Getting to plot 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**:

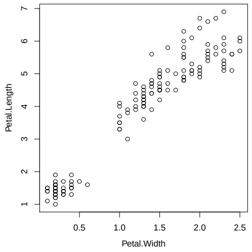
- traditional graphics (based on `graphics`)
- `lattice` (based on `grid`)
- `ggplot2` (based on `grid`)

Note:

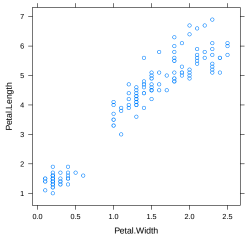
- `graphics` and `grid` are part of any basic installation of **R**
- `lattice` is part of the so-called list of CRAN recommended packages
- `ggplot2` is part of the tidyverse universe (from RStudio)
- we will focus on traditional graphics and `ggplot2`, but `lattice` is excellent too!
- some other packages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

An example (using default settings)

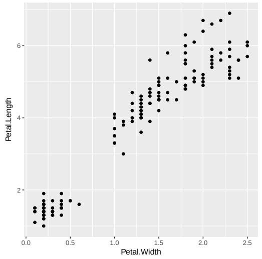
```r
plot(Petal.Length ~ Petal.Width,
     data = iris)
```



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



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



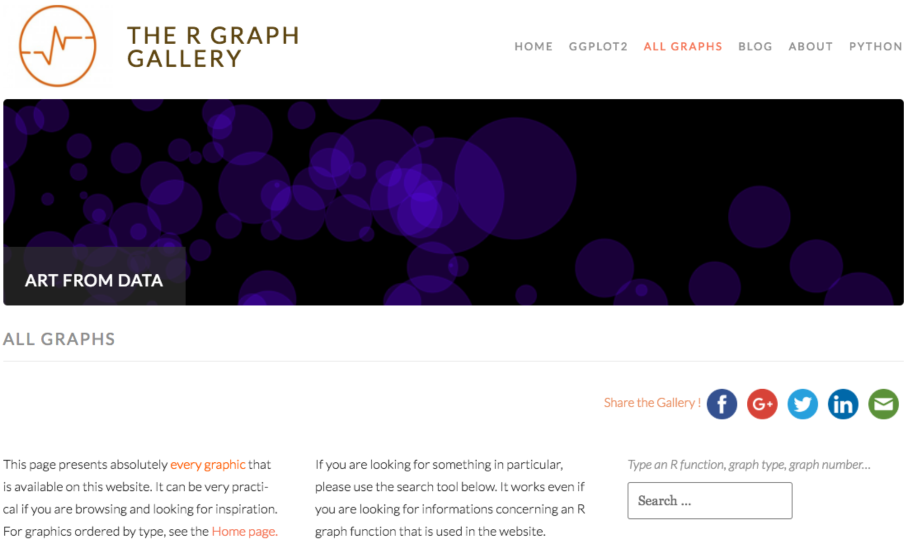
How to learn on your own?

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

```r
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/`)



ALL GRAPHS

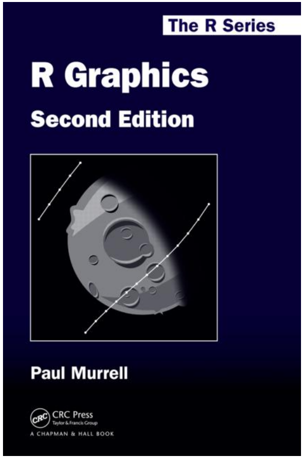
This page presents absolutely every graphic that is available on this website. It can be very practical if you are browsing and looking for inspiration. For graphics ordered by type, see the Home page.

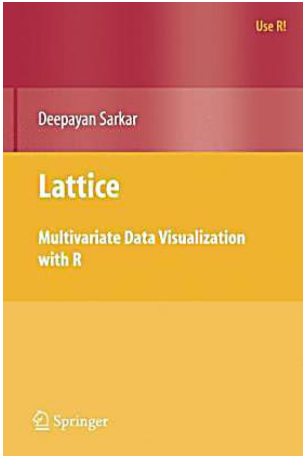
If you are looking for something in particular, please use the search tool below. It works even if you are looking for informations concerning an R graph function that is used in the website.

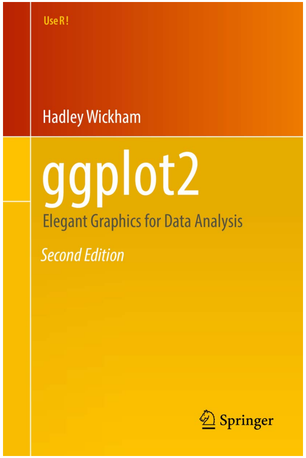
Type an R function, graph type, graph number...

How to learn on your own?

3. Read books:







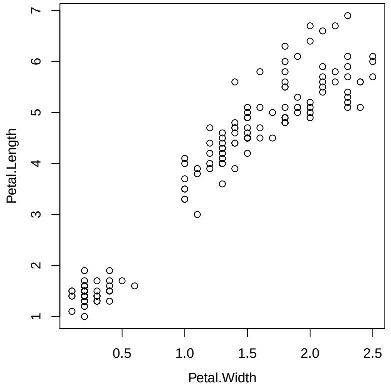
Plotting in **R**

Plotting in **R**

Traditional graphics: scatter plots

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

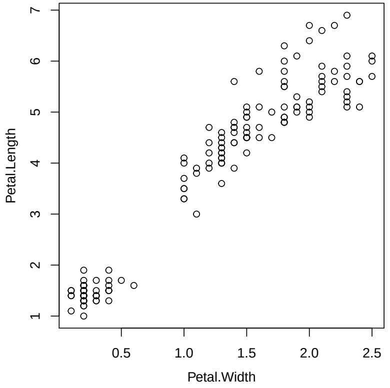
```
plot(Petal.Length ~ Petal.Width, data = iris)
```



Traditional graphics: scatter plots

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

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

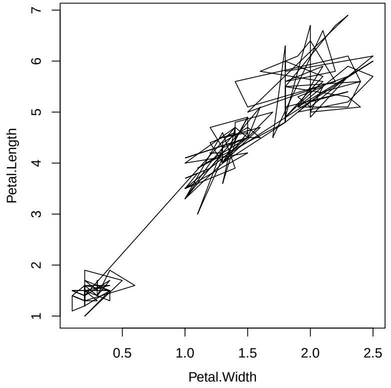


Note: see `"?plot.default"`.

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")
```

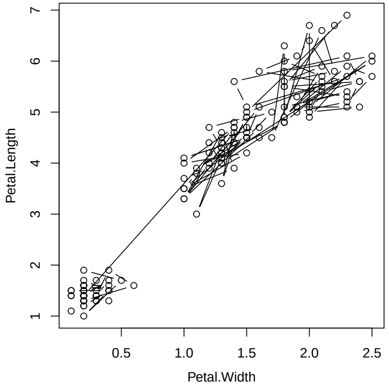


Note: it makes more sense when data are ordered. . .

Traditional graphics: scatter plots

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

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

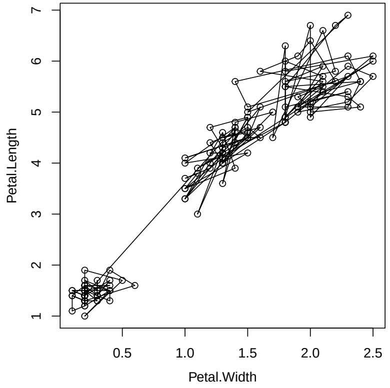


Note: it makes more sense when data are ordered. . .

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")
```

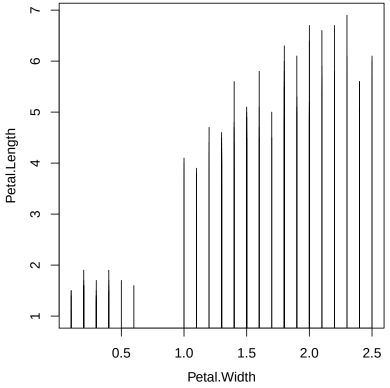


Note: it makes more sense when data are ordered. . .

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")
```

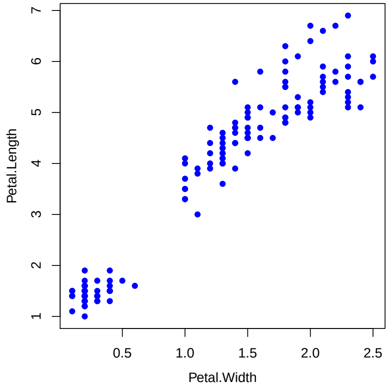


Note: it makes more sense when x-values are unique. . .

Traditional graphics: scatter plots

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

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

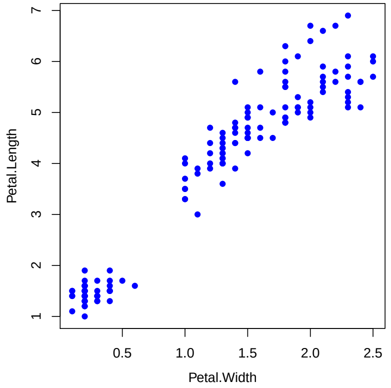


Note: you can use colour names.

Traditional graphics: scatter plots

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

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

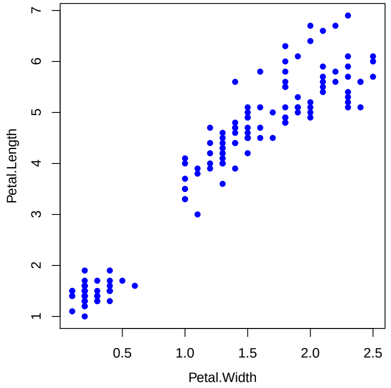


Note: you can use number of basic colours.

Traditional graphics: scatter plots

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

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



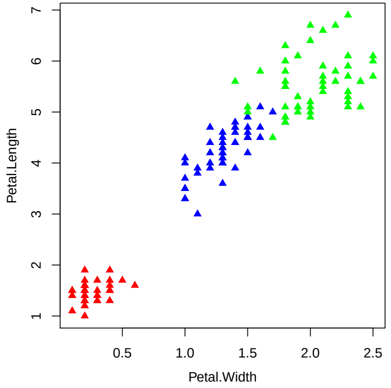
```
rgb(red = 0, green = 0, blue = 255, alpha = 255, maxColorValue = 255)
## [1] "#0000FFFF"
```

Note: you can have full control using hexadecimal!!

Traditional graphics: scatter plots

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

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

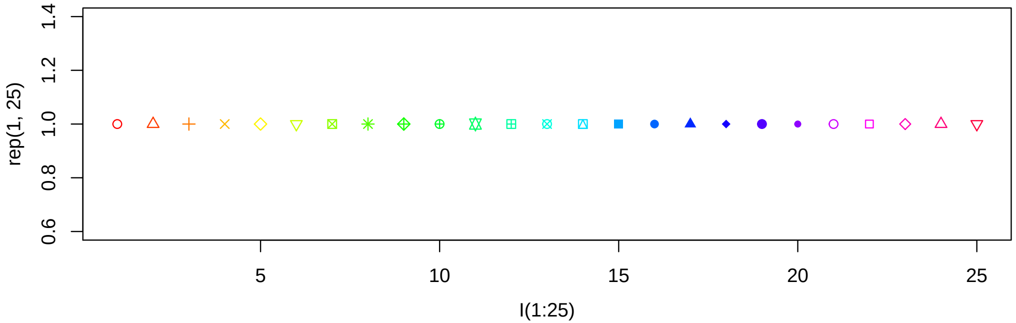


Note: you can use a palette to match the levels of a factor.

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:

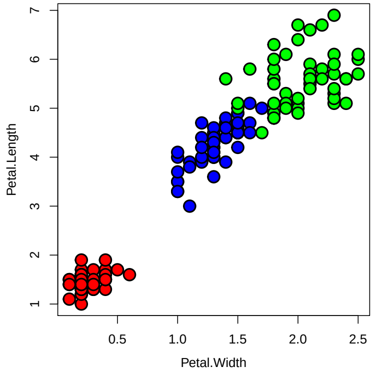
- there are 25 basic symbols (but other ways allow to use many more)
- check `"?rainbow"` for a list of different color palettes
- the `I()` allows for the creation of the vector before being interpreted by `plot()`

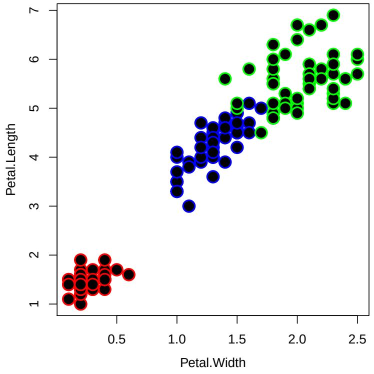
Traditional graphics: scatter plots

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

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

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

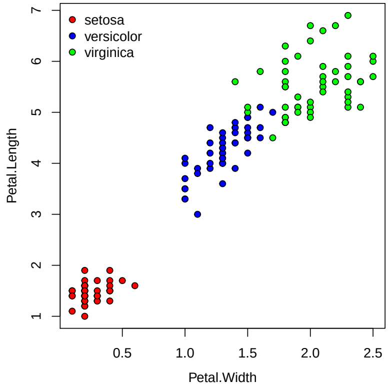




Traditional graphics: scatter plots

Add a legend to make colours understandable:

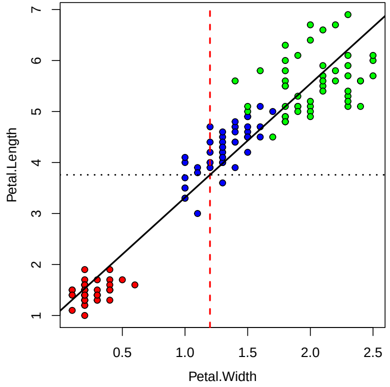
```r
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = iris$Species)
legend(x = "topleft", legend = c("setosa", "versicolor", "virginica"), pch = 21, pt.bg = c("red", "blue", "green"), bty = "n")
```



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)
```

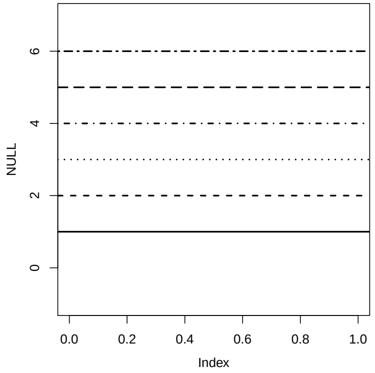


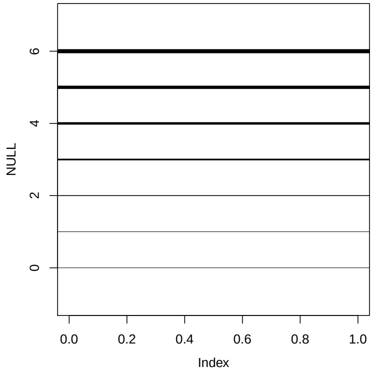
Traditional graphics: scatter plots

You can add lines to the plot:

```r
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)
```

```r
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)
```

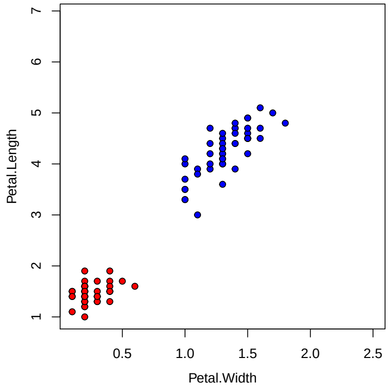




Traditional graphics: scatter plots

You can also add additional points to the plot:

```r
versicolor <- subset(iris, Species == "versicolor")
setosa     <- subset(iris, Species == "setosa")
plot(Petal.Length ~ Petal.Width, data = versicolor, pch = 21, bg = "blue", xlim = range(iris$Petal.Width), ylim = range(iris$Petal.Length))
points(Petal.Length ~ Petal.Width, data = setosa, pch = 21, bg = "red", xlim = range(iris$Petal.Width), ylim = range(iris$Petal.Length))
```

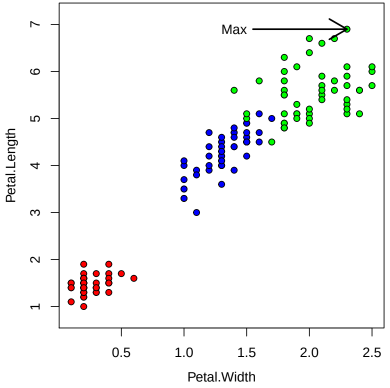


Note: You need to make sure the axis limits are the same!

Traditional graphics: scatter plots

You can add text and arrows:

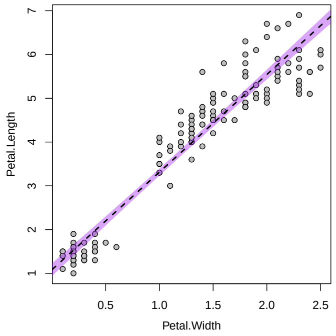
```r
palette(c("red", "blue", "green"))
plot(Petal.Length ~ Petal.Width, data = iris, pch = 21, bg = iris$Species, ylim = c(0.8, 7.2))
max.value <- iris[iris$Petal.Length == max(iris$Petal.Length), ]
arrows(x0 = max.value$Petal.Width - 0.75, y0 = max.value$Petal.Length,
       x1 = max.value$Petal.Width, 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:

```r
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)
```

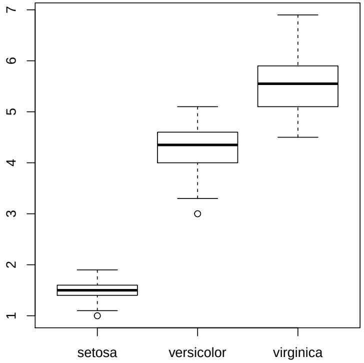


Plotting in **R**

Traditional graphics: box plots

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

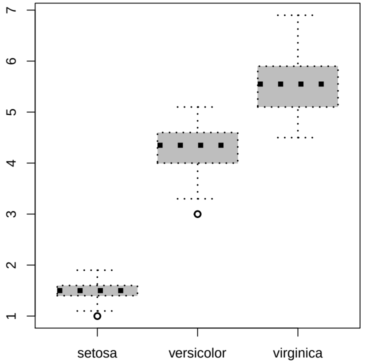
```r
boxplot(Petal.Length ~ Species, data = iris)
```



Traditional graphics: box plots

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

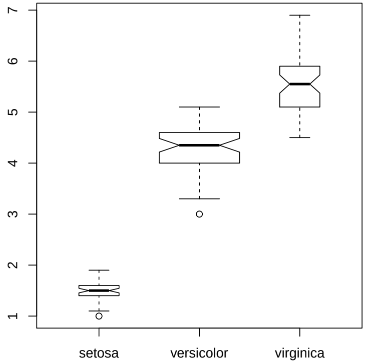
```
boxplot(Petal.Length ~ Species, data = iris, col = "grey", lwd = 2, lty = 3)
```



Traditional graphics: box plots

There are also some boxplot specific arguments:

```r
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
```

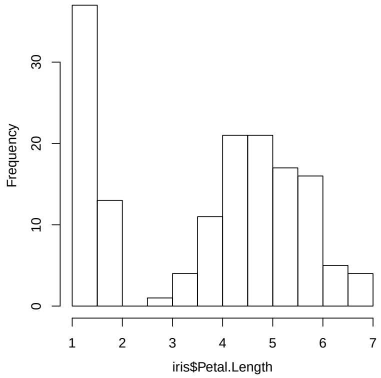
```
## $stats
##      [,1] [,2] [,3]
## [1,]  1.1 3.30 4.50
## [2,]  1.4 4.00 5.10
## [3,]  1.5 4.35 5.55
## [4,]  1.6 4.60 5.90
## [5,]  1.9 5.10 6.90
##
## $n
## [1] 50 50 50
##
## $conf
##          [,1]     [,2]     [,3]
## [1,] 1.455311 4.215933 5.371243
## [2,] 1.544689 4.484067 5.728757
##
## $out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

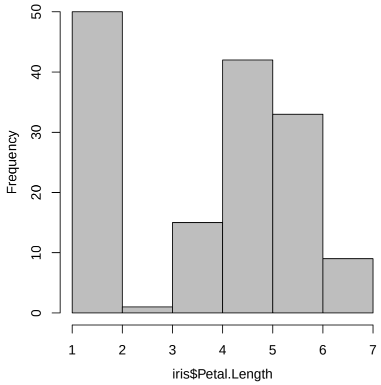
```r
hist(iris$Petal.Length, main = "") ## main used here to remove the automatic title
```



Traditional graphics: histograms

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

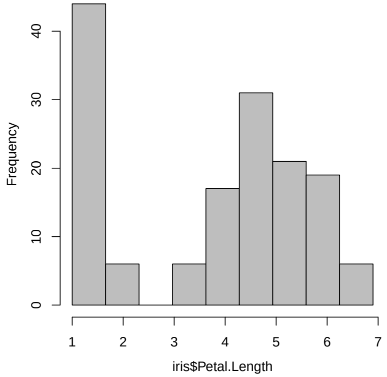
```
hist(iris$Petal.Length, main = "", breaks = 5, col = "grey")
```



Traditional graphics: histograms

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

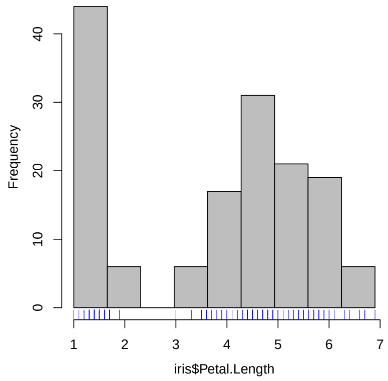
```
hist_breaks <- seq(min(iris$Petal.Length), max(iris$Petal.Length), length.out = 10)
hist(iris$Petal.Length, main = "", breaks = hist_breaks, col = "grey")
```



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

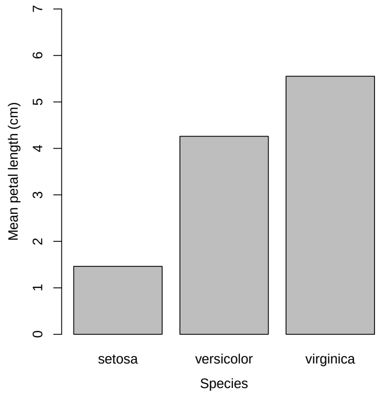
```
spp_means <- data.frame(Species = c("setosa", "versicolor", "virginica"),
                        mean = as.numeric(by(iris$Petal.Length, iris$Species, mean)),
                        SE = as.numeric(by(iris$Petal.Length, iris$Species, function(x)sd(x)/sqrt(length(x))))
                        )

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

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

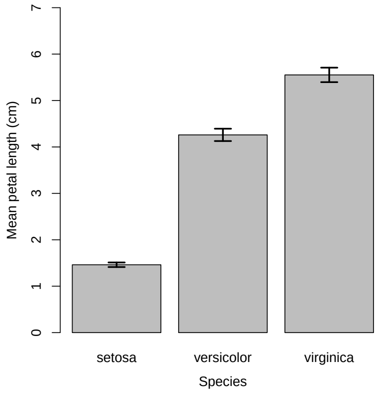
```
barplot(height = spp_means$mean, names.arg = spp_means$Species, ylim = c(0, 7),
        xlab = "Species", ylab = "Mean petal length (cm)")
```



Traditional graphics: bar plots

Adding error bars can be done with the arrows function:

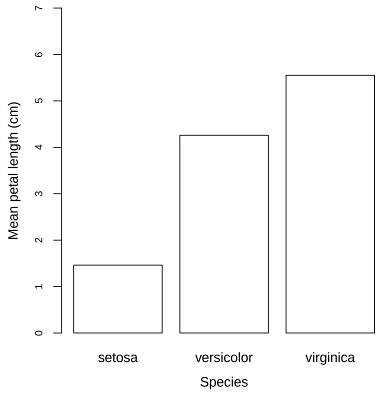
```
bar_locations <- barplot(height = spp_means$mean, names.arg = spp_means$Species, plot = FALSE)
barplot(height = spp_means$mean, names.arg = spp_means$Species, ylim = c(0, 7),
        xlab = "Species", ylab = "Mean petal length (cm)")
arrows(x0 = bar_locations[, 1], x1 = bar_locations[, 1],
       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)
```

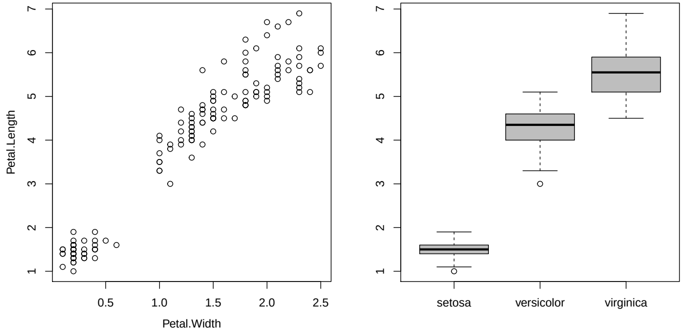


Plotting in **R**

Combining plots

You can easily combine multiple traditional graphics using global parameters:

```r
oldpar <- par(mfrow = c(1, 2))
plot(Petal.Length ~ Petal.Width, data = iris)
boxplot(Petal.Length ~ Species, data = iris, col = "grey")
```



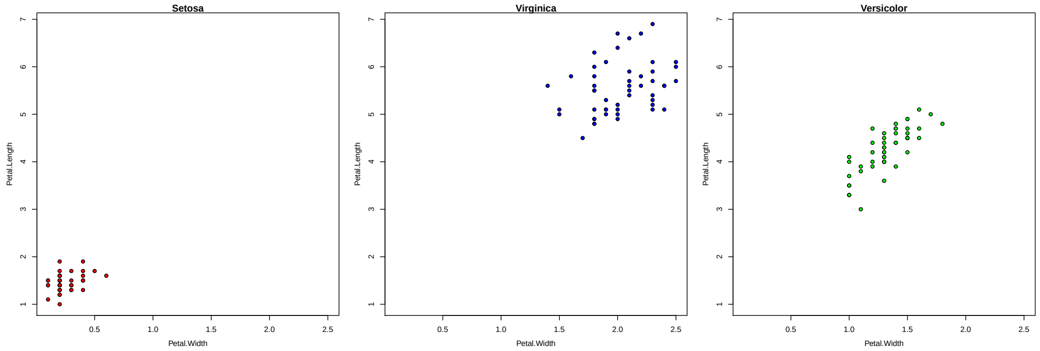
```r
par(oldpar) ## or par(mfrow = c(1, 1))
```

Note: if you want to combine graphics of different sizes it is a bit more complex (see `?layout`).

Facetting

This can be used to created 'facet' plots:

```
oldpar <- 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))
```

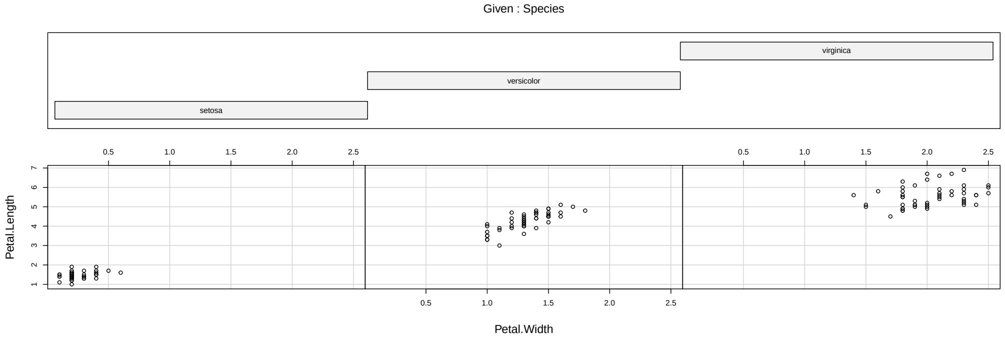


```
par(oldpar)
```

Facetting

Note: there is also an easier function for that:

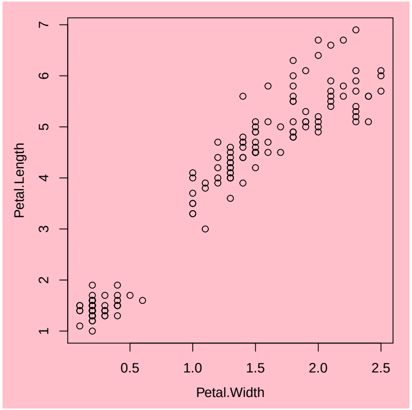
```r
coplot(Petal.Length ~ Petal.Width | Species, data = iris, rows = 1)
```



Background

Global parameters can also be used to change the background colour:

```r
oldpar <- par(bg = "pink")
plot(Petal.Length ~ Petal.Width, data = iris)
```



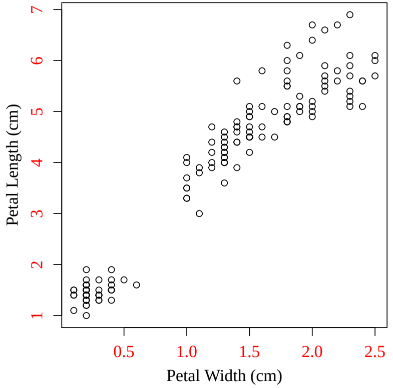
```r
par(oldpar)
```

Note: you can also plot a background image but you need to use specific packages for that.

Modifying typefaces

Global parameters can also be used to change the typeface:

```
oldpar <- par(family = "serif")
plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.25, cex.axis = 1.25, col.axis = "red")
```

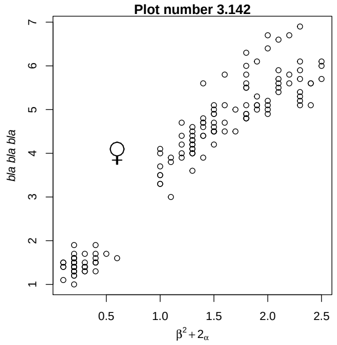


```
par(oldpar)
```

Special characters

You can use weird characters:

```
foo <- round(pi, 3)
plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = expression(beta^2 + 2[alpha]), ylab = expression(paste(italic("bla bla bla"))),
     main = paste("Plot number", foo))
#text(0.4, 4, "\u2640", cex = 3) ## require font with unicode installed (try it!)
text(0.6, 4, "\\VE", vfont = c("serif", "plain"), cex = 3)
```

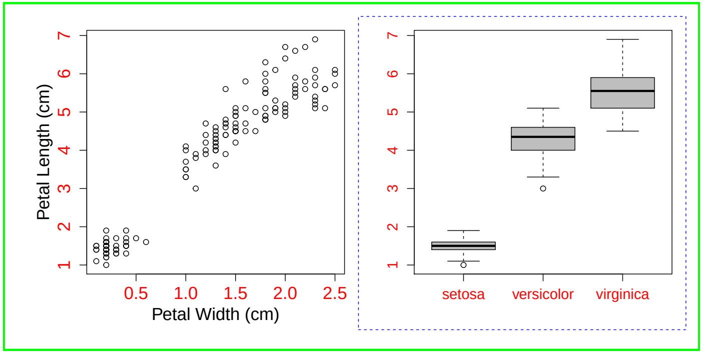


Note: unicode characters are nice but source of problems, it may appear on the screen but not necessarily in pdf if the postscript font is missing. How to get the font and how to make it work depends on the OS.

Plot margins

Global parameters can also be used to change individual plot margins (blue) and outer margins (green):

```r
oldpar <- 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 = 1.25, col.axis = "red")
box('figure', col = 'blue', lty = "dashed")
box('outer', col = 'green', lwd = 5)
```

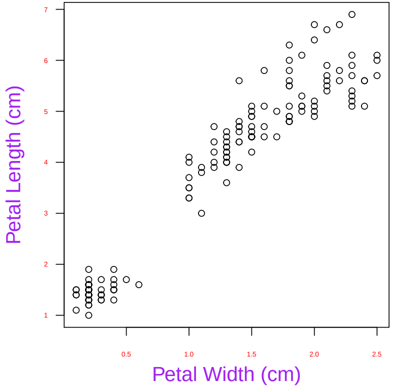


```r
par(oldpar)
```

Change axes

You can change the size, colour and orientation of the axis labels and text easily:

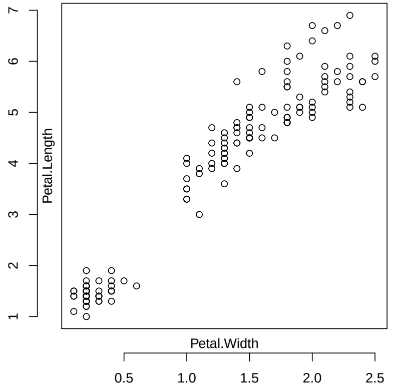
```
plot(Petal.Length ~ Petal.Width, data = iris,
     xlab = "Petal Width (cm)", ylab = "Petal Length (cm)",
     cex.lab = 1.5, cex.axis = 0.5, col.axis = "red", col.lab = "purple", las = 1)
```



Change axes

You can change the position of the axes:

```r
oldpar <- par(mgp = c(0.4, 2, 1.5))
plot(Petal.Length ~ Petal.Width, data = iris)
```



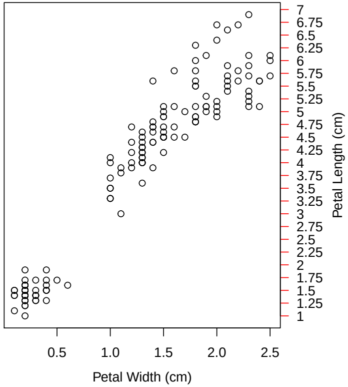
```r
par(oldpar)
```

Note: check `?par` for a huge list of the things you can easily change!

Change axes

You can change the axes themselves:

```
oldpar <- par(mar = c(4, 4, 1, 4))
plot(Petal.Length ~ Petal.Width, data = iris, axes = FALSE,
     xlab = "Petal Width (cm)", ylab = "")
axis(side = 1)
axis(side = 4, at = seq(1, 7, 0.25), labels =  seq(1, 7, 0.25), col.ticks = "red", las = 1)
mtext("Petal Length (cm)", side = 4, line = 3)
box()
```



```
par(oldpar)
```

Plotting in R

Traditional graphics: exporting

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

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

Plotting in **R**

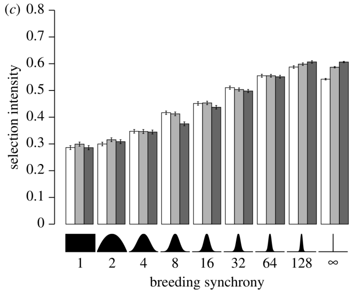
Challenge

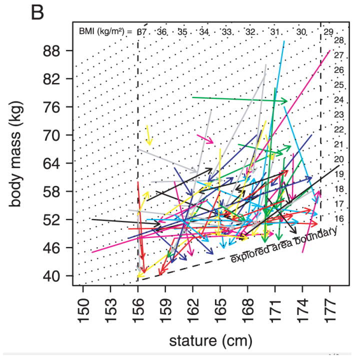
Run the examples of the following traditional plot functions:

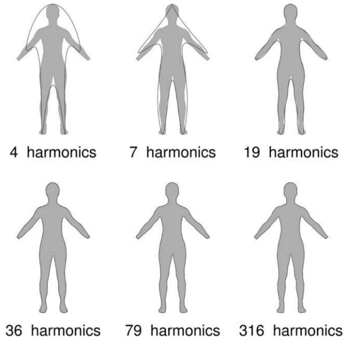
- `plot.ecdf()`
- `curve()`
- `pie()`
- `cdplot()`
- `image()`
- `contour()`
- `persp()`
- `dotchart()`
- `mosaicplot()`
- `stars()`
- `matplot()`
- `pairs()`
- `sunflowerplot()`

Note: there are a few other ones, but they do not seem very interesting. . .

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







Plotting in **R**

Plotting in **R**

ggplot: introduction

ggplot (defunct) and ggplot2 were both created by Hadley Wickham (now chief Scientist at Rstudio) during his PhD: `http://had.co.nz/thesis/practical-tools-hadley-wickham.pdf`

ggplot: introduction

ggplot (defunct) and ggplot2 were both created by Hadley Wickham (now chief Scientist at Rstudio) during his PhD: http://had.co.nz/thesis/practical-tools-hadley-wickham.pdf

The idea was to create a grammar of graphics for **R**.

It is inspired from the seminal Leland Wilkson's book **The Grammar of Graphics**: *"This book [...] presents a unique foundation for producing almost every quantitative graphic found in scientific journals, newspapers, statistical packages, and data visualization systems. This foundation was designed for a distributed computing environment (Internet, Intranet, client-server), with special attention given to conserving computer code and system resources."*

ggplot: introduction

`ggplot` (defunct) and `ggplot2` were both created by Hadley Wickham (now chief Scientist at Rstudio) during his PhD: `http://had.co.nz/thesis/practical-tools-hadley-wickham.pdf`

The idea was to create a grammar of graphics for **R**.

It is inspired from the seminal Leland Wilkson's book **The Grammar of Graphics**: *"This book [...] presents a unique foundation for producing almost every quantitative graphic found in scientific journals, newspapers, statistical packages, and data visualization systems. This foundation was designed for a distributed computing environment (Internet, Intranet, client-server), with special attention given to conserving computer code and system resources."*

In plain english, that means that the graphics are build by considering successive layers.

The originality is that the user directly handles different functions corresponding to each layer.

Other graphics systems do use layers too, but only behind the curtain.

The conceptual unit is thus shifted from the type of plot to the type of layer!

ggplot: introduction

ggplot (defunct) and ggplot2 were both created by Hadley Wickham (now chief Scientist at Rstudio) during his PhD: `http://had.co.nz/thesis/practical-tools-hadley-wickham.pdf`

The idea was to create a grammar of graphics for **R**.

It is inspired from the seminal Leland Wilkson's book **The Grammar of Graphics**: *"This book [...] presents a unique foundation for producing almost every quantitative graphic found in scientific journals, newspapers, statistical packages, and data visualization systems. This foundation was designed for a distributed computing environment (Internet, Intranet, client-server), with special attention given to conserving computer code and system resources."*

In plain english, that means that the graphics are build by considering successive layers.

The originality is that the user directly handles different functions corresponding to each layer.

Other graphics systems do use layers too, but only behind the curtain.

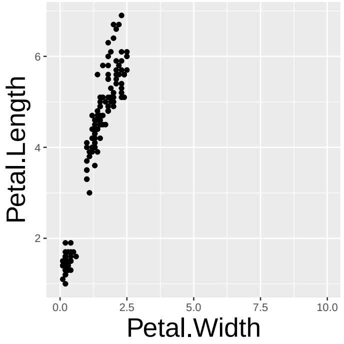
The conceptual unit is thus shifted from the type of plot to the type of layer!

It has pros and cons.

ggplot: introduction

Overall structure of a ggplot call:

```
ggplot(data = iris, mapping = aes(x = Petal.Width, y = Petal.Length)) + ## we specify the source of data and map the variables
  geom_point() + ## we use a function that specifies the type of plot (e.g. scatterplot, histogram). A geom_* function is always required!
  scale_x_continuous(limits = c(0, 10)) + ## we can fiddle with the scale_*_* functions to adjust the axes, colouration etc.
  theme(axis.title = element_text(size = 22)) # we can use one of the theme_* functions to change things like legends, font size etc.
```



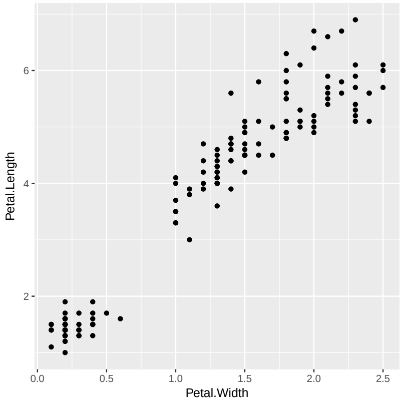
Note: to better understand, execute the code progressively adding one command at a time!

Plotting in R

ggplot: scatter plots

We add points on the empty plot:

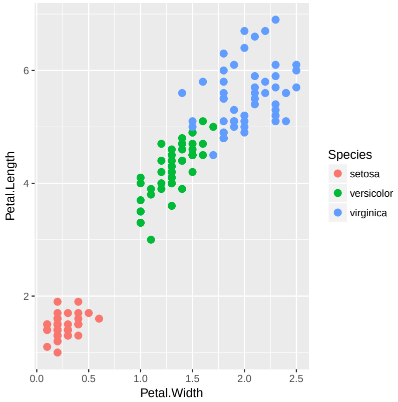
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point()
```



ggplot: scatter plots

You can change point shapes `shape` (= `pch` in traditional graphics) and `colour` (= `col` in traditional graphics):

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

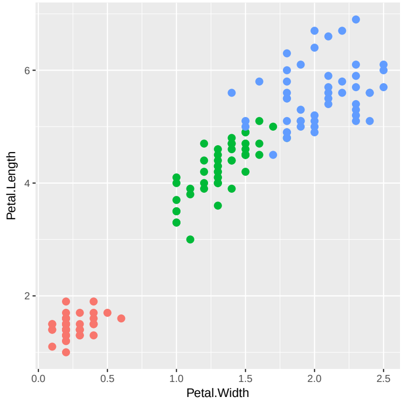


Note: everytime we want to create a connection between the data and some elements of the plot, we use the same function: `aes()`!

ggplot: scatter plots

In ggplot the legend is included by default but you can remove it:

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

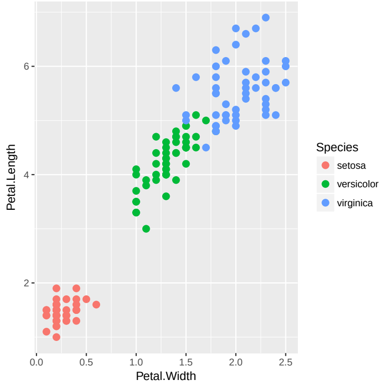


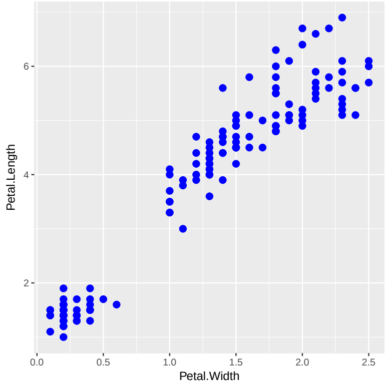
ggplot: scatter plots

The aesthetic function in ggplot is a powerful tool for changing plot aesthetics.
If you specify an aesthetic argument inside aes(), it will give each 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:

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

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



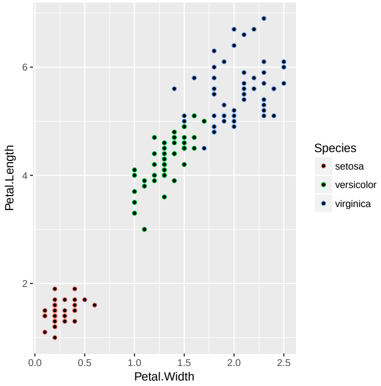


Note: we have used aes() to apply colours but it can be used to make other changes (see later).

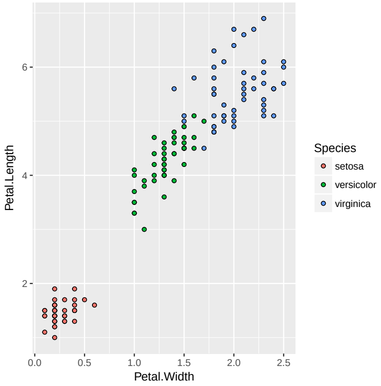
ggplot: scatter plots

Again, for many elements you can set both an outline `colour` (= `col` for traditional graphics) and background `fill` colour (= `bg` for traditional graphics):

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



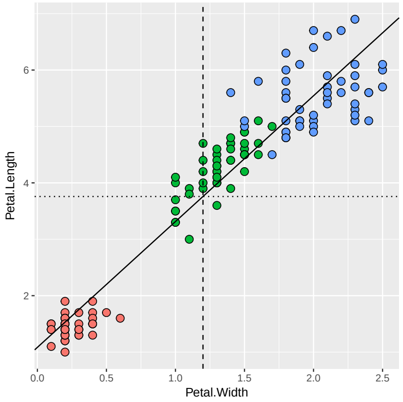
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), colour = "black", shape = 21)
```



ggplot: scatter plots

You can also add lines to the plot:

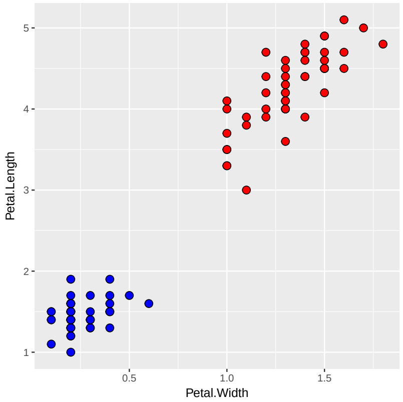
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), shape = 21, size = 3) +
  theme(legend.position = "none") +
  geom_hline(yintercept = mean(iris$Petal.Length), lty = 3) +
  geom_vline(xintercept = mean(iris$Petal.Width), lty = 2) +
  geom_abline(intercept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



ggplot: scatter plots

You can add extra points to the plot by using two `geom_point` layers:

```
versicolor <- subset(iris, Species == "versicolor")
setosa     <- subset(iris, Species == "setosa")
ggplot() +
  geom_point(data = versicolor, aes(x = Petal.Width, y = Petal.Length), fill = "red", shape = 21, size = 3) +
  geom_point(data = setosa, aes(x = Petal.Width, y = Petal.Length), fill = "blue", shape = 21, size = 3)
```

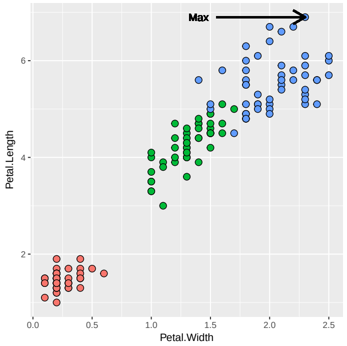


Note: Unlike traditional plots, `ggplot` will automatically adjust the axis limits.

Traditional graphics: scatter plots

You can add text and arrows:

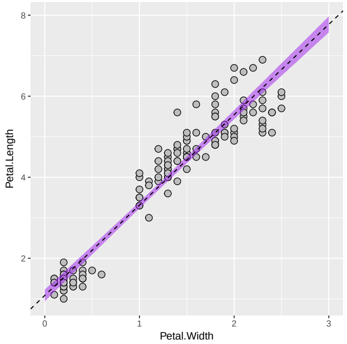
```
max.value <- iris[iris$Petal.Length == max(iris$Petal.Length), ]
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), shape = 21, size = 3) +
  theme(legend.position = "none") +
  geom_segment(aes(x = max.value$Petal.Width - 0.75,
                   xend = max.value$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 = newdat, aes(x = Petal.Width, ymin = pred[, 2], ymax = pred[, 3]), fill = "purple", alpha = 0.5) +
  geom_abline(intercept = coef(test_mod)[1], slope = coef(test_mod)[2], lty = 2)
```



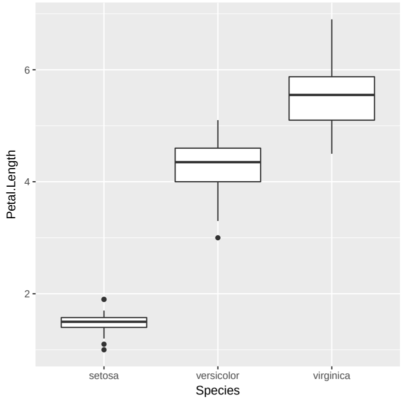
Note: because we are building the plot with two different datasets (i.e. `iris` & `newdat`), we specify the data separately in each line of the ggplot code.

Plotting in **R**

ggplot: box plots

In ggplot, use `geom_boxplot()` to draw a box plot:

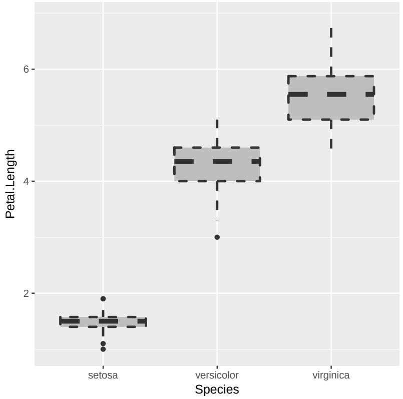
```r
ggplot(data = iris, aes(x = Species, y = Petal.Length)) +
  geom_boxplot()
```



ggplot: box plots

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

```r
ggplot(data = iris, aes(x = Species, y = Petal.Length)) +
  geom_boxplot(fill = "grey", size = 1, lty = 2)
```

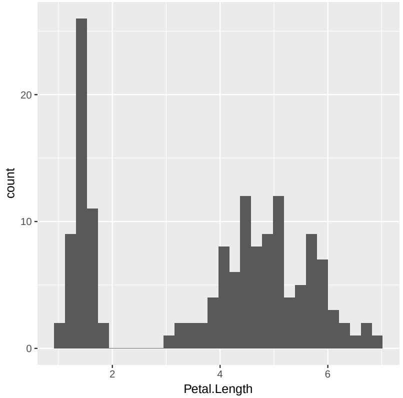


Plotting in R

ggplot: histograms

In ggplot, use `geom_histogram()` to draw an histogram:

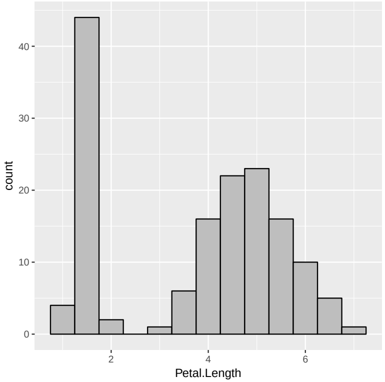
```
ggplot(data = iris, aes(x = Petal.Length)) +
  geom_histogram()
## `stat_bin()` using `bins = 30`.  Pick better value with
## `binwidth`.
```



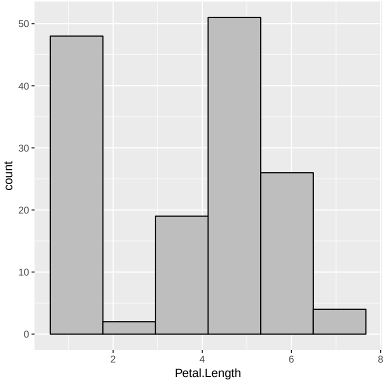
ggplot: histograms

You can change the width of bins with `binwidth` or their number with `bins`:

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



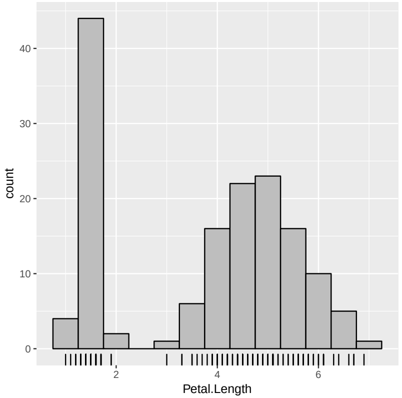
```r
ggplot(data = iris, aes(x = Petal.Length)) +
  geom_histogram(bins = 6, colour = "black", fill = "grey")
```



ggplot: histograms

Add a rug below the histogram with `geom_rug`:

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

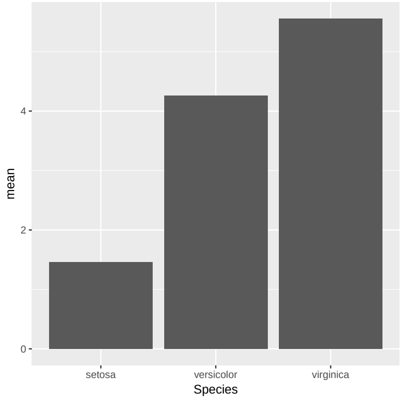


Plotting in R

ggplot: bar plots

In ggplot, use `geom_col()` to draw a bar plot:

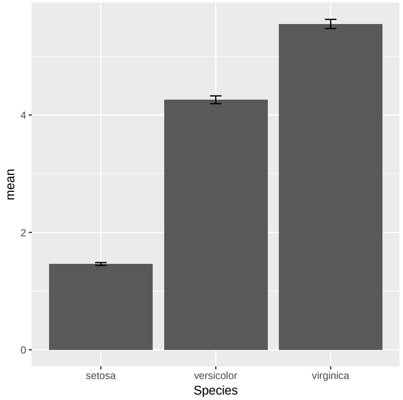
```
ggplot(data = spp_means, aes(x = Species, y = mean)) +
  geom_col()
```



ggplot: bar plots

Adding error bars in ggplot is much easier than in traditional graphics:

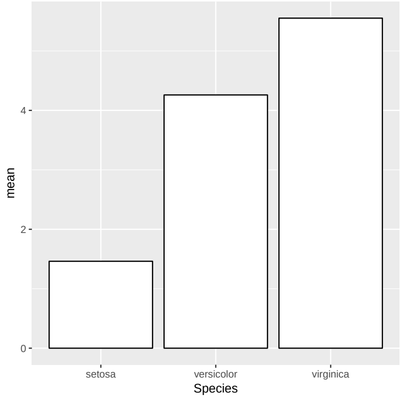
```r
ggplot(data = spp_means, aes(x = Species, y = mean)) +
  geom_col() +
  geom_errorbar(aes(ymin = mean - SE, ymax = mean + SE), width = 0.1)
```



ggplot: bar plots

As before, modifications are straigthforward:

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



Plotting in **R**

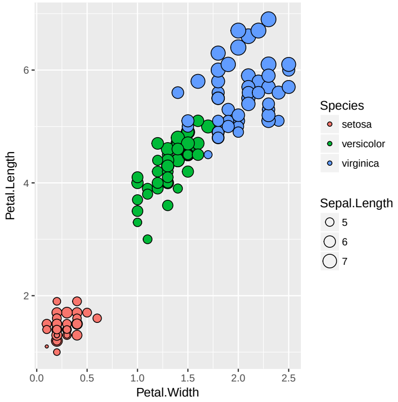
Introduction to aesthetics

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

Using the aesthetic function more

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

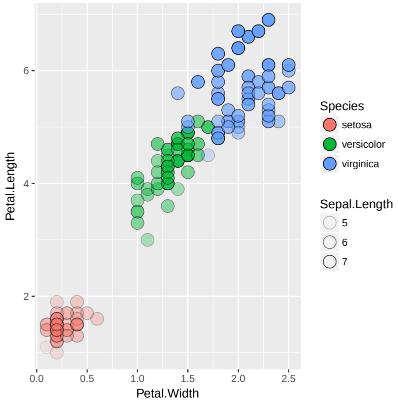
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species, size = Sepal.Length), shape = 21)
```



Using the aesthetic function more

Here we change the transparency:

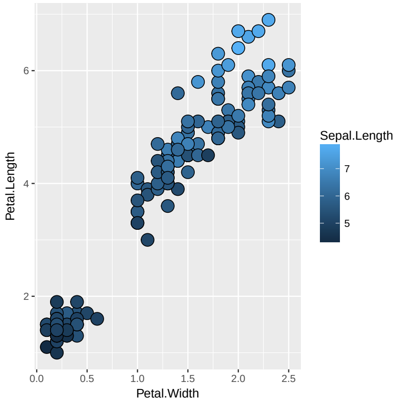
```r
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species, alpha = Sepal.Length), shape = 21, size = 5)
```



Using the aesthetic argument more

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

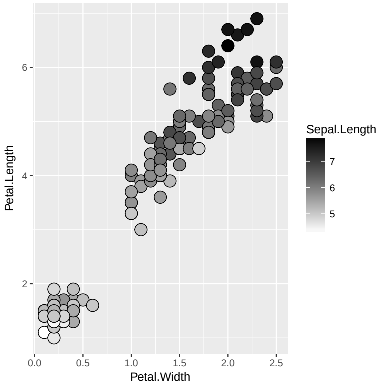
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Sepal.Length), shape = 21, size = 5)
```



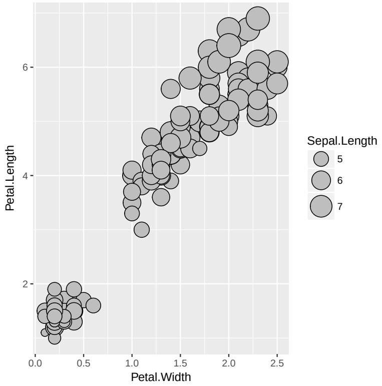
Using the aesthetic argument more

We can adjust the way the aesthetics are applied:

```r
ggplot(data = 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")
```



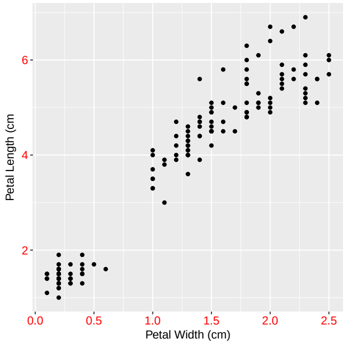
```r
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(size = Sepal.Length), shape = 21, fill = "grey") +
  scale_size_continuous(range = c(3, 10))
```



Change text

You can change size and colour of axis text easily:

```r
ggplot(data = 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))
```

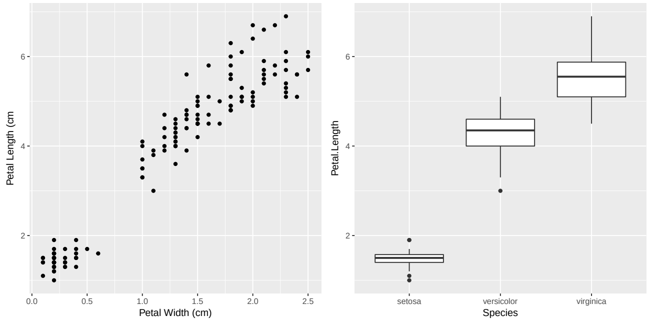


Note: the text size uses different measurement units than traditional graphics.

Combining plots

Combining plots is less straightforward in ggplot...
You need to use an additional package:

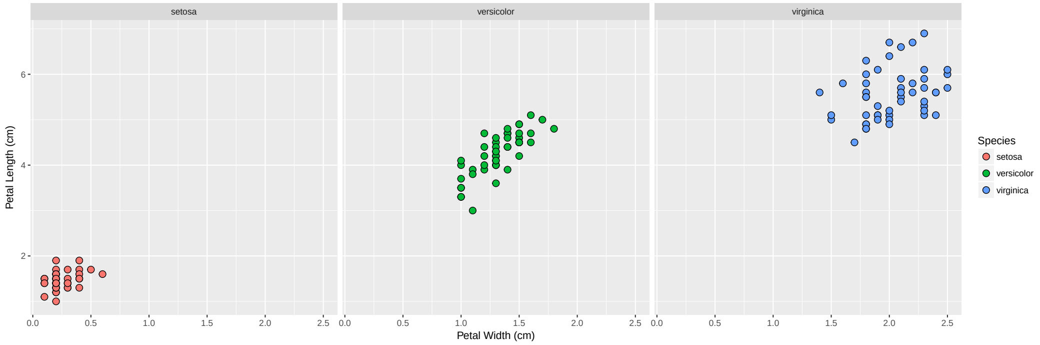
```
scatter <- ggplot(iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() + xlab("Petal Width (cm)") + ylab("Petal Length (cm")

box     <- ggplot(iris, aes(x = Species, y = Petal.Length)) +
  geom_boxplot()

library(gridExtra)
grid.arrange(scatter, box, nrow = 1)
```



Faceting

Although combining multiple plots is cumbersome, there is an inbuilt function to create facets:

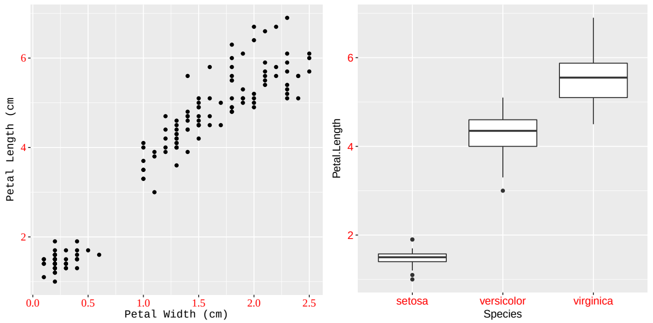
```
ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point(aes(fill = Species), shape = 21, colour = "black", size = 3) +
  xlab("Petal Width (cm)") +
  ylab("Petal Length (cm)") +
  facet_wrap(~ Species)
```



Modifying typefaces

Unlike traditional graphics, in ggplot you can easily change the typeface of individual elements in theme:

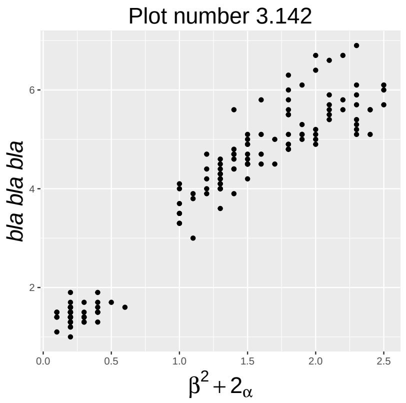
```r
scatter <- ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  xlab("Petal Width (cm)") +
  ylab("Petal Length (cm") +
  theme(axis.text = element_text(size = 12, colour = "red", family = "serif"),
        axis.title = element_text(size = 12, family = "mono"))

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"))

grid.arrange(scatter, box, nrow = 1)
```



Special characters

You can use weird characters:

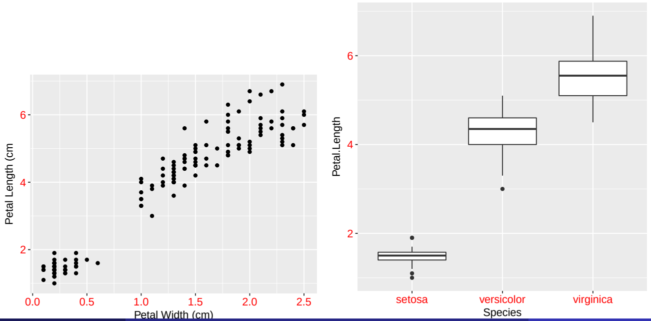
```
foo <- round(pi, 3)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  labs(x = expression(beta^2 + 2[alpha]), y = expression(paste(italic("bla bla bla"))), title = paste("Plot number", foo)) +
  #annotate("text", x = c(0.4, 0.5), y = c(4, 4), label = c("\u2640", "\u2642"), size = 12)+ # require font with unicode installed (try it!)
  theme(plot.title = element_text(hjust = 0.5, size = 20),
        axis.title = element_text(size = 20))
```



Plot margins

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

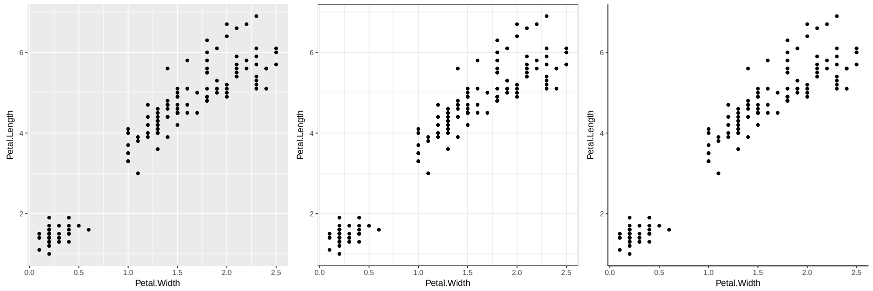
```
scatter <- ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  xlab("Petal Width (cm)") +
  ylab("Petal Length (cm") +
  theme(axis.text = element_text(size = 12, colour = "red"),
        axis.title = element_text(size = 12),
        plot.margin = unit(c(30, 4, 1, 1), "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))

grid.arrange(scatter, box, nrow = 1)
```



Preset themes

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

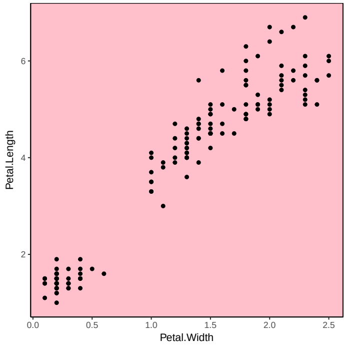
```
grey <- ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  theme_grey()

bw <- ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  theme_bw()

classic <- ggplot(data = iris, aes(x = Petal.Width, y = Petal.Length)) +
  geom_point() +
  theme_classic()

grid.arrange(grey, bw, classic, nrow = 1)
```



Preset themes

You can create your own theme once for all!

```
theme_pink <- theme_classic() %+replace% theme(panel.background = element_rect(fill = "pink"))

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

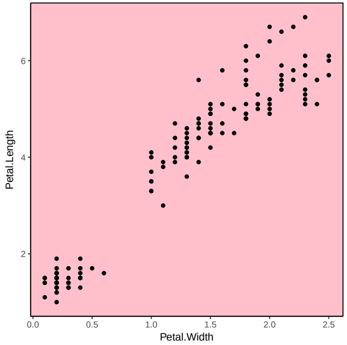


Note: check `?theme` for the list of options that can be changed.

Preset themes

You can set a theme for all plots with `theme_set`

```
theme_old  <- theme_set(theme_pink)

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



```
theme_set(theme_old) ## restore original theme
```

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**

There are other geom out there!

There are a few other plot types in `ggplot` that are worth looking at (check the cheatsheet!):

- `geom_violin()`
- `geom_area()`
- `geom_smooth()`
- `geom_density()`

There are other geom out there!

There are a few other plot types in ggplot that are worth looking at (check the cheatsheet!):

- geom_violin()
- geom_area()
- geom_smooth()
- geom_density()

But the true force of having developed something as modular as ggplot is that there are many more packages each week providing ggplot extensions!

A few random examples of ggplot extensions:

- ggExtra (for marginal distribution)
- ggthemes (for extra themes, including theme_excel()!! :-/)
- ggmap (for maps)
- ggrepel (for adding labels to plots)
- ggalt (for creating cluster plots)
- cowplot (for creating nested figures)

Plotting in **R**

Which plotting tool should you use?

Traditional graphics:

- no new packages required
- arguments often differ between functions
- help files useful
- easier for doing something simple
- more difficult for doing something complex

Which plotting tool should you use?

Traditional graphics:

- no new packages required
- arguments often differ between functions
- help files useful
- easier for doing something simple
- more difficult for doing something complex

`ggplot`:

- requires multiple packages for best results
- uniform grammar
- help files often useless (due to modularity)
- more difficult for doing something simple
- easier for doing something complex
- faster for large datasets
- more accessible than lattice (or grid itself)

Which plotting tool should you use?

Traditional graphics:

- no new packages required
- arguments often differ between functions
- help files useful
- easier for doing something simple
- more difficult for doing something complex

ggplot:

- requires multiple packages for best results
- uniform grammar
- help files often useless (due to modularity)
- more difficult for doing something simple
- easier for doing something complex
- faster for large datasets
- more accessible than lattice (or grid itself)

→ It may be useful to know both!!