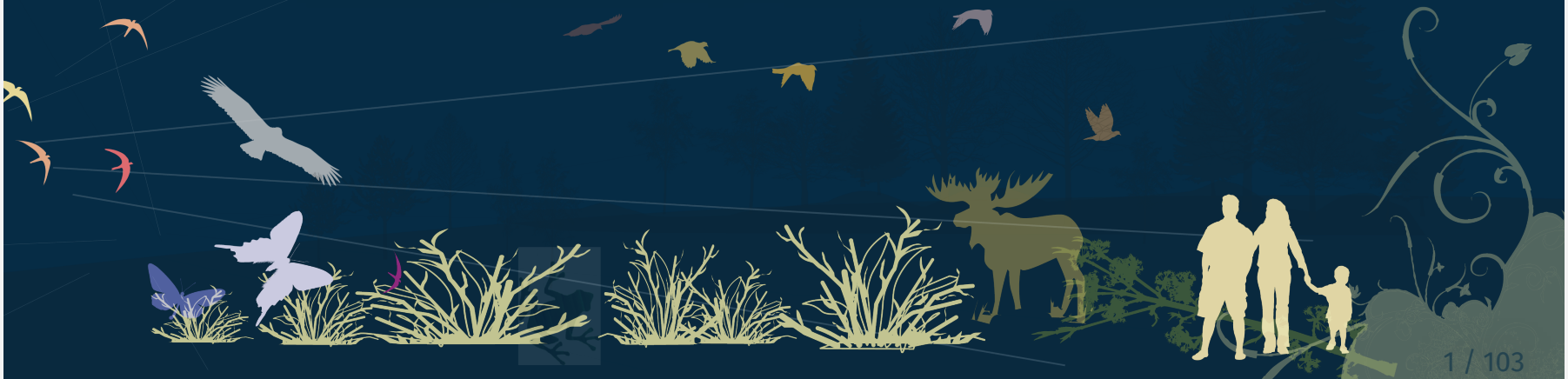
Workshop 3: Introduction to `ggplot2`

QCBS R Workshop Series

Québec Centre for Biodiversity Science



À propos de cet atelier

REPO DEV | WIKI 06 | DIAPOS 06 | DIAPOS 06 | SCRIPT 06

Packages requis

- grid
- gridExtra
- ggplot2
- ggpubr
- ggsignif
- ggdendro
- maps
- mapproj
- RColorBrewer
- psych
- plotly

```
install.packages(c('grid', 'gridExtra', 'ggplot2', 'ggpubr', 'ggsignif', 'ggdendro',
```

Leaning objectives

1. Use R to do various plots

Introduction

Introduction

To follow along:

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

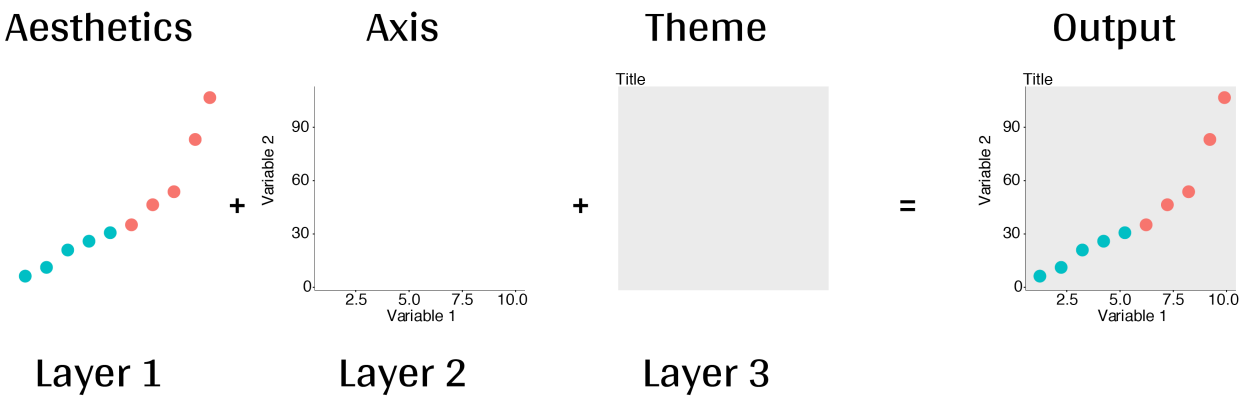
Recommendations:

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.
- A good representation of one's data can be unique and thus no single function or package can meet everyone's needs.
- Creativity is important to scientists!
- A basic understanding of design to communicate effectively with graphs.

Visualization in science

Visualization in science

Why are we using data visualization?

What makes an effective visualization?

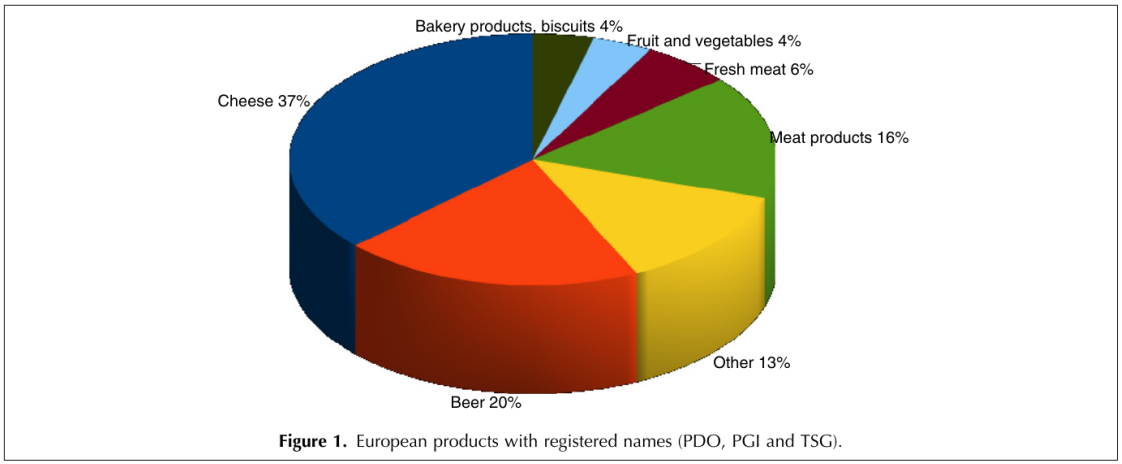


Figure 1. European products with registered names (PDO, PGI and TSG).

What do you thing about this one ↑?

Visualization in science

1. Represent results of statistical analyses
2. Research and discuss (preliminary data from the literature, summaries, formulate hypotheses)
3. Explore your own data (exploratory data analysis, outlier detection)
4. Communicate and report
    - Clearly (design principles)
    - Precisely and accurately
    - Effectively and efficiently

Visualization in science

Important questions:

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

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

Additional resources

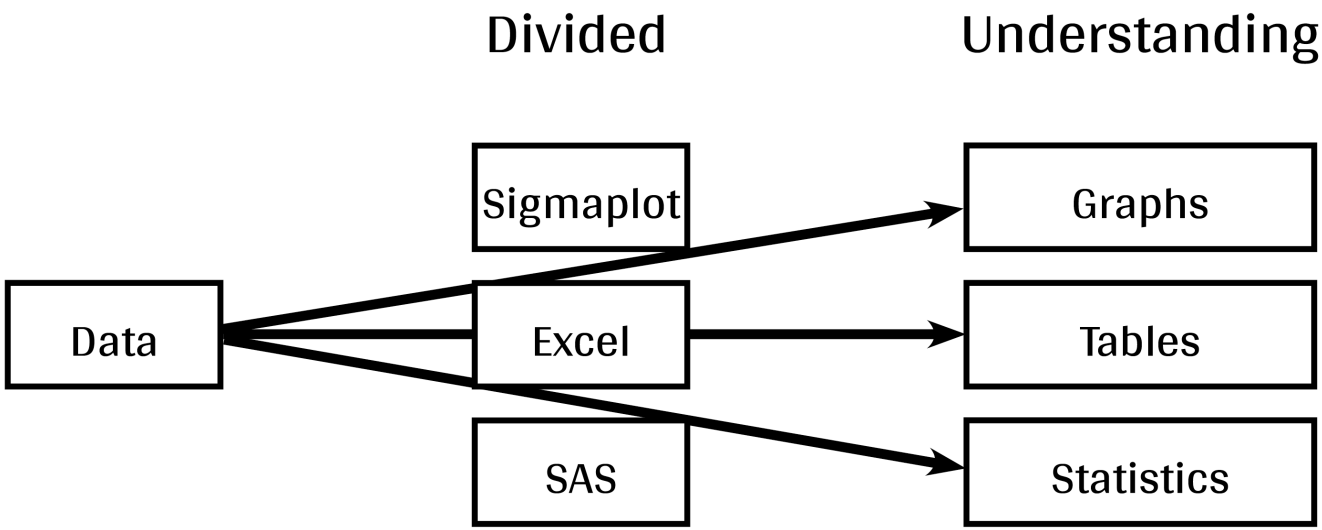
- Fundamentals of Data Visualization (ggplot)
- A Compendium of Clean Graphs in R (base plot)

WARNING!

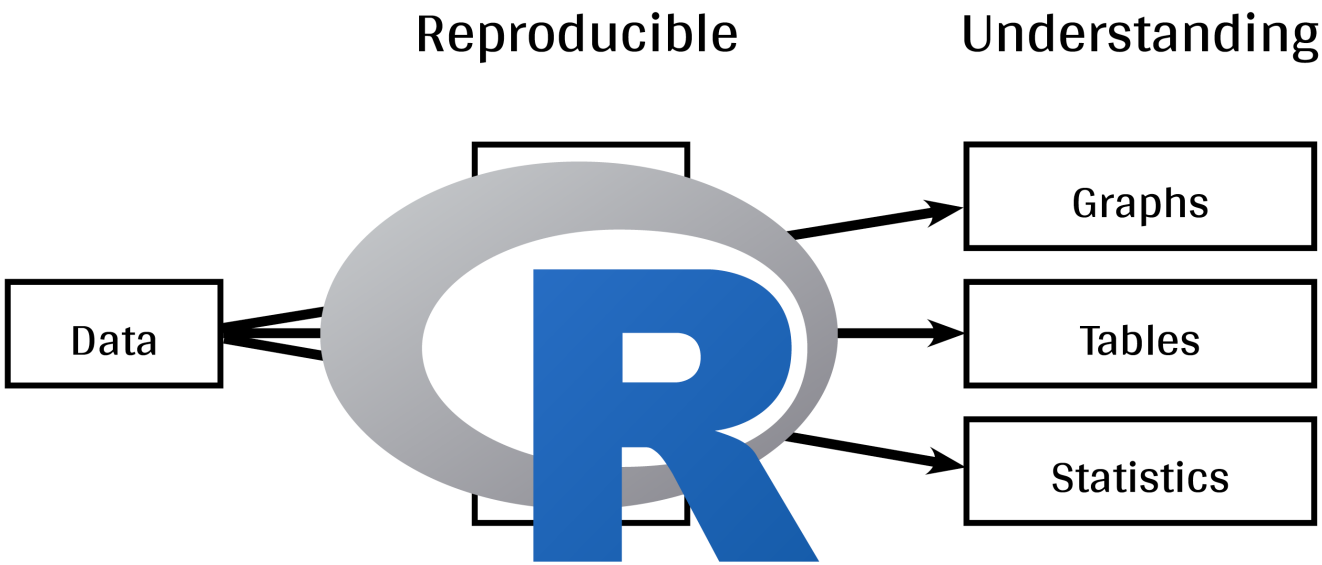
⚠ R is not made for drawings.

Other drawing softwares are probably more interesting 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



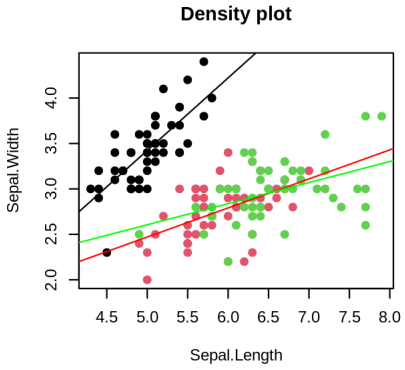
Reproducible science comes with effort:

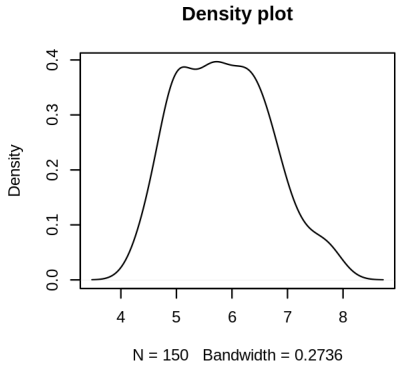
- comment the script
- add relevant information in the figures (titles, labels, captions)

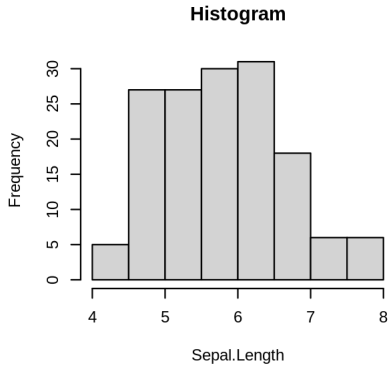
Why R for graphs?

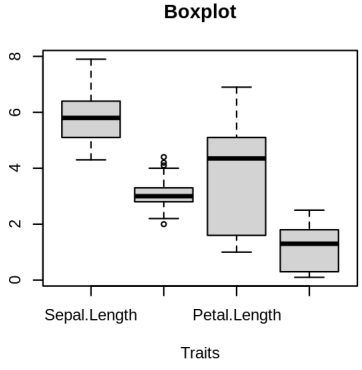
Because of its powerful features!

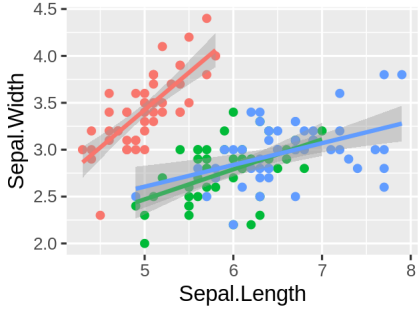
In this workshop, we focus only on `ggplot2`, but multiple packages and functions can be used for great visualization (e.g., "base R", plotly, sjPlot, mapview, igraph).

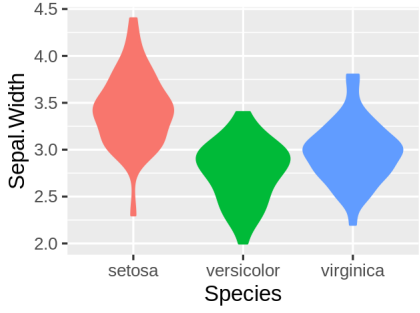






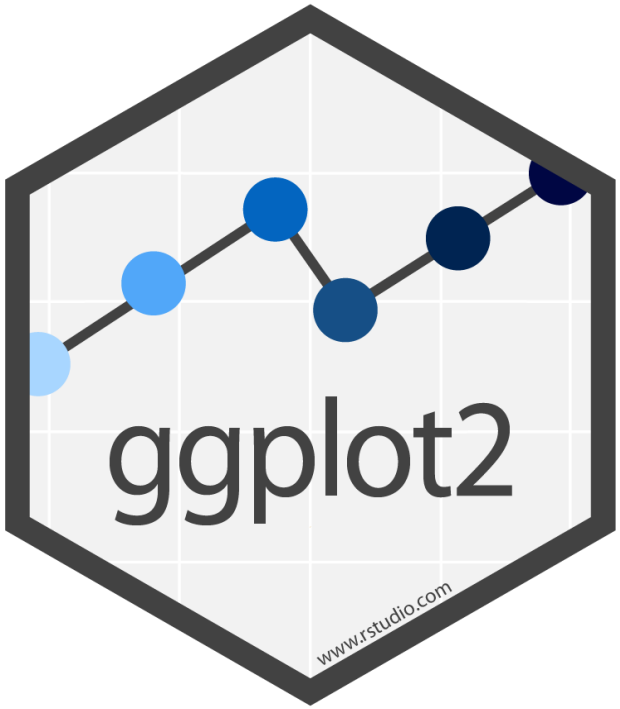






`ggplot2` is versatile

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

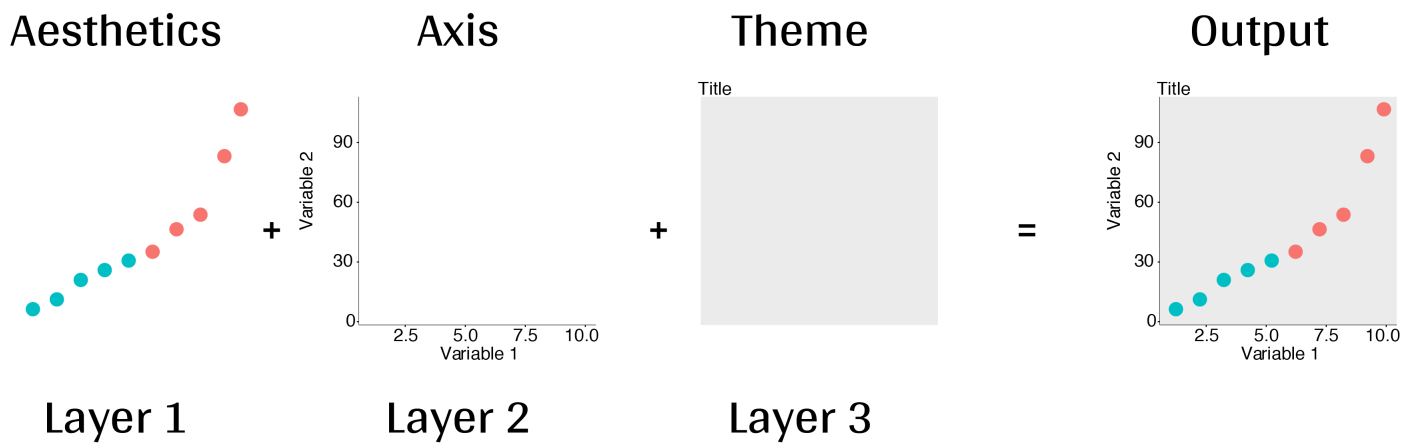


`ggplot2` mechanics: the basics

Grammar of Graphics (GG) basics

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

A graphic is made of different layers:



We will learn how to build graphs step by step to make it look as we desire.

Grammar of Graphics (GG) basics

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

Major Components of the Grammar of Graphics

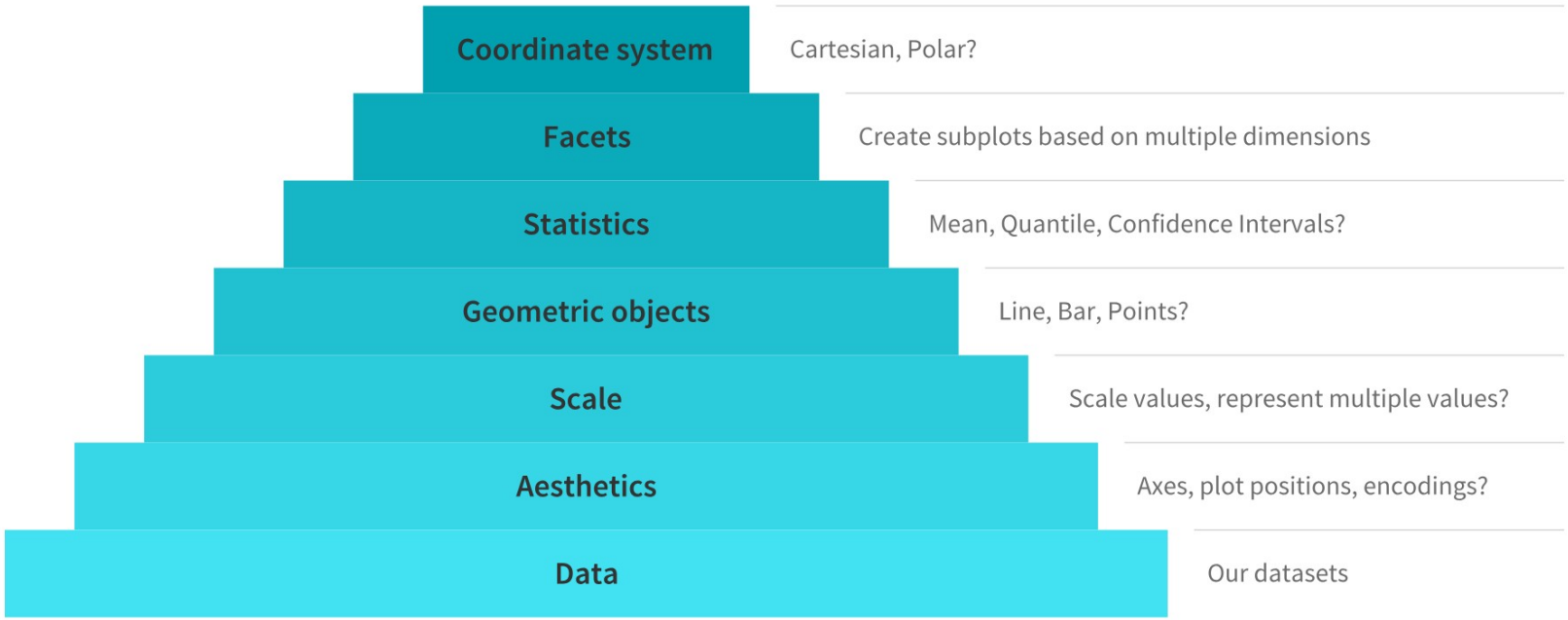
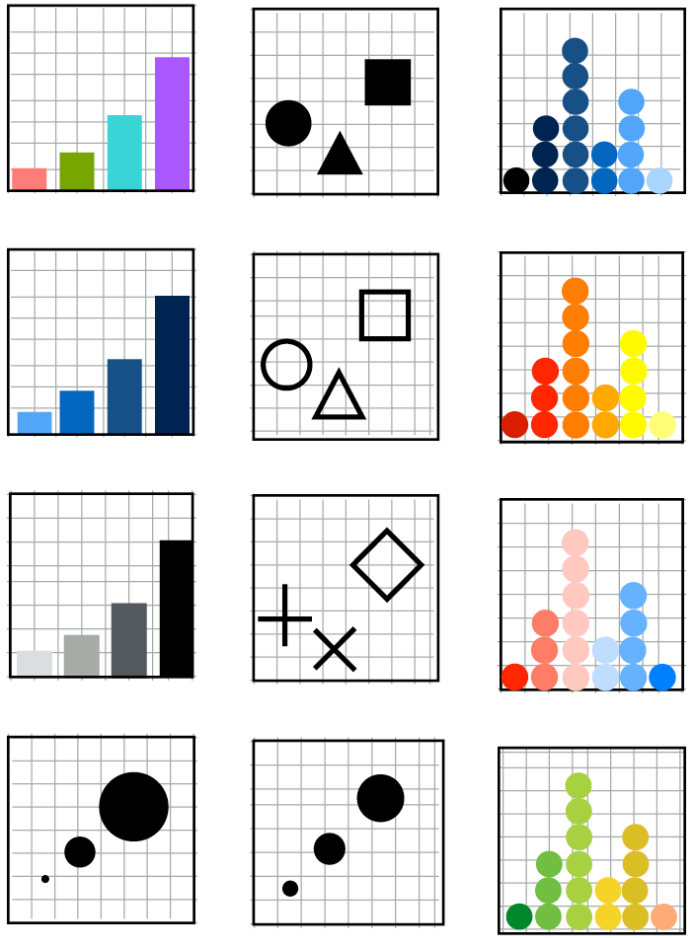


image from Towards Data Science

Grammar of Graphics (GG) basics

Scales



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)

A graphic is made of elements (layers)

- Data
- Aesthetics (aes), to make data visible
  - `x`, `y`: position along the x and y axis
  - `colour`: the colour of the point
  - `group`: what group a point belongs to
  - `shape`: the figure used to plot a point
  - `linetype`: the type of line used (solid, dashed, etc)
  - `size`: the size of the point or line
  - `alpha`: the transparency of the point

Grammar of Graphics (GG)

A graphic is made of elements (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

How layer in ggplot works

1. Create a simple plot object:
   - `plot.object <- ggplot()`
2. Add graphical layers:
   - `plot.object <- plot.object + layer()`
3. Repeat step 2 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
```

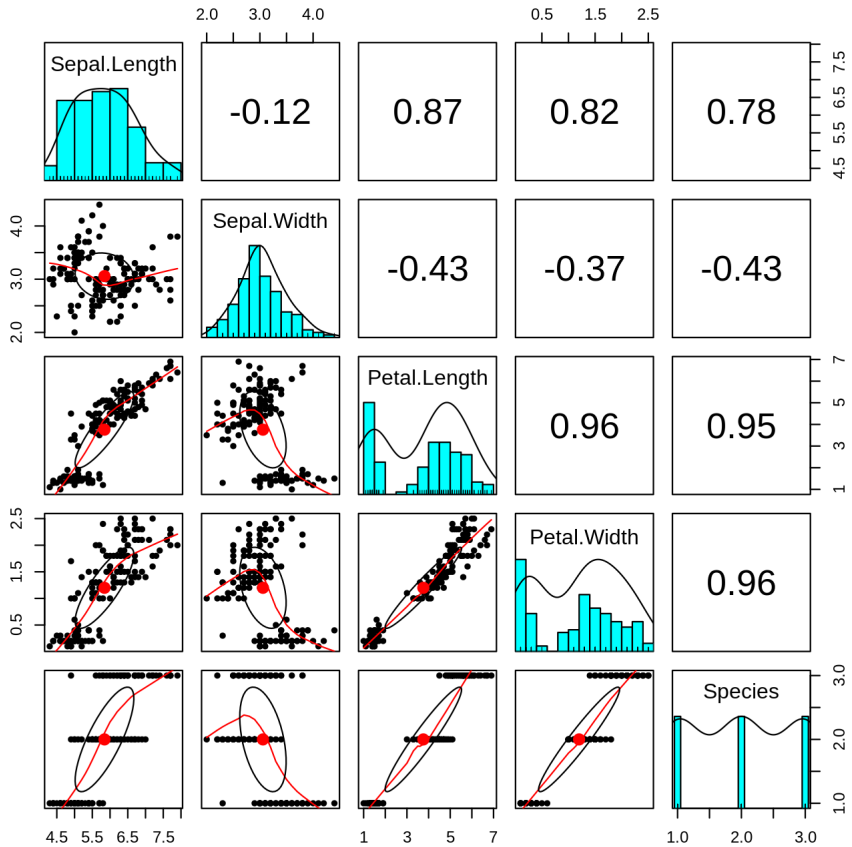
Scientific questions

- 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** ?

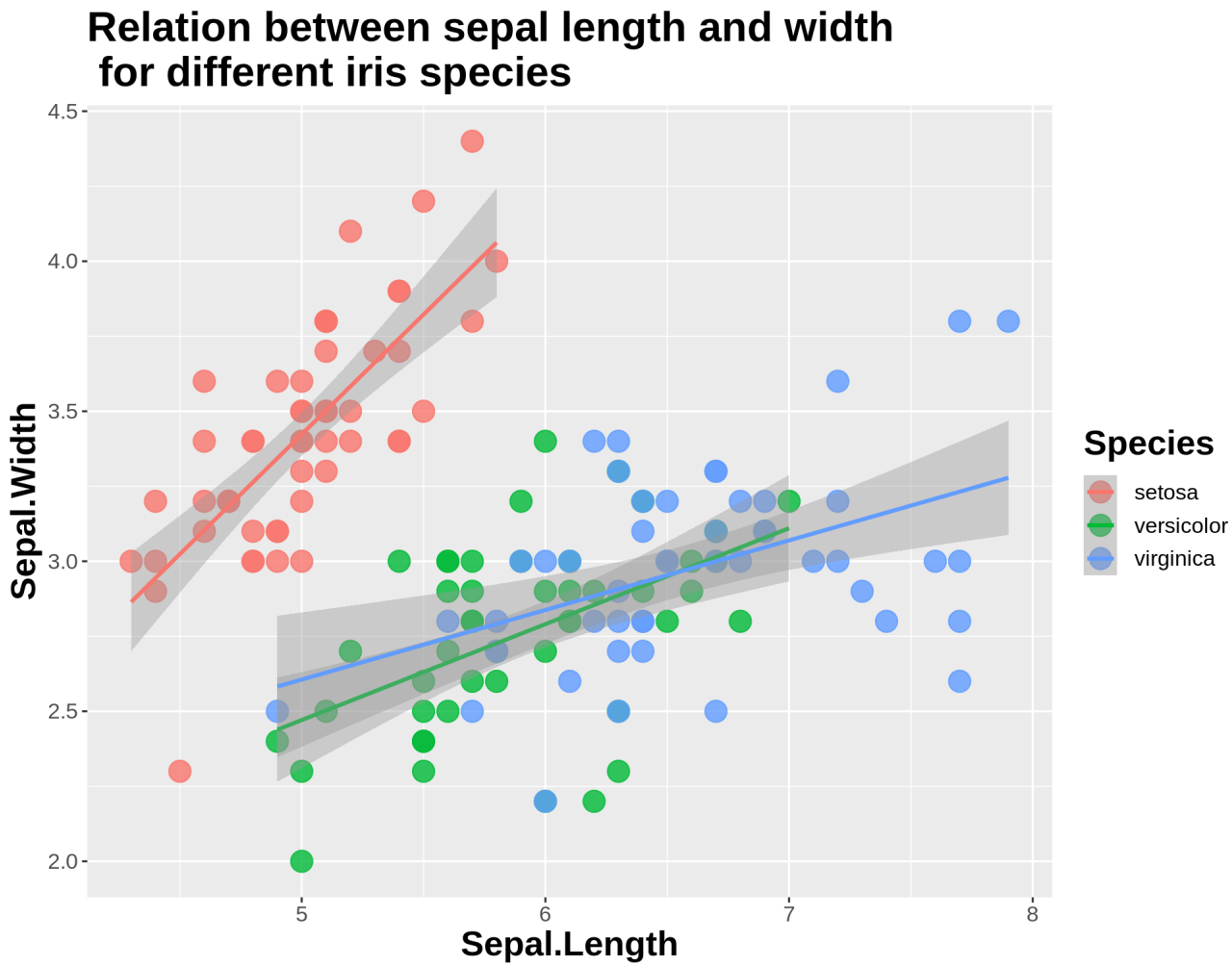
How to graphically address these questions with ggplot ?

Exploring data structure

```
library(psych)
pairs.panels(iris)
```



Exploring data structure

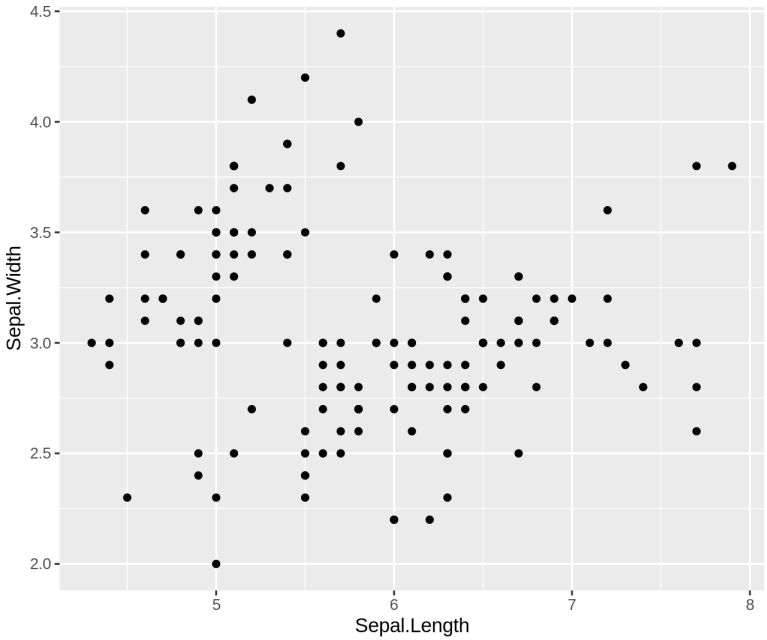


♻ See *Loading and manipulating data workshop* to learn how to clean your dataset.

Inheritance in ggplot2

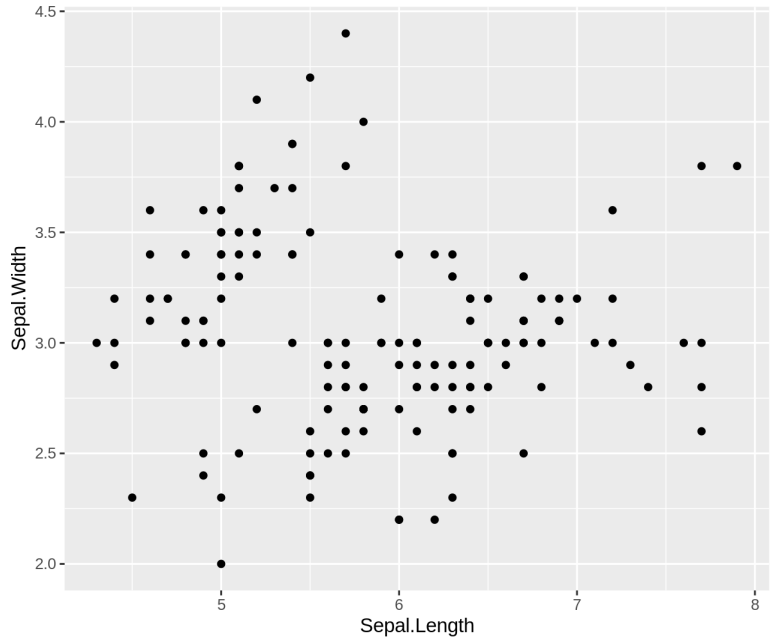
Inheritance from ggplot

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



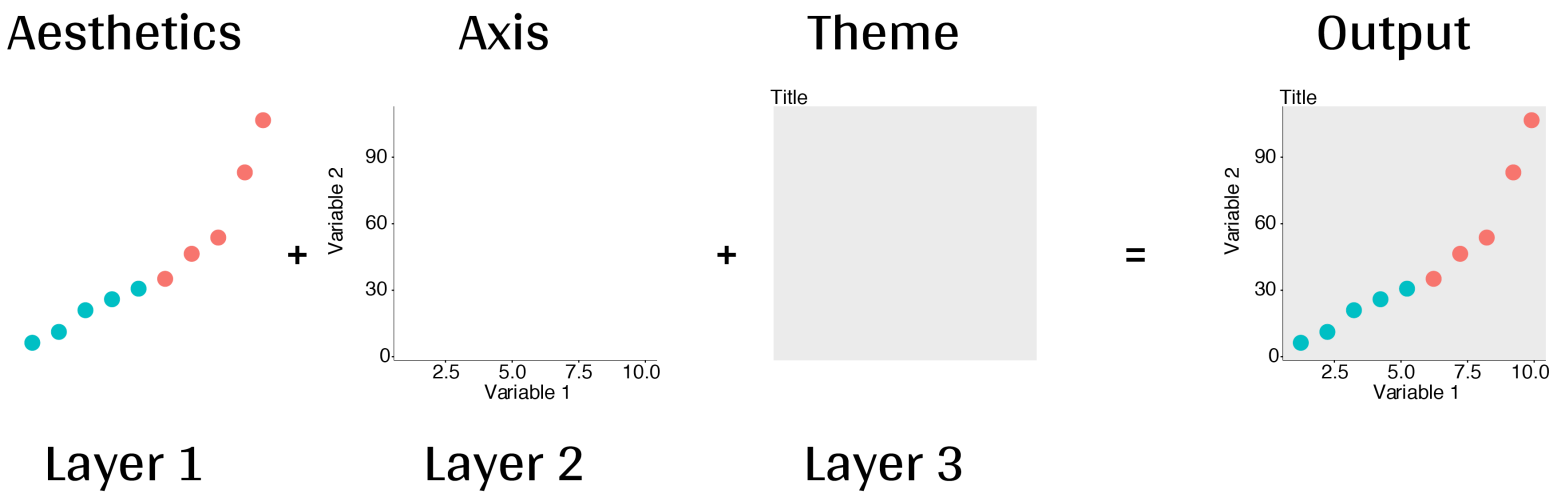
No inheritance from ggplot

```
s <- ggplot()
s <- s + geom_point(data = iris,
                    aes(x = Sepal.Leng
                        y = Sepal.Widt
s # Print your final plot
```



Grammar of Graphics: *recall*

Remember: a graphic is made of different layers:

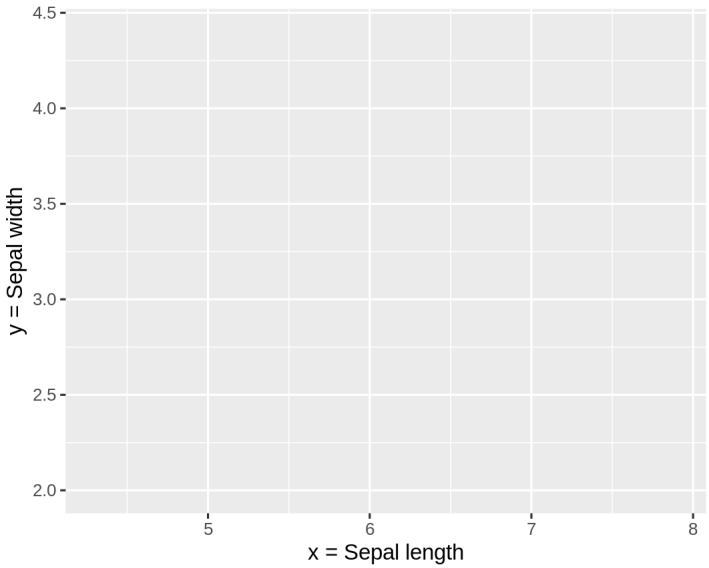


`ggplot()` dynamics: base layer

```
ggplot()
```

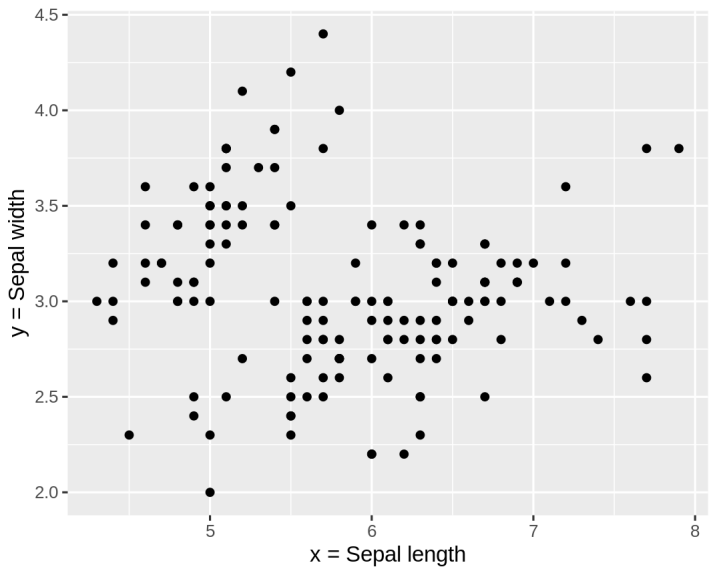
`ggplot()` dynamics: data layer

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  xlab("x = Sepal length") +
  ylab("y = Sepal width")
```



`ggplot()` dynamics: geometric Layer

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  xlab("x = Sepal length") +
  ylab("y = Sepal width") +
  geom_point()
```



Challenge #1 (5min)

Draw your 1st ggplot!

Question

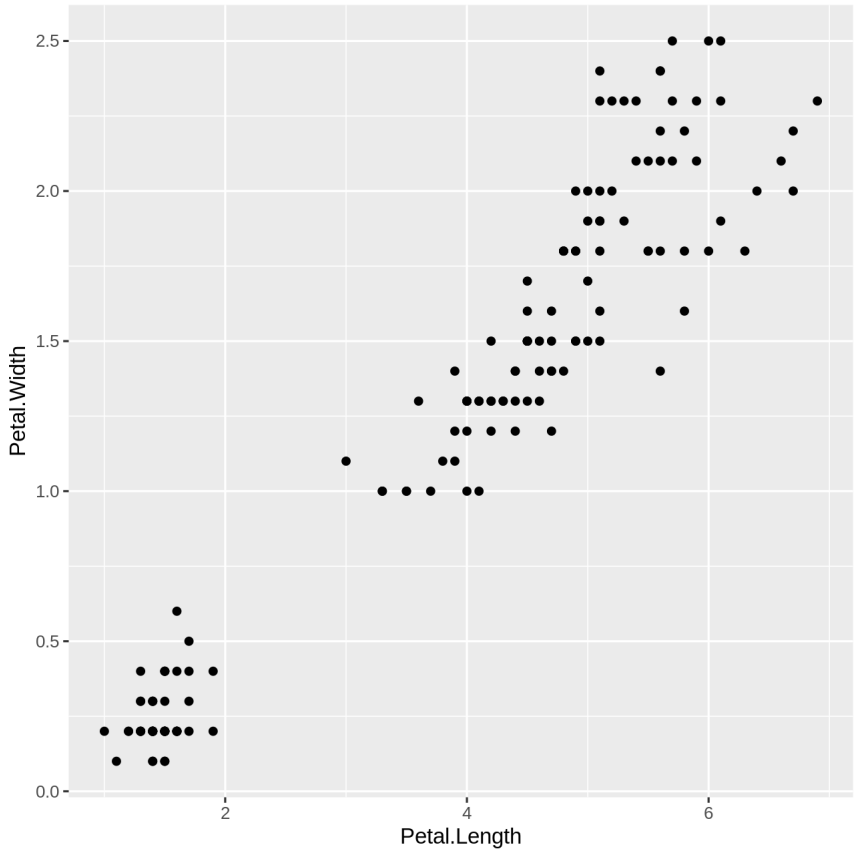
> Is there a relation between the **length** & the **width** of the iris **petal** ?
>
> Does the *width* of the petal increase with its *length* ?

Parameters to consider when addressing this question:

data	geom	x value	y value
iris	geom_point	Petal length	Petal width

Challenge 1#: Solution

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

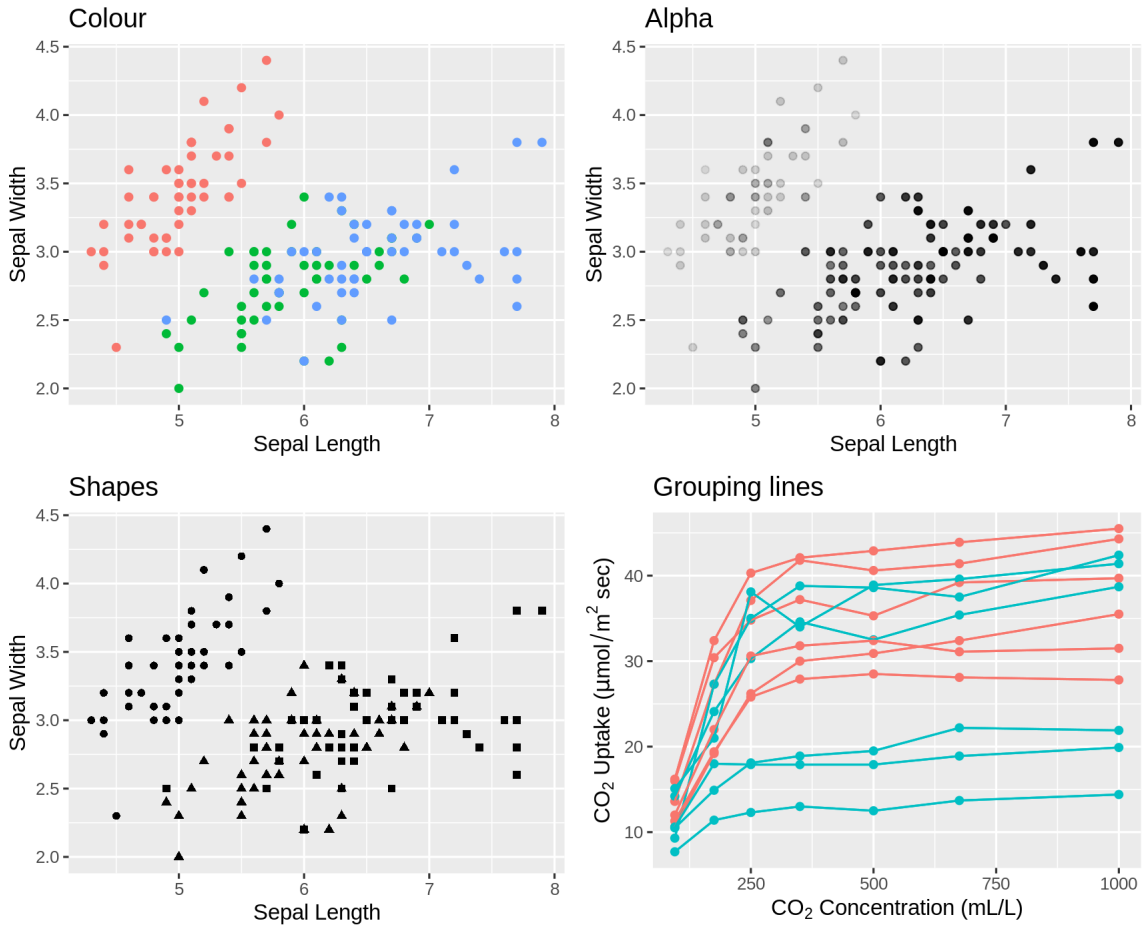


Aesthetic mapping

colour, shape, size, labels and transparency

Aesthetic

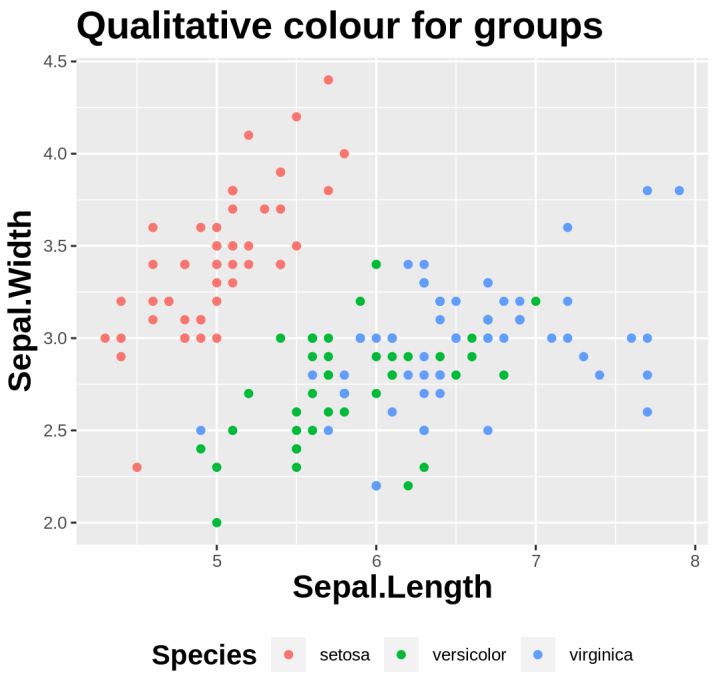
Use aesthetics (`aes()`) to distinguish classes, groups and structure

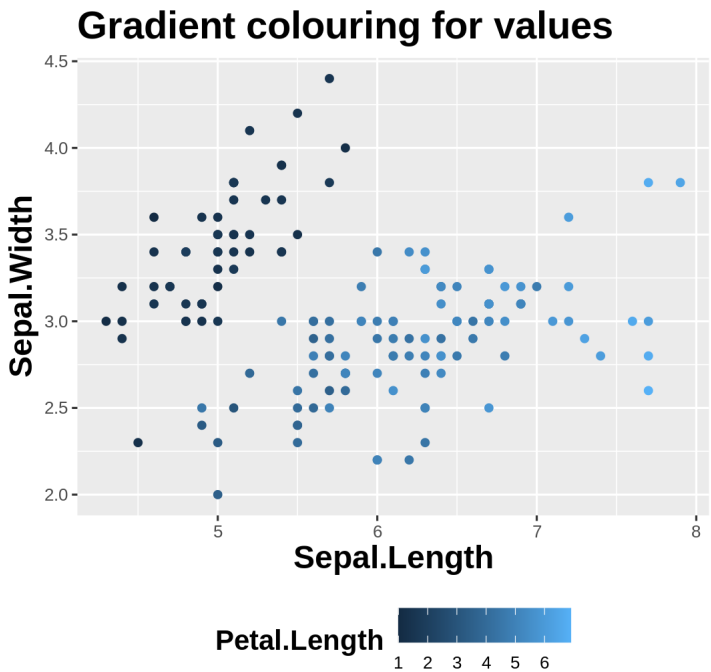


Colouring: make your points talk

Change **colour** to

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



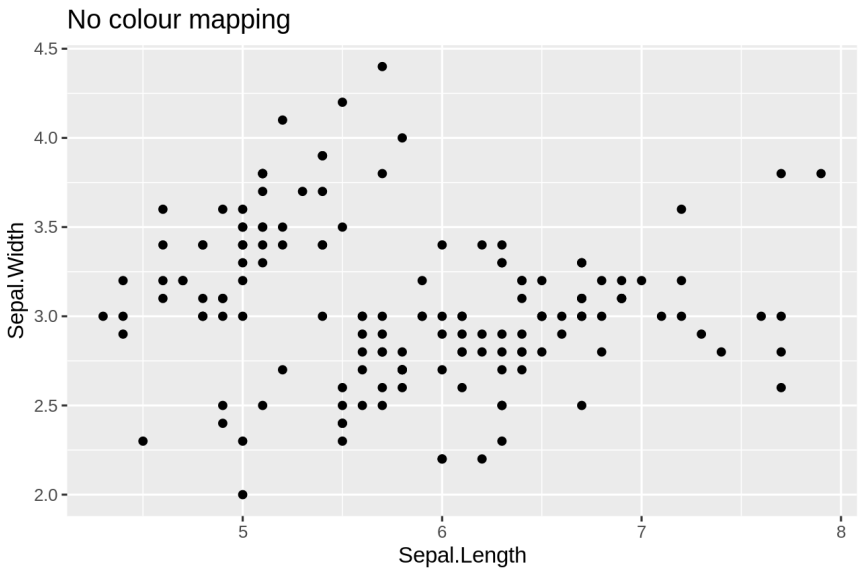


See Fundamentals of Data Visualization

Using `aes()` use to change colour

Do the sepal length and width vary differently across species?

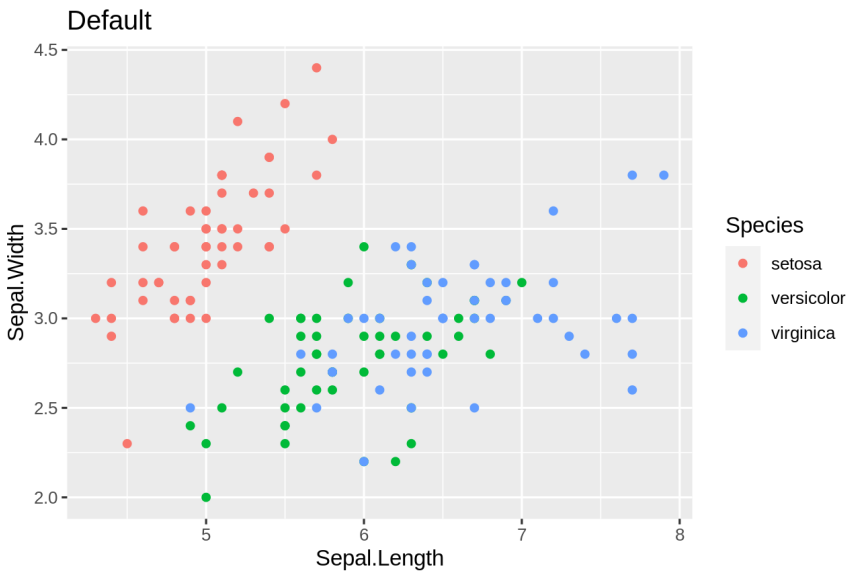
```
# No colour mapping
ggplot(data = iris) +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  labs(title = "No colour mapping")
```




```
# With colour mapping
ggplot(data = iris) +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +
```

Change colour manually

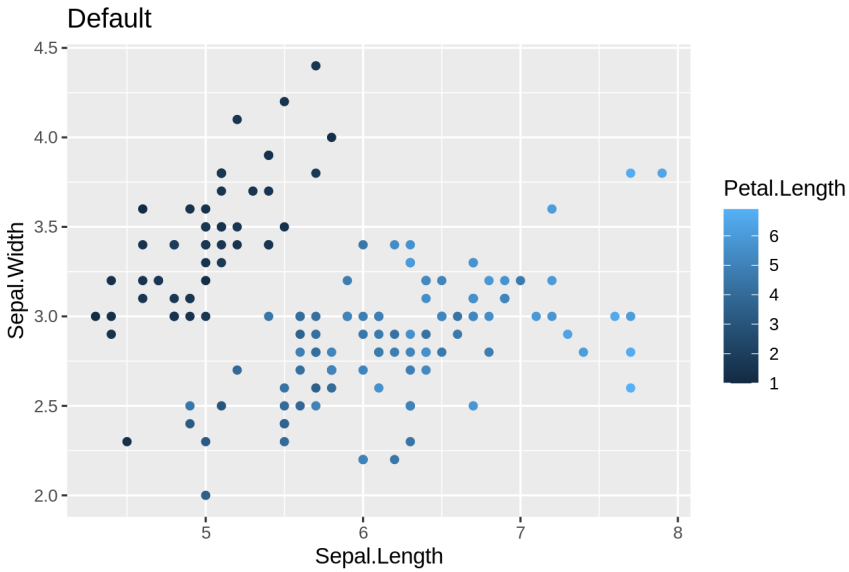
```
# Default
pp <- ggplot(data = iris) +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, colour = Species))
pp + labs(title = "Default")
```




```
# Manual
pp +
  scale_colour_manual(values = c("grey55", "orange", "skyblue")) +
  labs(title = "Manual")
```

Colour gradient

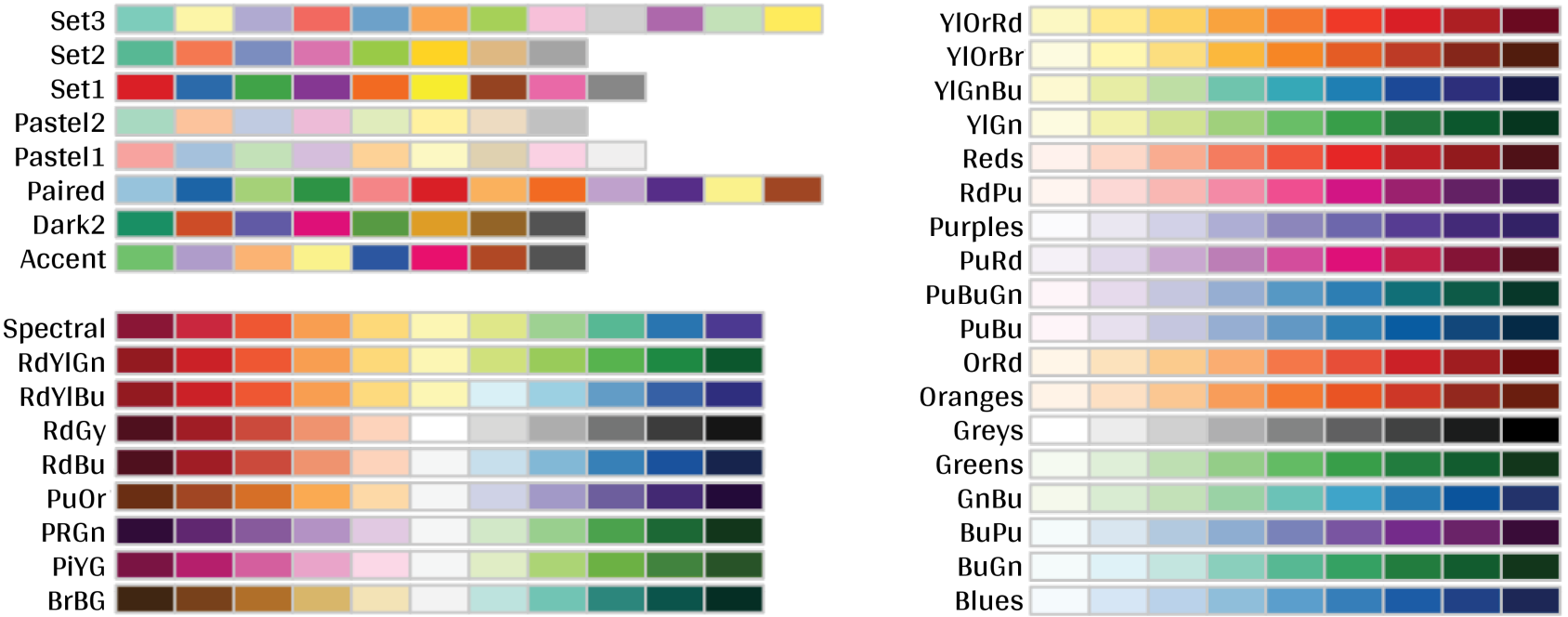
```
# Default
pp2 <- ggplot(data = iris) +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width,
                           colour = Petal.Length))
pp2 + labs(title = "Default")
```




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

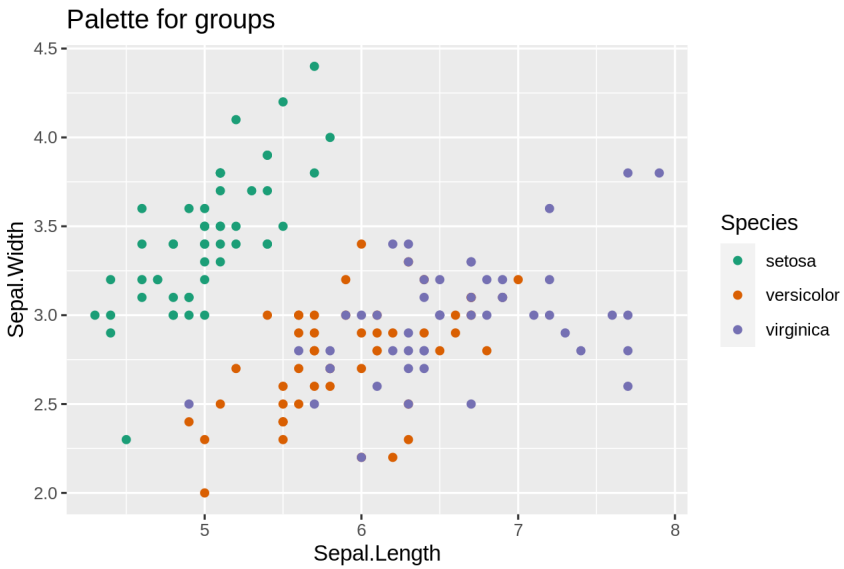
Use a predefined colour palette

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



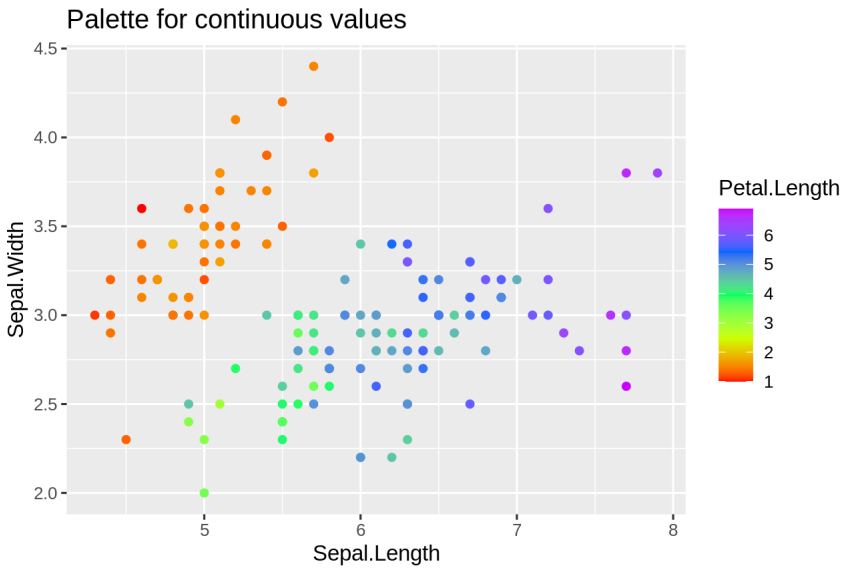
Use a predefined colour palette

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



Use a predefined colour palette

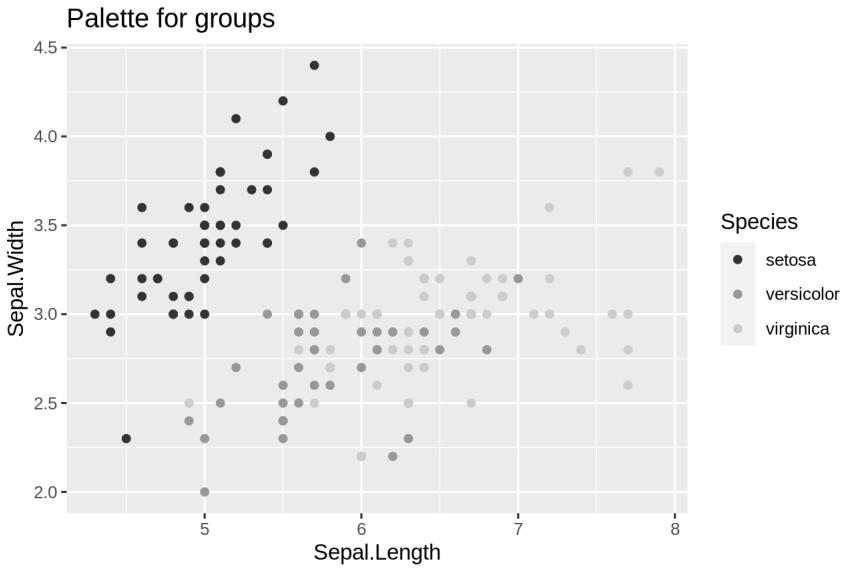
```
# Palette for continuous values
pp2 + scale_colour_gradientn(colours = rainbow(5)) +
  labs(title = "Palette for continuous values")
```



Use a predefined colour palette

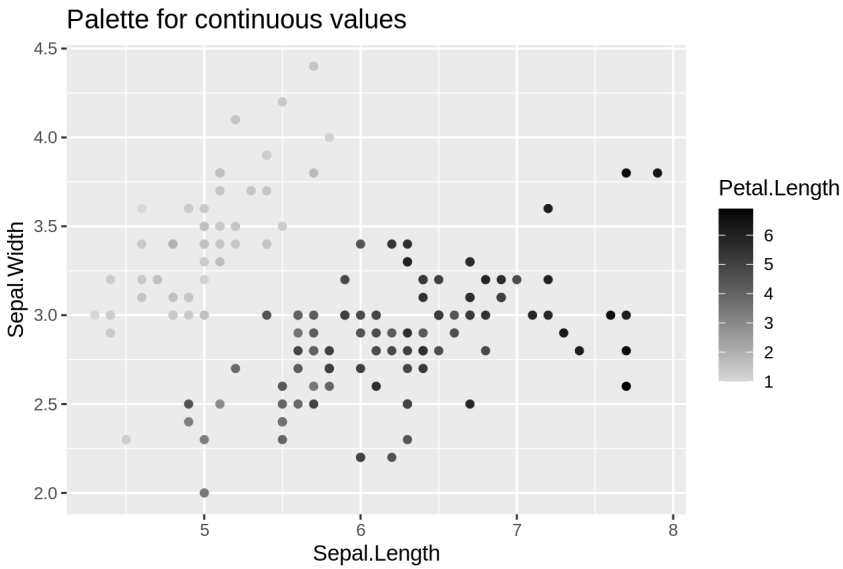
Grey-scale palette for publication purposes

```
# 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")
```



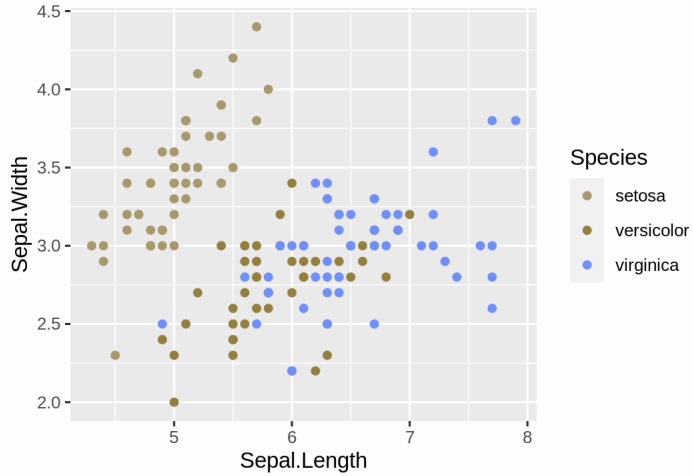
Use colourblind-friendly palettes

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.

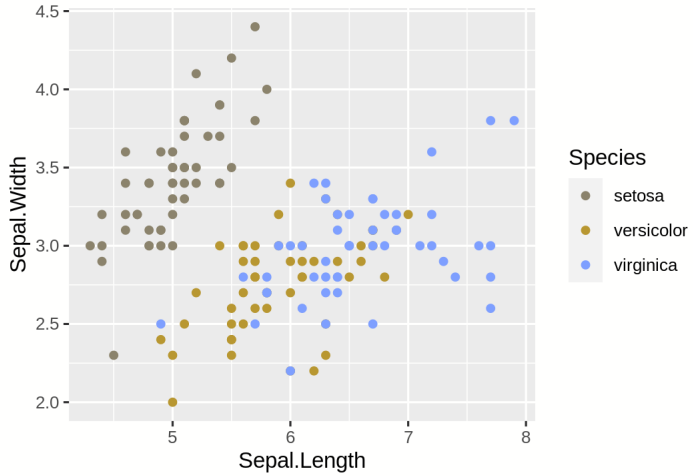
```
remotes::install_github("clauswilke/colorblindr", quiet = TRUE)
library(colorblindr)
```

Use colourblind-friendly palettes

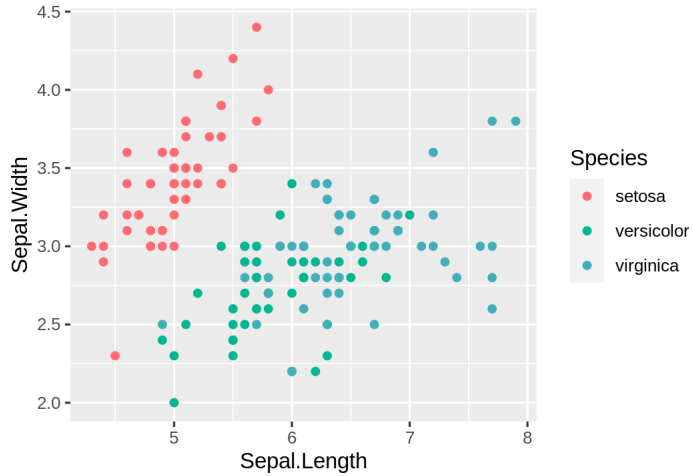
Deutanomaly



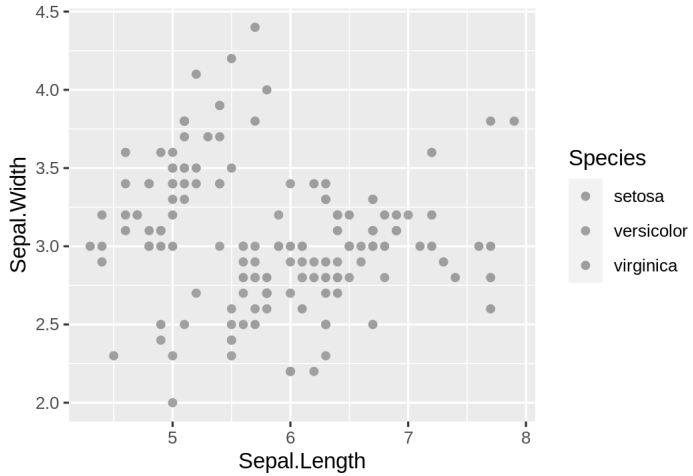
Protanomaly



Tritanomaly

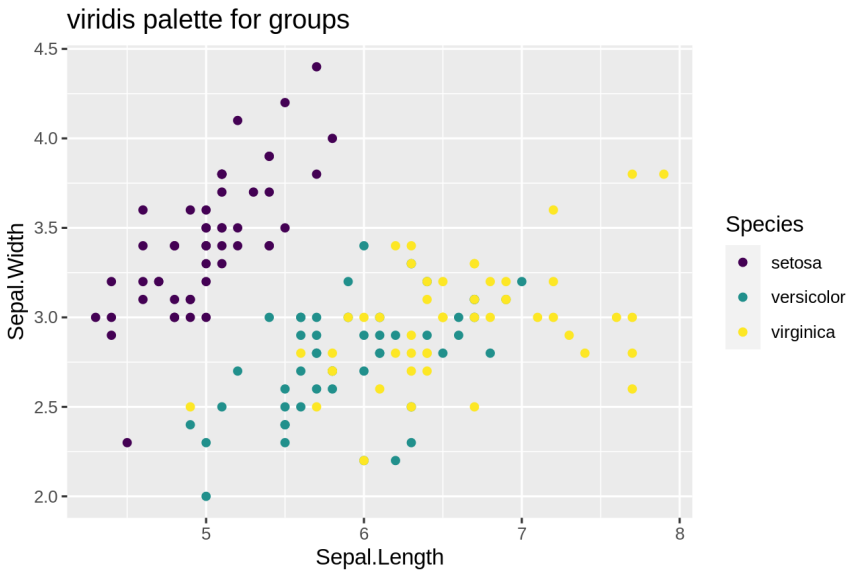


Desaturated



Use colourblind-friendly palettes

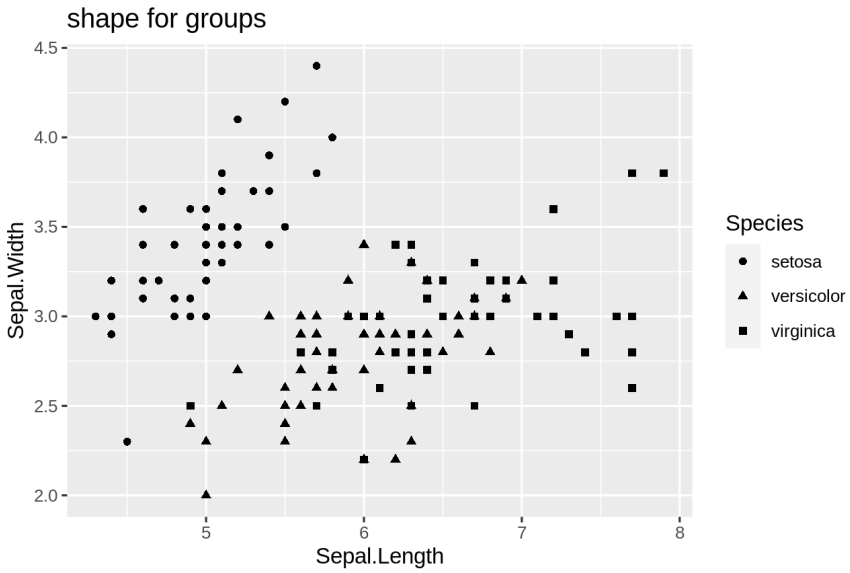
```
# Palette for groups
pp + scale_colour_viridis_d() +
  labs(title = "viridis palette for groups")
```




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

Changing `shape`, `size` and `alpha`

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




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

Challenge #2

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

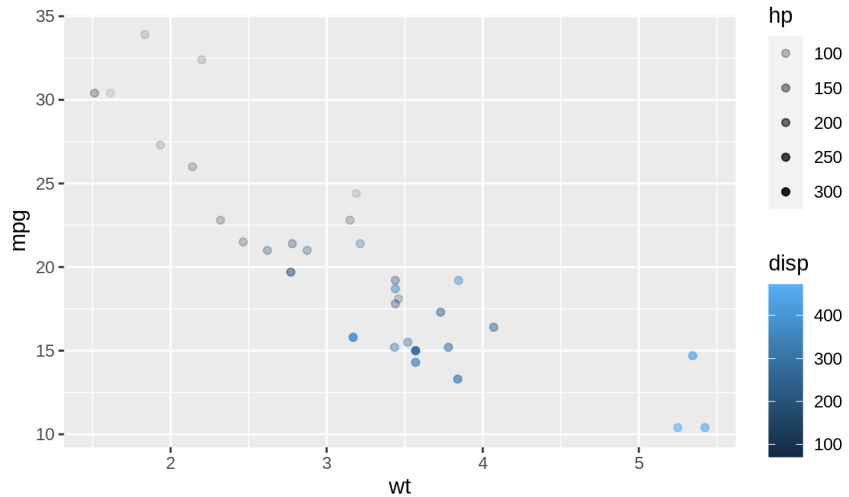
Data	x	y	Aesthetics
mtcars	*wt*	*mpg*	*disp* and *hp*
CO2	*conc*	*uptake*	*Treatment* and *Type*
msleep	log10(*bodywt*)	*awake*	*vore* and *conservation*

⚠ Pay attention to the data types!

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))
```

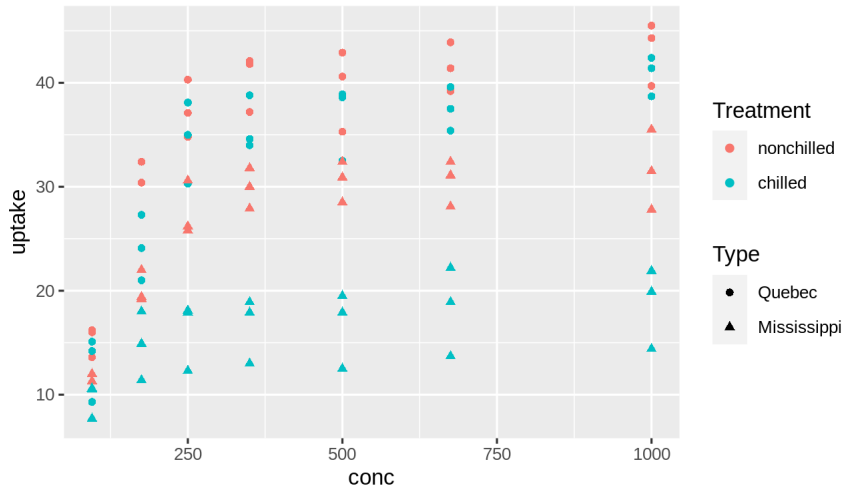


Could you use `size` instead of `alpha`? What about `shape`?

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))
```

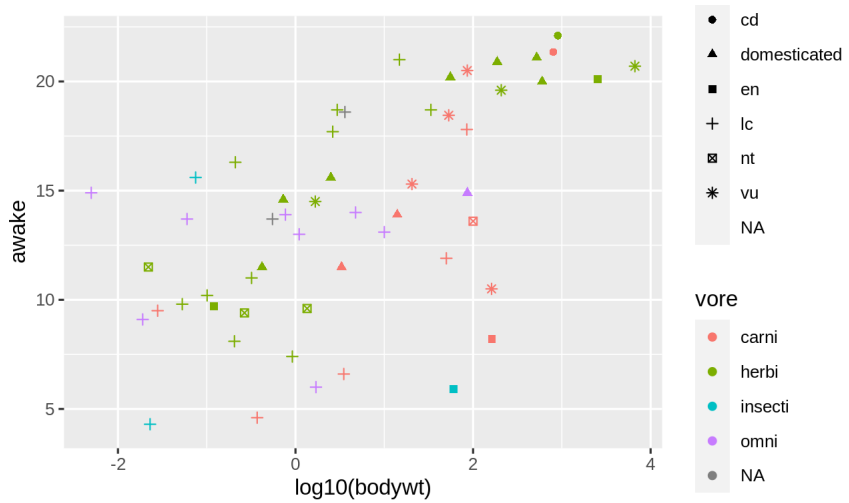


Why not use `size = Type`*?*

Challenge #2 - Solution example #3

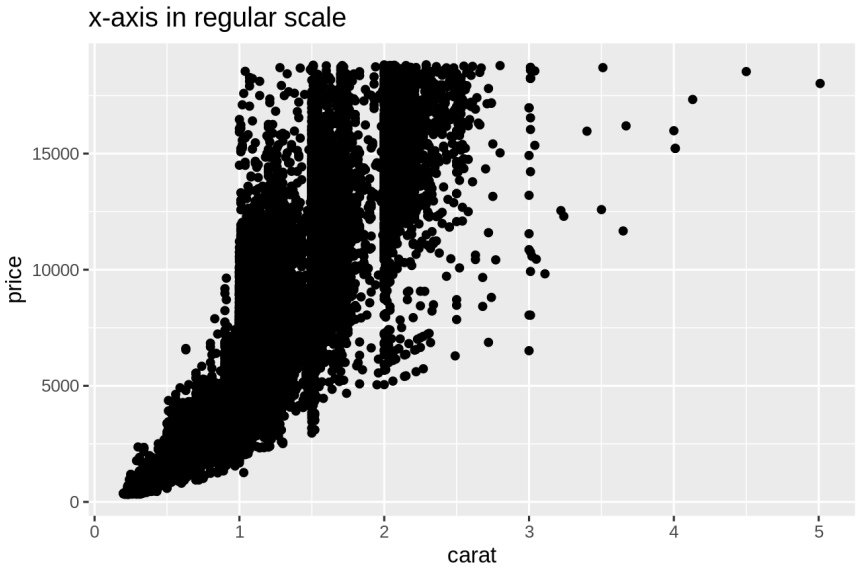
Data	x	y	Aesthetics
msleep	log10(*bodywt*)	*awake*	*vore* and *conservation*

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



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")
```



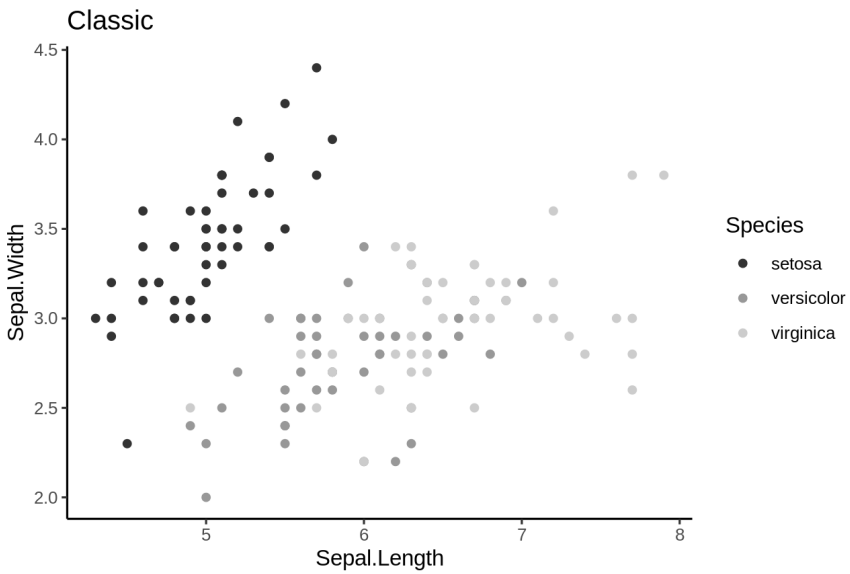

```
# x-axis and y-axis in log10() scale
ggplot(diamonds) + geom_point(mapping = aes(x = carat, y = price)) +
  scale_x_log10() +
  scale_y_log10() +
  labs(title = "x- and y-axes in log10 scale")
```

Fine-tuning your plots

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

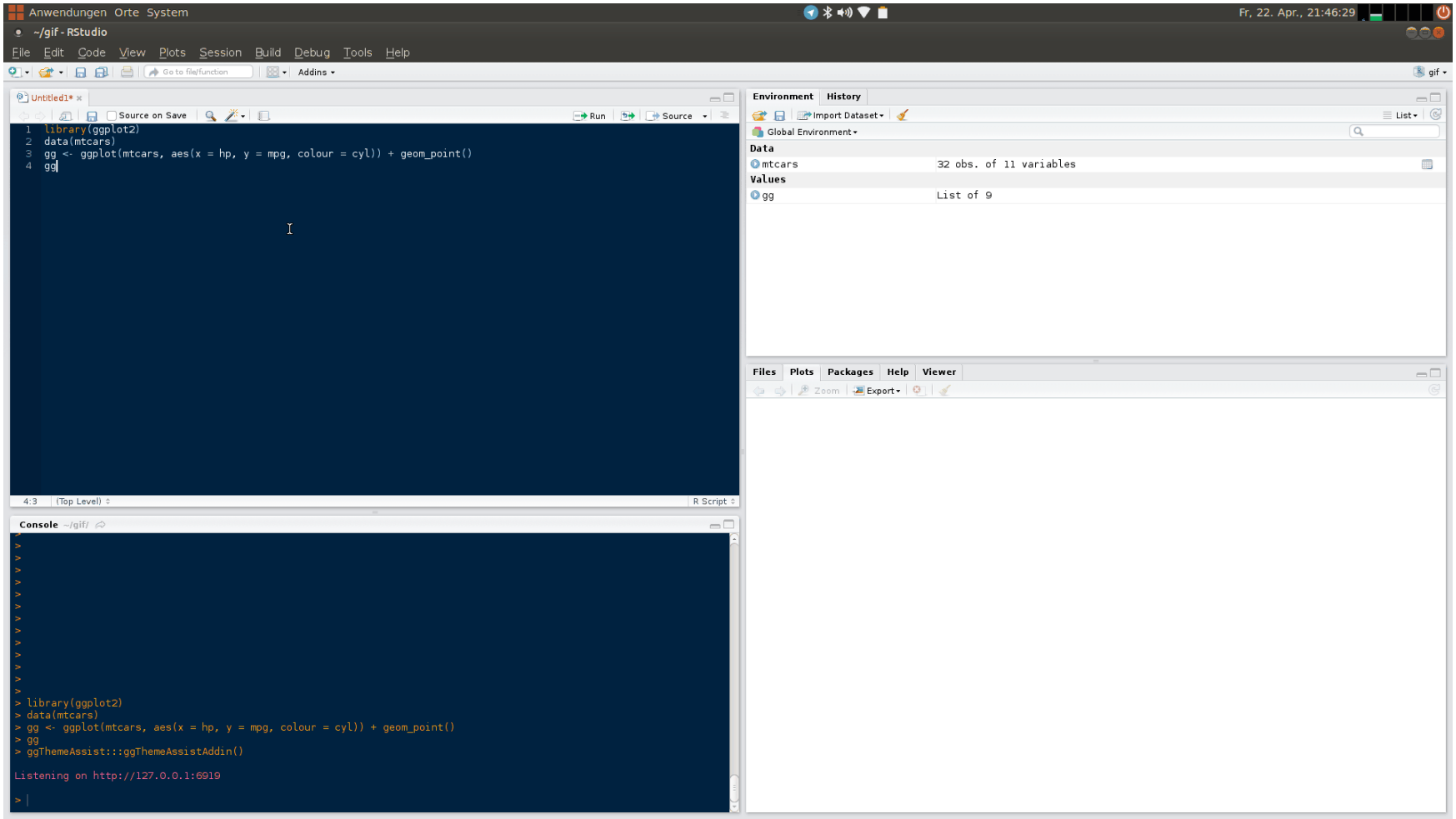
theme()

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




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

ggThemeAssist: RStudio-addin

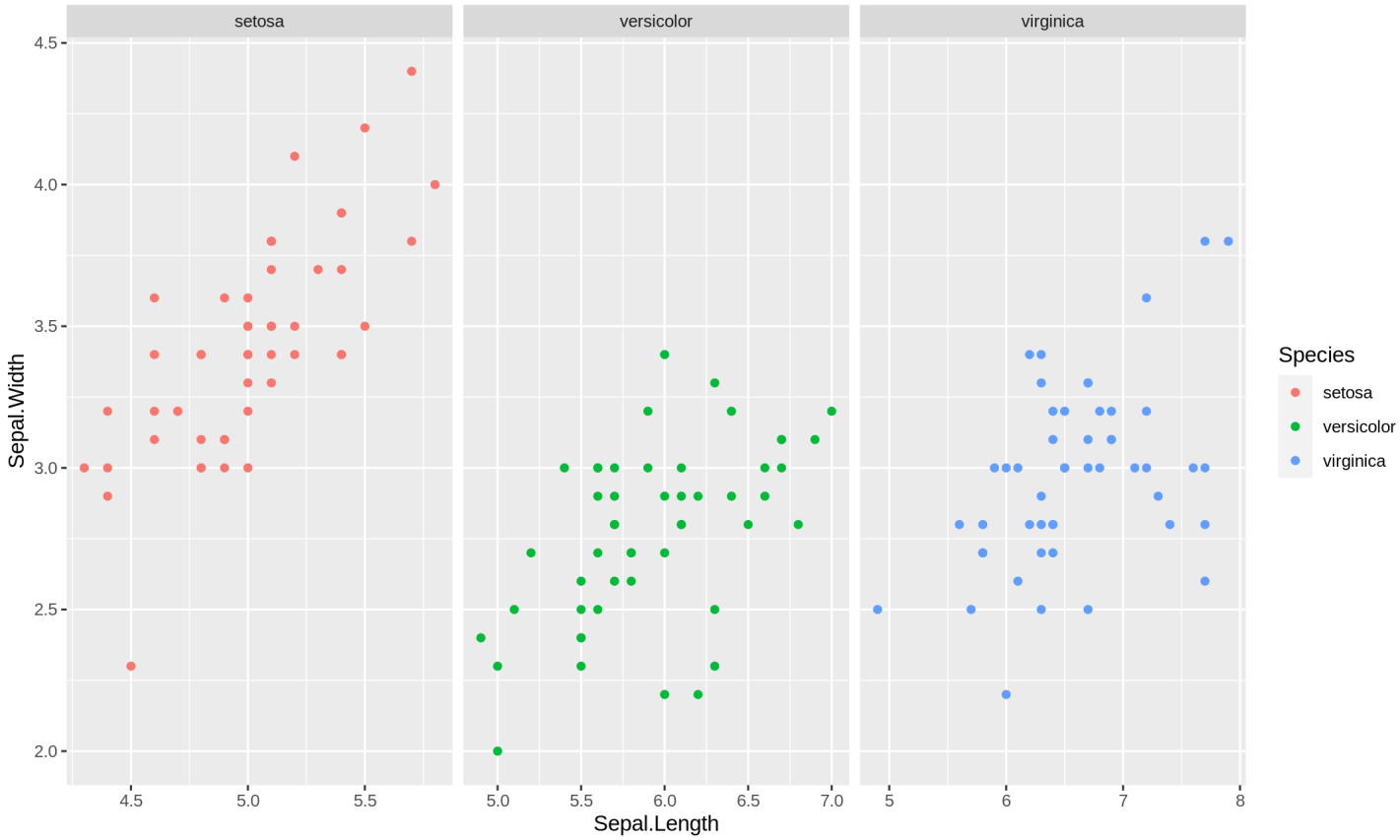


Fine-tuning your plots

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

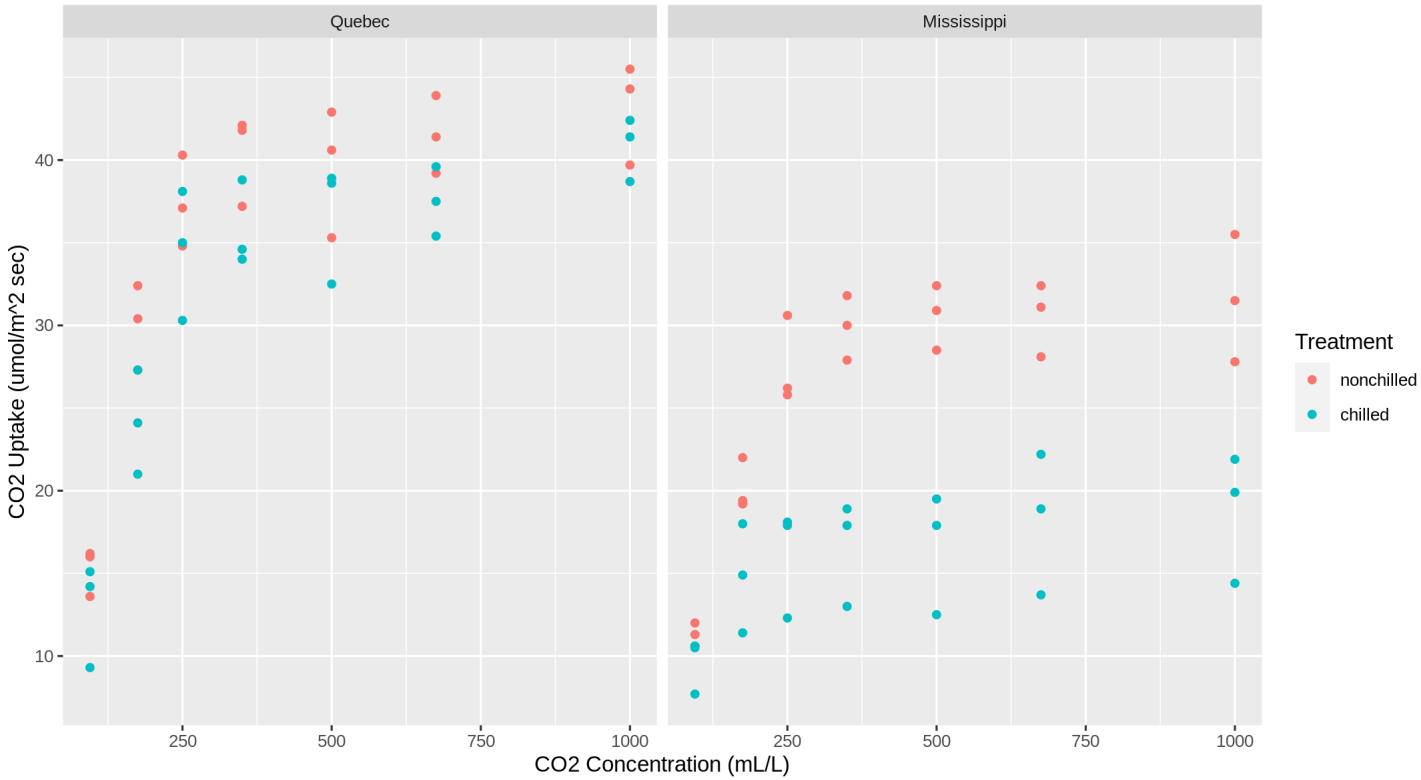
iris dataset: per-species facets

```
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

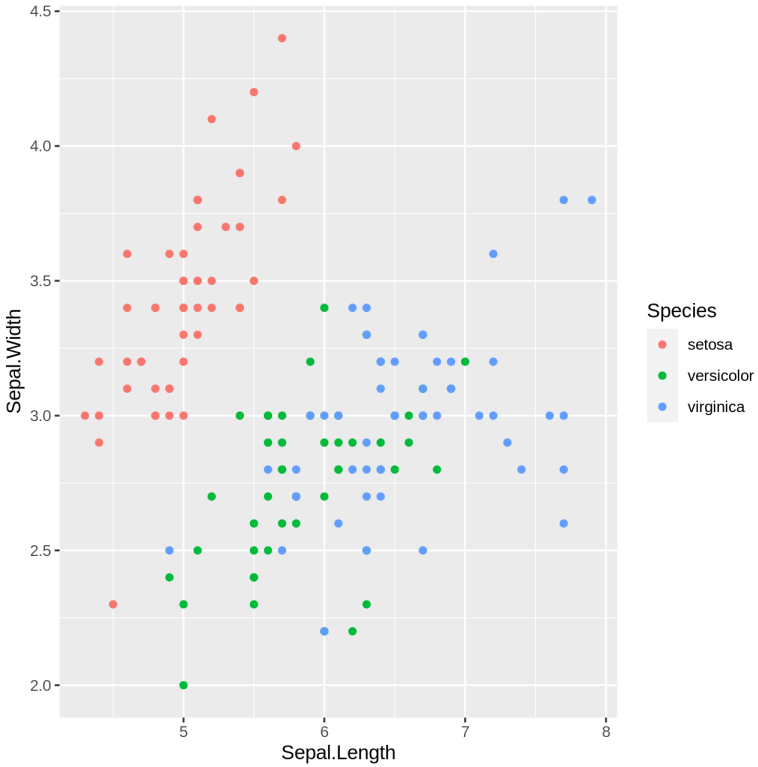
```
ggplot(data = CO2) +
  geom_point(mapping = aes(x = conc, y = uptake, colour = Treatment)) +
  xlab("CO2 Concentration (mL/L)") + ylab("CO2 Uptake (umol/m^2 sec)") +
  facet_grid(~ Type)
```



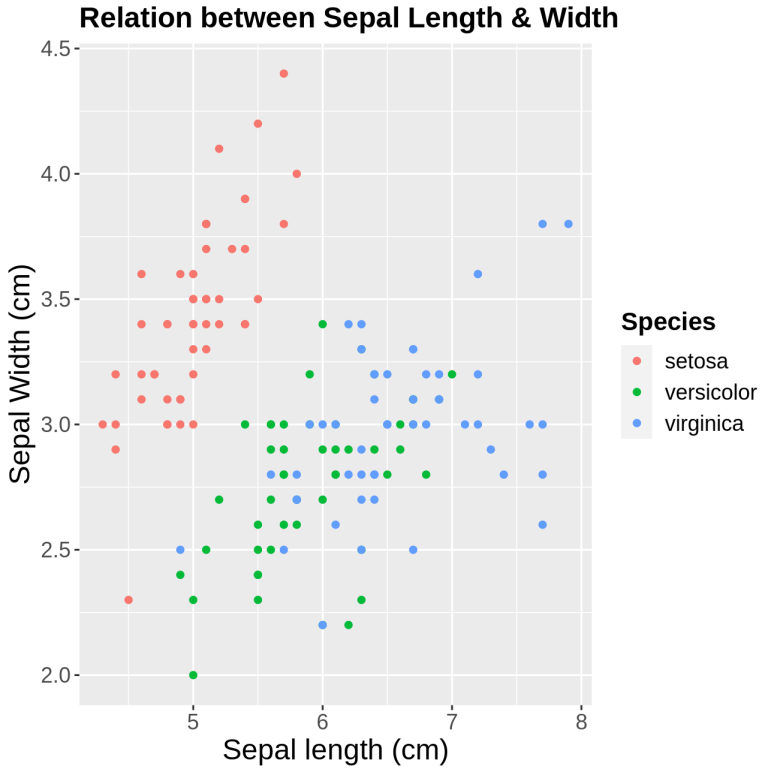
Title and axes components: changing size, colour and face

Title and axes components: size, colour and face

Default



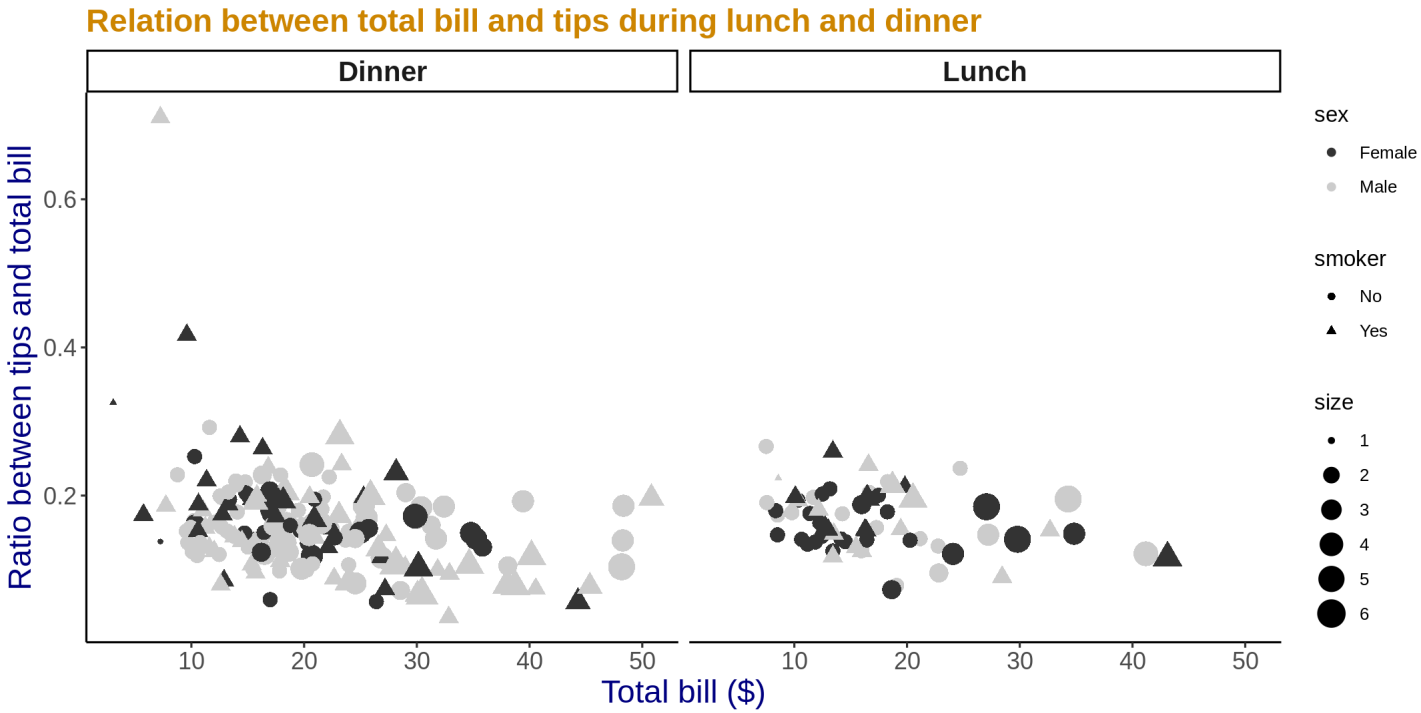
Axes and title tuning



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: 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()`

```
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

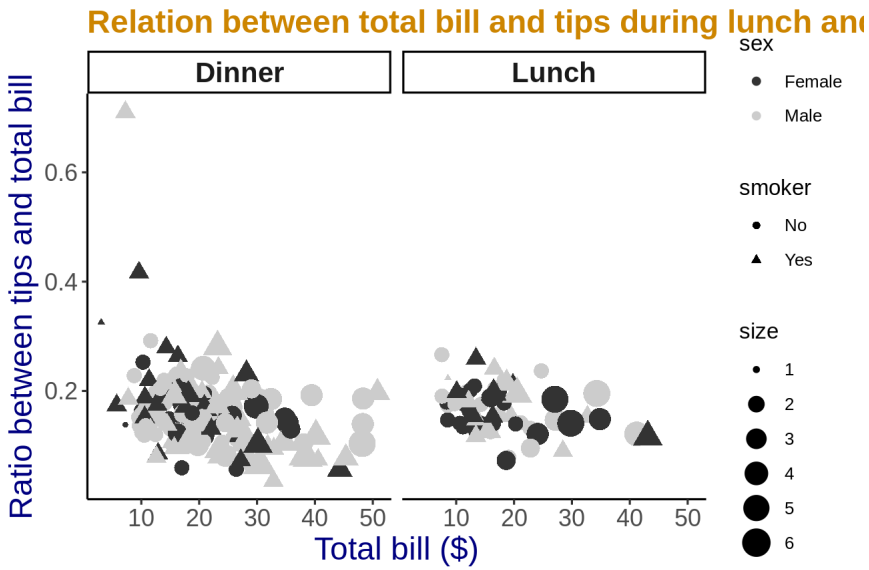
Adding plot title and labels

```
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

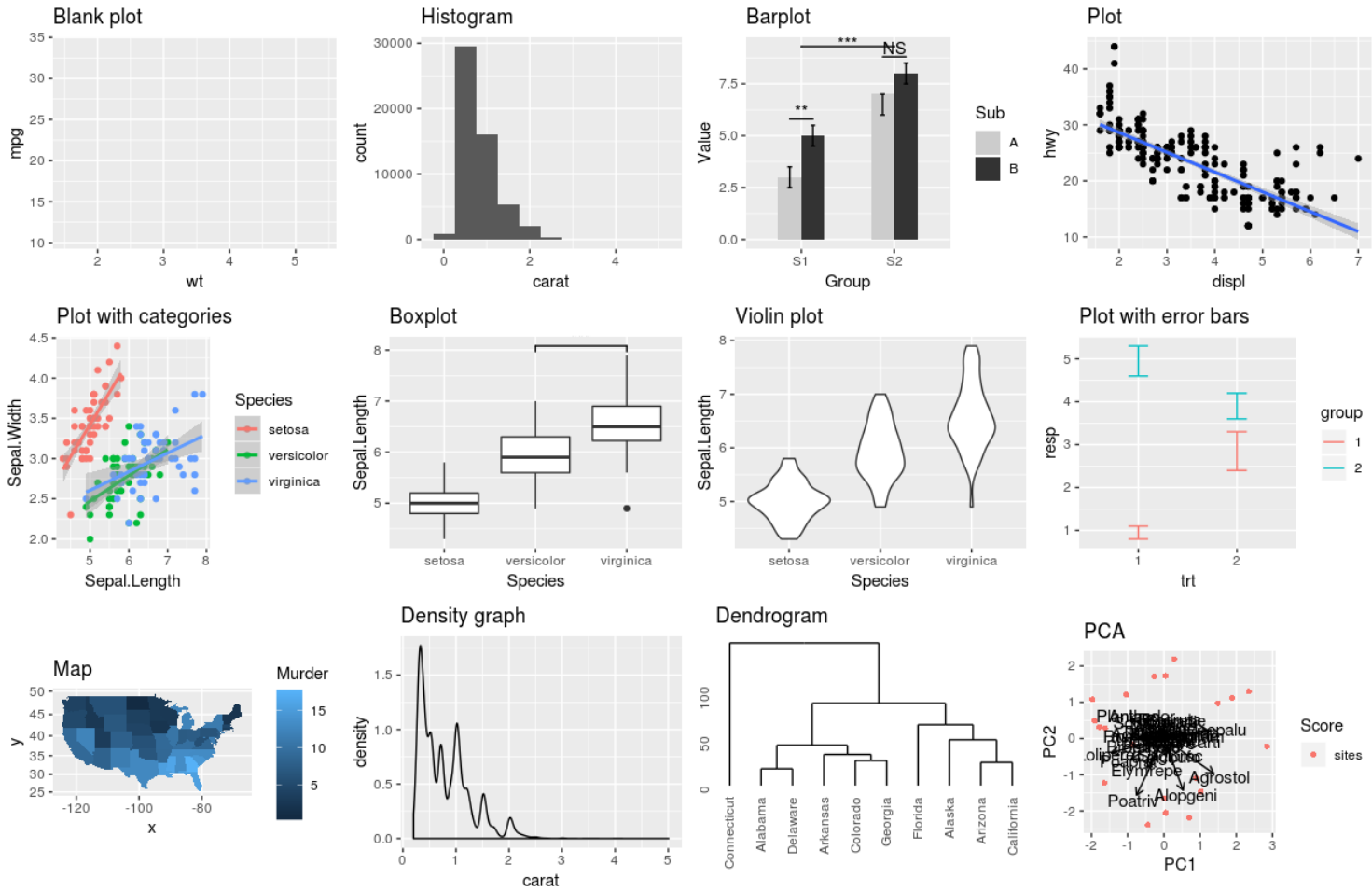
```
tips.gg <- tips.gg +
  theme_classic() +
  theme(axis.title = element_text(size = 16, colour = "navy"),
        axis.text = element_text(size = 12),
        plot.title = element_text(size = 16, colour = "orange3", face = "bold"),
        strip.text.x = element_text(size = 14, face = "bold"))
tips.gg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



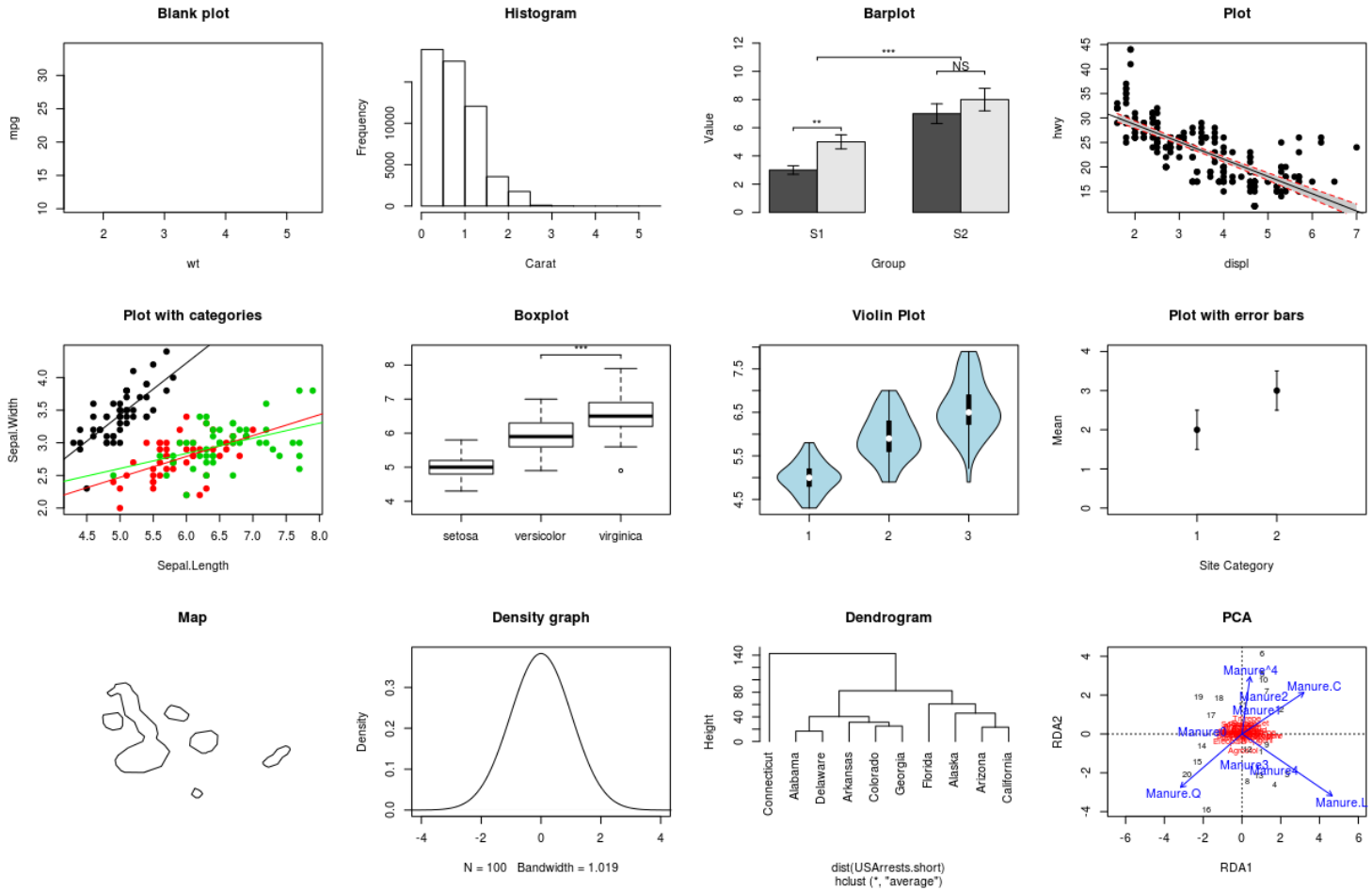
Fine-tuning your plots

Using `geom_*()` to create different plots

ggplot2::geom_*()

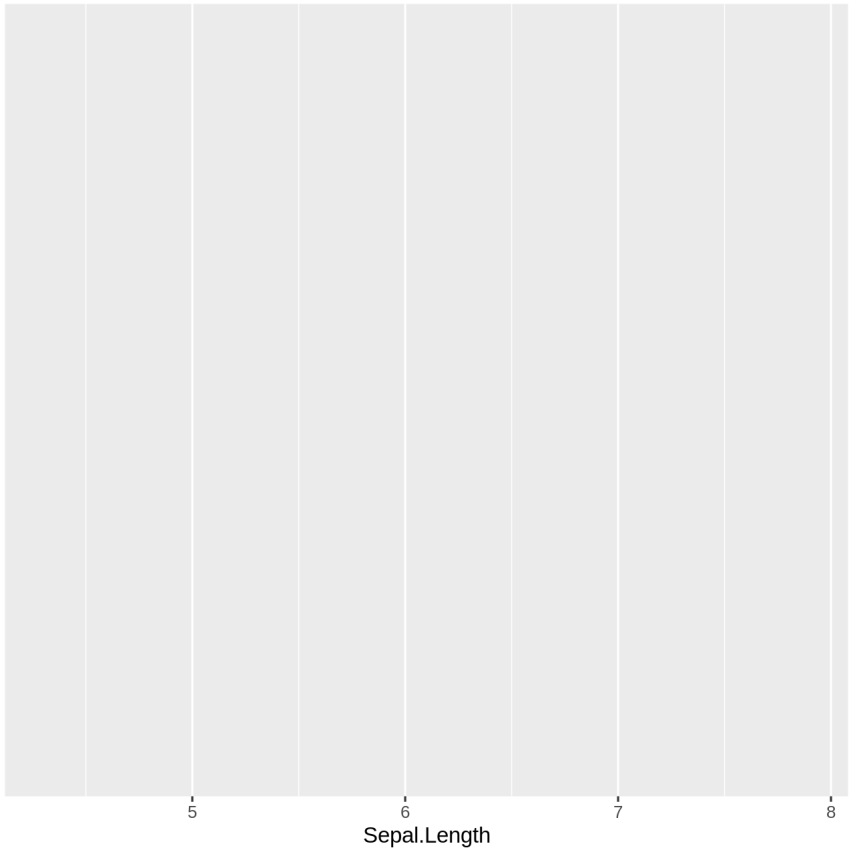


base::plot()


Blank plot
mpg
wt
Histogram
Frequency
Carat
Barplot
Value
Group
S1
S2
NS
Plot
hwy
displ
Plot with categories
Sepal.Width
Sepal.Length
Boxplot
setosa
versicolor
virginica
Violin Plot
Plot with error bars
Mean
Site Category
Map
Density graph
Density
N = 100 Bandwidth = 1.019
Dendrogram
Height
Connecticut
Alabama
Delaware
Arkansas
Colorado
Georgia
Florida
Alaska
Arizona
California
dist(USArrests.short)
hclust (*, "average")
PCA
RDA2
RDA1
Manure.C
Manure.Q
Manure.L

ggplot2::ggplot()

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



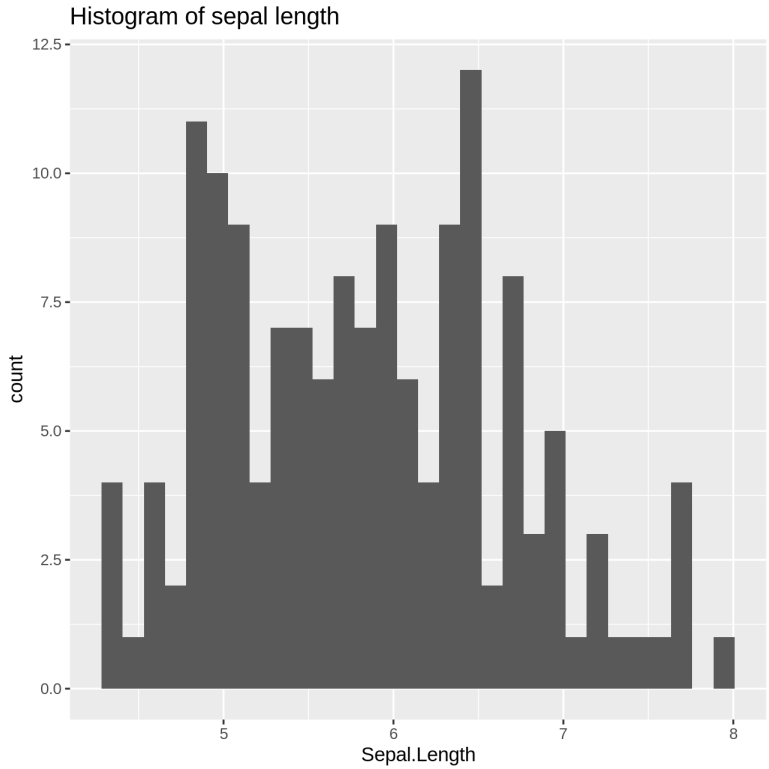
Histograms: `geom_histogram()`

A **histogram** is an accurate graphical representation of the distribution of numeric data

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

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

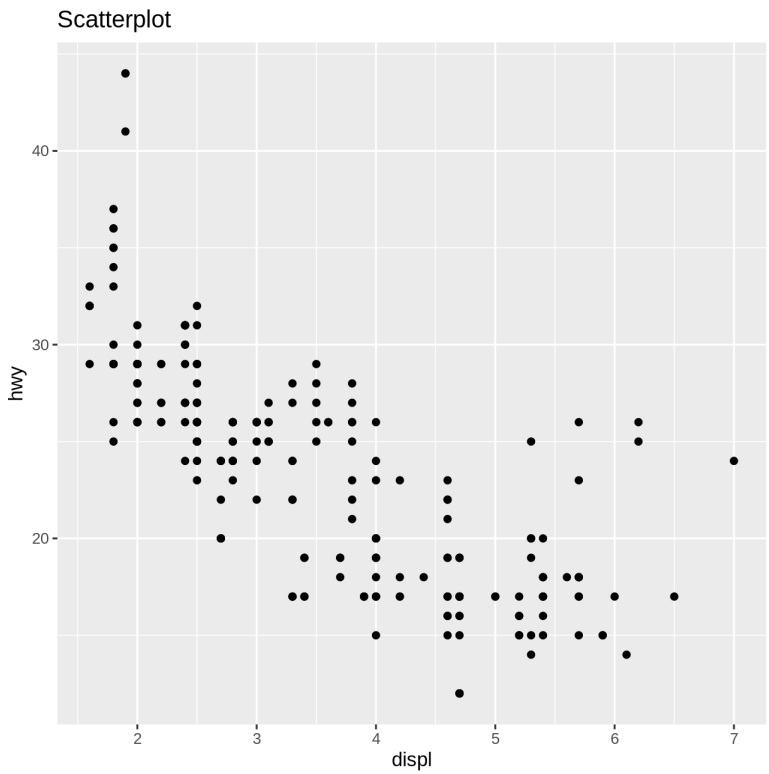
```
ggplot(iris, aes(Sepal.Length)) +
  geom_histogram() +
  ggtitle("Histogram of sepal length")
```



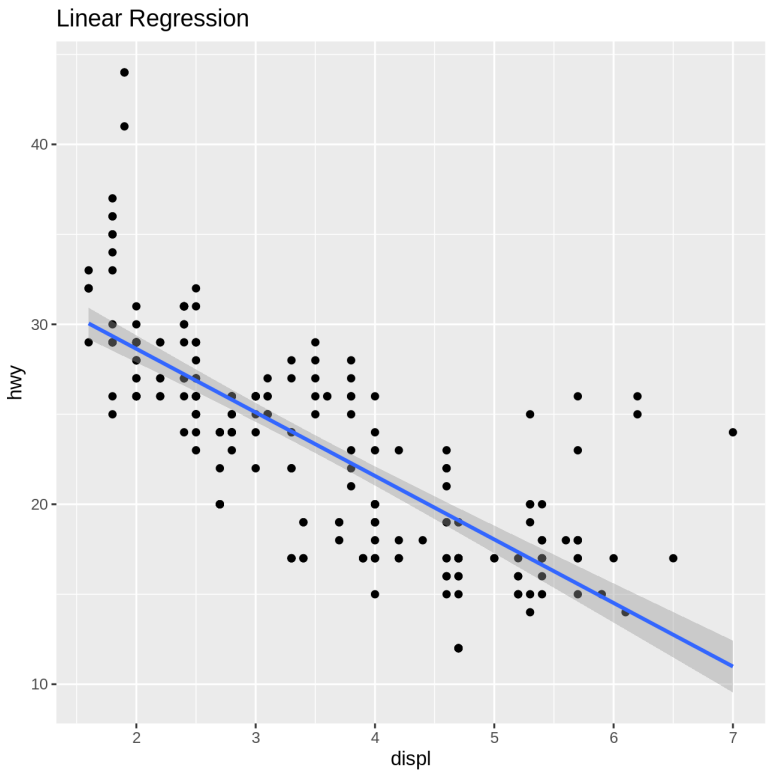

```
hist(iris$Sepal.Length,
     main = "Histogram of sepal length"
```

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

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

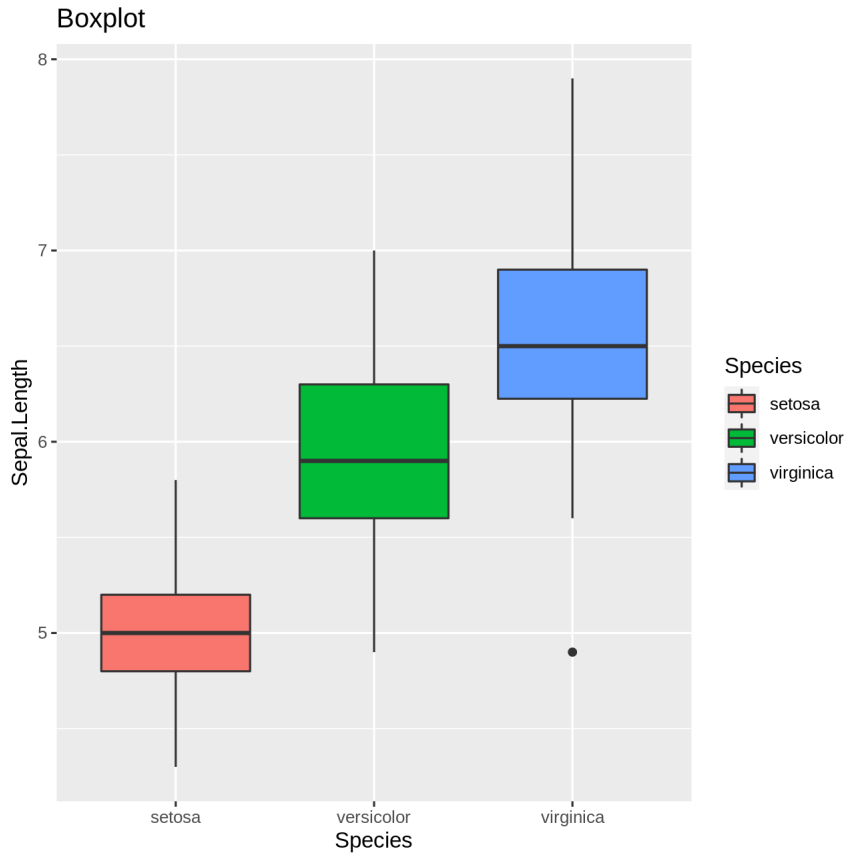



```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  labs(title = "Linear Regression") +
  geom_smooth(method = lm)
```

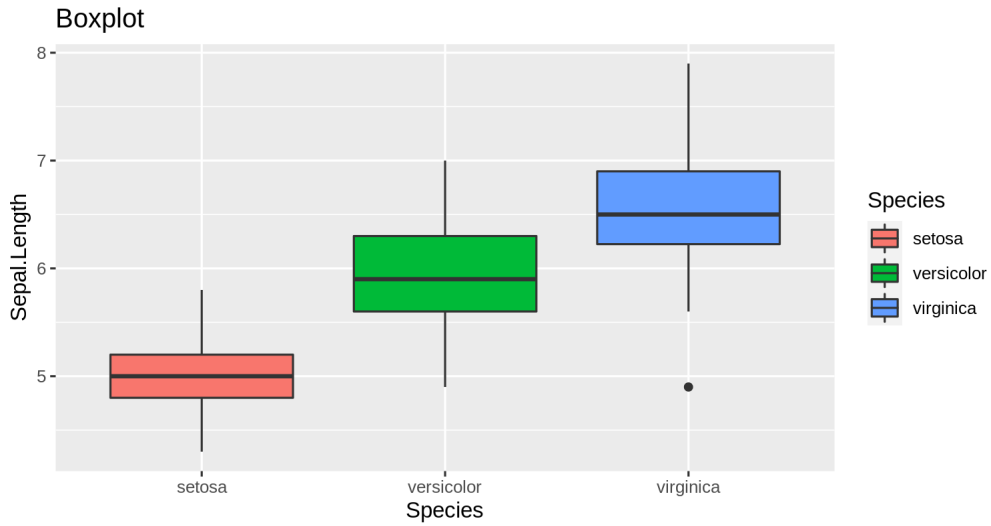


Boxplot: `geom_boxplot()`

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



Boxplot: `geom_boxplot()`



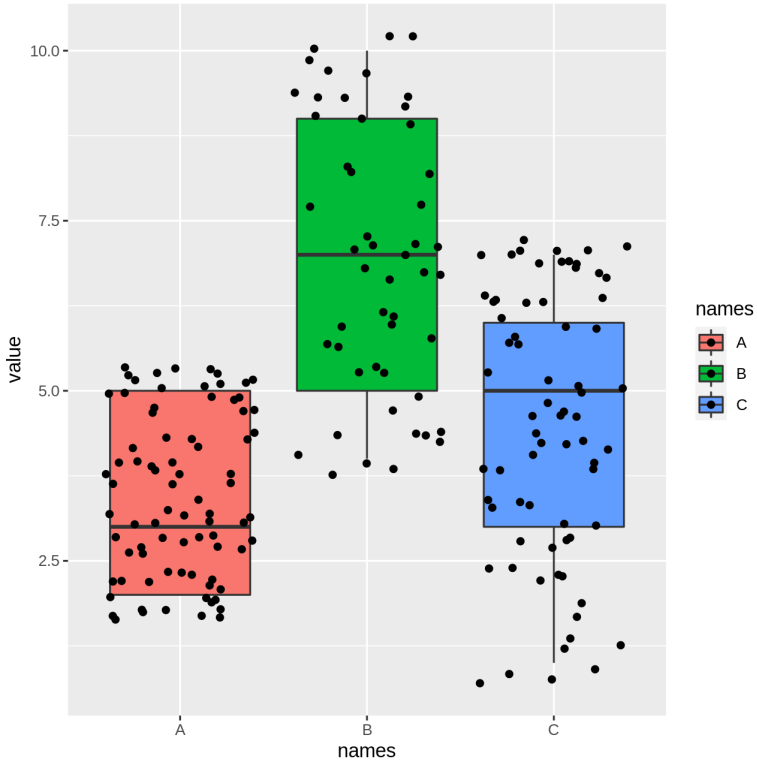
Boxplot with annotations: `geom_boxplot()` and `geom_signif()`

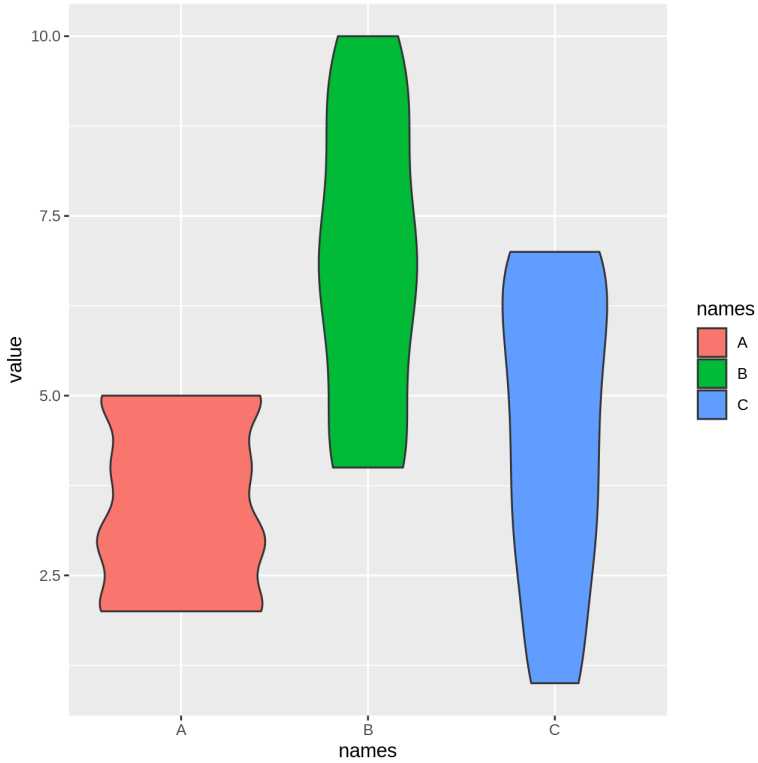
```
library(ggsignif)
ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_boxplot() +
  geom_signif(comparisons = list(c("versicolor", "virginica")),
              map_signif_level=TRUE)
```

Violin plot: `geom_violin()`

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

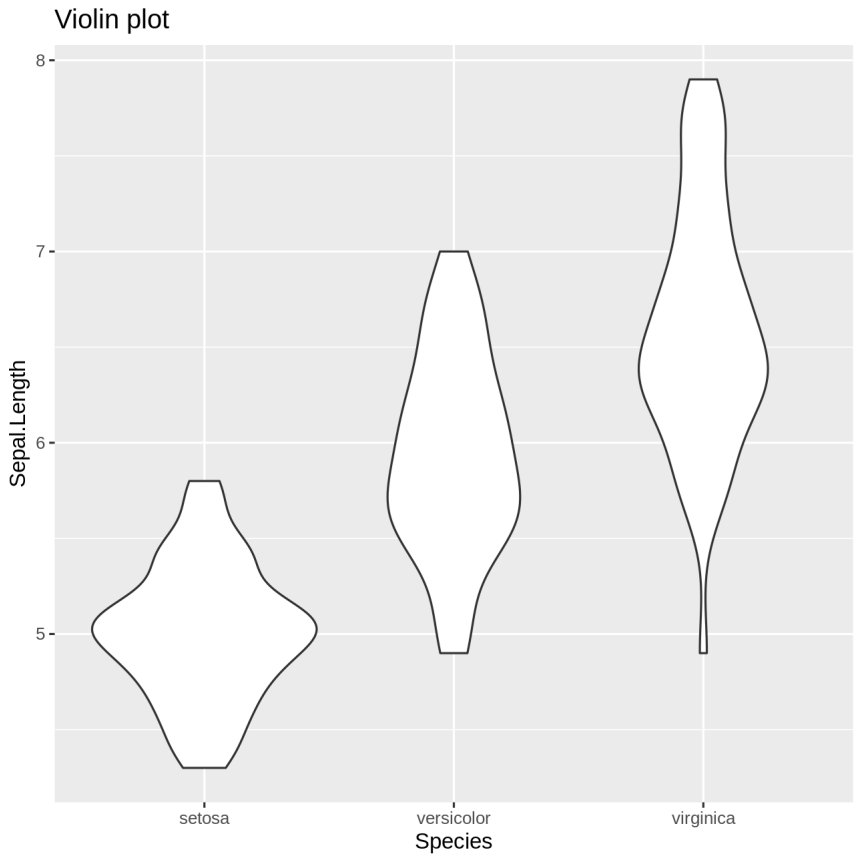
> Violin plot allows to visualize the distribution of a numeric variable for one or several groups. [...] It is really close to a boxplot, but allows a deeper understanding of the distribution.





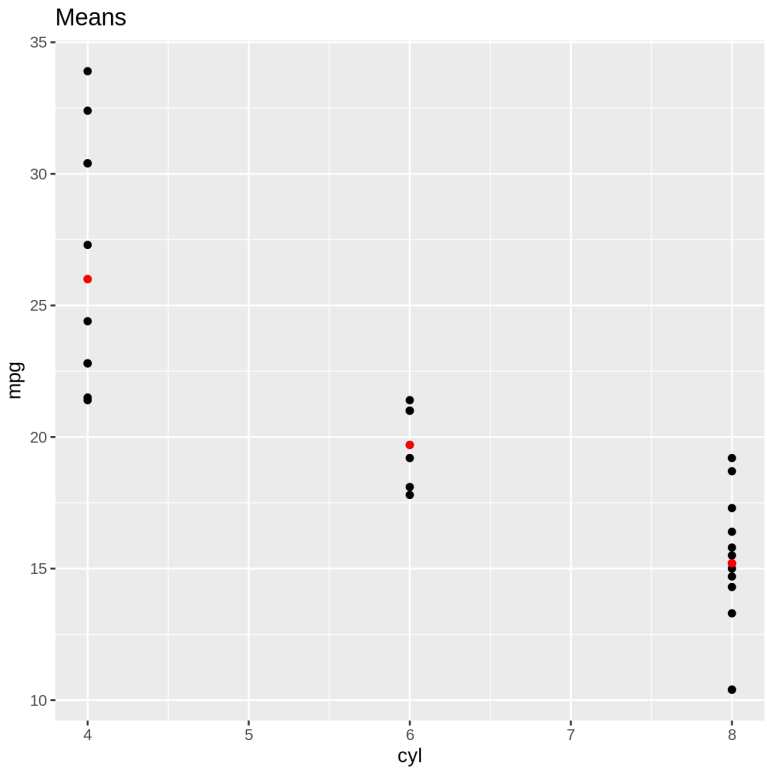
Violin plot: `geom_violin()`

```
ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_violin() +
  labs(title = "Violin plot")
```

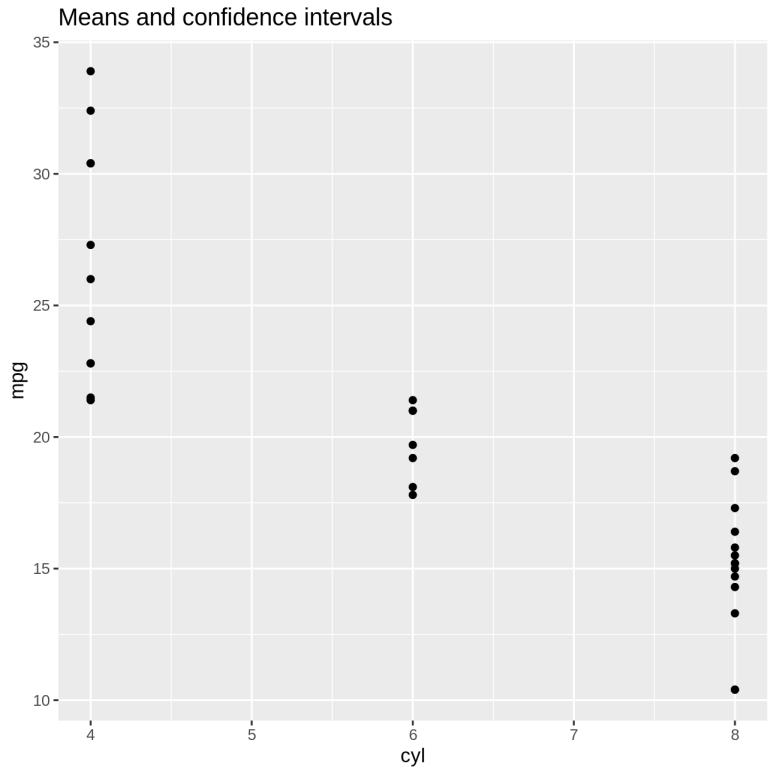


Summarise y values: `stat_summary()`

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




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



Summarise values

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

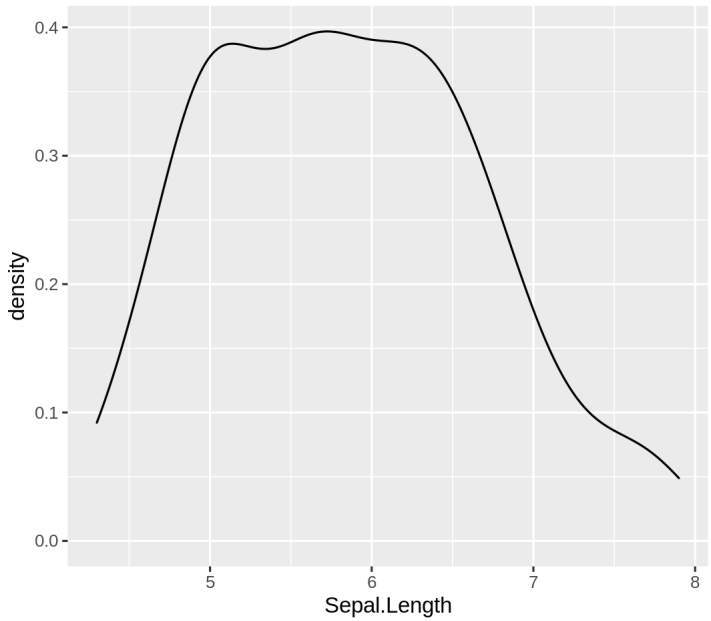
Representing maps: `geom_map()`

```
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()
```

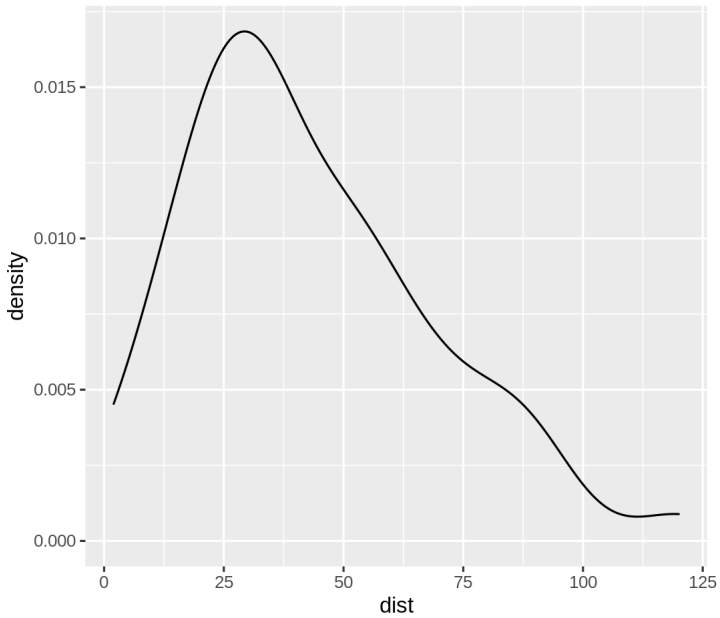
Density plots: `geom_density()`

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.Le
  geom_density()
iris.dens
```

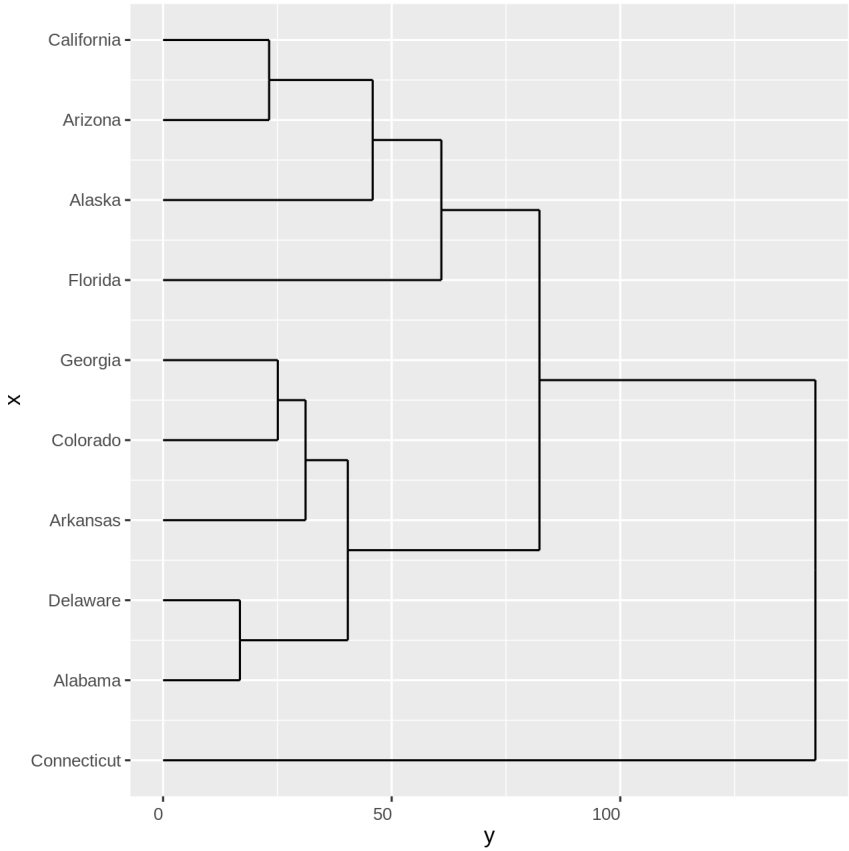



```
cars.dens <- ggplot(cars, aes(dist)) +
  geom_density()
cars.dens
```



Dendogram: `ggdendrogram()`

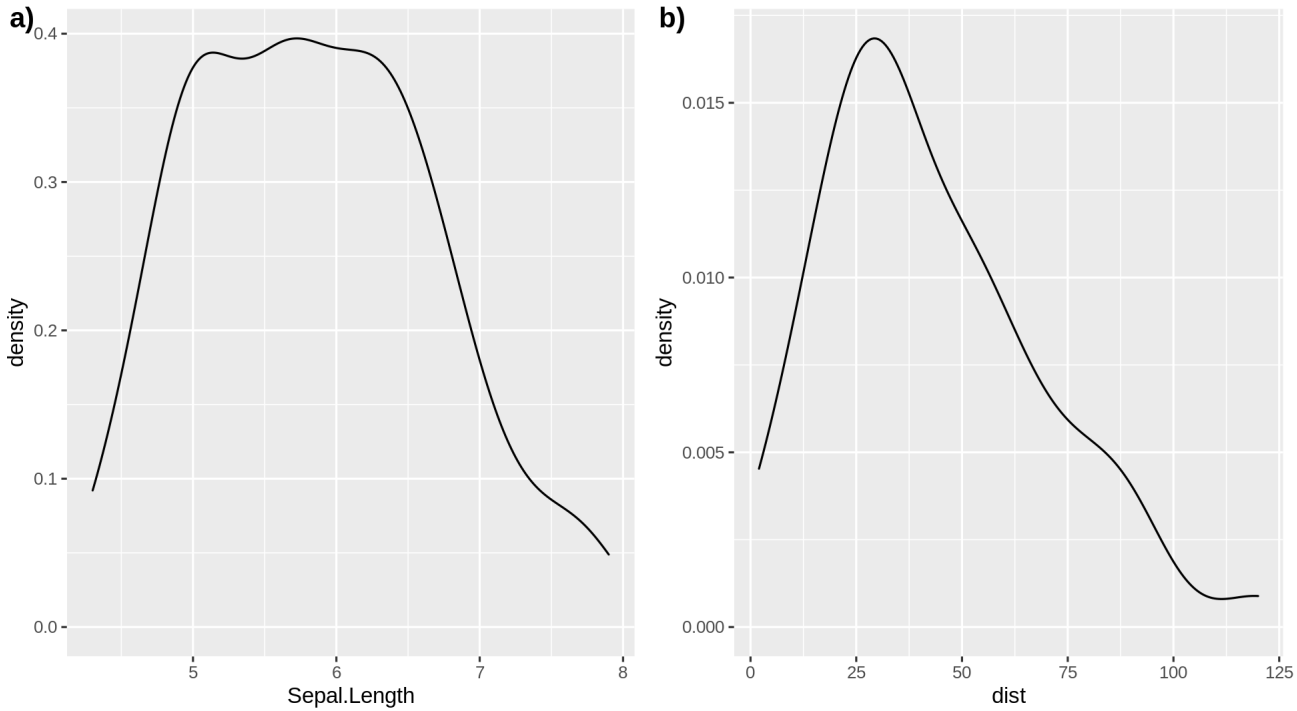
```
library(ggdendro)
USArrests.short = USArrests[1:10,]
hc <- hclust(dist(USArrests.short), "ave")

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



Arrange plots: `ggarrange()`

Add multiple graphs on the same plot

```
library(ggpubr)
ggarrange(iris.dens, cars.dens, labels = c('a)','b)'))
```



Final Challenge

Create your own graph and follow these recommendations:

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

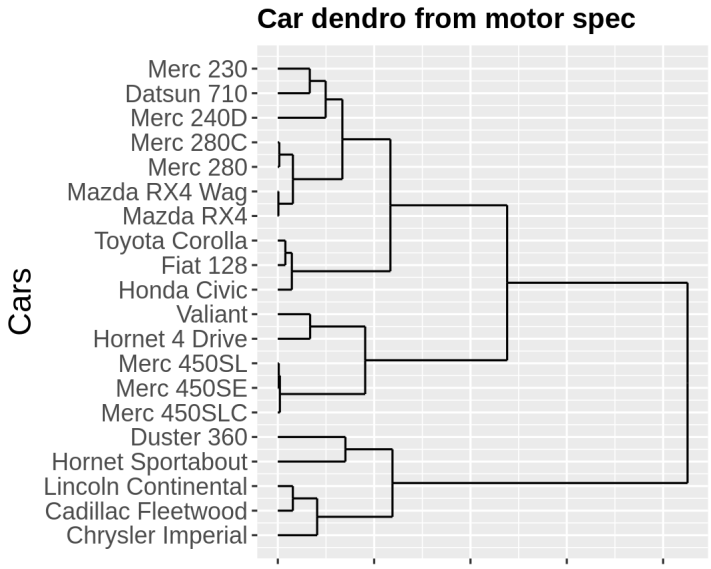
Use the following links for tips:
https://ggplot2.tidyverse.org/reference/index.html
http://shinyapps.stat.ubc.ca/r-graph-catalog/

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

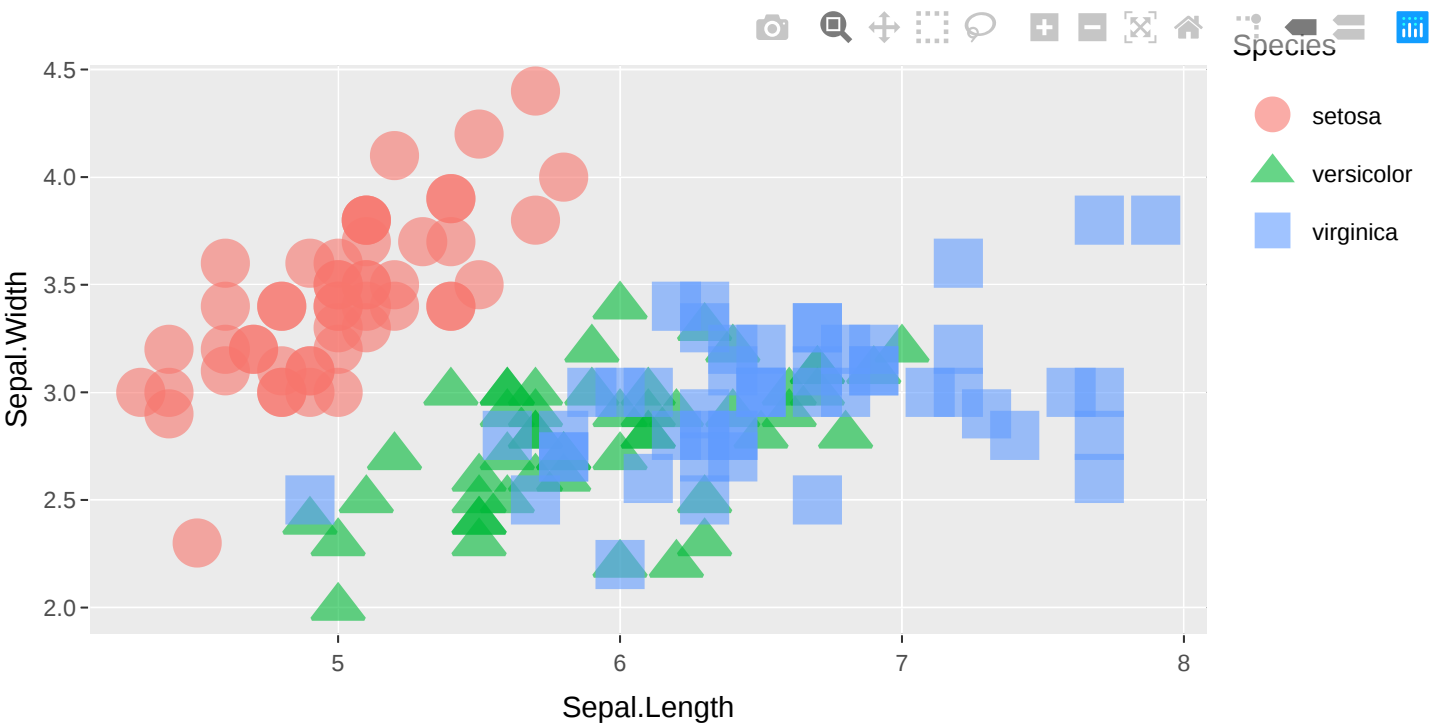
```
data(mtcars)
mtcars.short <- mtcars[1:20,]
mtcars.short.hc <- hclust(dist(mtcars.short), "complete")
dendro.challenge4 <- ggdendrogram(mtcars.short.hc, rotate = TRUE, theme_dendro = FAL
dendro.challenge4 + ggtitle("Car dendro from motor spec") +
  xlab("Cars") +
  theme(axis.title.y = element_text(size = 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()`

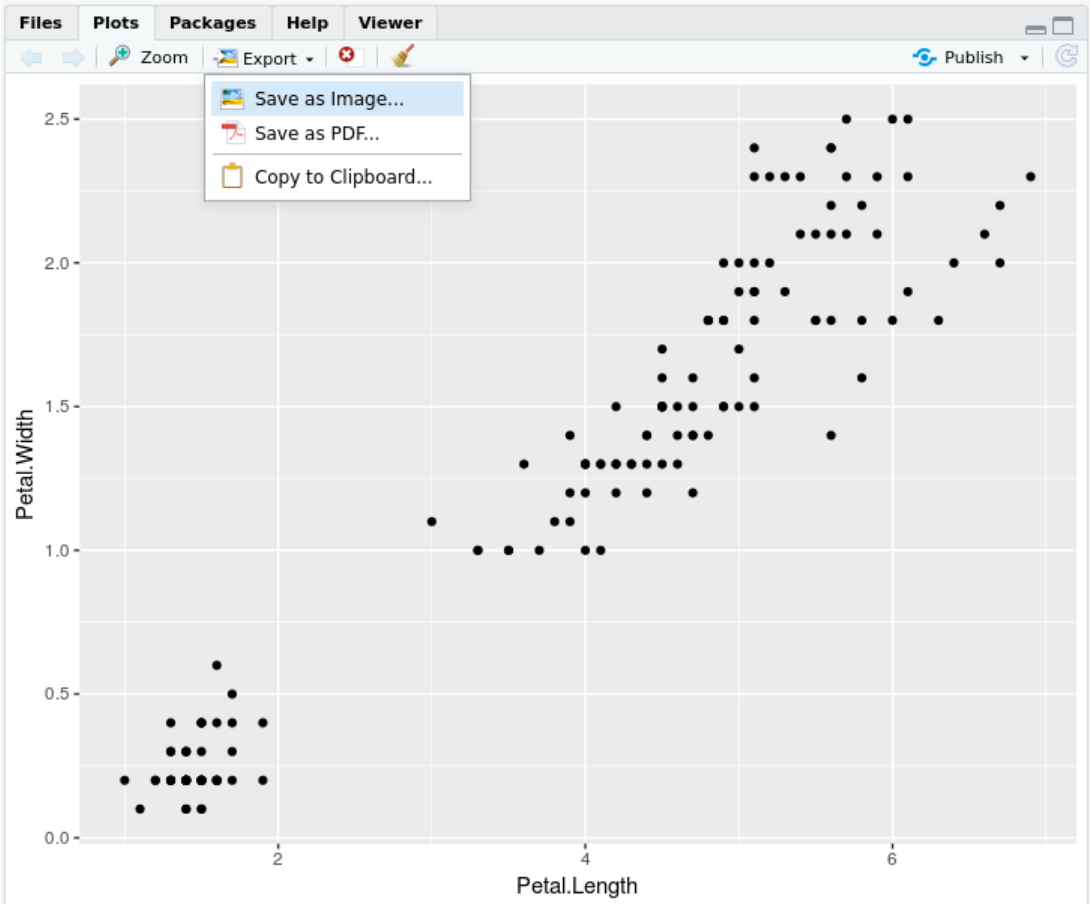
Select the species data to display directly from the legend

```
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("my1stPlot.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

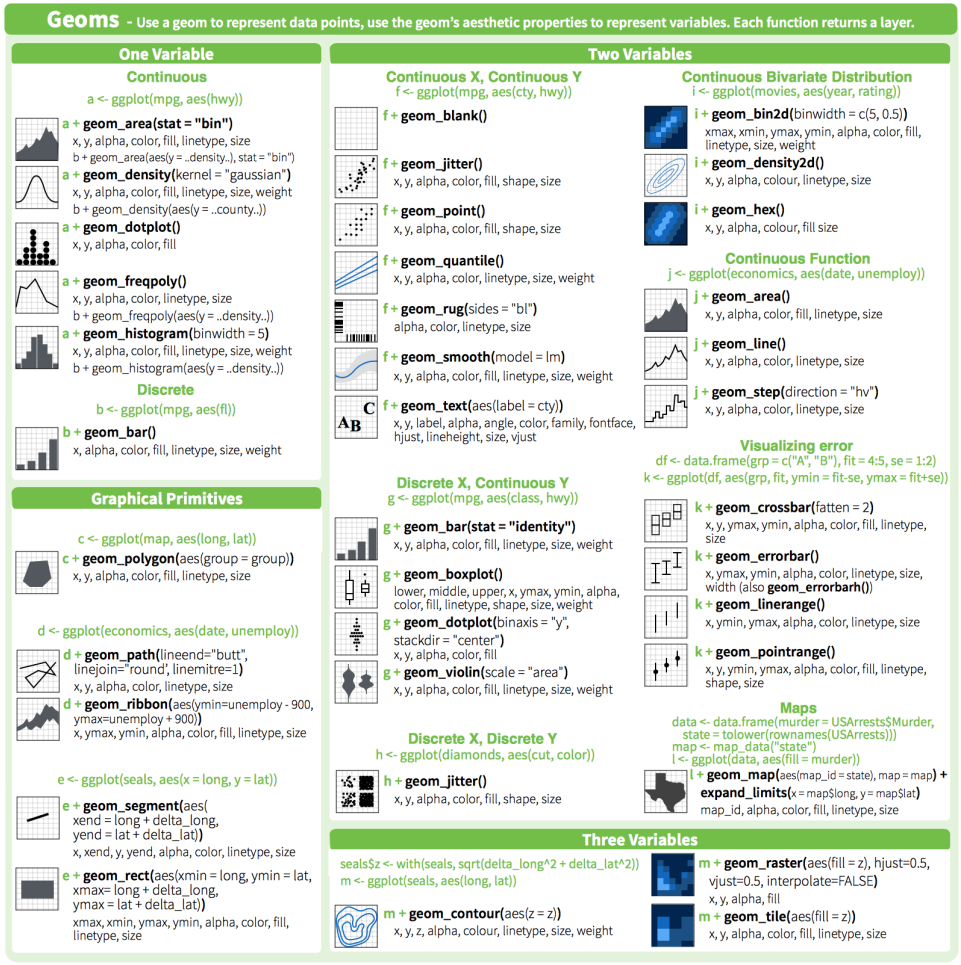
Other methods to save image: see `?pdf` `?jpeg`

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

Going further

Available elements

Data Visualization with ggplot2 Cheat Sheet



Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

- a + **geom_area(stat = "bin")** — x, y, alpha, color, fill, linetype, size; b + geom_area(aes(y = ..density..), stat = "bin")
- a + **geom_density(**kernel = "gaussian"**)** — x, y, alpha, color, fill, linetype, size, weight; b + geom_density(aes(y = ..county..))
- a + **geom_dotplot()** — x, y, alpha, color, fill
- a + **geom_freqpoly()** — x, y, alpha, color, linetype, size; b + geom_freqpoly(aes(y = ..density..))
- a + **geom_histogram(**binwidth = 5**)** — x, y, alpha, color, fill, linetype, size, weight; b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

- b + **geom_bar()** — x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

- c + **geom_polygon(**aes(group = group)**)** — x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

- d + **geom_path(**lineend="butt", linejoin="round', linemitre=1**)** — x, y, alpha, color, linetype, size
- d + **geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** — x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

- e + **geom_segment(**aes(xend = long + delta_long, yend = lat + delta_lat)**)** — x, xend, y, yend, alpha, color, linetype, size
- e + **geom_rect(**aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)** — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

- f + **geom_blank()**
- f + **geom_jitter()** — x, y, alpha, color, fill, shape, size
- f + **geom_point()** — x, y, alpha, color, fill, shape, size
- f + **geom_quantile()** — x, y, alpha, color, linetype, size, weight
- f + **geom_rug(**sides = "bl"**)** — alpha, color, linetype, size
- f + **geom_smooth(**model = lm**)** — x, y, alpha, color, fill, linetype, size, weight
- f + **geom_text(**aes(label = cty)**)** — x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

- g + **geom_bar(**stat = "identity"**)** — x, y, alpha, color, fill, linetype, size, weight
- g + **geom_boxplot()** — lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight
- g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)** — x, y, alpha, color, fill
- g + **geom_violin(**scale = "area"**)** — x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

- h + **geom_jitter()** — x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

- i + **geom_bin2d(**binwidth = c(5, 0.5)**)** — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight
- i + **geom_density2d()** — x, y, alpha, colour, linetype, size
- i + **geom_hex()** — x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

- j + **geom_area()** — x, y, alpha, color, fill, linetype, size
- j + **geom_line()** — x, y, alpha, color, linetype, size
- j + **geom_step(**direction = "hv"**)** — x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

- k + **geom_crossbar(**fatten = 2**)** — x, y, ymax, ymin, alpha, color, fill, linetype, size
- k + **geom_errorbar()** — x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)
- k + **geom_linerange()** — x, ymin, ymax, alpha, color, linetype, size
- k + **geom_pointrange()** — x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

- l + **geom_map(**aes(map_id = state), map = map**)** + **expand_limits(**x = map$long, y = map$lat**)** — map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

- m + **geom_contour(**aes(z = z)**)** — x, y, z, alpha, colour, linetype, size, weight
- m + **geom_raster(**aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)** — x, y, alpha, fill
- m + **geom_tile(**aes(fill = z)**)** — x, y, alpha, color, fill, linetype, size

https://www.rstudio.com/resources/cheatsheets

Additional resources

- `help(package = ggplot2)`
- http://ggplot2.tidyverse.org/reference/
- Fundamentals of Data Visualization https://serialmentor.com/dataviz/

Concluding remarks

1. There are other useful packages that can be used with `ggplot2`! To name a few: `ggbio`, `ggpmisc`, `geomnet`, `gganimate`, `ggnetwork` and `ggtree`. Have a look at www.ggplot2-exts.org for other `ggplot2` extensions.
1. Note that you can learn more about design and image manipulation with the "Introduction to graphic design and image manipulation with open source tools": https://qcbs.ca/wiki/graphics

Thank you for attending!