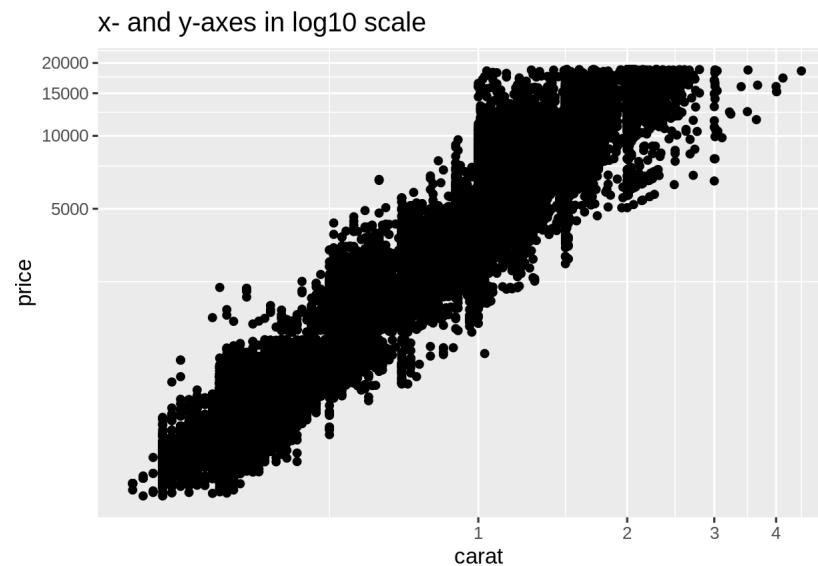
x axis in regular scale

```
ggplot(diamonds) +  
  geom_point(mapping = aes(x = carat, y = price)) +  
  labs(title = "x-axis in regular scale")
```



x-axis and y-axis in log10() scale

```
ggplot(diamonds) +  
  geom_point(mapping = aes(x = carat, y = price)) +  
  coord_trans(x = "log10",  
              y = "log10") +  
  labs(title = "x- and y-axes in log10 scale")
```



It is also possible to transform the coordinate system using  
`scale_x_log10()` and `scale_y_log10()`

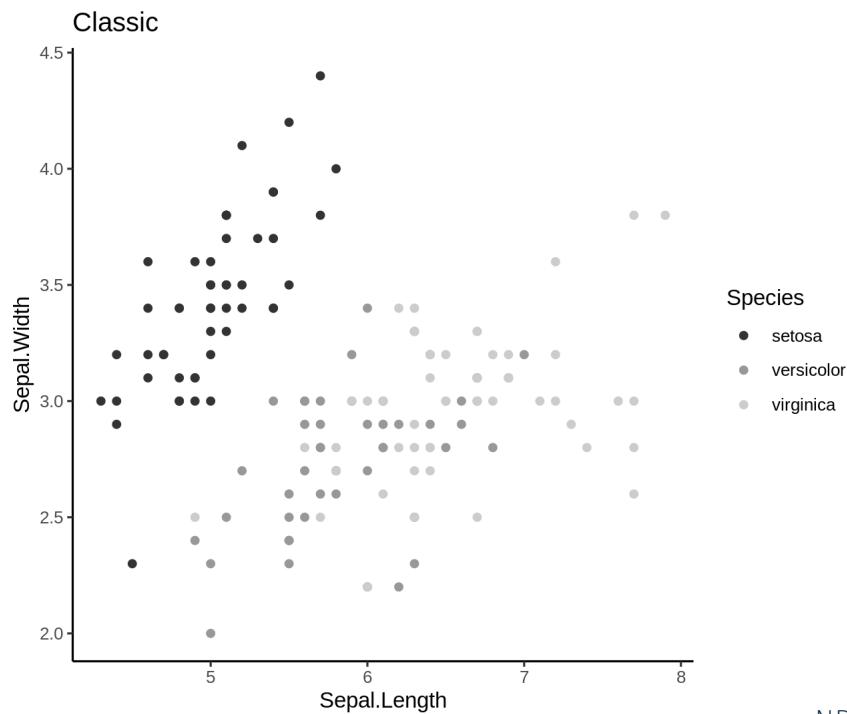
# Fine-tuning your plots

Using `theme()` to make it look good!

# theme()

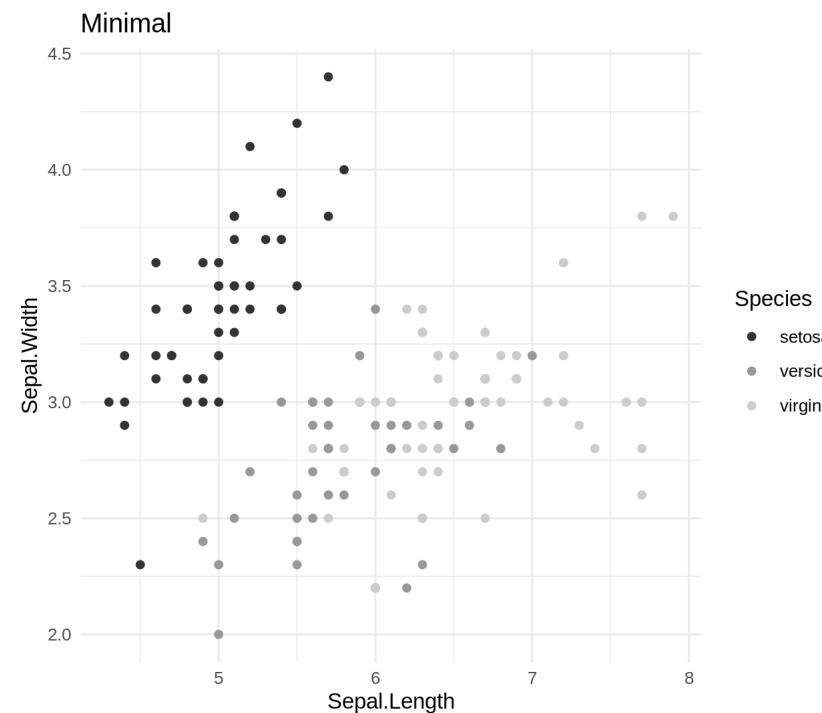
## Theme Classic

```
# Theme classic  
pp + scale_colour_grey() +  
  theme_classic() +  
  labs(title = "Classic")
```



## Theme Minimal

```
pp + scale_colour_grey() +  
  theme_minimal() +  
  labs(title = "Minimal")
```



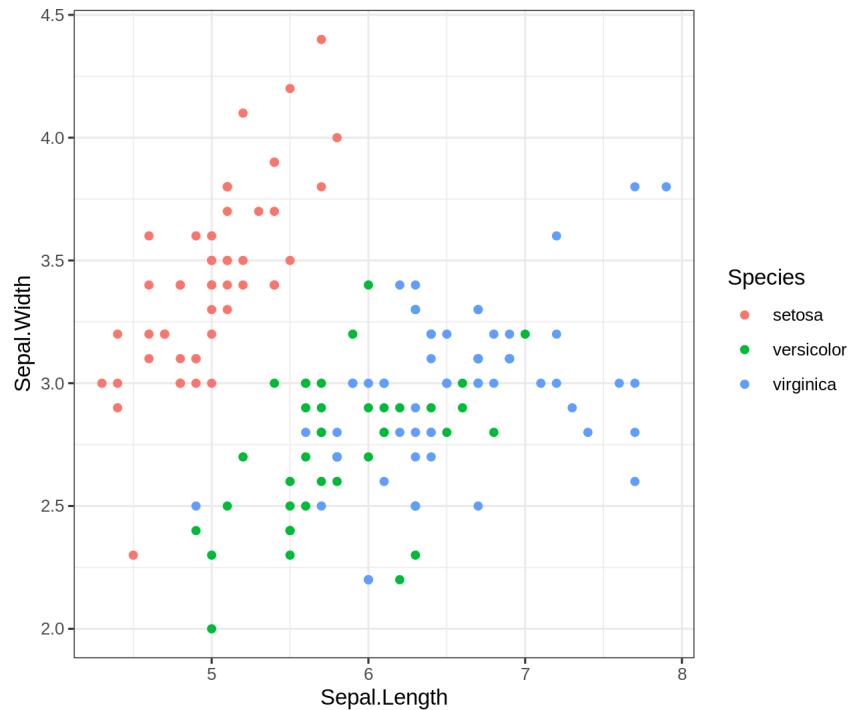
NB: Good choices for publication purposes!

# theme()

Use `theme_set()` to change theme for all your future plots, or `theme_update()` to edit aspects of a theme setting.

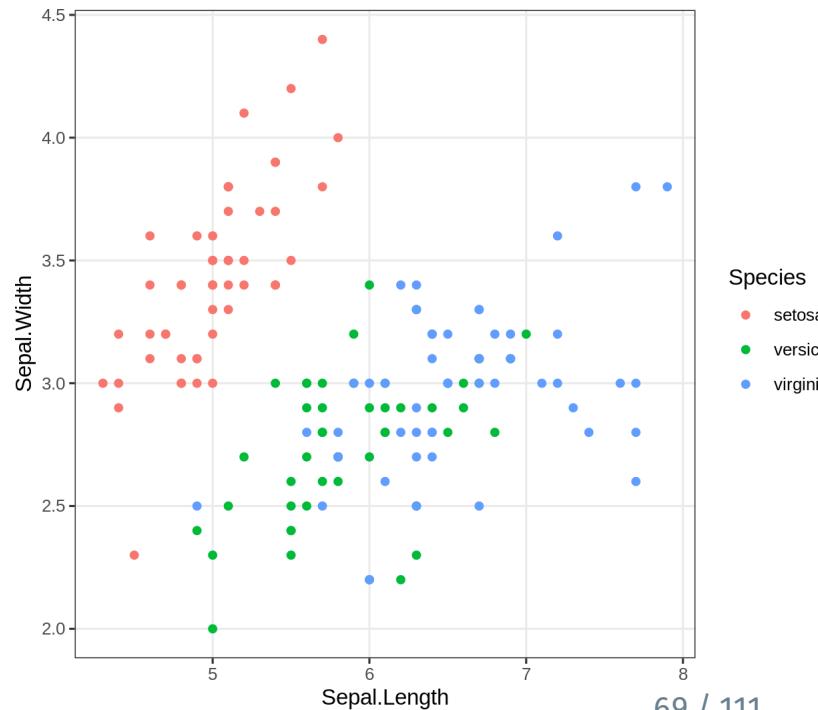
## Set theme

```
# Set Classic as default  
theme_set(theme_bw())  
pp
```



## Update theme

```
# remove minor gridlines  
theme_update(panel.grid.minor = element_none)  
pp
```



# theme()

## Elements of a theme

### ggplot2 theme elements reference

Set minimal as the baseline theme:

```
theme_minimal() +  
  theme(element = element_type())
```

Use element\_blank() to remove an element

Axis titles, text, ticks, and lines can be specified per axis using theme inheritance by putting .x/.y at the end of the theme element.

```
axis.line.y = element_line()
```

```
axis.title.y = element_text()
```

```
panel.grid.major = element_line()
```

```
panel.grid.minor = element_line()
```

```
axis.text.y  
axis.text = element_text()
```

plot.title.position = "plot"  
plot.caption.position = "plot"

plot.title = element\_text()  
plot.subtitle = element\_text()

Miles per Gallon & Horsepower  
of 32 Automobiles(1973-74 models)

Horsepower

300

200

100

10

Miles per gallon

Data: 1974 Motor Trend US magazine

"plot" means that they will be aligned to the entire plot (instead of the panel)  
plot.margin = margin(25, 25, 25, 25)

legend.title = element\_text()

legend.background = element\_rect()

legend.text = element\_text()

legend.position = c(.85,.85) / "none" /  
"left" / "right" /  
"bottom" / "top"

plot.background = element\_rect()

plot.caption = element\_text()

isabella-b

text = element\_text() ← modifications will be applied to all text elements

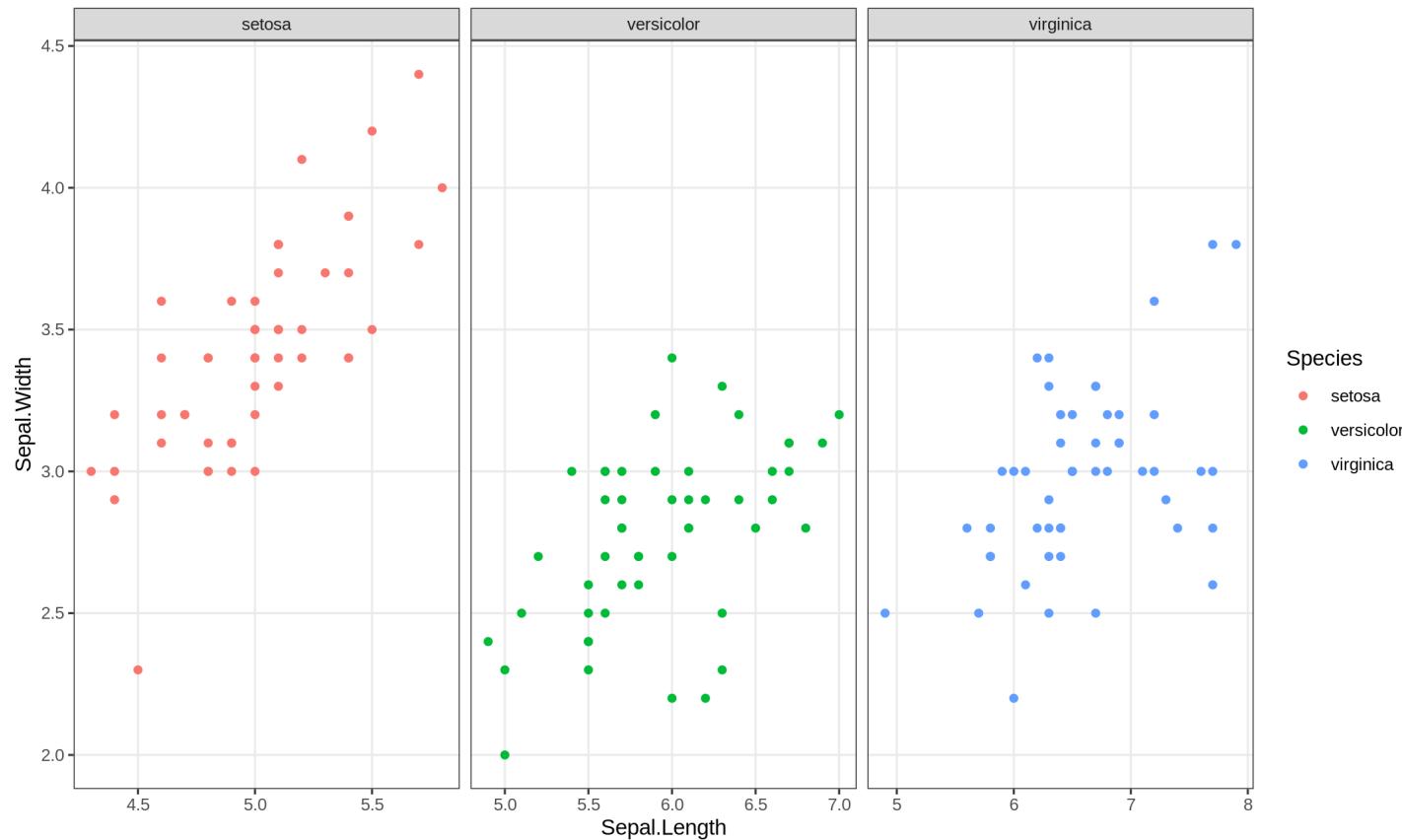
Full list of elements at [ggplot2.tidyverse.org/reference/theme](http://ggplot2.tidyverse.org/reference/theme)

# Fine-tuning your plots

Using `facet_grid()` to change the arrangement of plots

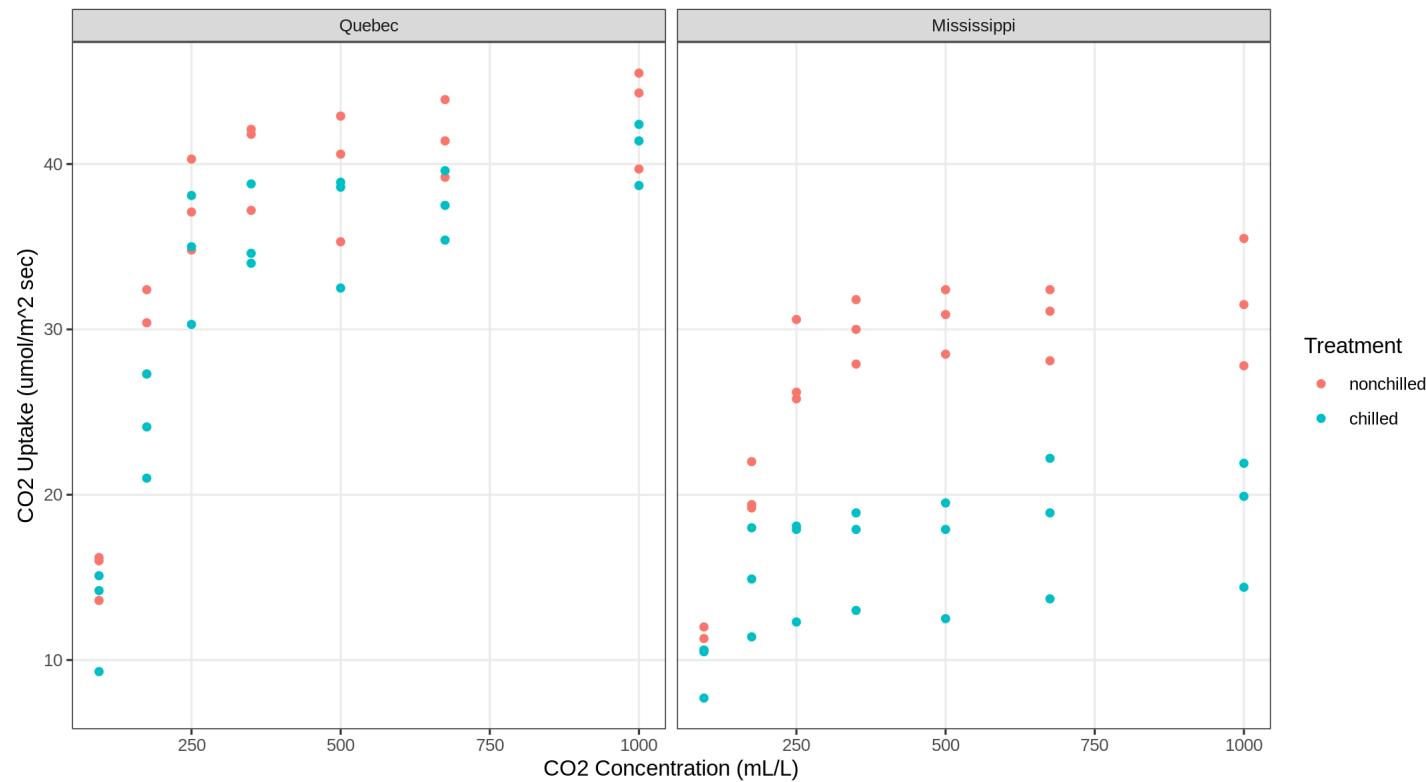
# iris dataset: per-species facets

```
ggplot(data = iris) +  
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +  
  facet_grid(~ Species, scales = "free")
```



# CO2 dataset: per-type facets

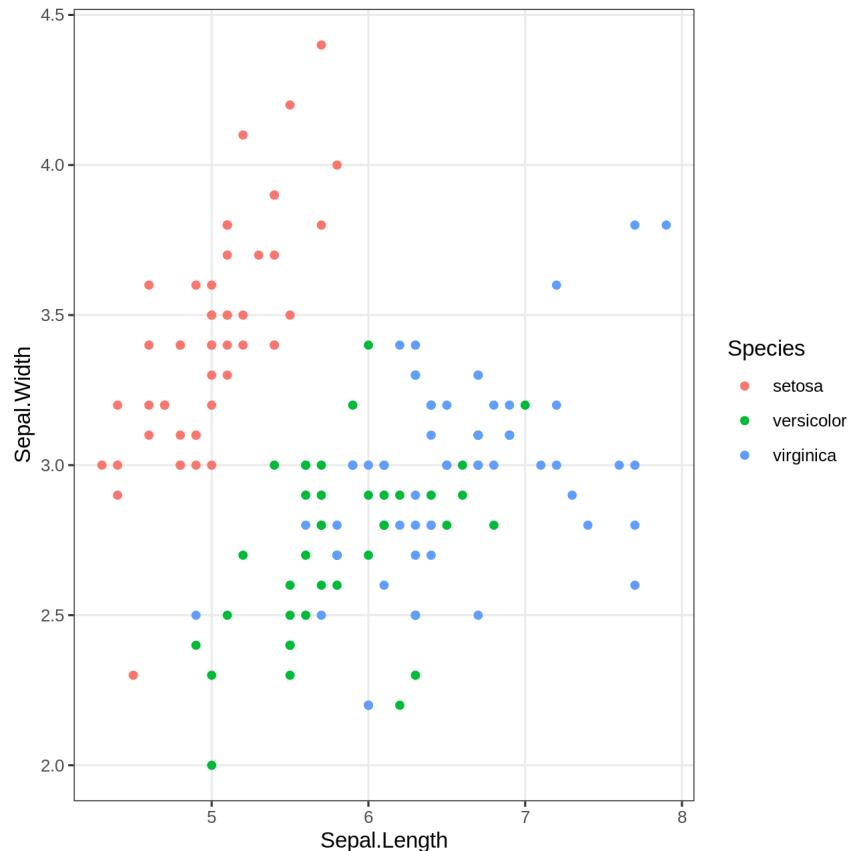
```
ggplot(data = CO2) +  
  geom_point(mapping = aes(x = conc, y = uptake, colour = Treatment)) +  
  xlab("CO2 Concentration (mL/L)") + ylab("CO2 Uptake (umol/m^2 sec)") +  
  facet_grid(~ Type)
```



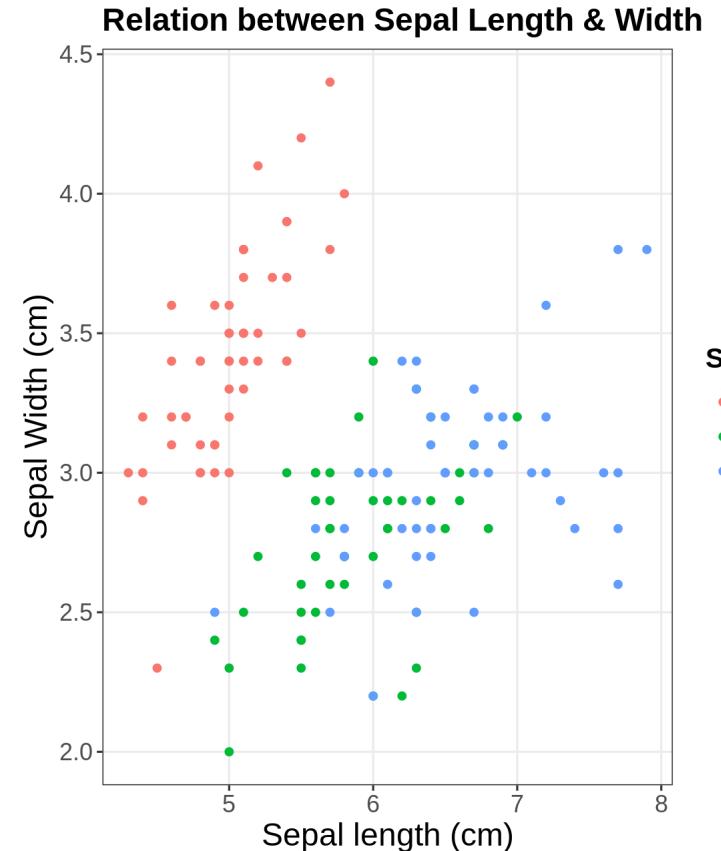
# Title and axes components: changing size, colour and face

# Title and axes components: size, colour and face

## Default



## Axes and title tuning

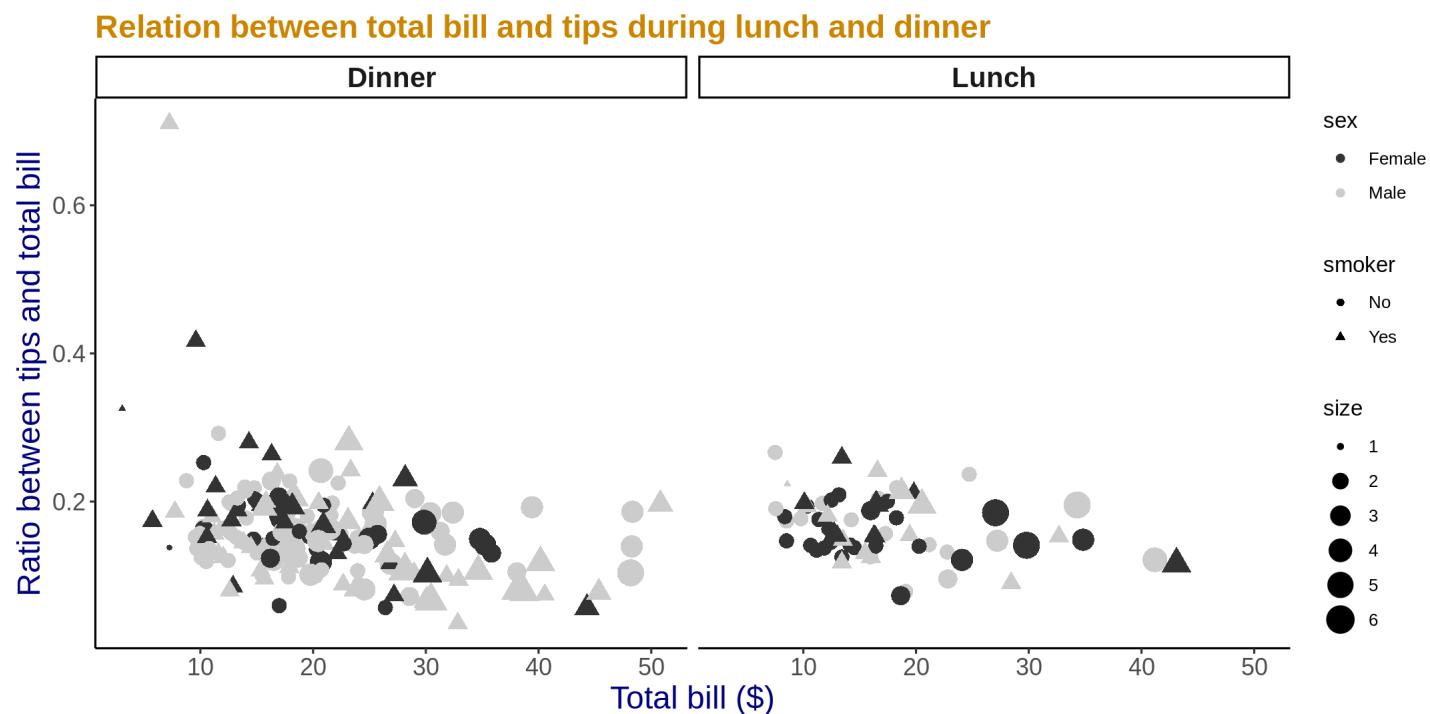




# Challenge #3

Use the `tips` dataset found in `reshape2` to reproduce the plot below.

Our tip: start from `theme_classic()` and add `theme()` to make your additional changes.





# Challenge #3

**Breakout rooms! - 15 min**



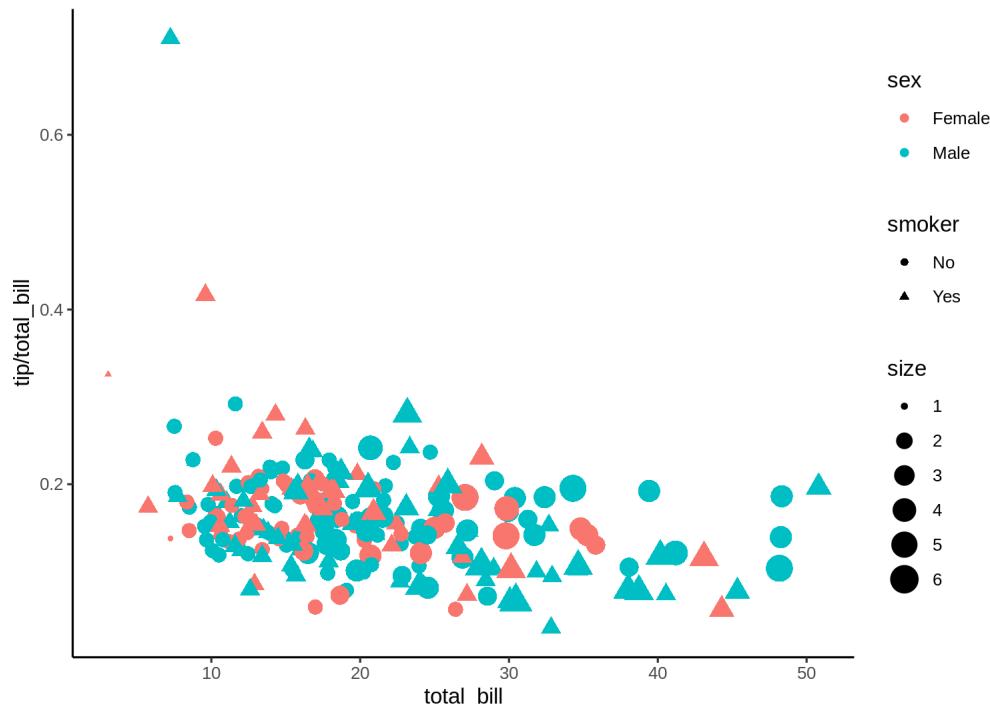
# Challenge #3: Solution

```
library(reshape2)
tips.gg <- ggplot(tips, aes(x = total_bill,
                               y = tip/total_bill,
                               shape = smoker,
                               colour = sex,
                               size = size)) +
  geom_point() +
  facet_grid( ~ time) +
  scale_colour_grey() +
  labs(title = "Relation between total bill and tips during lunch and dinner",
       x = "Total bill ($)", y = "Ratio between tips and total bill") +
  theme_classic() +
  theme(axis.title = element_text(size = 16,
                                   colour = "navy"),
        axis.text = element_text(size = 12),
        plot.title = element_text(size = 16,
                                  colour = "orange3",
                                  face = "bold"),
        strip.text.x = element_text(size = 14, face="bold"))
tips.gg
```

# Challenge #3: step-by-step solution

aes()

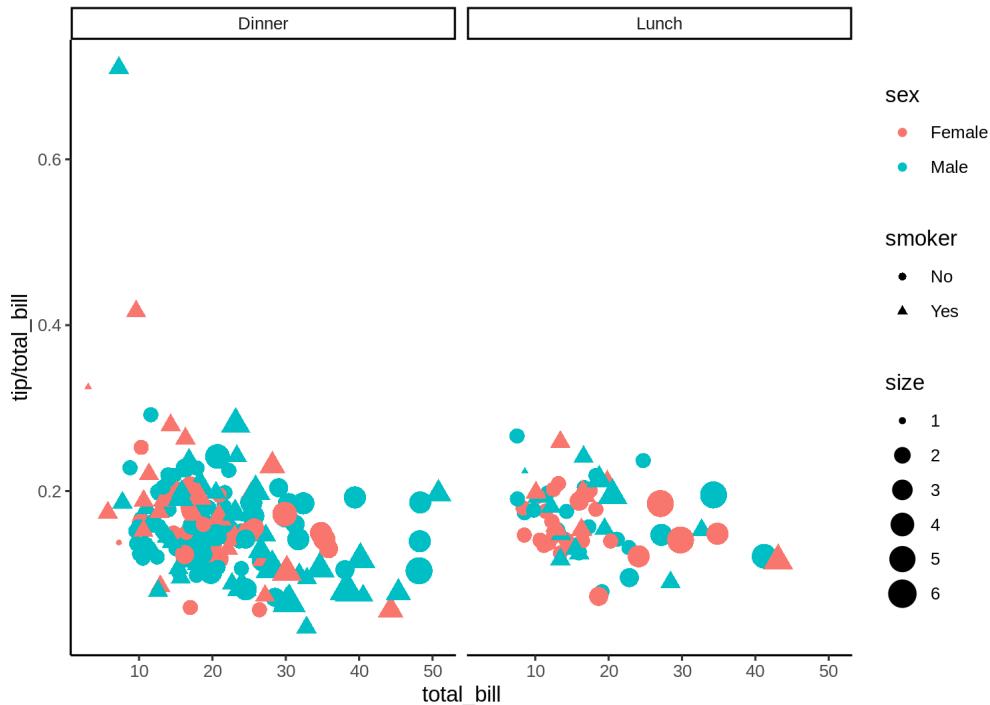
```
theme_set(theme_classic())
tips.gg <- ggplot(tips) +
  geom_point(mapping = aes(x = total_bill, y = tip/total_bill,
                           shape = smoker, colour = sex, size = size))
tips.gg
```



# Challenge #3: step-by-step solution

facet\_grid()

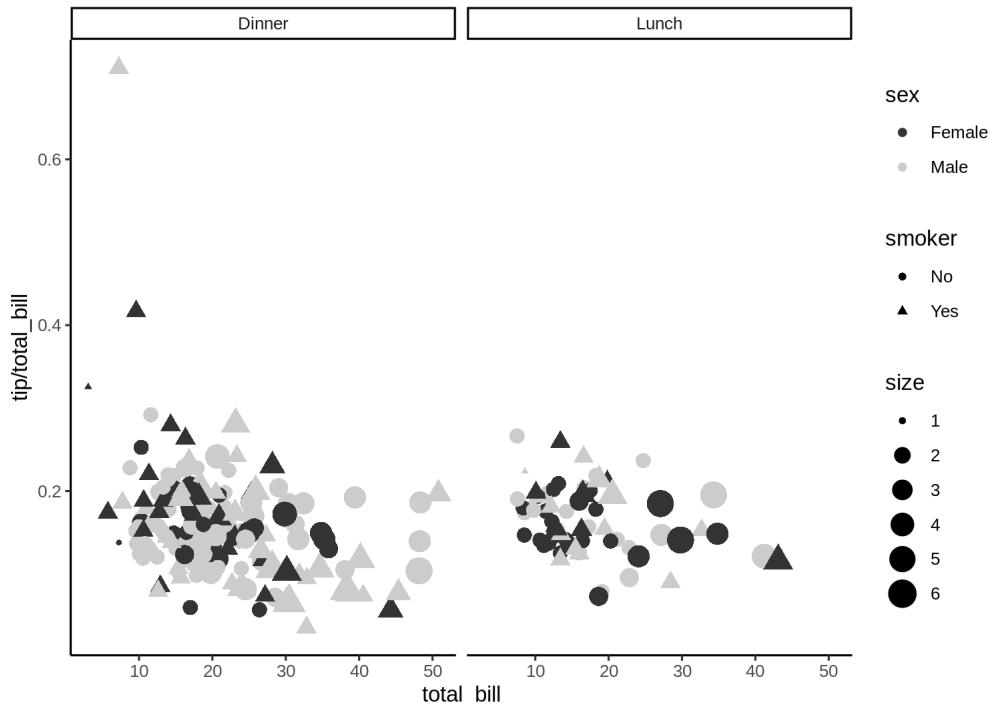
```
tips.gg <- tips.gg +  
  facet_grid( ~ time)  
tips.gg
```



# Challenge #3: step-by-step solution

## Grey-scale palette

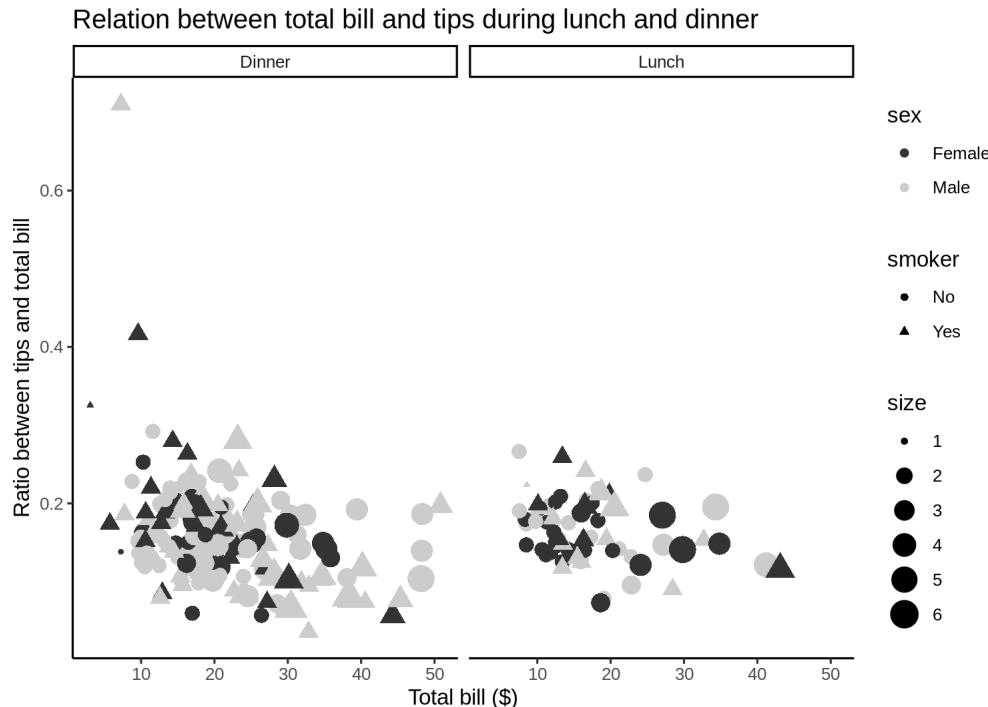
```
tips.gg <- tips.gg +  
  scale_colour_grey()  
tips.gg
```



# Challenge #3: step-by-step solution

## Adding plot title and labels

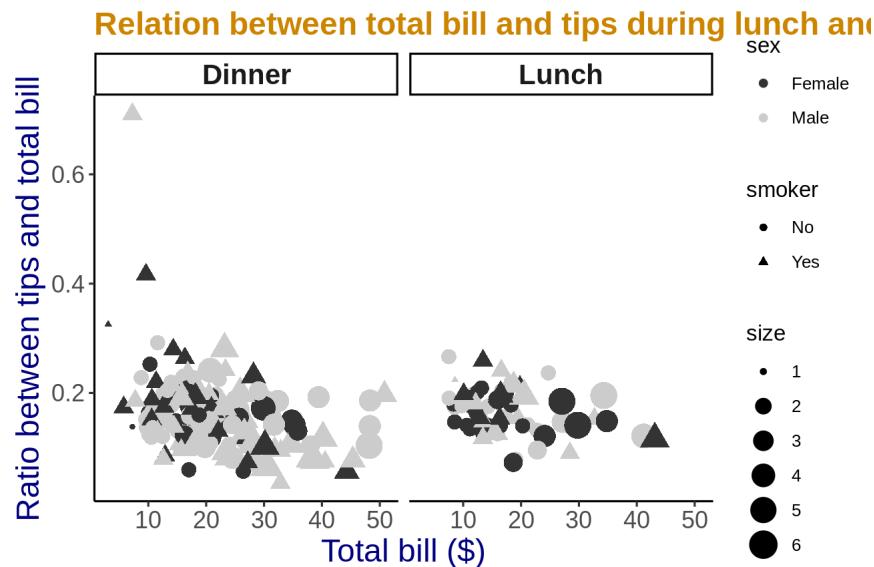
```
tips.gg <- tips.gg +  
  labs(title = "Relation between total bill and tips during lunch and dinner",  
        x = "Total bill ($)", y = "Ratio between tips and total bill")  
tips.gg
```



# Challenge #3: step-by-step solution

## Adding a theme and fine tuning it

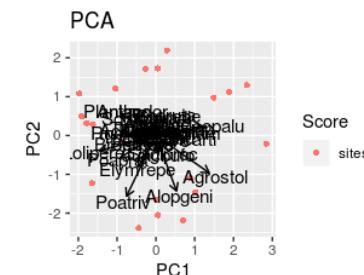
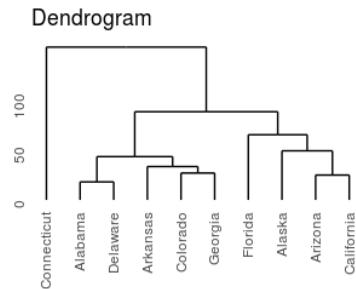
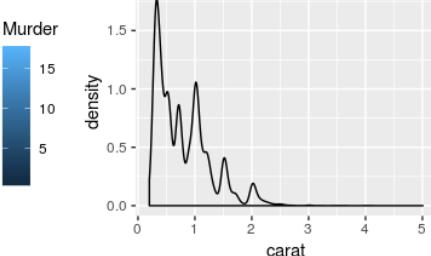
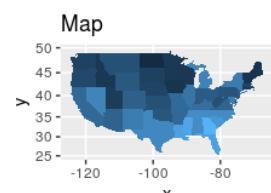
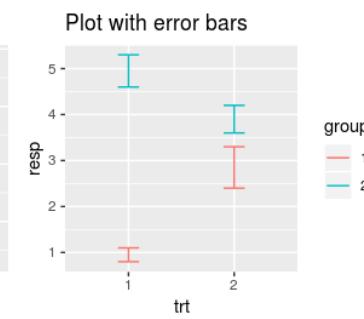
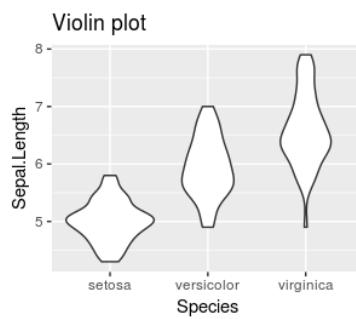
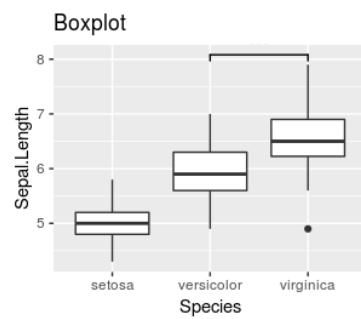
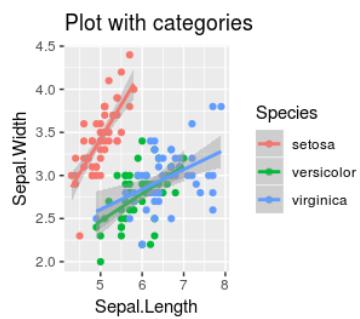
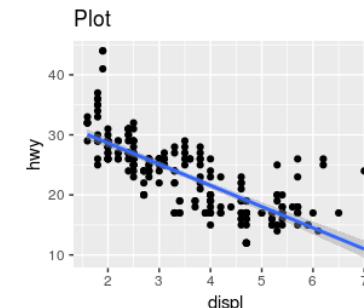
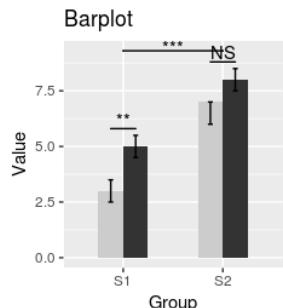
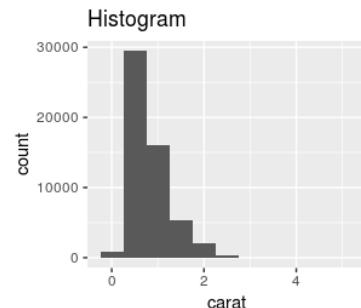
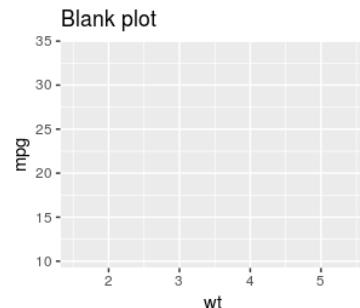
```
tips.gg <- tips.gg +  
  theme_classic() +  
  theme(axis.title = element_text(size = 16, colour = "navy"),  
        axis.text = element_text(size = 12),  
        plot.title = element_text(size = 16, colour = "orange3", face = "bold"),  
        strip.text.x = element_text(size = 14, face = "bold"))  
tips.gg
```



# Fine-tuning your plots

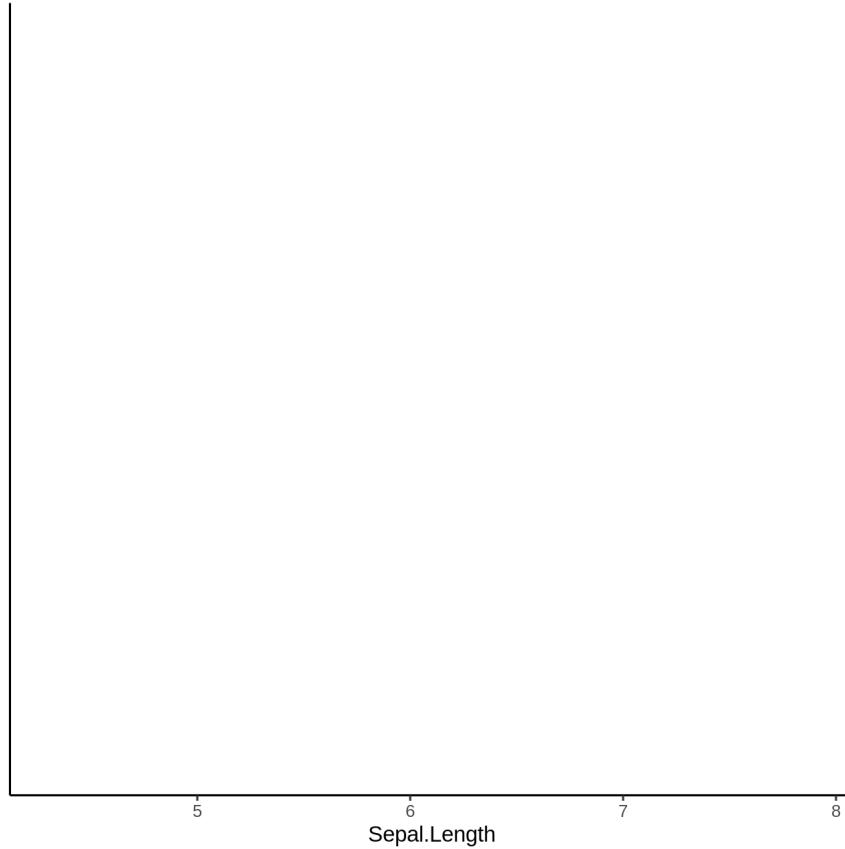
Using `geom_*`( ) to create different plots

# ggplot2::geom\_\*( )



## ggplot2::ggplot()

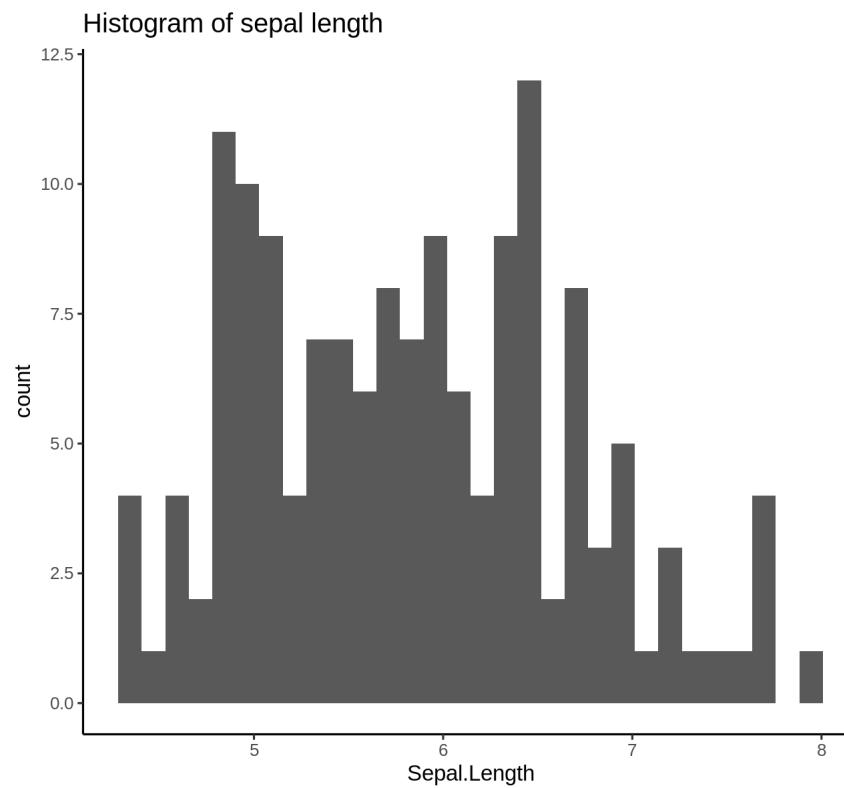
```
ggplot(iris, aes(Sepal.Length))
```



# Histograms: `geom_histogram()`

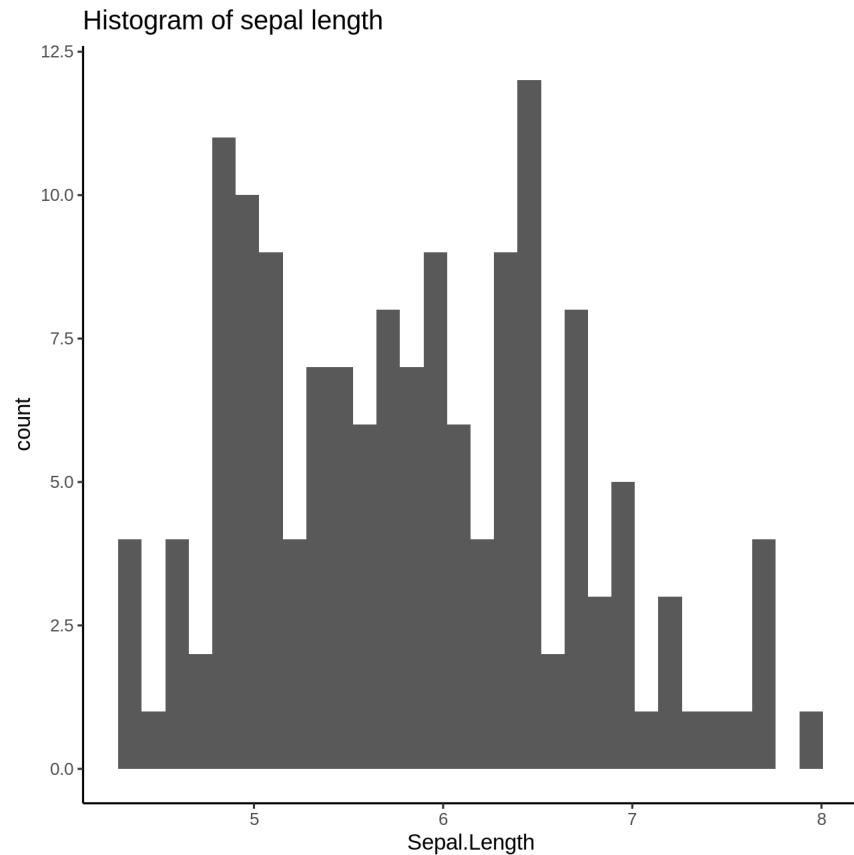
A **histogram** is an accurate graphical representation of the distribution of numeric data - only one aesthetic required

```
ggplot(iris, aes(Sepal.Length)) +  
  geom_histogram() +  
  ggtitle("Histogram of sepal length ")
```



# `geom_histogram()` versus `graphics::hist()`

```
ggplot(iris, aes(Sepal.Length)) +  
  geom_histogram() +  
  ggtitle("Histogram of sepal length")
```



```
hist(iris$Sepal.Length,  
     main = "Histogram of sepal length")
```



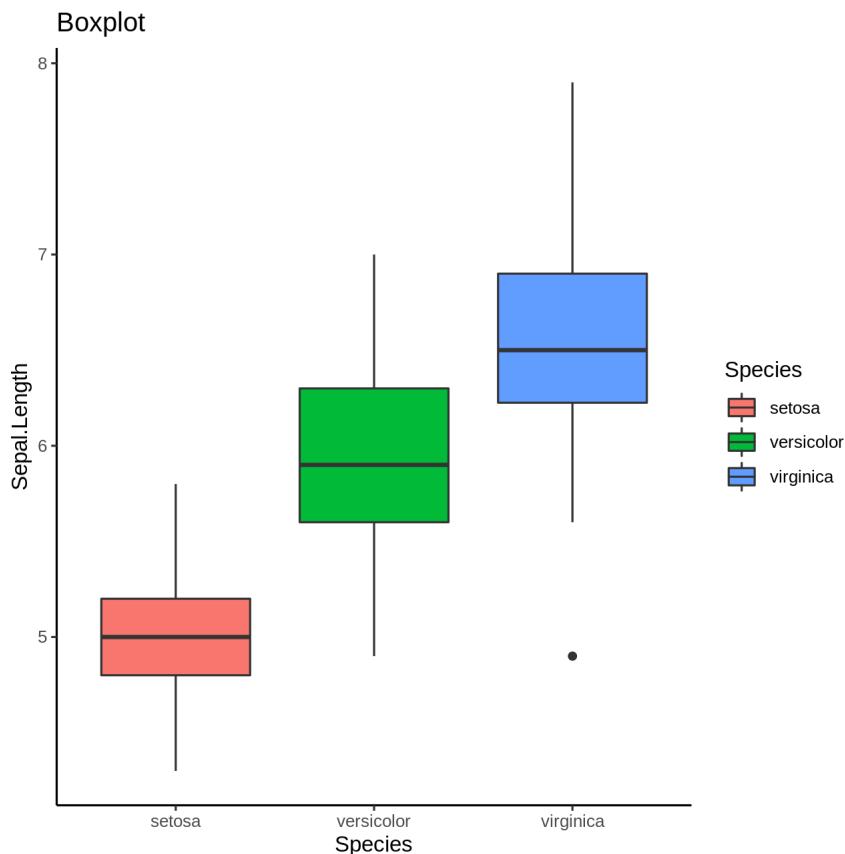
# Scatterplot and linear-fit: `geom_point()` and `geom_smooth()`

```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() +  
  labs(title = "Scatterplot")
```

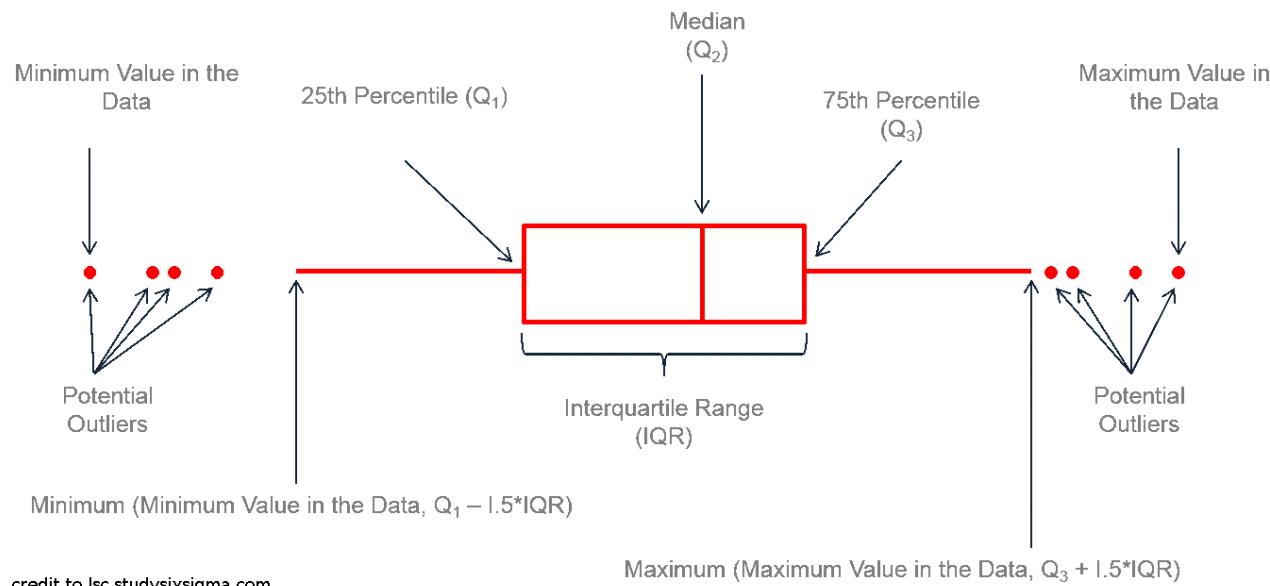
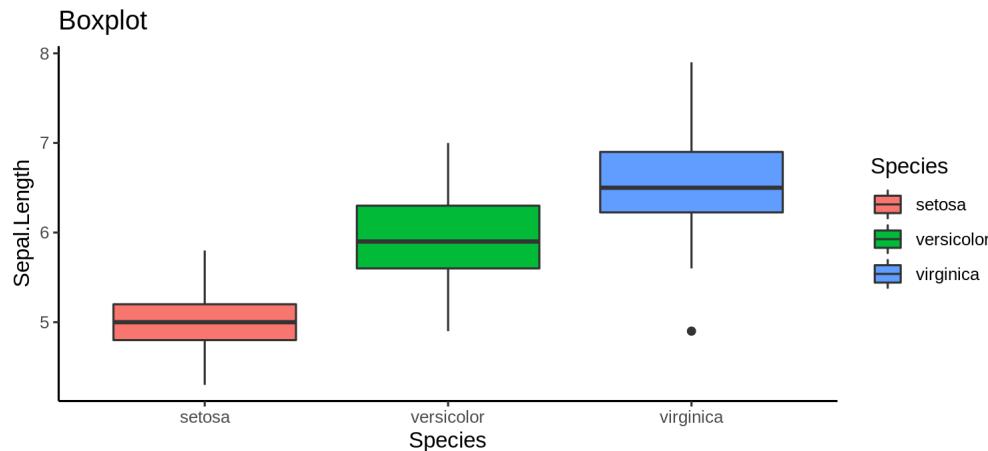
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() +  
  labs(title = "Linear Regression") +  
  geom_smooth(method = lm)
```

# Boxplot: `geom_boxplot()`

```
ggplot(data = iris, aes(Species, Sepal.Length,  
fill = Species)) +  
  geom_boxplot() +  
  labs(title = "Boxplot")
```



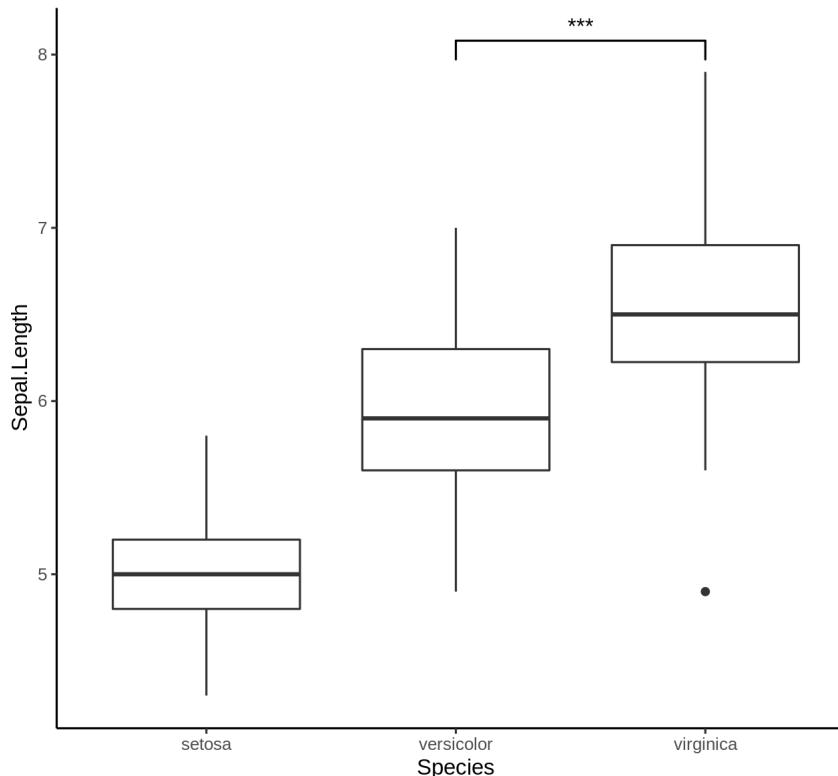
# Boxplot: `geom_boxplot()`



credit to [sc.studysixsigma.com](http://sc.studysixsigma.com)

# Boxplot with annotations: `geom_boxplot()` and `geom_signif()`

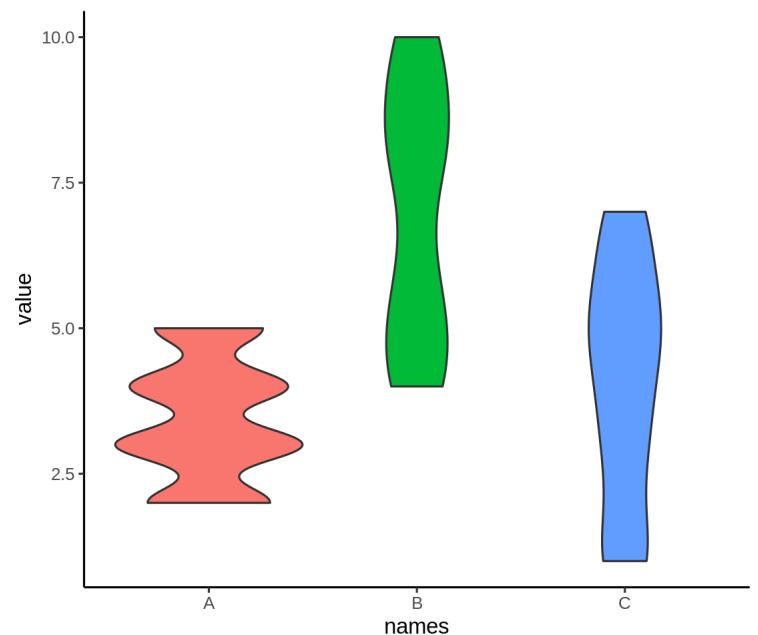
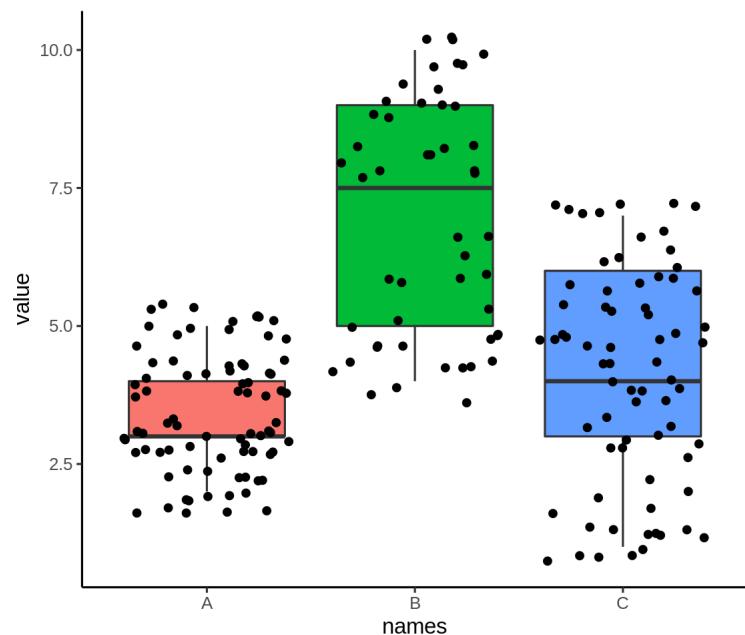
```
library(ggsignif)
ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_boxplot() +
  geom_signif(comparisons = list(c("versicolor", "virginica")),
              map_signif_level=TRUE)
```



# Violin plot: `geom_violin()`

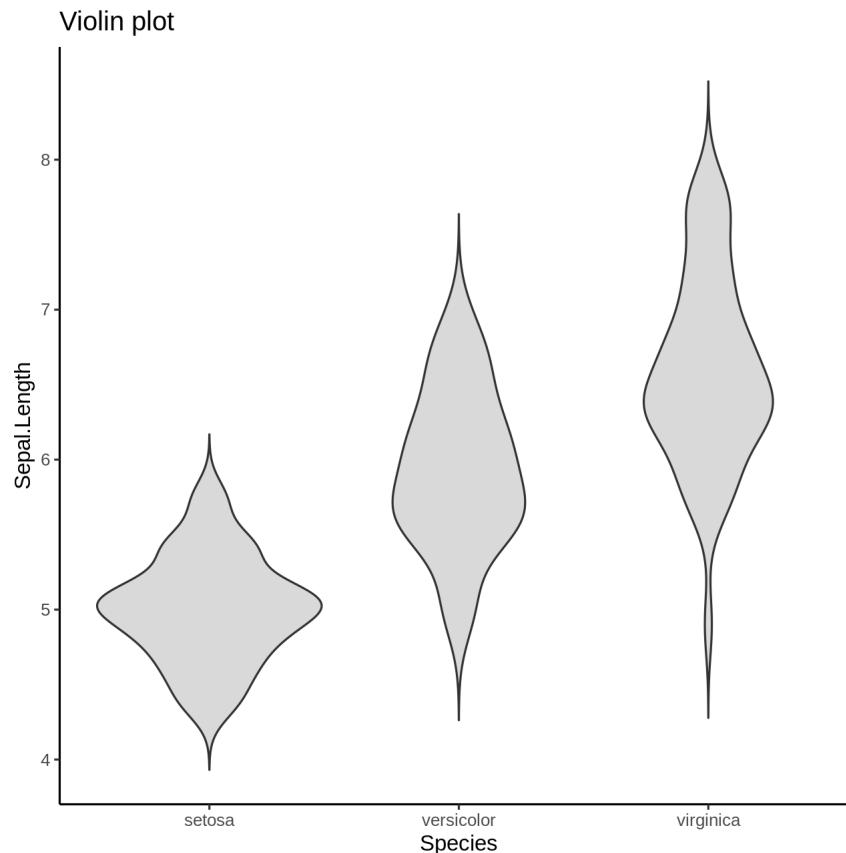
As explained on <https://www.data-to-viz.com/graph/violin.html>

Violin plot allows to visualize the distribution of a numeric variable for one or several groups. [...] It is really close to a boxplot, but allows a deeper understanding of the distribution.



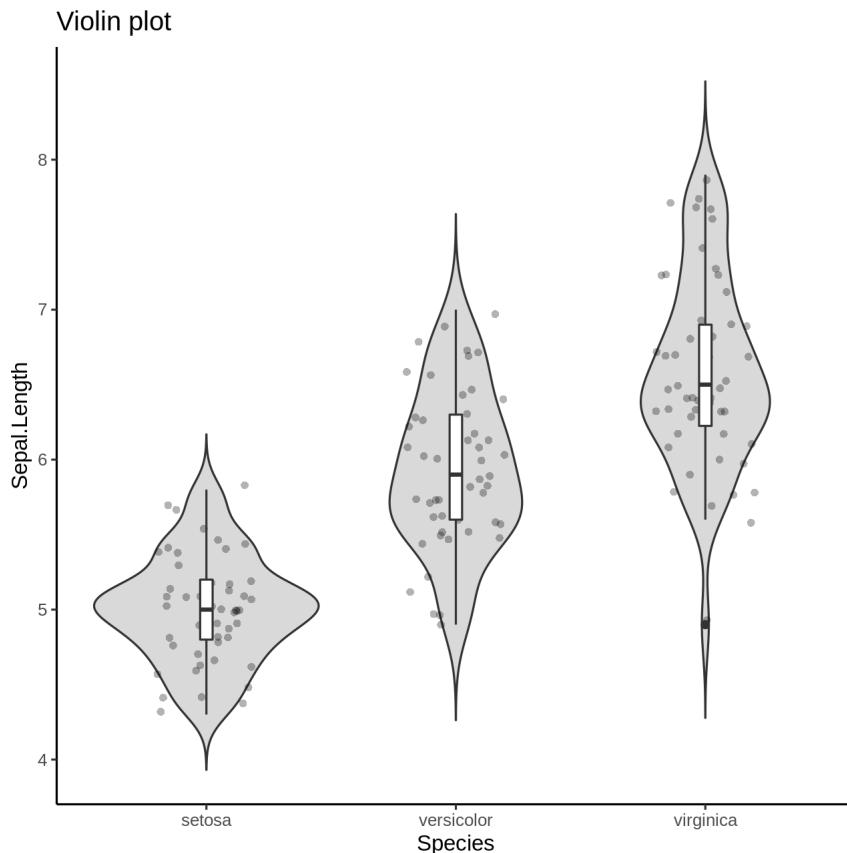
# Violin plot: `geom_violin()`

```
violin <- ggplot(data = iris, aes(x=Species, y=Sepal.Length)) +  
  geom_violin(trim=F, fill="grey70", alpha=.5) +  
  labs(title = "Violin plot")  
violin
```



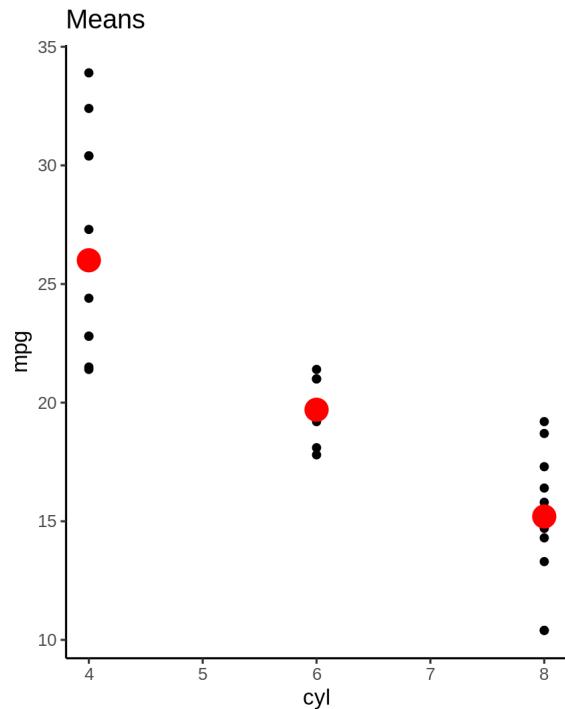
# Violin plot: `geom_violin()` + `_boxplot()` + `_jitter()`

```
violin +
  geom_jitter(shape=16, position=position_jitter(0.2),
              alpha=.3) +
  geom_boxplot(width=.05)
```

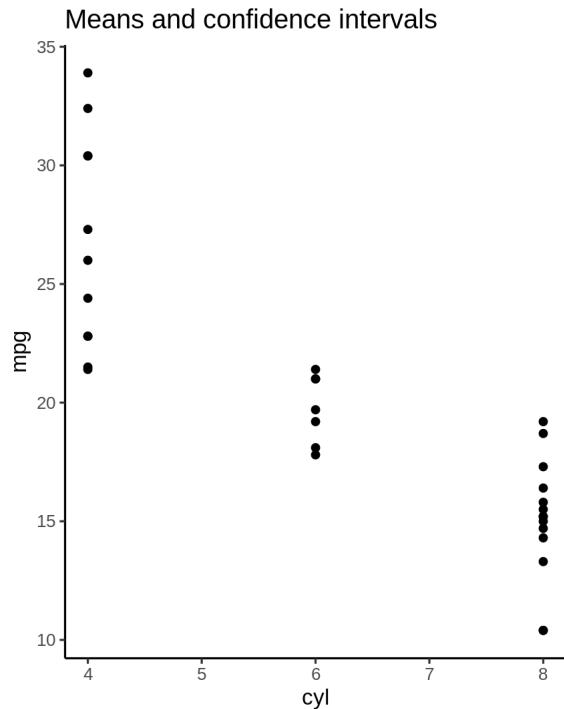


# Summarise y values: stat\_summary()

```
ggplot(mtcars, aes(cyl, mpg)) +  
  geom_point() +  
  stat_summary(fun.y = "median", geom = "point",  
              colour = "red", size=5) +  
  labs(title = "Means")
```



```
ggplot(mtcars, aes(cyl, mpg)) +  
  geom_point() +  
  stat_summary(fun.data = "mean_cl_boot",  
              colour = "red", size=2) +  
  labs(title = "Means and confidence intervals")
```



# Summarise values

See also `geom_errorbar()`, `geom_pointrange()`, `geom_linerange()`,  
`geom_crossbar()` for geoms to display summarised data.

# Representing maps: geom\_map( )

```
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
crimesm <- reshape2::melt(crimes, id = 1)

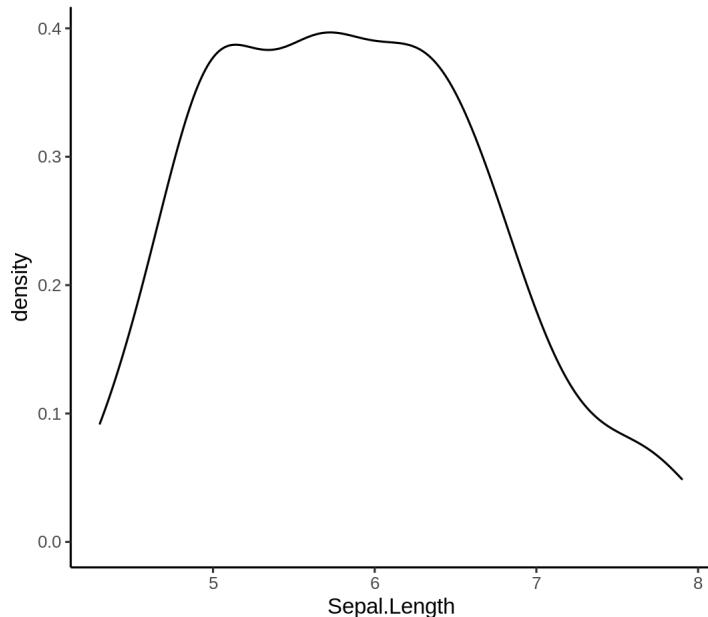
library(maps)
states_map <- map_data("state")

ggplot(crimes, aes(map_id = state)) +
  geom_map(aes(fill = Murder), map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  coord_map()
```

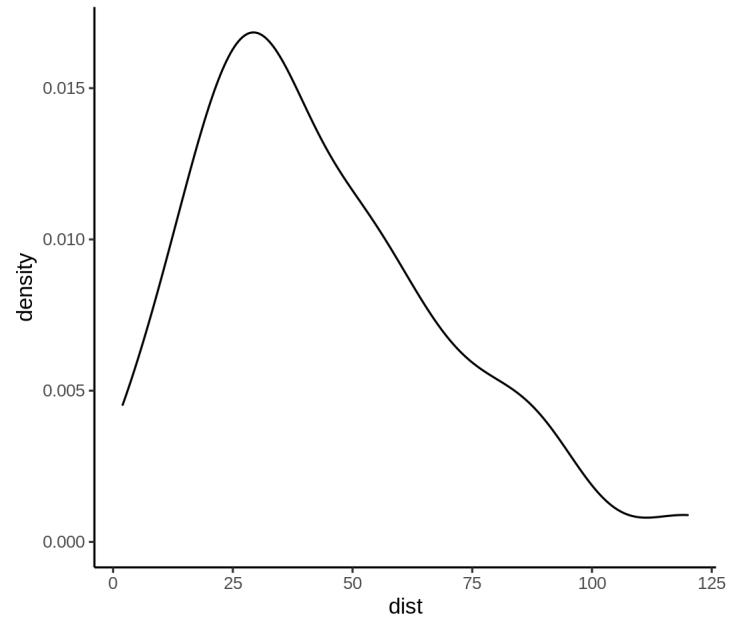
# Density plots: `geom_density()`

A density plot shows the distribution of a numerical variable and it takes only set of numeric values as input.

```
iris.dens <- ggplot(iris, aes(Sepal.Length)) +  
  geom_density()  
iris.dens
```



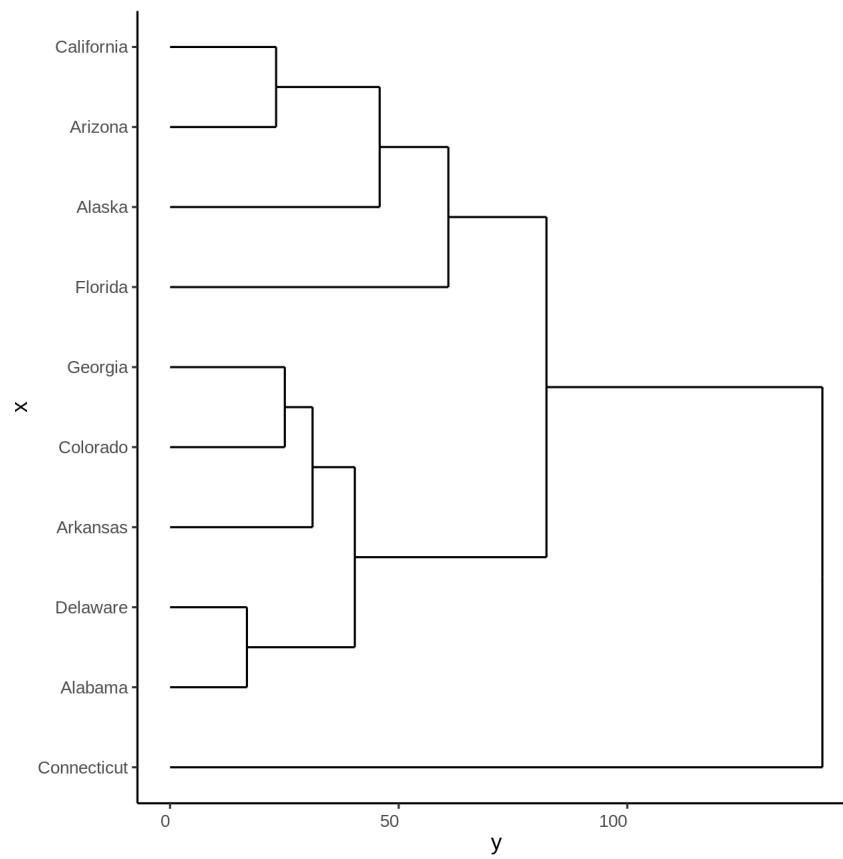
```
cars.dens <- ggplot(cars, aes(dist)) +  
  geom_density()  
cars.dens
```



# Dendrogram: ggdendrogram( )

```
library(ggdendro)
USArrests.short = USArrests[1:10, ]
hc <- hclust(dist(USArrests.short), "ave")

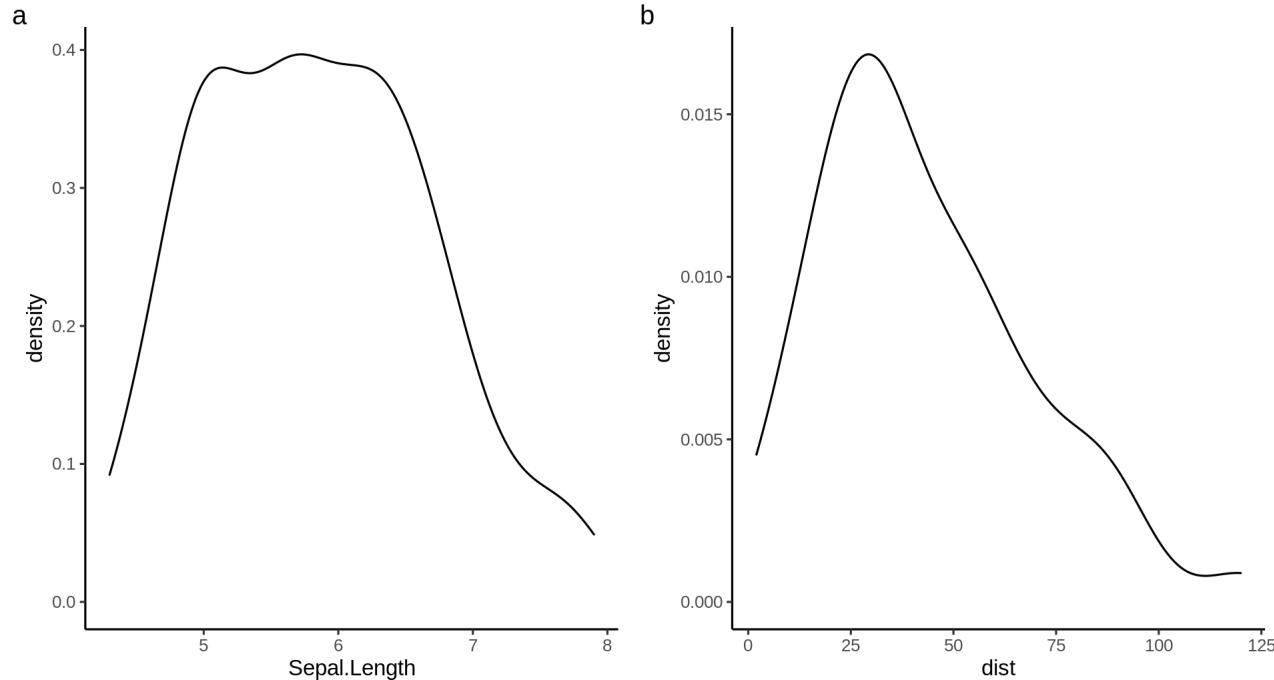
ggdendrogram(hc, rotate = TRUE, theme_dendro = FALSE)
```



# Arrange plots: patchwork

## Add multiple graphs on the same plot

```
install.packages("patchwork")
library(patchwork)
iris.dens + cars.dens +
  plot_annotation(tag_levels = 'a')
```





# Final Challenge

Create your own graph and follow these recommendations:

- Dataset: any (recommended: use your dataset)
- Explore a new geom\_\* and other plot elements

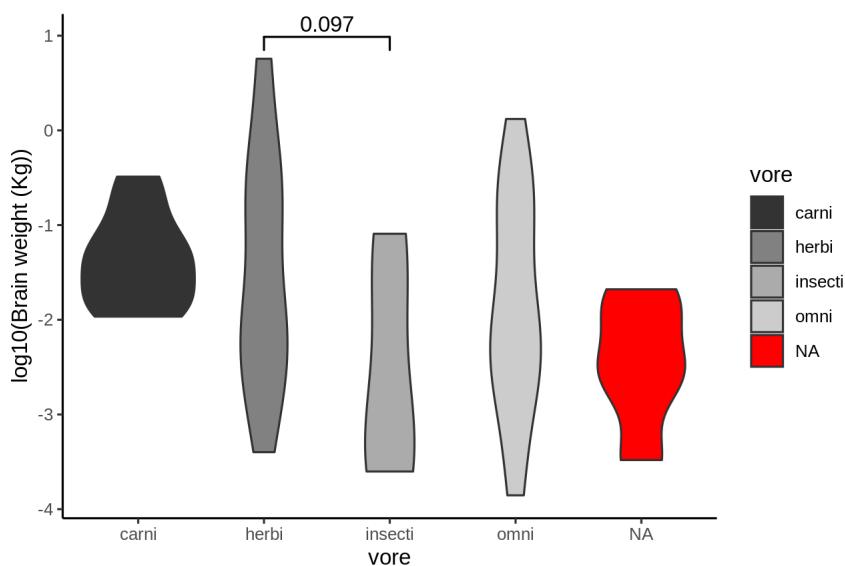
Use the following links for tips:

<https://ggplot2.tidyverse.org/reference/index.html>

<http://shinyapps.stat.ubc.ca/r-graph-catalog/>

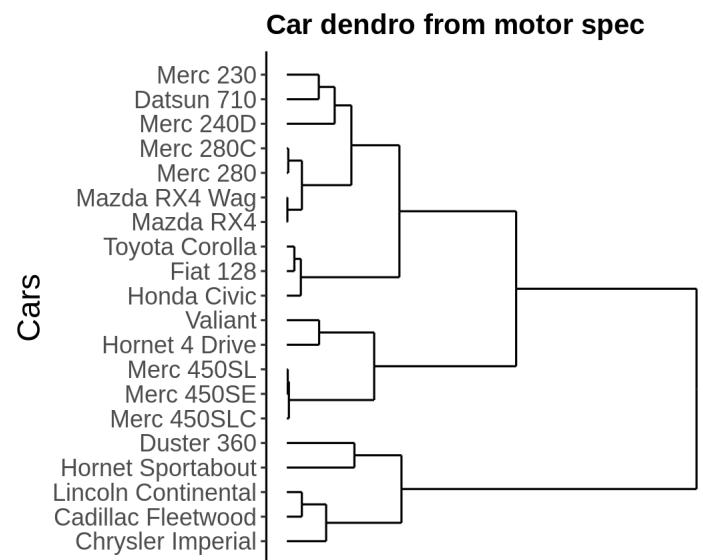
# Final Challenge: Solution example #1

```
data(msleep)
msleep.challenge4 <- ggplot(msleep, aes(vore, log10(brainwt), fill=vore))
msleep.challenge4 + geom_violin() +
  geom_signif(comparisons = list(c("herbi", "insecti")))) +
  labs(main = "Brain weight among different vore", y = "log10(Brain weight (Kg))") +
  scale_fill_grey() +
  theme_classic()
```



# Final Challenge: Solution example #2

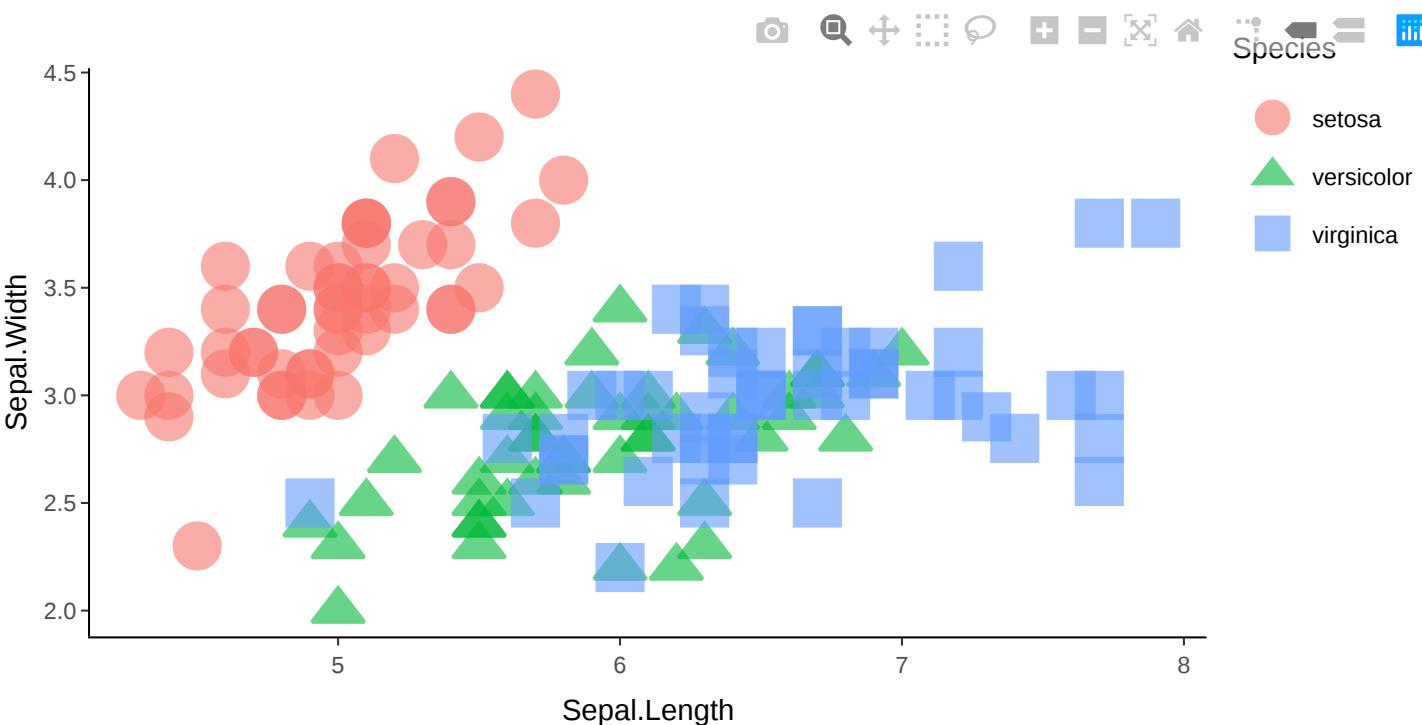
```
data(mtcars)
mtcars.short <- mtcars[1:20,]
mtcars.short.hc <- hclust(dist(mtcars.short), "complete")
dendro.challenge4 <- ggdendrogram(mtcars.short.hc, rotate = TRUE, theme_dendro = FALSE)
dendro.challenge4 + ggtitle("Car dendro from motor spec") +
  xlab("Cars") +
  theme(axis.title.y = element_text(size = 16),
        axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.text.y = element_text(size = 12),
        plot.title = element_text(size = 14, face="bold"))
```



# Miscellaneous: interactive plots using `plotly()`

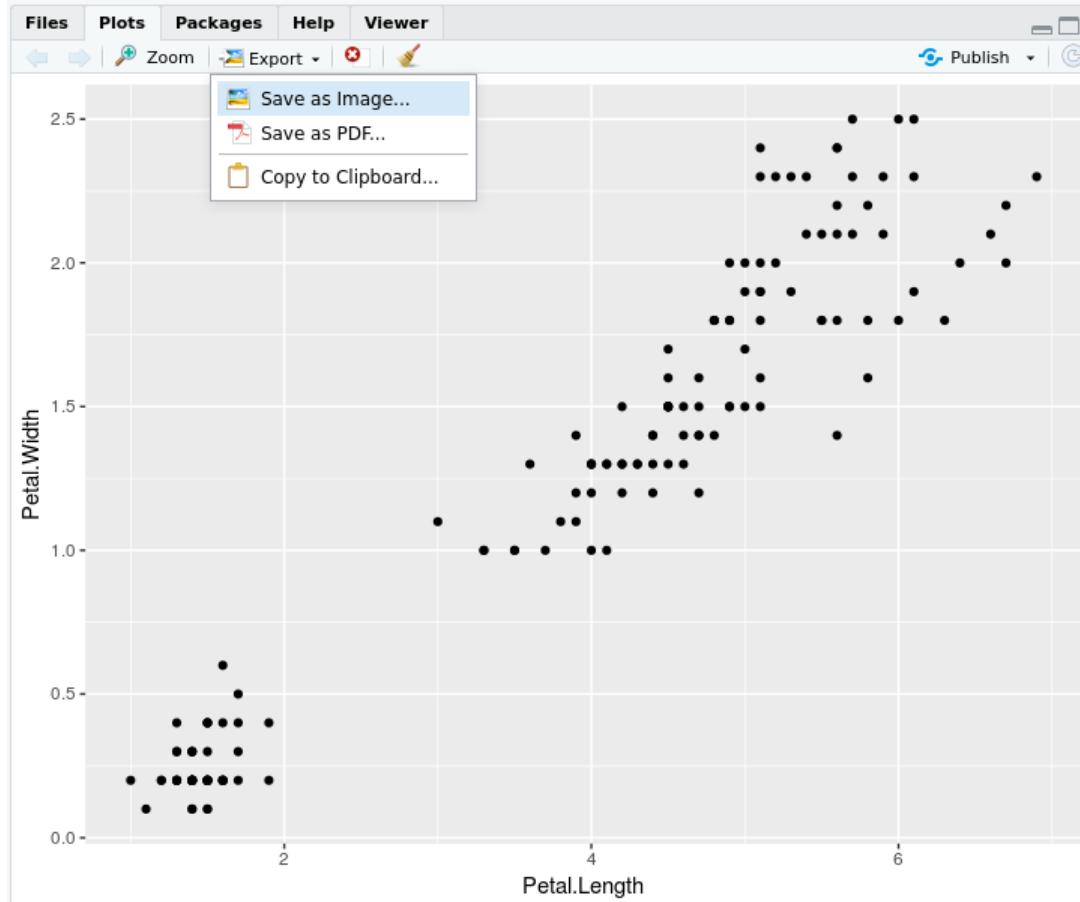
Select the species data to display directly from the legend

```
p = ggplot(iris,  
           aes(x=Sepal.Length, y=Sepal.Width, colour=Species, shape=Species)) +  
     geom_point(size=6, alpha=0.6)  
  
plotly::ggplotly(p)
```



# Saving your plots with ggplot2

# Saving plots in RStudio



Think about the margin of the document you are using. If you resize the image after saving it, the labels and text will change size as well which could be hard to read.

# Saving plots in code

`ggsave()` writes directly to your working directory and allows you to specify the name of the file, the dimensions of the plot, the resolution, etc.

```
my1rstPlot <- ggplot(iris, aes(Petal.Length, Petal.Width)) +  
  geom_point()  
  
ggsave("my1rstPlot.pdf",  
       my1rstPlot,  
       height = 8.5, width = 11, units = "in", res = 300)
```

NB: Vectors (e.g., pdf, svg) format are more flexible than raster format (jpeg, png, ...) if the image needs modification afterwards.

# Saving plots in code

Other methods to save image: see [?pdf](#) [?jpeg](#)

```
pdf("./graph_du_jour.pdf")
print(my1rstPlot) # print function is necessary
graphics.off()
```

# Saving plots in code

**Tip:** `quartz()` (mac) or `window()` (pc) functions make sizing easier before `ggsave( )!`

**Thank you for attending!**

