Plotting in **R**

Alexandre Courtiol & Liam D. Bailey

Leibniz Institute of Zoo and Wildlife Research

June 2018

Plotting in **R**

Why plot in **R**?

- Powerful (large range of plot types)
- Fully customizable (make your own style)
- Practical (integrate your plots and your code together)

Graphics paradigms in **R**

They are three dominant graphics paradigms in **R**:

- traditional graphics (based on `graphics`)
- `lattice` (based on `grid`)
- `ggplot2` (based on `grid`)

Graphics paradigms in **R**

They are three dominant graphics paradigms in **R**:

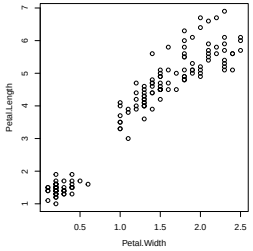
- traditional graphics (based on `graphics`)
- `lattice` (based on `grid`)
- `ggplot2` (based on `grid`)

Note:

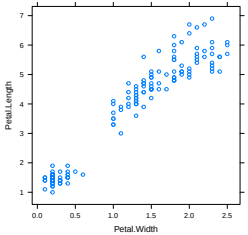
- `graphics` and `grid` are part of any basic installation of **R**
- `lattice` is part of the so-called list of CRAN recommended packages
- `ggplot2` is part of the tidyverse universe (from Rstudio)
- we will focus on traditional graphics and `ggplot2`, but `lattice` is excellent too!
- some other packages are sometimes usefull too (e.g. `rgl`, `plotly`)

An example (using default settings)

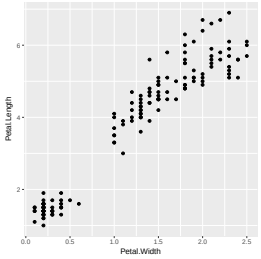
```
plot(Petal.Length ~ Petal.Width,
     data = iris)
```




```
library(lattice)
xyplot(Petal.Length ~ Petal.Width,
       data = iris)
```




```
library(ggplot2)
ggplot(data = iris,
  aes(x = Petal.Width, y = Petal.Length)) +
  geom_point()
```



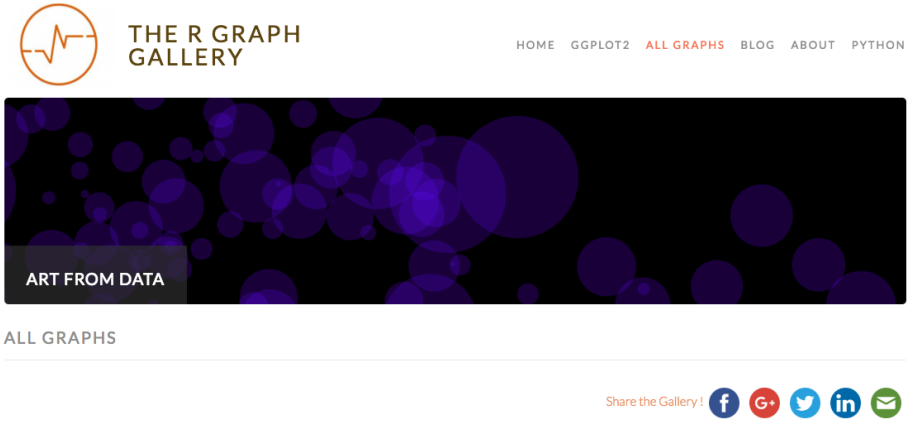
How to learn on your own?

1. Check the examples readily available in **R**, e.g.

```
demo(graphics)
demo(image)
demo(persp)
demo(colors)
demo(plotmath)
demo(Hershey)

example(plot)
example(boxplot)
example(hist)
example(bartplot)

browseVignettes(package = "ggplot2")
```

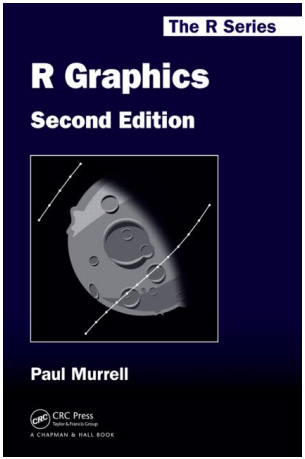
How to learn on your own?

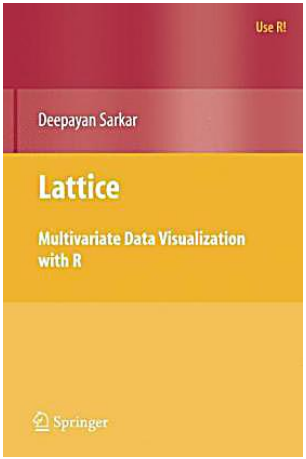
2. Scroll the web:
(e.g. `http://www.r-graph-gallery.com/all-graphs/`)

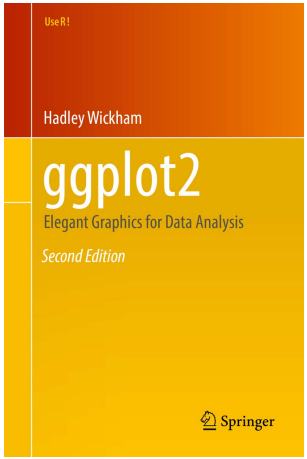


How to learn on your own?

3. Read books:







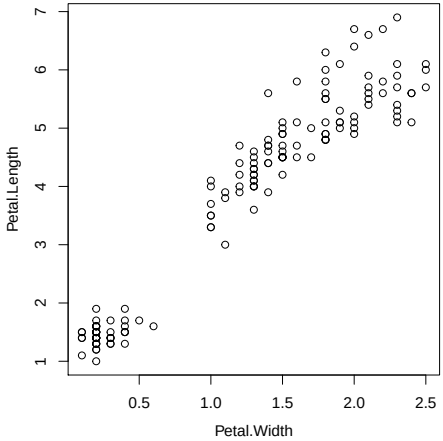
Plotting in **R**

Plotting in **R**

Traditional graphics: Scatter plots

In traditional graphics, use `plot()` to draw a scatter plot:

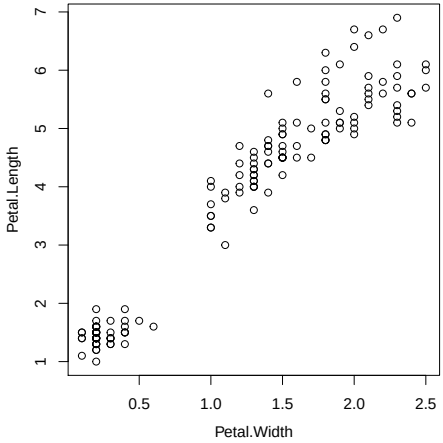
```
plot(Petal.Length ~ Petal.Width, data = iris)
```



Traditional graphics: Scatter plots

You can choose what type of scatter plot to display with argument type:

```
plot(Petal.Length ~ Petal.Width, data = iris, type = "p")
```

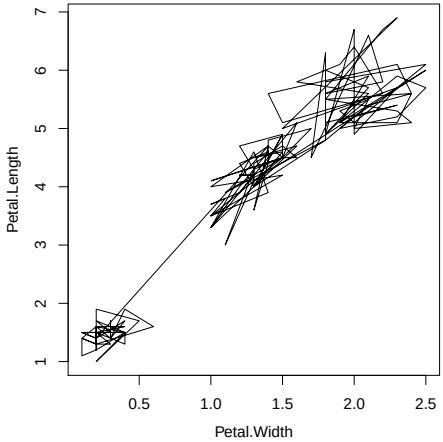


Note: see `"?plot.default"`

Traditional graphics: Scatter plots

You can choose what type of scatter plot to display with argument type:

```
plot(Petal.Length ~ Petal.Width, data = iris, type = "l")
```

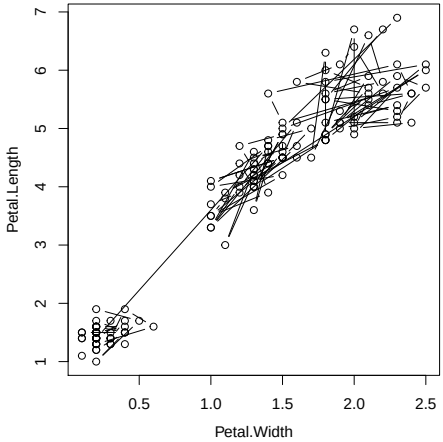


Note: it makes more sense when data are ordered...

Traditional graphics: Scatter plots

You can choose what type of scatter plot to display with argument type:

```
plot(Petal.Length ~ Petal.Width, data = iris, type = "b")
```

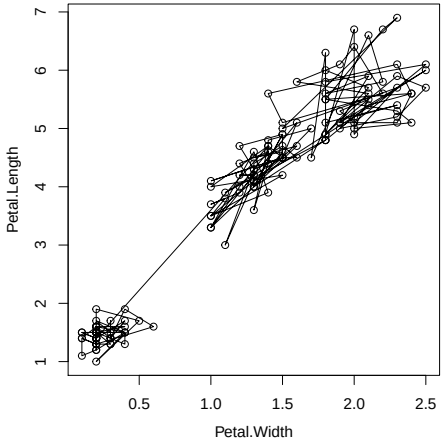


Note: it makes more sense when data are ordered...

Traditional graphics: Scatter plots

You can choose what type of scatter plot to display with argument type:

```
plot(Petal.Length ~ Petal.Width, data = iris, type = "o")
```

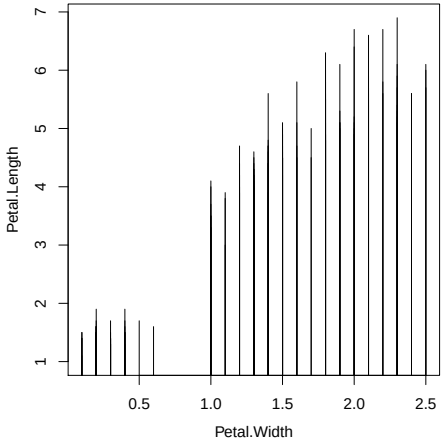


Note: it makes more sense when data are ordered...

Traditional graphics: Scatter plots

You can choose what type of scatter plot to display with argument type:

```
plot(Petal.Length ~ Petal.Width, data = iris, type = "h")
```

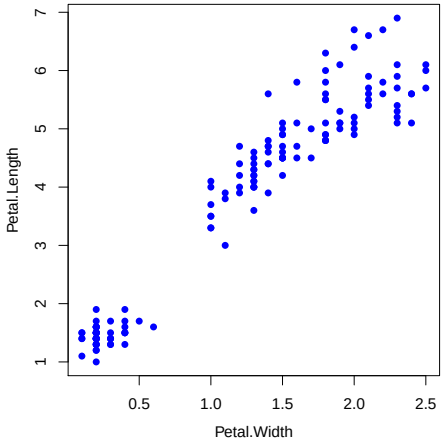


Note: it makes more sense when x-values are unique...

Traditional graphics: Scatter plots

You can change point shapes (pch) and colour (col):

```
plot(Petal.Length ~ Petal.Width, data = iris, pch = 16, col = "blue")
```

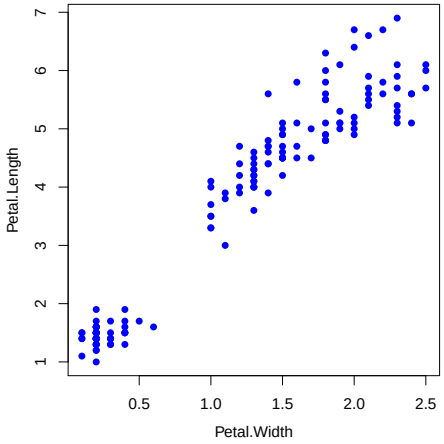


Note: you can use colour names

Traditional graphics: Scatter plots

You can change point shapes (pch) and colour (col):

```
plot(Petal.Length ~ Petal.Width, data = iris, pch = 16, col = 4)
```

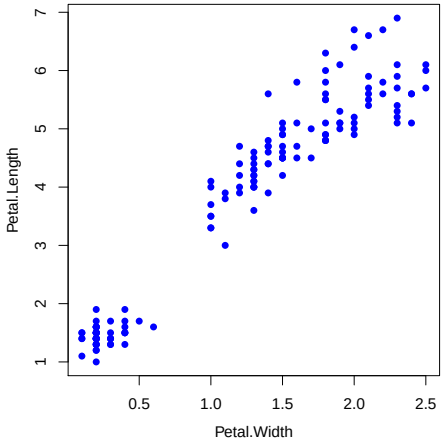


Note: you can use number of basic colours

Traditional graphics: Scatter plots

You can change point shapes (pch) and colour (col):

```
plot(Petal.Length ~ Petal.Width, data = iris, pch = 16, col = "#0000FFFF")
```



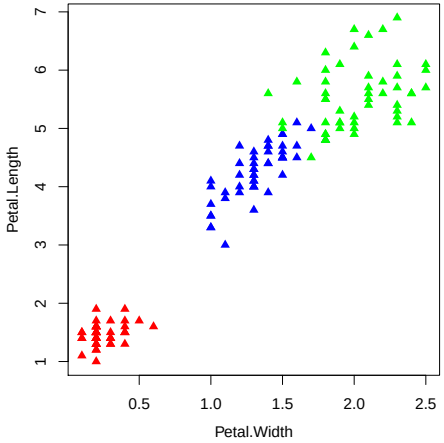

```
rgb(red = 0, green = 0, blue = 255, alpha = 255, maxColorValue = 255)
## [1] "#0000FFFF"
```

Note: you can have full control using hexadecimal!!

Traditional graphics: Scatter plots

You can change point shapes (pch) and colour (col):

```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 17, col = iris$Species)
```

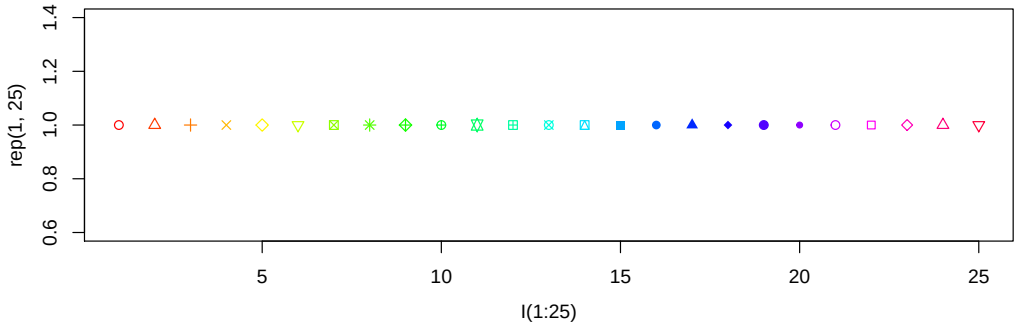


Note: you can use a palette to match the levels of a factor

Traditional graphics: Scatter plots

You can change point shapes (pch) and colour (col):

```
plot(rep(1, 25) ~ I(1:25), data = iris, pch = 1:25, col = rainbow(25))
```



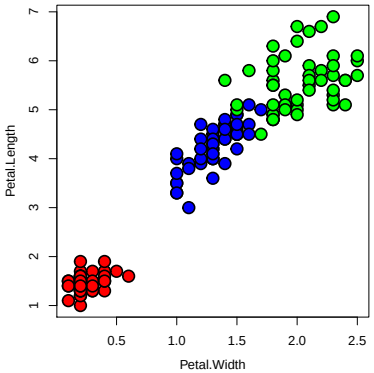
Note:

- there are 25 basic symbols (but other ways allow to use many more)
- check `"?rainbow"` for a list of different color palettes
- the `I()` allows for the creation of the vector before being interpreted by `plot()`

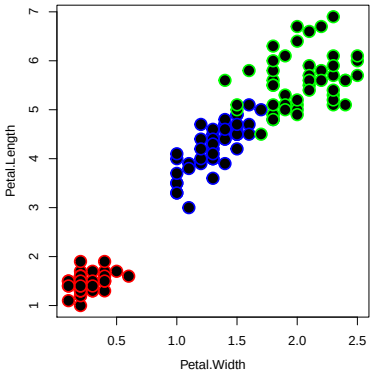
Traditional graphics: Scatter plots

For many elements you can set both an outline colour (col) and background colour (bg):

```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, cex = 2,
     pch = 21, col = "black", bg = iris$Species, lwd = 2)
```



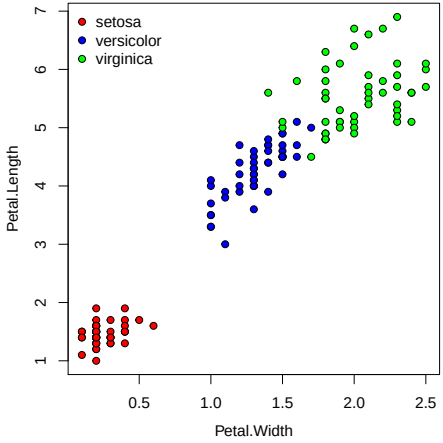

```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, cex = 2,
     pch = 21, col = iris$Species, bg = "black", lwd = 2)
```



Traditional graphics: Scatter plots

Add a legend to make colours understandable:

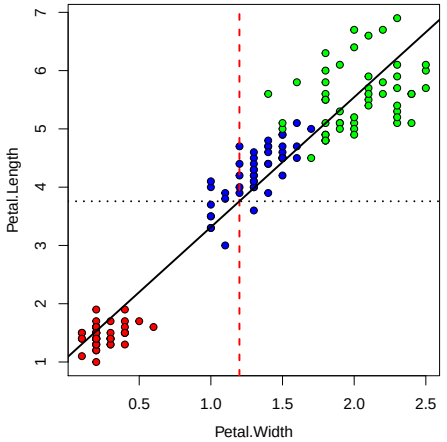
```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = iris$Species)
legend(x = "topleft", legend = c("setosa", "versicolor", "virginica"), pch = 21, pt.bg = c("red", "blue", "green"), bty = "n")
```



Traditional graphics: Scatter plots

You can add lines to the plot:

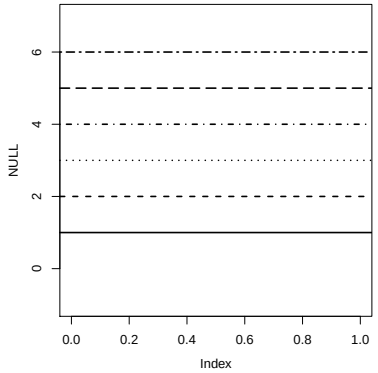
```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = iris$Species)
abline(v = mean(iris$Petal.Width), lty = 2, col = "red", lwd = 2)
abline(h = mean(iris$Petal.Length), lty = 3, col = "black", lwd = 2)
abline(a = 1.084, b = 2.23, lty = 1, lwd = 2)
```



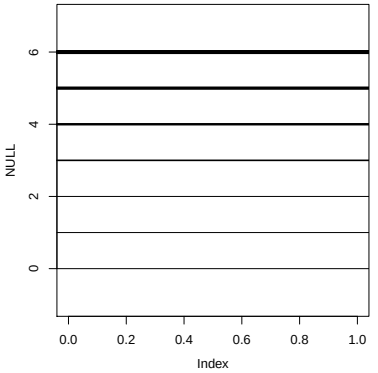
Traditional graphics: Scatter plots

You can add lines to the plot:

```
plot(NULL, xlim = c(0, 1), ylim = c(-1, 7))
abline(h = 0, lty = 0, lwd = 2)
abline(h = 1, lty = 1, lwd = 2)
abline(h = 2, lty = 2, lwd = 2)
abline(h = 3, lty = 3, lwd = 2)
abline(h = 4, lty = 4, lwd = 2)
abline(h = 5, lty = 5, lwd = 2)
abline(h = 6, lty = 6, lwd = 2)
```



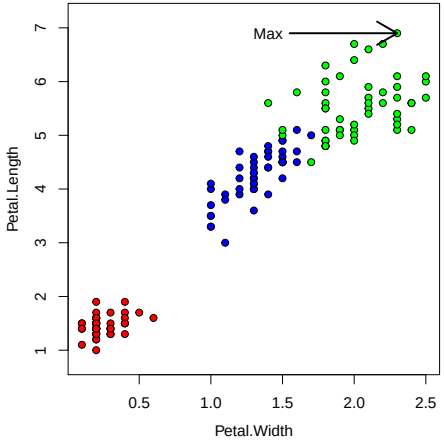

```
plot(NULL, xlim = c(0, 1), ylim = c(-1, 7))
abline(h = 0, lty = 1, lwd = 0.2)
abline(h = 1, lty = 1, lwd = 0.5)
abline(h = 2, lty = 1, lwd = 1)
abline(h = 3, lty = 1, lwd = 2)
abline(h = 4, lty = 1, lwd = 3)
abline(h = 5, lty = 1, lwd = 4)
abline(h = 6, lty = 1, lwd = 5)
```



Traditional graphics: Scatter plots

You can add text and arrows:

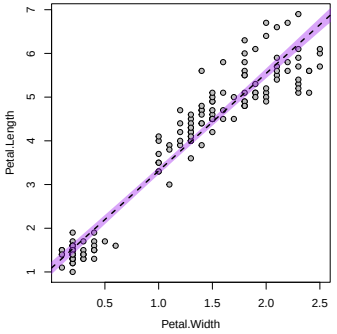
```
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = iris$Species, ylim = c(0.8, 7.2))
max.value <- iris[iris$Petal.Length == max(iris$Petal.Length), ]
arrows(x0 = max.value$Petal.Width - 0.75, y0 = max.value$Petal.Length,
       x1 = max.value$Petal.Width, y1 = max.value$Petal.Length, lwd = 2)
text(x = max.value$Petal.Width - 0.9, y = max.value$Petal.Length, labels = "Max")
```



Traditional graphics: Scatter plots

Including an interval around a prediction line requires you to build a polygon:

```
test_mod <- lm(Petal.Length ~ Petal.Width, data = iris)
newdat   <- data.frame(Petal.Width = seq(0, 3, length.out = 100))
pred     <- predict(test_mod, newdata = newdat, interval = "confidence")

plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = "grey")
polygon(x = c(newdat$Petal.Width, rev(newdat$Petal.Width)),
        y = c(pred[, "lwr"], rev(pred[, "upr"])),
        col = alpha("purple", alpha = 0.4),  ## alpha sets transparency!
        border = NA)  ## removes black line around the polygon
abline(test_mod, lty = 2, lwd = 2)
```

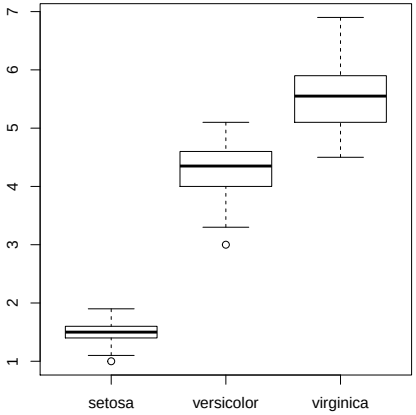


Plotting in **R**

Traditional graphics: Box plots

In traditional graphics, use `boxplot()` to draw a box plot:

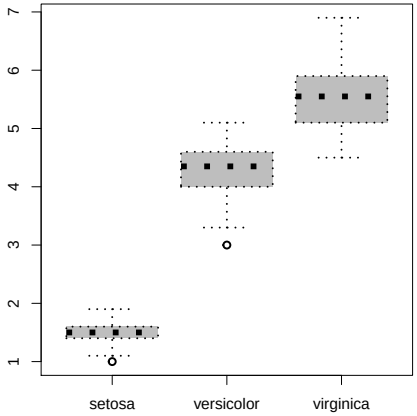
```
boxplot(Petal.Length ~ Species, data = iris)
```



Traditional graphics: Box plots

Many of the same changes made to scatter plots can be made here:

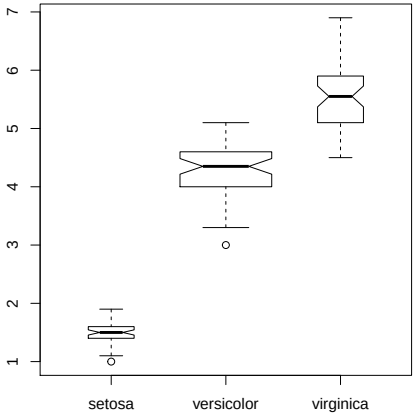
```
boxplot(Petal.Length ~ Species, data = iris, col = "grey", lwd = 2, lty = 3)
```



Traditional graphics: Box plots

There are also some boxplot specific arguments:

```
boxplot(Petal.Length ~ Species, data = iris, width = c(1, 2, 1), notch = TRUE)
```



Traditional graphics: Box plots

You can retrieve information by storing the output in an object:

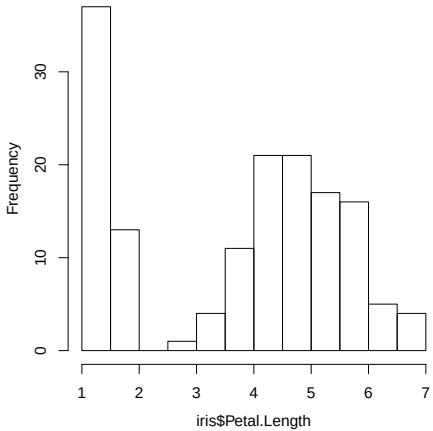
```
my_boxcox <- boxplot(Petal.Length ~ Species, data = iris, plot = FALSE)

my_boxcox
## $stats
##      [,1] [,2] [,3]
## [1,]  1.1 3.30 4.50
## [2,]  1.4 4.00 5.10
## [3,]  1.5 4.35 5.55
## [4,]  1.6 4.60 5.90
## [5,]  1.9 5.10 6.90
##
## $n
## [1] 50 50 50
##
## $conf
##          [,1]     [,2]     [,3]
## [1,] 1.455311 4.215933 5.371243
## [2,] 1.544689 4.484067 5.728757
##
## $out
## [1] 1 3
##
## $group
## [1] 1 2
##
## $names
## [1] "setosa"     "versicolor" "virginica"
```

Plotting in **R**

Traditional graphics: Histograms

In traditional graphics, use `hist()` to draw an histogram:

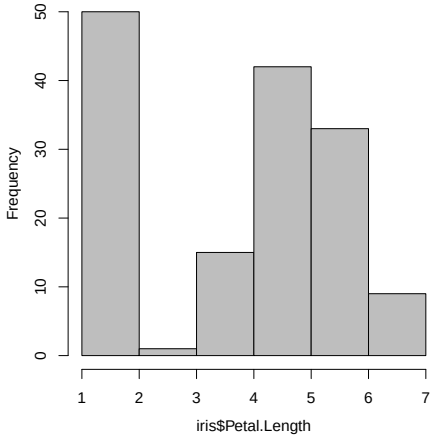
```
hist(iris$Petal.Length, main = "") ## main used here to remove the automatic title
```



Traditional graphics: Histograms

You can change the number and location of breaks between bins:

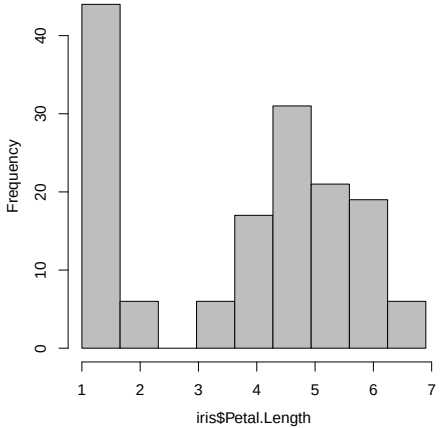
```
hist(iris$Petal.Length, main = "", breaks = 5, col = "grey")
```



Traditional graphics: Histograms

You can change the number and location of breaks between bins:

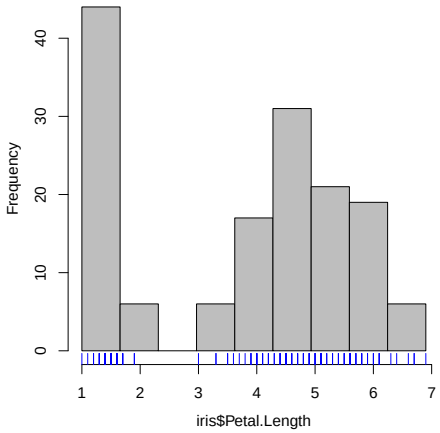
```
hist_breaks <- seq(min(iris$Petal.Length), max(iris$Petal.Length), length.out = 10)
hist(iris$Petal.Length, main = "", breaks = hist_breaks, col = "grey")
```



Traditional graphics: Histograms

You can change the number and location of breaks between bins:

```
hist_breaks <- seq(min(iris$Petal.Length), max(iris$Petal.Length), length.out = 10)
hist(iris$Petal.Length, main = "", breaks = hist_breaks, col = "grey")
rug(x = iris$Petal.Length, col = "blue")
```



Note: it never hurts to add a rug under an histogram!

Plotting in **R**

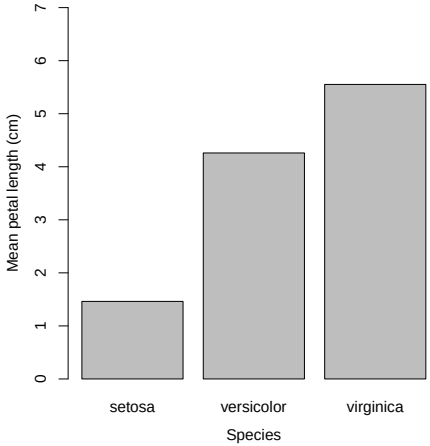
Traditional graphics: Bar plots

```
spp_means <- data.frame(Species = c("setosa", "versicolor", "virginica"),
                        mean = as.numeric(by(iris$Petal.Length, iris$Species, mean)),
                        SE = as.numeric(by(iris$Petal.Length, iris$Species, function(x)sd(x)/sqrt(length(x))))
                        )

spp_means
##      Species  mean         SE
## 1     setosa 1.462 0.02455980
## 2 versicolor 4.260 0.06645545
## 3  virginica 5.552 0.07804970
```

Traditional graphics: Bar plots

Bar plots

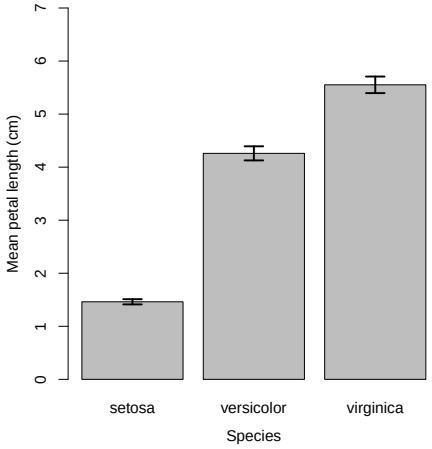
```
barplot(height = spp_means$mean, names.arg = spp_means$Species, ylim = c(0, 7),
        xlab = "Species", ylab = "Mean petal length (cm)")
```



Traditional graphics: Bar plots

Adding error bars can be done with the arrows function:

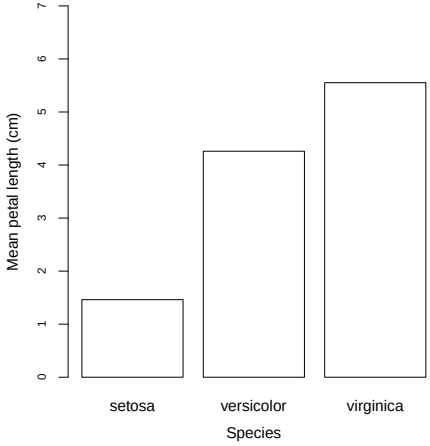
```
bar_locations <- barplot(height = spp_means$mean, names.arg = spp_means$Species, plot = FALSE)
barplot(height = spp_means$mean, names.arg = spp_means$Species, ylim = c(0, 7),
        xlab = "Species", ylab = "Mean petal length (cm)")
arrows(x0 = bar_locations[, 1], x1 = bar_locations[, 1],
       y0 = spp_means$mean - 2*spp_means$SE, y1 = spp_means$mean + 2*spp_means$SE,
       angle = 90, lwd = 2, code = 3, length = 0.1)
```



Traditional graphics: Bar plots

As before, there are similar arguments available:

```
barplot(height = spp_means$mean, names.arg = spp_means$Species, ylim = c(0, 7),
        xlab = "Species", ylab = "Mean petal length (cm)",
        col = "white", cex.axis = 0.75)
```

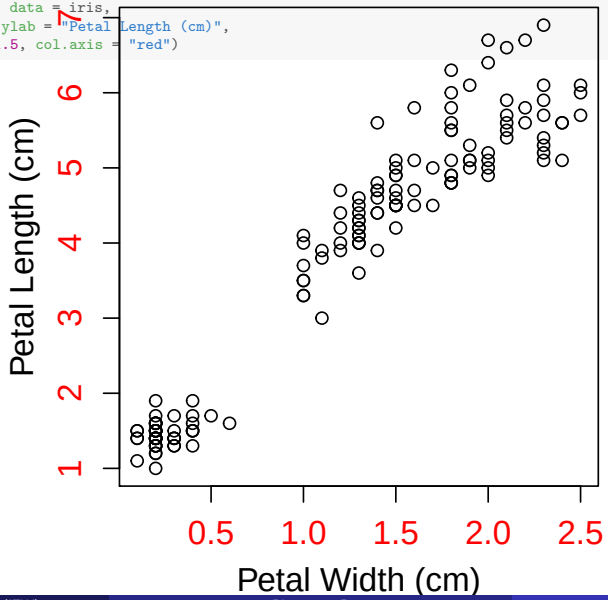


Plotting in **R**

Change text

You can size and colour of axis text easily.

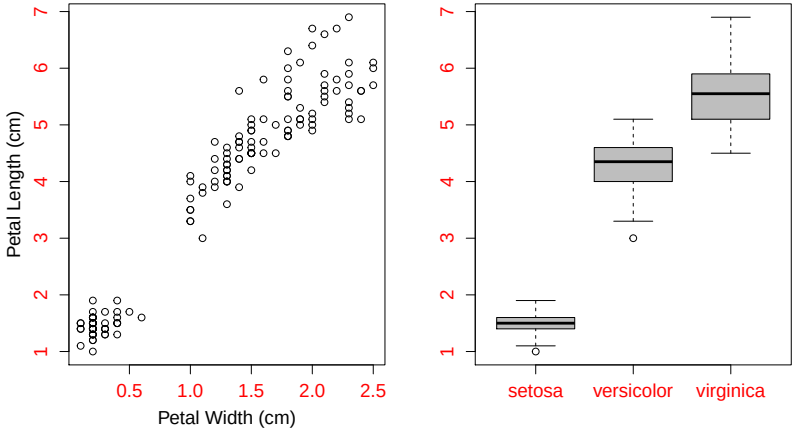
```
plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.5, cex.axis = 1.5, col.axis = "red")
```



Combining plots

You can easily combine multiple traditional graphicss together by changing the global parameters.

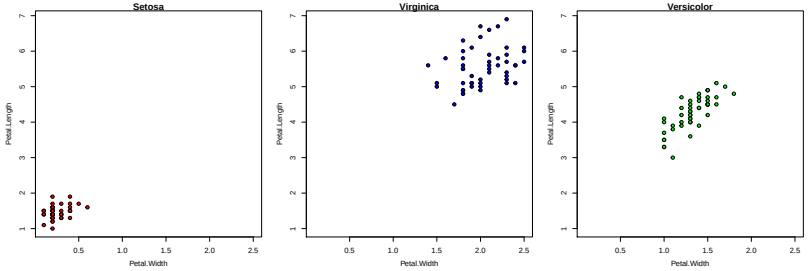
```
par(mfrow = c(1, 2))
plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.5, cex.axis = 1.5, col.axis = "red")
boxplot(Petal.Length ~ Species, data = iris, col = "grey")
par(mfrow = c(1, 1))
```



Facetting

This can be used to created 'facet' plots.

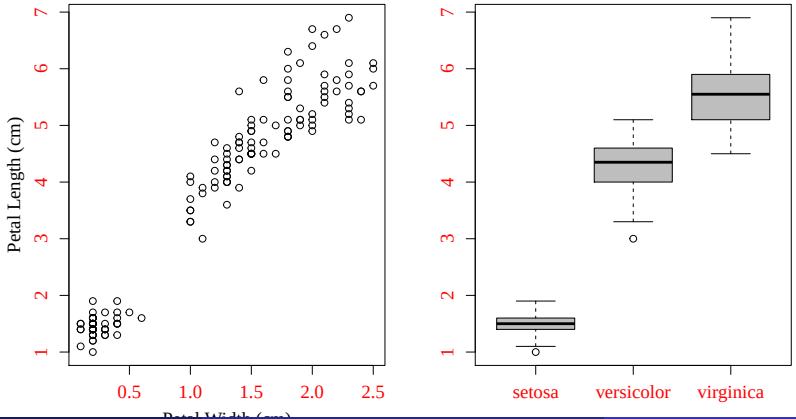
```
par(mgp = c(2.5, 1, 0))
par(mfrow = c(1, 3))
split_data <- split(iris, iris$Species)
plot(Petal.Length ~ Petal.Width, data = split_data$setosa, main = "Setosa", pch = 21, bg = "red", col = "black",
     xlim = range(iris$Petal.Width), ylim = range(iris$Petal.Length))
plot(Petal.Length ~ Petal.Width, data = split_data$virginica, main = "Virginica", pch = 21, bg = "blue", col = "black",
     xlim = range(iris$Petal.Width), ylim = range(iris$Petal.Length))
plot(Petal.Length ~ Petal.Width, data = split_data$versicolor, main = "Versicolor", pch = 21, bg = "green", col = "black",
     xlim = range(iris$Petal.Width), ylim = range(iris$Petal.Length))
```



Text family

Global parameters can also be used to change the font family.

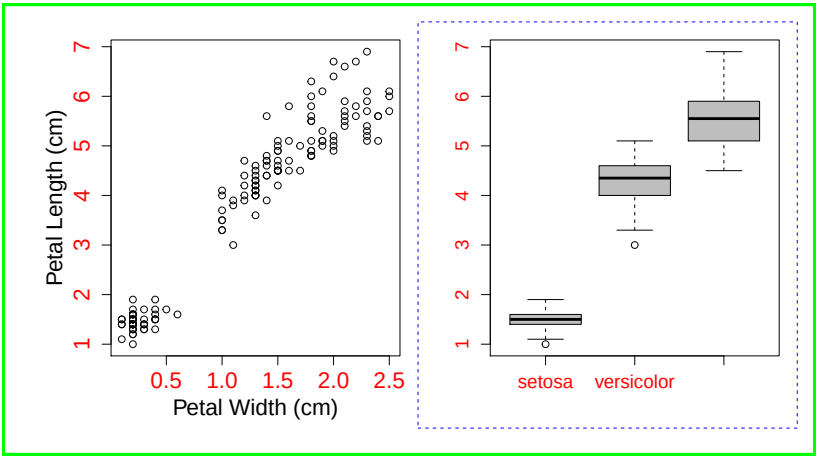
```
par(mfrow = c(1, 2), family = "serif")

plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.5, cex.axis = 1.5, col.axis = "red")
boxplot(Petal.Length ~ Species, data = iris, col = "grey")

par(mfrow = c(1, 1), family = "sans")
```



Plot margins

You can change individual plot margins (blue) and outer margins (green).

```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1),
    oma = c(1.5, 2, 1, 1))
plot(Petal.Length ~ Petal.Width, data = iris, xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.5, cex.axis = 1.5, col.axis = "red")
boxplot(Petal.Length ~ Species, data = iris, col = "grey", cex.lab = 0.75, cex.axis = 1.5, col.axis = "red")
```



Plotting in **R**

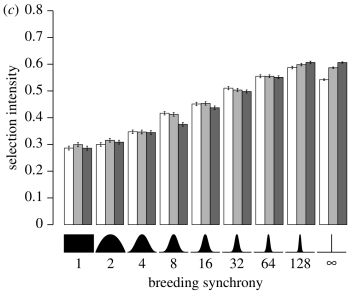
Traditional graphics: Exporting

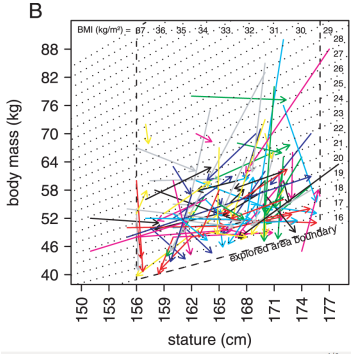
```
?pdf ?jpeg ?tiff ?bmp ?postscript
```

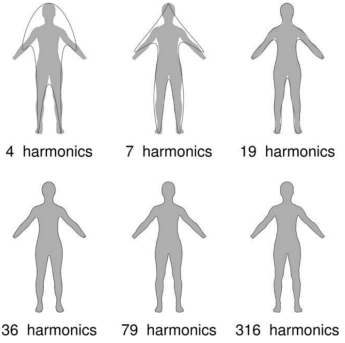
```
pdf("base_plot.pdf", width = 15, height = 5)
plot(Petal.Length ~ Petal.Width, data = iris)
dev.off()
```

Plotting in **R**

You can create home-made **R** graphics too





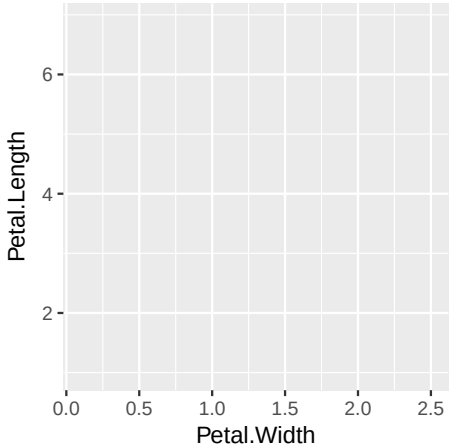


Plotting in **R**

ggplot: An introduction

Unlike traditional graphics, ggplot works around a single function. We use different functions to add layers onto a ggplot.

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

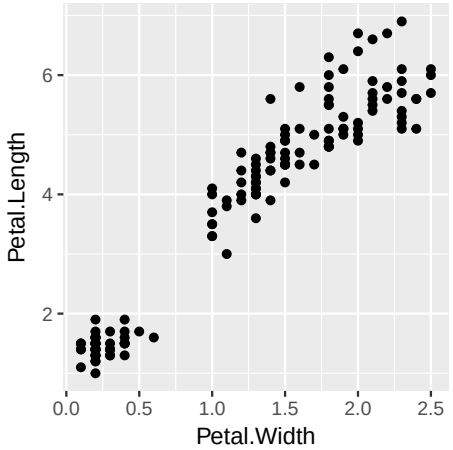


Plotting in **R**

ggplot: Scatter plots

We build onto our initial ggplot argument.

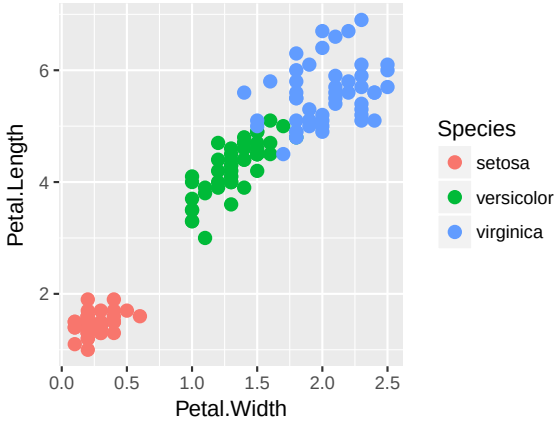
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point()
```



ggplot: Scatter plots

You can change point shapes (shape) and colour (colour). The numbers used here are **the same** as the ones we used above.

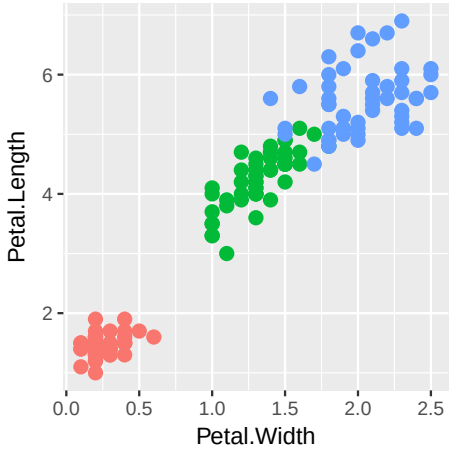
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), shape = 16, size = 3)
```



ggplot: Scatter plots

In ggplot the legend is included by default. You will need to manually remove it.

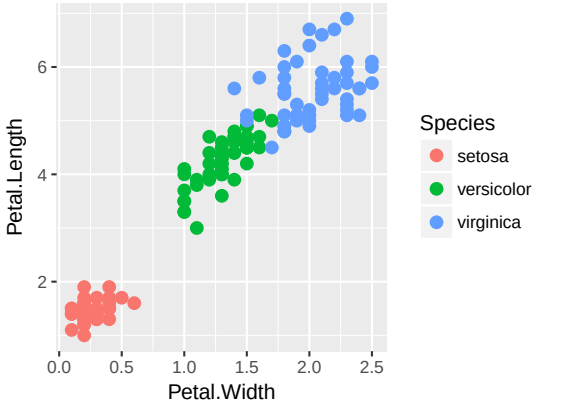
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), shape = 16, size = 3)+
  theme(legend.position = "none")
```



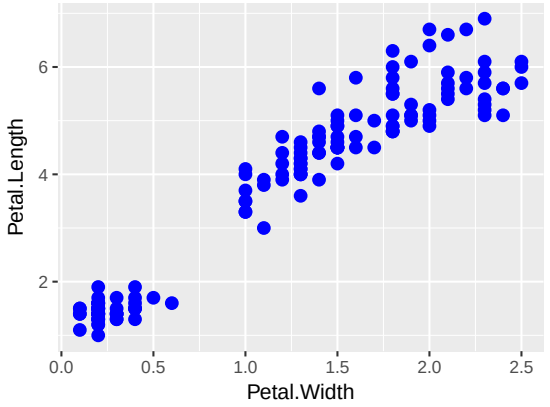
The aesthetic argument

The aesthetic argument in ggplot (aes) is a powerful tool for changing plot aesthetics. If you specify an aesthetic argument inside aes() it will give each data point a different aesthetic based on its value. If you specify the same aesthetic argument outside aes() it will give all data points the same aesthetic.

```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), shape = 16, size = 3)
```



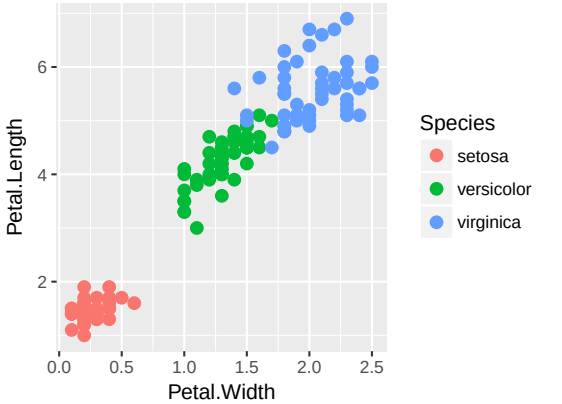

```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(colour = "blue", shape = 16, size = 3)
```



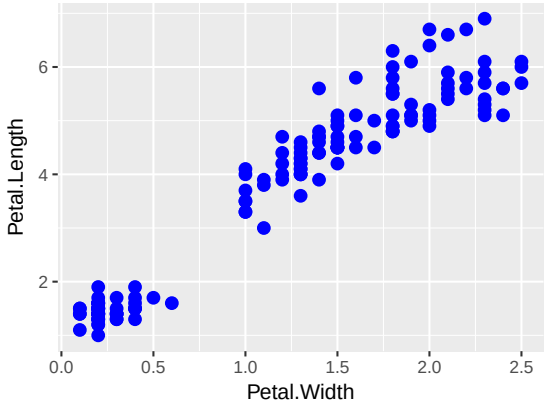
The aesthetic argument

The aesthetic argument in ggplot (aes) is a powerful tool for changing plot aesthetics. If you specify an aesthetic argument inside aes() it will give each data point a different aesthetic based on its value. If you specify the same aesthetic argument outside aes() it will give all data points the same aesthetic.

```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), shape = 16, size = 3)
```




```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(colour = "blue", shape = 16, size = 3)
```

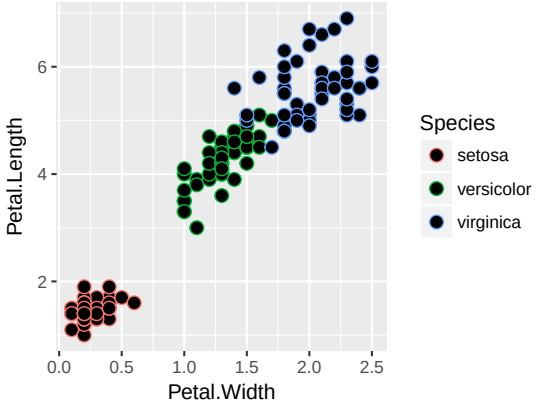


Here, we have used aes() to apply colours. However, we will use it later to make other changes.

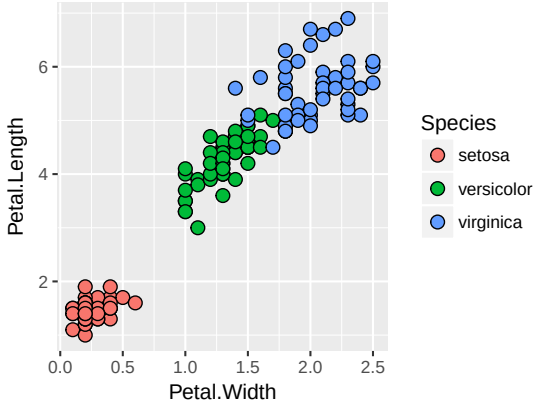
Beware

Again, there is a difference between outline colour (colour) and backgroun colour (fill).

```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(colour = Species), fill = "black",
             shape = 21, size = 3)
```



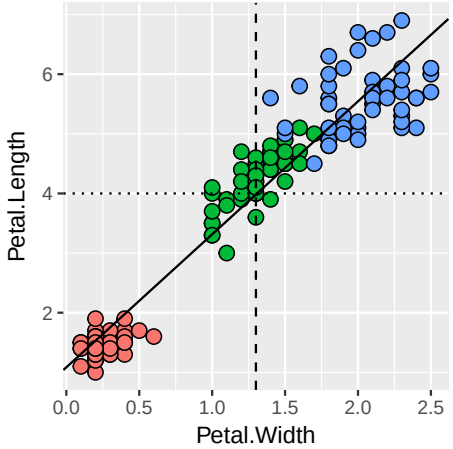

```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), colour = "black",
             shape = 21, size = 3)
```



ggplot: Scatter plots

You can add lines to the plot.

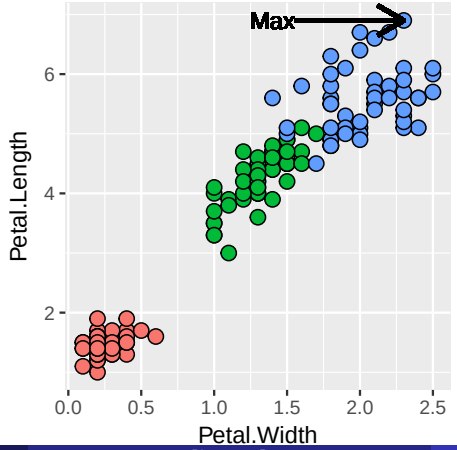
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), shape = 21, size = 3)+
  theme(legend.position = "none")+
  geom_hline(yintercept = 4, lty = 3)+
  geom_vline(xintecept = 1.3, lty = 2)+
  geom_abline(intercept = 1.084, slope = 2.23, lty = 1)
```



Traditional graphics: Scatter plots

You can add text and arrows.

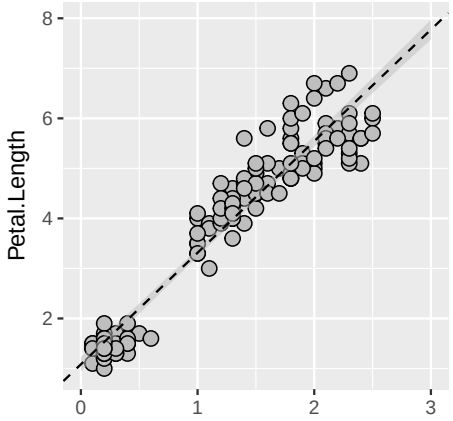
```
max.value <- iris[iris$Petal.Length == max(iris$Petal.Length), ]
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), shape = 21, size = 3)+
  theme(legend.position = "none")+
  geom_segment(aes(x = max.value$Petal.Width - 0.75, xend = max.value$Petal.Width,
                   y = max.value$Petal.Length, yend = max.value$Petal.Length), size = 1, arrow = arrow(length = unit(0.5, "cm")))+
  geom_text(aes(x = max.value$Petal.Width - 0.9, y = max.value$Petal.Length, label = "Max"))
```



ggplot: Scatter plots

In ggplot, confidence intervals can be added with specialised function geom_*ribbon*.

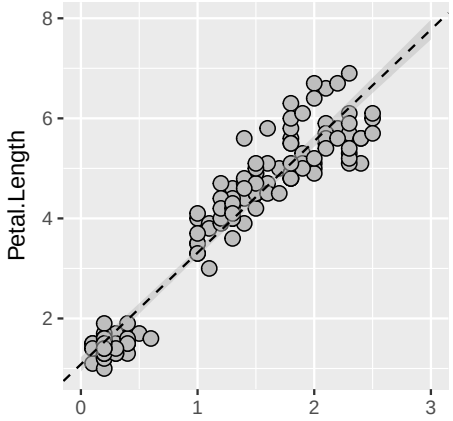
```
test_mod <- lm(Petal.Length ~ Petal.Width, data = iris)
newdat   <- data.frame(Petal.Width = seq(0, 3, length.out = 100))
pred     <- predict(test_mod, newdata = newdat, interval = "confidence")
ggplot() +
  geom_point(data = iris, aes(x = Petal.Width, y = Petal.Length), fill = "grey", shape = 21, size = 3) +
  geom_ribbon(data = NULL, aes(x = newdat$Petal.Width, ymin = pred[, 2], ymax = pred[, 3]), fill = "grey", alpha = 0.5) +
  geom_abline(intercept = coef(test_mod)[1], slope = coef(test_mod)[2], lty = 2) +
  theme(legend.position = "none")
```



ggplot: Scatter plots

In ggplot, confidence intervals can be added with specialised function geom_*ribbon*.

```
test_mod <- lm(Petal.Length ~ Petal.Width, data = iris)
newdat   <- data.frame(Petal.Width = seq(0, 3, length.out = 100))
pred     <- predict(test_mod, newdata = newdat, interval = "confidence")
ggplot() +
  geom_point(data = iris, aes(x = Petal.Width, y = Petal.Length), fill = "grey", shape = 21, size = 3) +
  geom_ribbon(data = NULL, aes(x = newdat$Petal.Width, ymin = pred[, 2], ymax = pred[, 3]), fill = "grey", alpha = 0.5) +
  geom_abline(intercept = coef(test_mod)[1], slope = coef(test_mod)[2], lty = 2) +
  theme(legend.position = "none")
```

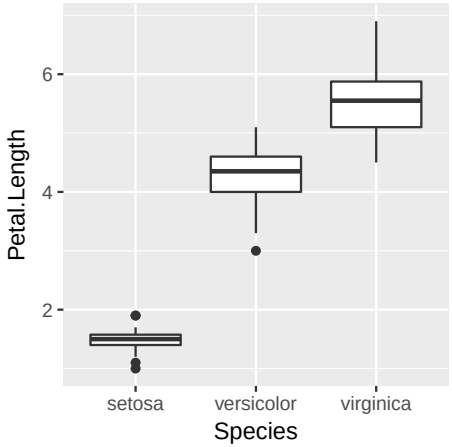


Plotting in **R**

ggplot: Boxplot

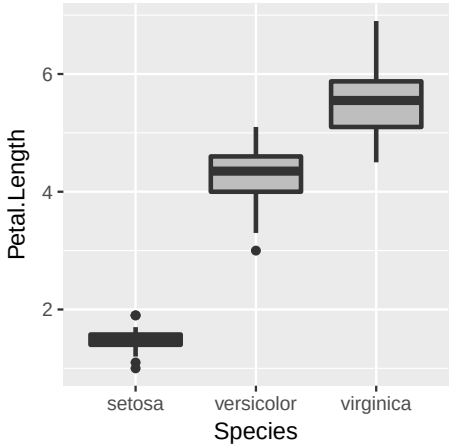
We build different plot types in a similar way.

```
ggplot(iris, aes(x = Species, y = Petal.Length))+
  geom_boxplot()
```



ggplot: Boxplot

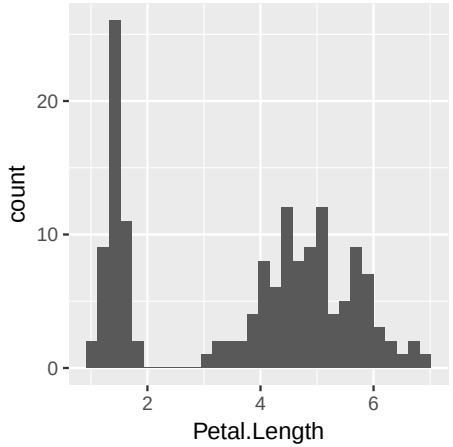
```
ggplot(iris, aes(x = Species, y = Petal.Length))+
  geom_boxplot(fill = "grey", size = 1, lty = 1)
```



Plotting in **R**

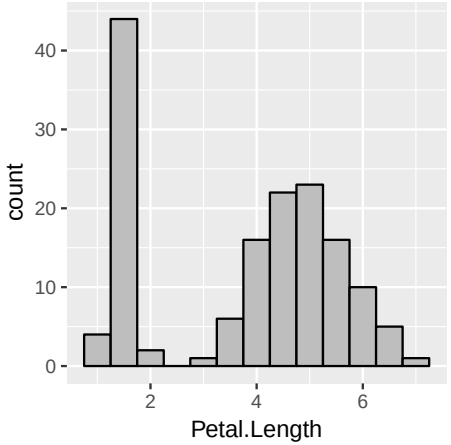
ggplot: Histogram

```
ggplot(iris, aes(x = Petal.Length))+
  geom_histogram()
```



ggplot: Histogram

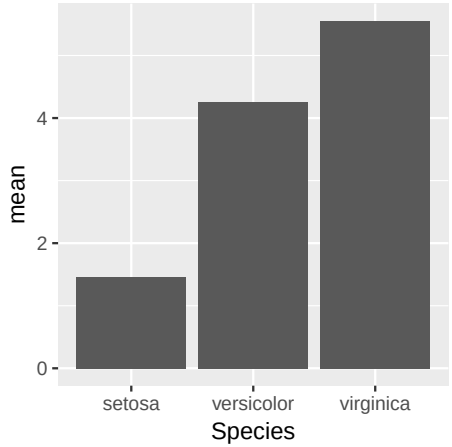
```
ggplot(iris, aes(x = Petal.Length))+
  geom_histogram(binwidth = 0.5, colour = "black", fill = "grey")
```



Plotting in **R**

ggplot: Bar plots

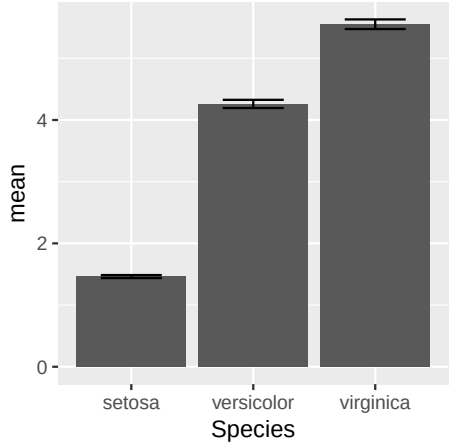
```
ggplot(spp_means, aes(x = Species, y = mean))+
  geom_col()
```



ggplot: Bar plots

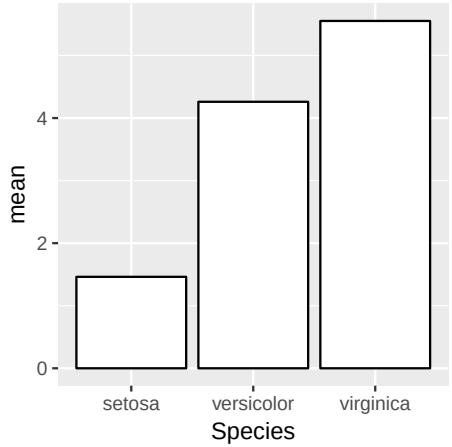
Adding errorbars in ggplot is much easier than traditional graphics.

```
ggplot(spp_means, aes(x = Species, y = mean))+
  geom_col()+
  geom_errorbar(aes(ymin = mean - SE, ymax = mean + SE), size = 1, width = 0.5)
```



ggplot: Bar plots

```
ggplot(spp_means, aes(x = Species, y = mean))+
  geom_col(fill = "white", colour = "black")
```



Plotting in **R**

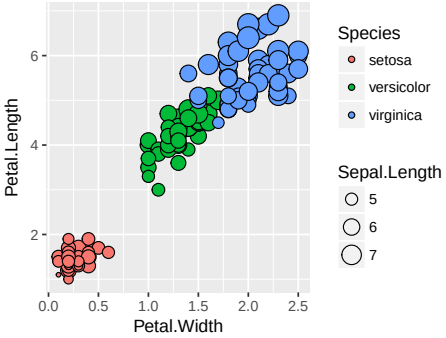
Introduction to themes

In ggplot, you can change aesthetics in individual segments of the code *or* you can change information for the whole plot using theme.

Using the aesthetic argument more

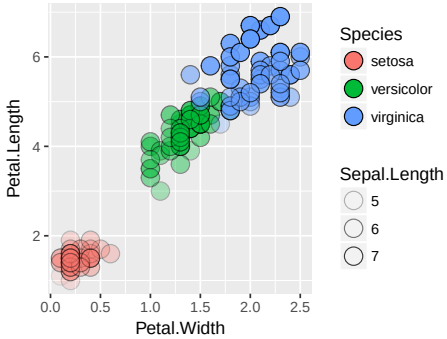
We'll start by looking back at the aesthetic (aes) argument. We can use it to change multiple different aesthetics of a plot.

```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(fill = Species, size = Sepal.Length), shape = 21)
```



Using the aesthetic argument more

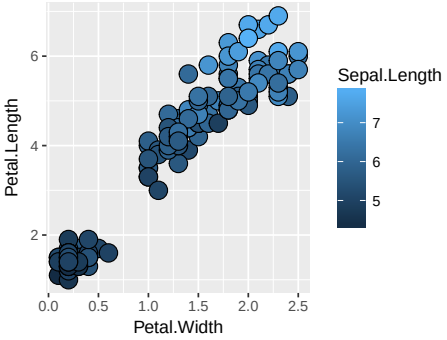
```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(fill = Species, alpha = Sepal.Length), shape = 21, size = 5)
```



Using the aesthetic argument more

Applying aesthetics to continuous variables will be different to categorical variables.

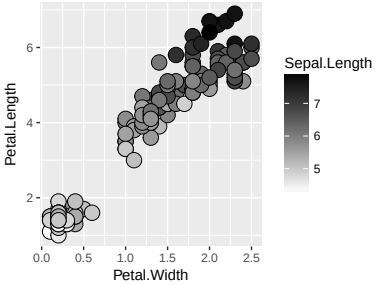
```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(fill = Sepal.Length), shape = 21, size = 5)
```



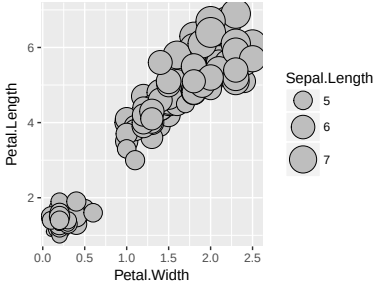
Using the aesthetic argument more

We can adjust the way the aesthetics are applied.

```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(fill = Sepal.Length), shape = 21, size = 5)+
  scale_fill_continuous(low = "white", high = "black")
```



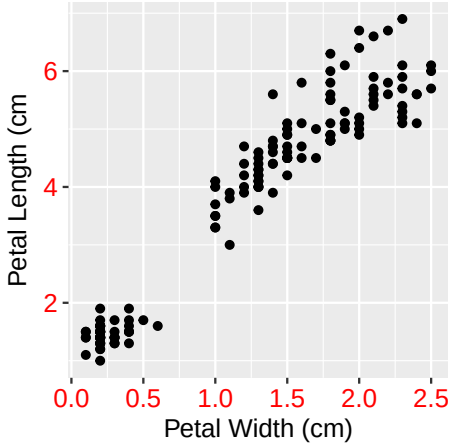

```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(size = Sepal.Length), shape = 21, fill = "grey")+
  scale_size_continuous(range = c(3, 10))
```



Change text

You can change size and colour of axis text easily. Note that text size uses different measurement units.

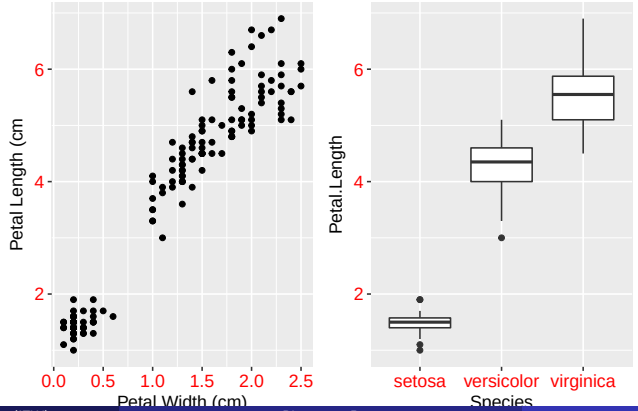
```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  xlab("Petal Width (cm)")+
  ylab("Petal Length (cm")+
  theme(axis.text = element_text(size = 12, colour = "red"),
        axis.title = element_text(size = 12))
```



Combining plots

Combining plots it much less straightforward in ggplot. You need to use an additional package.

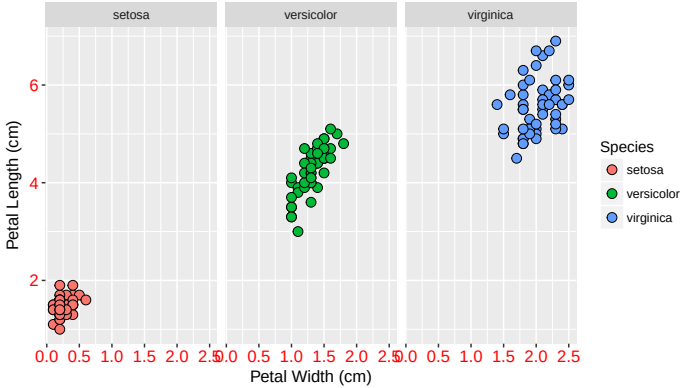
```
scatter <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  xlab("Petal Width (cm)")+
  ylab("Petal Length (cm")+
  theme(axis.text = element_text(size = 12, colour = "red"), axis.title = element_text(size = 12))
box     <- ggplot(iris, aes(x = Species, y = Petal.Length))+
  geom_boxplot()+
  theme(axis.text = element_text(size = 12, colour = "red"), axis.title = element_text(size = 12))
gridExtra::grid.arrange(scatter, box, nrow = 1)
```



Faceting

Although combining multiple plots is cumbersome, there is an inbuilt option to create facets.

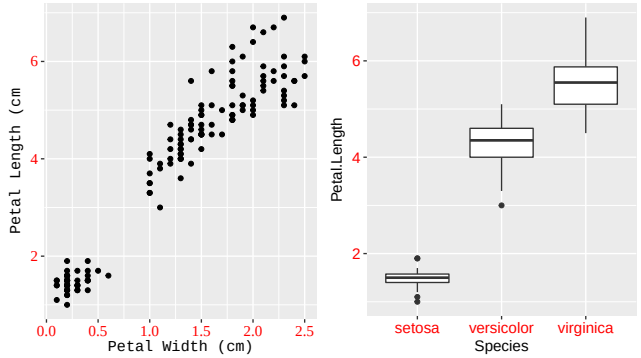
```
ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point(aes(fill = Species), shape = 21, colour = "black", size = 3)+
  xlab("Petal Width (cm)")+
  ylab("Petal Length (cm)")+
  facet_wrap(~Species) +
  theme(axis.text = element_text(size = 12, colour = "red"), axis.title = element_text(size = 12))
```



Text family

Unlike traditional graphics, in ggplot you can change font family of individual elements in theme.

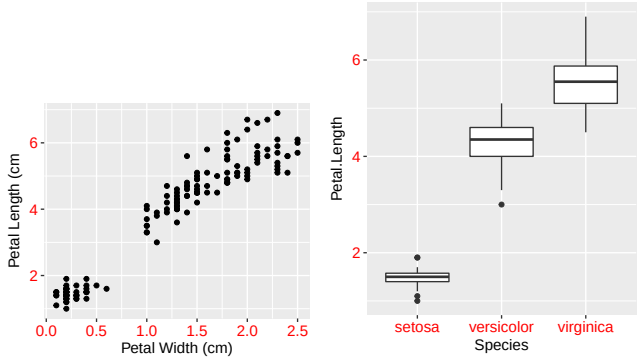
```
scatter <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  xlab("Petal Width (cm)")+
  ylab("Petal Length (cm)")+
  theme(axis.text = element_text(size = 12, colour = "red", family = "serif"), axis.title = element_text(size = 12, family = "mono"))
box     <- ggplot(iris, aes(x = Species, y = Petal.Length))+
  geom_boxplot()+
  theme(axis.text = element_text(size = 12, colour = "red", family = "sans"), axis.title = element_text(size = 12, family = "sans"))
gridExtra::grid.arrange(scatter, box, nrow = 1)
```



Plot margins

Plot margins are also controlled in theme of each plot individually.

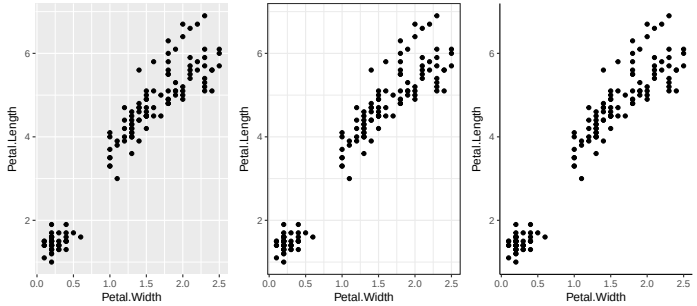
```
scatter <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  xlab("Petal Width (cm)")+
  ylab("Petal Length (cm")+
  theme(axis.text = element_text(size = 12, colour = "red"), axis.title = element_text(size = 12),
        plot.margin = unit(c(4, 4, 1, 1), "mm"))
box     <- ggplot(iris, aes(x = Species, y = Petal.Length))+
  geom_boxplot()+
  theme(axis.text = element_text(size = 12, colour = "red"), axis.title = element_text(size = 12))
gridExtra::grid.arrange(scatter, box, nrow = 1)
```



Preset themes

ggplot also has a number of preset themes that you can use.

```
grey <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  theme_grey()
bw <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  theme_bw()
classic <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  theme_classic()
gridExtra::grid.arrange(grey, bw, classic, nrow = 1)
```



Plotting in **R**

ggplot: Exporting

```
?ggsave
```

```
classic <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length))+
  geom_point()+
  theme_classic()
ggsave("ggplot.pdf", plot = classic, width = 15, height = 5)
```

Plotting in **R**

Which plotting tool should you use?

Traditional graphics:

- No new packages required
- Easy to combine many plots
- Looks good out of the box (but harder to customise)

Which plotting tool should you use?

Traditional graphics:

- No new packages required
- Easy to combine many plots
- Looks good out of the box (but harder to customise)

ggplot:

- Requires multiple packages for best results
- More difficult to combine many plots
- Easy to customise (but looks poor out of the box)
- Faster. Better for bigger datasets

Which plotting tool should you use?

It is useful to know both!!