

# **Notes on using R**

Pedro J. Aphalo

Git: tag ‘none’, committed with hash ‘none’ on ‘none’



# Contents

<b>1 Plots with <code>ggplot</code></b>	<b>7</b>
1.1 Packages used in this chapter . . . . .	7
1.2 Introduction . . . . .	7
1.3 Bases of plotting with ‘ <code>ggplot2</code> ’: points and lines . . . . .	7
1.4 Adding fitted curves, including splines . . . . .	19
1.5 Adding statistical “summaries” . . . . .	21
1.6 Plotting functions . . . . .	26
1.7 Plotting text and expressions . . . . .	29
1.8 Using facets . . . . .	31
1.9 Scales . . . . .	41
1.10 Circular plots . . . . .	43
1.11 Bar plots . . . . .	44
1.12 Pie charts vs. bar plots example . . . . .	44
1.13 Frequencies and densities . . . . .	46
1.13.1 Marginal rug plots . . . . .	46
1.13.2 Histograms . . . . .	46
1.13.3 Density plots . . . . .	48
1.13.4 Box and whiskers plots . . . . .	50
1.13.5 Violin plots . . . . .	51
1.14 Adding annotations . . . . .	52
1.15 Themes . . . . .	53
1.15.1 Predefined themes . . . . .	53
1.15.2 Tweaking a theme . . . . .	53
1.15.3 Defining a new theme . . . . .	53
1.16 Advanced topics . . . . .	53
1.17 Using <code>plotmath</code> expressions . . . . .	53
1.18 Scales in detail . . . . .	55
1.19 Generating output files . . . . .	55
1.19.1 Using <code>LATEX</code> instead of <code>plotmath</code> . . . . .	56
1.19.2 Fonts . . . . .	56
1.20 Examples . . . . .	56
1.20.1 Anscombe’s regression examples . . . . .	56
1.20.2 Volcano plots . . . . .	58

*Contents*

---

<b>2 Plots with ggplot, ggrepel and ggpmisc</b>	<b>59</b>
2.1 Packages used in this chapter . . . . .	59
2.2 ggpmisc . . . . .	59
2.2.1 Plotting time-series . . . . .	59
2.2.2 New stats . . . . .	60
2.2.3 Peaks and valleys . . . . .	61
2.2.4 Equations as labels in plots . . . . .	63
2.2.5 Learning and/or debugging . . . . .	71
2.3 ggrepel . . . . .	72
2.3.1 New geoms . . . . .	72
2.4 Examples . . . . .	75
2.4.1 Anscombe's example revisited . . . . .	75
<b>3 Plotting maps with ggmap</b>	<b>77</b>
3.1 Plotting data onto maps . . . . .	77
<b>4 Further reading about R</b>	<b>91</b>
4.1 Introductory texts . . . . .	91
4.2 Texts on specific aspects . . . . .	91
4.3 Advanced texts . . . . .	91

# Preface

This series of Notes cover different aspects of the use of R. They are meant to be used as a complement to a course or book, as explanations are short and terse. We do not discuss here statistics, just R as a tool and language for data manipulation and display. The idea is a bit like how children learn a language: they work-out what the rules are simply by listening to people speak. I do give some explanations and comments, but the idea of this notes is mainly for you to use the numerous examples to find-out by yourself the overall patterns and coding philosophy behind the R language.

This is work-in-progress. I will appreciate suggestions for further examples, notification of errors and unclear things and any bigger contributions. Many of the examples here have been collected from diverse sources over many years and because of this not all sources are acknowledged. If you recognize any example as yours or someone else's please let me know so that I can add a proper acknowledgement.



# 1 Plots with ‘ggplot2’

## 1.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(ggplot2)
```

We set a font larger size than the default

```
theme_set(theme_grey(16))
```

## 1.2 Introduction

Being R extensible, in addition to the built-in plotting functions, there are several alternatives provided by packages. Of the general purpose ones, the most extensively used are `Lattice` and ‘`ggplot2`’. There are additional packages that add extra functionality to these packages.

In the examples in this chapter we use ‘`ggplot2`’. In later chapters we use ‘`ggplot2`’ together with ‘`ggmap`’, ‘`ggtern`’, ‘`ggrepel`’, and ‘`ggpmisc`’. Here we start with an introduction to the ‘grammar of graphics’ and ‘`ggplot2`’. There is ample literature on the use of ‘`ggplot2`’, starting with very good reference documentation at <http://ggplot2.org/>. The book ‘R Graphics Cookbook’ (Chang 2013) is very useful and should be always near you, when using the package, as it contains many worked out examples. Much of the literature available at this time is for older versions of ‘`ggplot2`’ but we here describe version 2.0.0, and highlight the most important incompatibilities that need to be taken into account when using versions of ‘`ggplot2`’ earlier than 2.0.0. There is little well-organized literature on packages extending ‘`ggplot2`’ so we will describe some of them in later chapters.

## 1.3 Bases of plotting with ‘ggplot2’: points and lines

What separates ‘`ggplot2`’ from base-R and trellis/lattice plotting functions is the use of a grammar of graphics (the reason behind ‘`gg`’ in the name of the package).

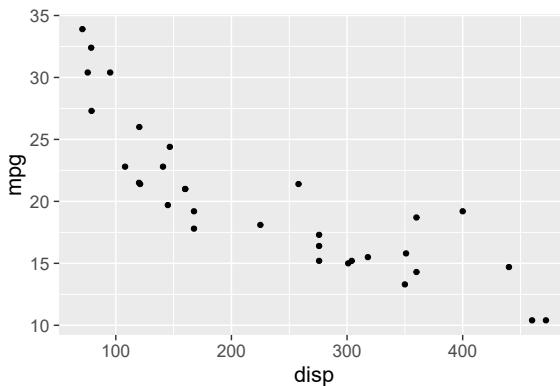
## 1 Plots with ggplot

---

What is meant by grammar in this case is that plots are assembled piece by piece from different ‘nouns’ and ‘verbs’. Instead of using a single function with many arguments, plots are assembled by combining different elements with operators `+` and `%+%`. Furthermore, the constructions is mostly semantic-based and to a large extent how the plot looks when is printed, displayed or exported to a bitmap or vector graphics file is controlled by themes.

The grammar of graphics is based on aesthetics (`aes`) as for example color, geometric elements `geom_....` such as lines, and points, statistics `stat_....`, scales `scale_....`, labels `labs`, and themes `theme_.....`. Plots are assembled from these elements, we start with a plot with two aesthetics, and one geometry. In the examples that follow we will use the `mtcars` data set included in R. To learn more about this data set, `help("mtcars")` at the R command prompt.

```
ggplot(data = mtcars, aes(x = disp, y = mpg)) +  
  geom_point()
```

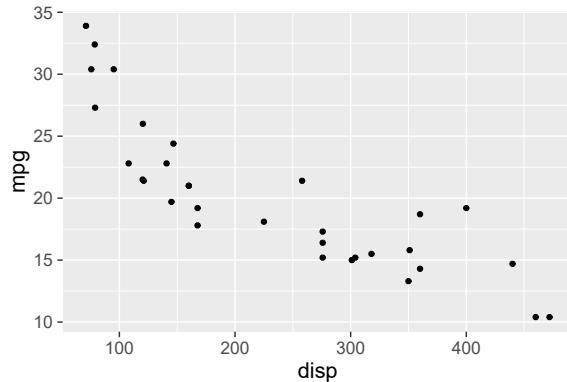


I could have written the code above passing the arguments by position but this makes the code more difficult to read and less tolerant to possible changes to the definitions of the functions used in future version of package ‘ggplot2’. It is not recommended to use this terse style in scripts or package coding. However, it can be used at the command prompt, although it can lead to mistakes when the sanity of results is difficult to quickly assess at first sight.

```
ggplot(mtcars, aes(disp, mpg)) +  
  geom_point()
```

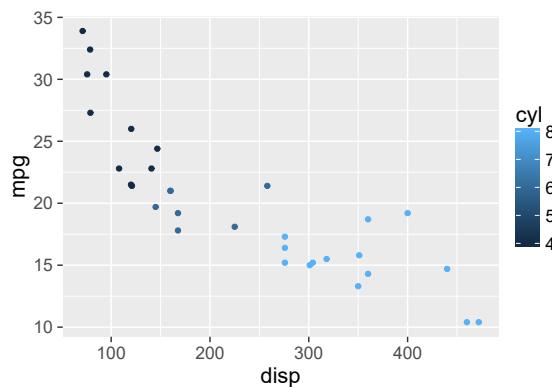
### 1.3 Bases of plotting with ‘ggplot2’: points and lines

---



Data variables can be ‘mapped’ to *aesthetics*, variables can be either continuous (numeric) or discrete (categorical, factor). Variable `cyl` is encoded in the `mtcars` data frame as numeric values. Even though only three values are present, a continuous color scale is used by default.

```
ggplot(data = mtcars,  
       aes(x = disp, y = mpg, colour = cyl)) +  
  geom_point()
```

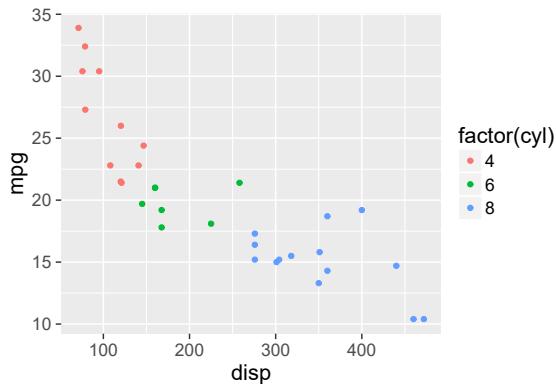


Some scales exist in two ‘flavours’, one suitable for continuous variables and another for discrete variables. For now we will just use the default scales, but later on we will see how to alter them. We can convert `cyl` into a factor ‘on-the-fly’ to force the use of a discrete color scale.

```
ggplot(data = mtcars,  
       aes(x = disp, y = mpg, colour = factor(cyl))) +  
  geom_point()
```

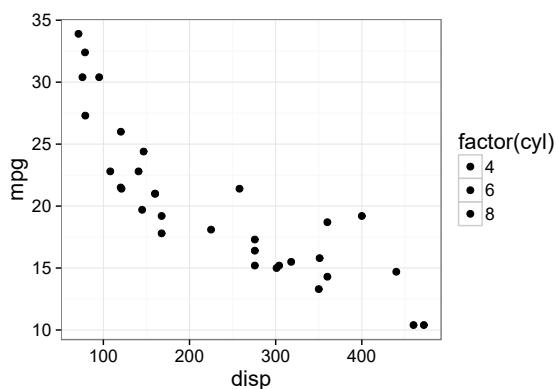
## 1 Plots with ggplot

---



Using an aesthetic, involves mapping of values in the data to aesthetic values such as colours. The mapping is defined by means of scales. If we now consider the `colour` aesthetic in the previous statement, a default discrete colour scale was used. In this case if we would like different colours used for the three values, but still have them selected automatically, we can select a different colour palette:

```
ggplot(data = mtcars,  
       aes(x = disp, y = mpg, fill = factor(cyl))) +  
  geom_point(size = rel(2)) +  
  scale_color_brewer(type = "div", palette = 8) +  
  theme_bw(16)
```

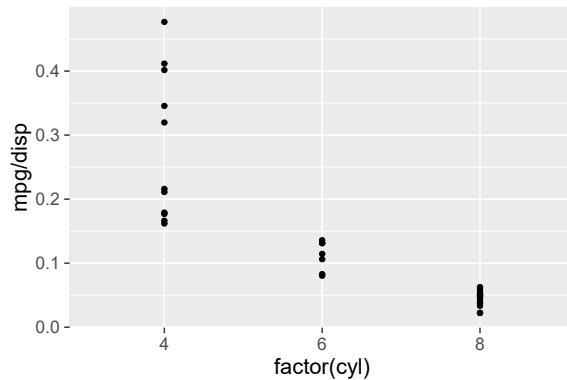


Data assigned to an aesthetic can be the ‘result of a computation’. In other words, the values to be plotted do not need to be stored in the data frame passed as argument to `data`, the first formal parameter of `ggplot()`

```
ggplot(data = mtcars,  
       aes(x = factor(cyl), y = mpg / disp)) +  
  geom_point()
```

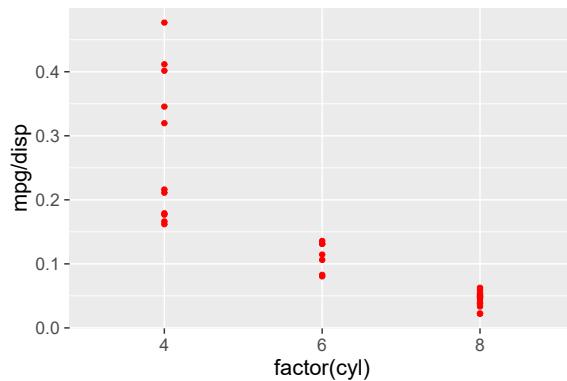
### 1.3 Bases of plotting with ‘ggplot2’: points and lines

---



Within `aes()` the aesthetics are interpreted as being a function of the values in the data. If given outside `aes()` they are interpreted as constant values, which apply to one geom if given within the call to `geom_xxx` but outside `aes()`. The aesthetics and data given as `ggplot()`’s arguments become the defaults for all the geoms, but geoms also accept aesthetics and data as arguments, which when supplied as arguments override the whole-plot defaults. In the example below, we override the default colour of the points.

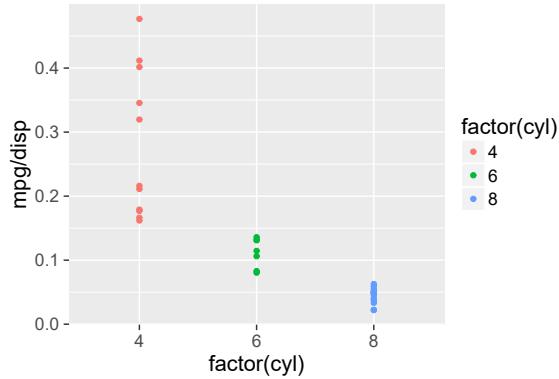
```
ggplot(data = mtcars,  
       aes(x = factor(cyl), y = mpg / disp)) +  
  geom_point(color = "red")
```



```
ggplot(data = mtcars,  
       aes(x = factor(cyl), y = mpg / disp, color = factor(cyl))) +  
  geom_point()
```

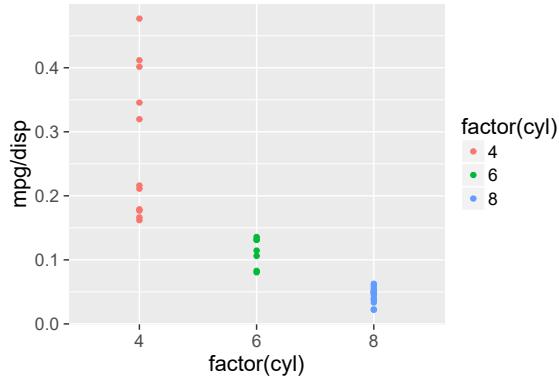
## 1 Plots with ggplot

---



The same plot can be also obtained by adding all pieces one by one (although seldom used except when defining new functions).

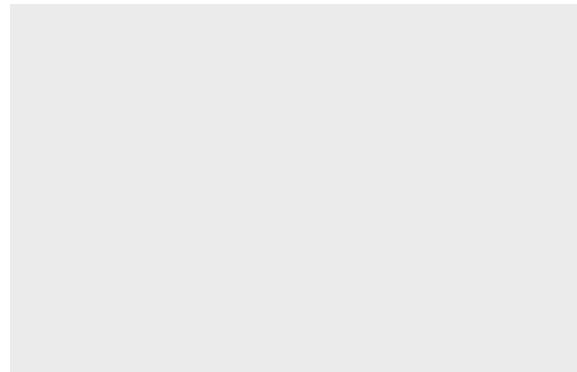
```
ggplot() +  
  aes(x = factor(cyl), y = mpg / disp,  
      colour = factor(cyl)) +  
  geom_point(data = mtcars)
```



The code in the next chunk is also valid, it returns a blank plot.

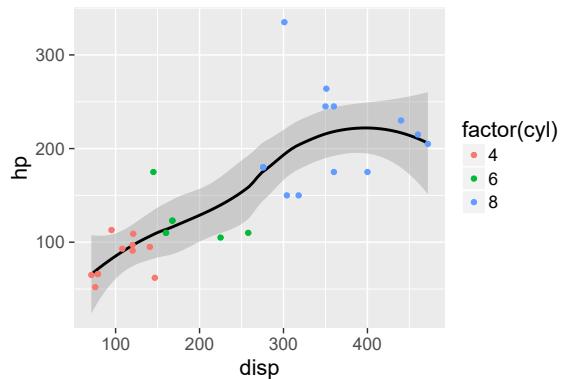
```
ggplot()
```

### 1.3 Bases of plotting with ‘ggplot2’: points and lines



In the next example we override the `color` aesthetic in `geom_smooth`<sup>1</sup>, causing all the data to be fitted together. The data points are labelled according to `cyl` but the smooth line is calculated jointly for all values of `cyl`. That we use `code = "black"` is not important, what is important is that we use a constant to override an aesthetic mapping set globally for the whole plot.

```
ggplot(data = mtcars, aes(x=disp, y=hp, colour=factor(cyl))) +  
  geom_smooth(colour="black") +  
  geom_point()
```



In the example above the order in which the two geoms are added is important, as this determines the position of their layers. In the example above the points are plotted on top of the smoother, while in the next example `geom_smooth` is plotted on top of the points.

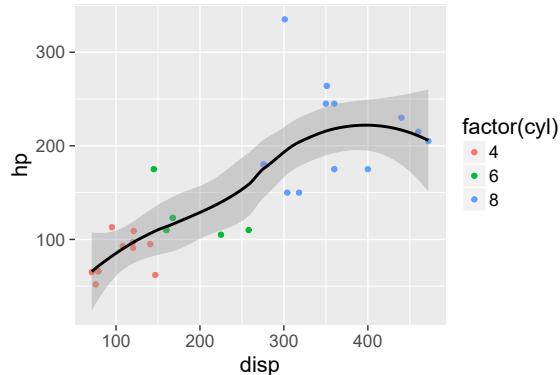
---

<sup>1</sup>Smoothing and curve fitted is discussed in more detail in section ??.

## 1 Plots with ggplot

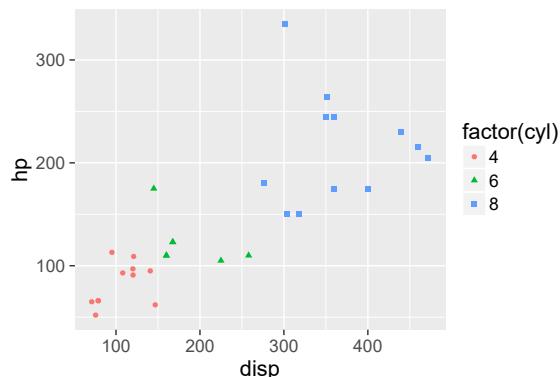
---

```
ggplot(data = mtcars, aes(x=disp, y=hp, colour=factor(cyl))) +  
  geom_point() +  
  geom_smooth(colour="black")
```



We can assign the same variable to more than one aesthetic, and the combined key will be produced automatically.

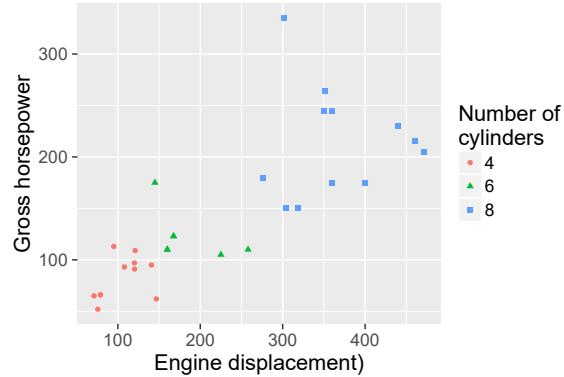
```
ggplot(data = mtcars,  
       aes(x=disp, y=hp, colour=factor(cyl),  
            shape=factor(cyl))) +  
  geom_point()
```



We can change the labels for the different aesthetics, and give a title (\n means ‘new line’ and can be used to continue a label in the next line). In this case, if two aesthetics are linked to the same variable, the labels supplied should be identical, otherwise two separate keys will be produced.

### 1.3 Bases of plotting with ‘ggplot2’: points and lines

```
ggplot(data = mtcars,
       aes(x=disp, y=hp, colour=factor(cyl),
           shape=factor(cyl))) +
  geom_point() +
  labs(x="Engine displacement",
       y="Gross horsepower",
       colour="Number of\ncylinders",
       shape="Number of\ncylinders")
```



We can assign a ggplot object or a part of it to a variable, and then assemble a new plot from the different pieces.

```
myplot <- ggplot(data = mtcars,
                  aes(x=disp, y=mpg,
                      colour=factor(cyl)) +
  geom_point()

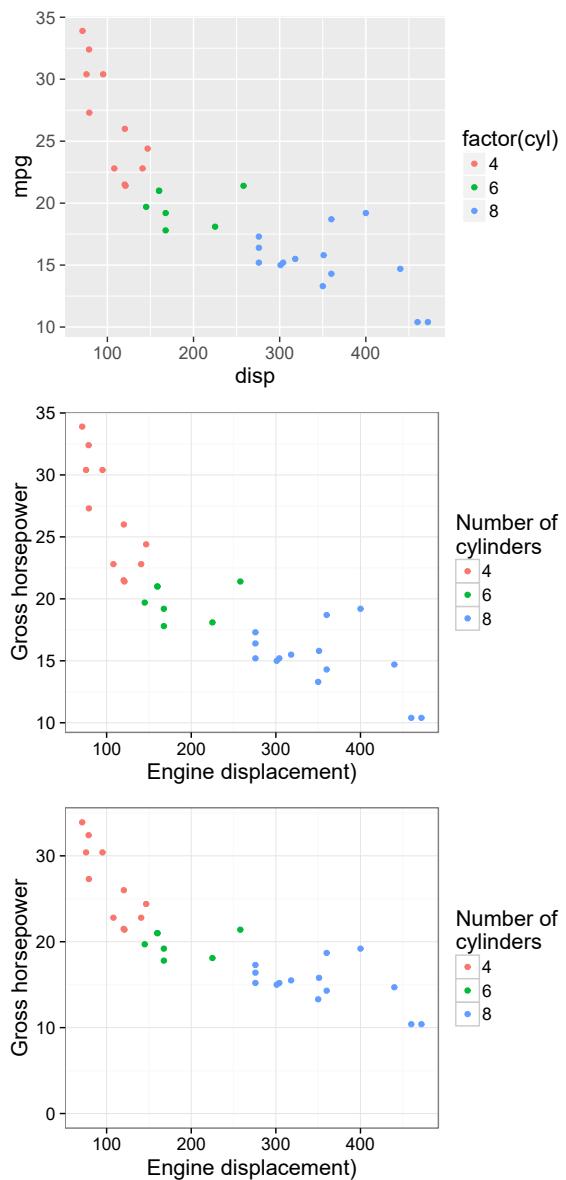
mylabs <- labs(x="Engine displacement",
               y="Gross horsepower",
               colour="Number of\ncylinders",
               shape="Number of\ncylinders")
```

And now we can assemble them into plots.

```
myplot
myplot + mylabs + theme_bw(16)
myplot + mylabs + theme_bw(16) + ylim(0, NA)
```

## 1 Plots with ggplot

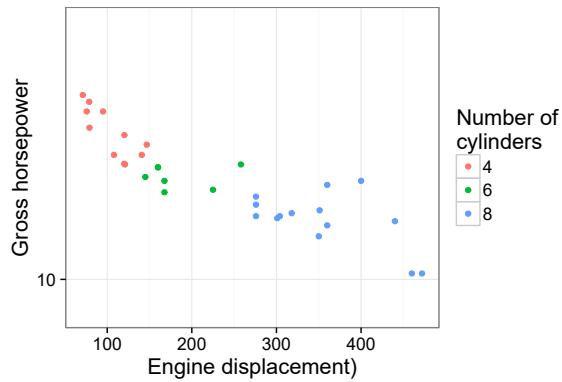
---



We can also save intermediate results.

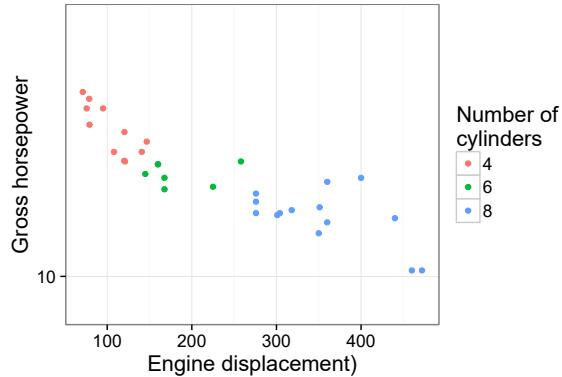
```
mylogplot <- myplot + scale_y_log10(limits=c(8,55))  
mylogplot + mylabs + theme_bw(16)
```

### 1.3 Bases of plotting with ‘ggplot2’: points and lines



If the pieces to put together do not include a “ggplot” object, we can put them into a “list” object.

```
myparts <- list(mylabs, theme_bw(16))
mylogplot + myparts
```

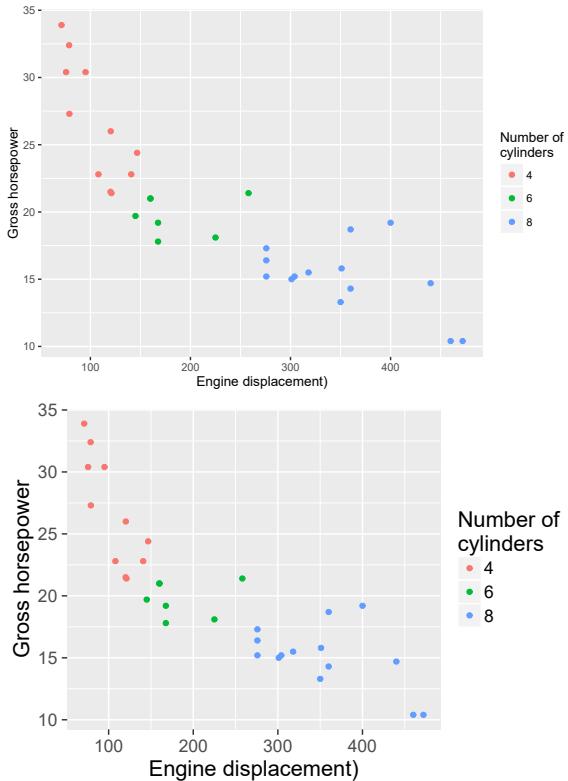


There are a few predefined themes in package ‘ggplot2’ and additional ones in other packages such as ‘cowplot’, even the default `theme_grey` can come in handy because the first parameter to themes is the point size used as reference to calculate all other font sizes. You can see in the two examples below, that the size of all text elements changes proportionally.

```
myplot + mylabs + theme_grey(10)
myplot + mylabs + theme_grey(16)
```

## 1 Plots with ggplot

---

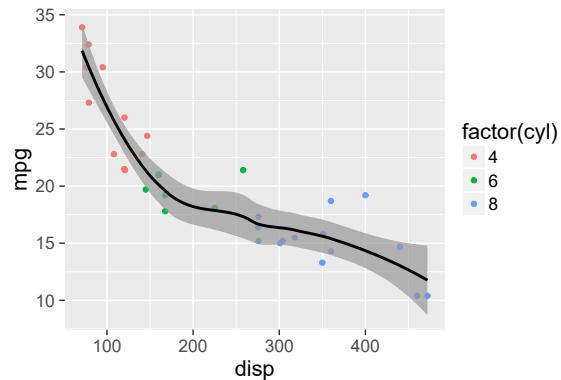


As exemplified above the different geoms and elements can be added in almost any order to a ggplot object, but they will be plotted in the order that they are added. The `alpha` (transparency) aesthetic can be mapped to a constant to make underlying layers visible, or they can be mapped to a data variable for example making the transparency of points in a plot depend on the number of observations used in its calculation.

```
ggplot(data = mtcars, aes(x=disp, y=mpg, colour=factor(cyl))) +  
  geom_point() + geom_smooth(colour="black", alpha=0.7)
```

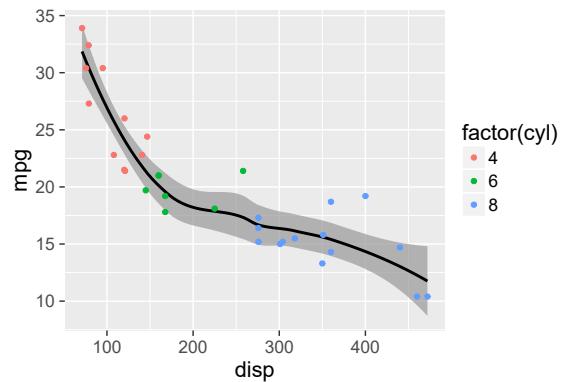
## 1.4 Adding fitted curves, including splines

---



The plot looks different if the order of the geoms is swapped. The data points overlapping the confidence band are more clearly visible in this second example because they are above the shaded area instead of below it.

```
ggplot(data = mtcars, aes(x=disp, y=mpg, colour=factor(cyl))) +  
  geom_smooth(colour="black", alpha=0.7) + geom_point()
```



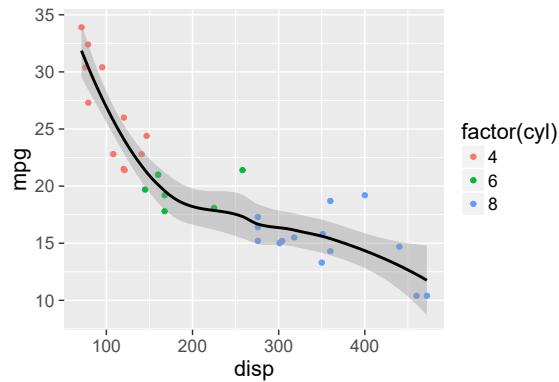
## 1.4 Adding fitted curves, including splines

We will now show an example of the use of `stat_smooth` accepting the default spline smoothing.

```
myplot + stat_smooth(colour="black")
```

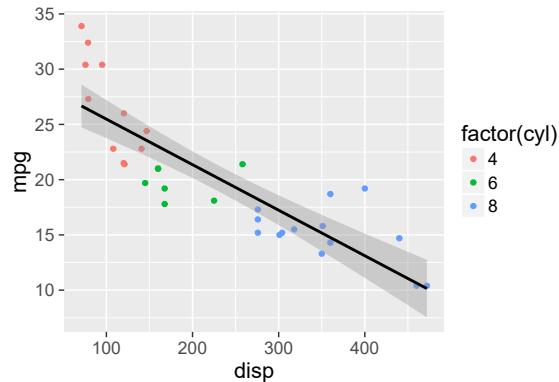
## 1 Plots with ggplot

---



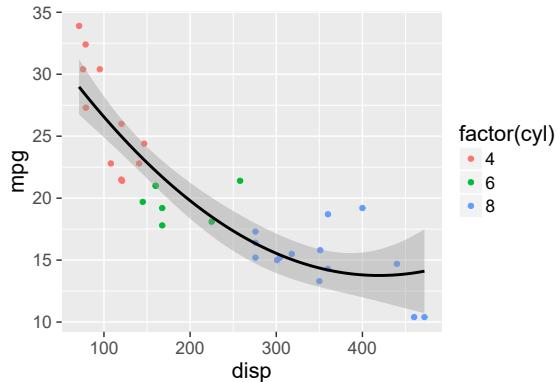
Instead of using the default spline, we can use a linear model fit. In this example we use a linear model, fitted by `lm`, as smoother:

```
myplot + stat_smooth(method="lm", colour="black")
```



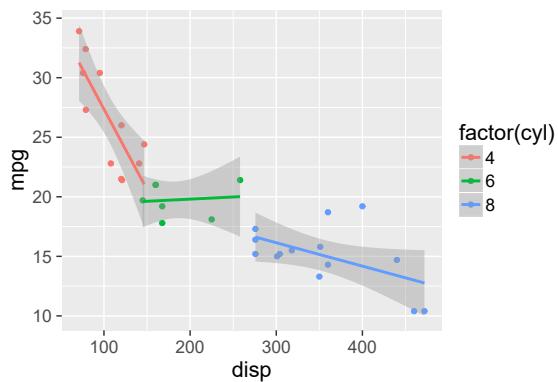
Instead of using the default linear regression as smoother, we can use a linear model fit. In this example we use a polynomial of order 2 fitted by `lm`.

```
myplot + stat_smooth(method="lm", formula=y~poly(x,2), colour="black")
```



If we do not use `colour="black"` then the colour aesthetics supplied to `ggplot` is used, and splits the data into three groups to which the model is fitted separately.

```
myplot + stat_smooth(method="lm")
```



It is possible to use other types of models, including GAM and GLM, as smoothers, but we will not give examples of the use these more advanced models.

## 1.5 Adding statistical “summaries”

It is also possible to summarize data on-the-fly when plotting, but before showing this we will generate some normally distributed artificial data:

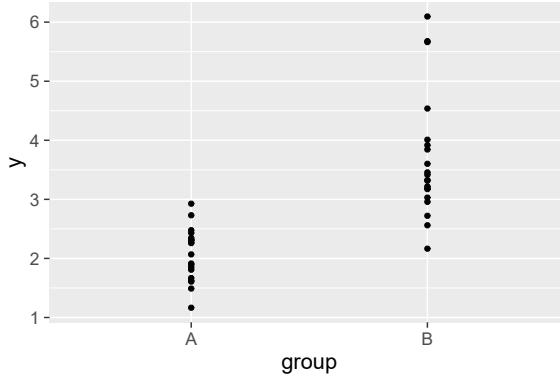
```
fake.data <- data.frame(
  y = c(rnorm(20, mean=2, sd=0.5),
        rnorm(20, mean=4, sd=0.7)),
  group = factor(c(rep("A", 20), rep("B", 20)))
)
```

## 1 Plots with ggplot

---

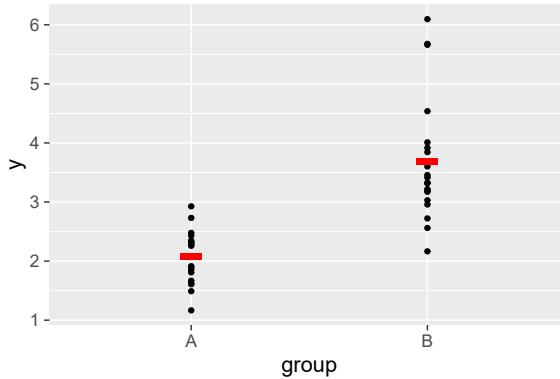
Now will we use these data to plot means and confidence intervals by group. However, to save typing we first produce a dot plot by group.

```
fig2 <- ggplot(data=fake.data, aes(y=y, x=group)) +  
  geom_point()  
fig2
```



We have saved the base figure in `fig2`, so now we can play with different summaries. We first add just the mean. In this case we need to add as argument to `stat_summary` the geom to use, as the default one expects data for plotting error bars, in later examples, this is not needed.

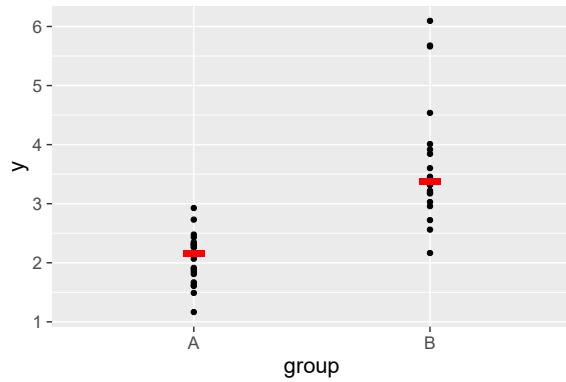
```
fig2 + stat_summary(fun.y = "mean", geom="point",  
  colour="red", shape="-", size=20)
```



Then the median, by changing the argument passed to `fun.y`.

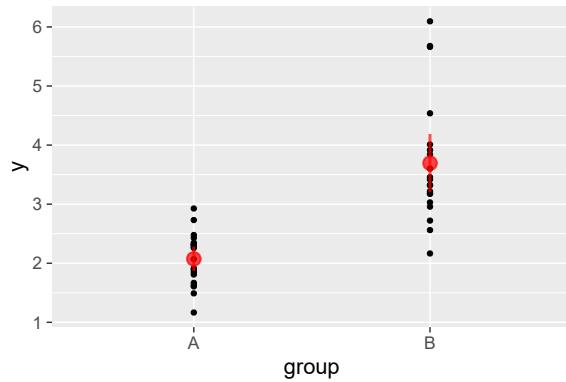
## 1.5 Adding statistical “summaries”

```
fig2 + stat_summary(fun.y = "median", geom="point",
                     colour="red", shape="-", size=20)
```



We can add the mean and  $p = 0.95$  confidence intervals assuming normality (using the  $t$  distribution):

```
fig2 + stat_summary(fun.data = "mean_cl_normal",
                     colour="red", size=1, alpha=0.7)
```

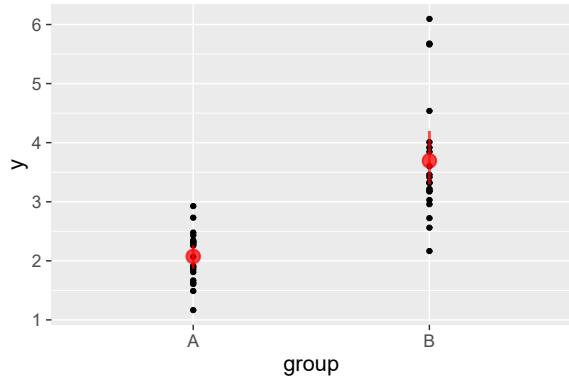


We can add the means and  $p = 0.95$  confidence intervals not assuming normality (using the actual distribution of the data by bootstrapping):

```
fig2 + stat_summary(fun.data = "mean_cl_boot",
                     colour="red", size=1, alpha=0.7)
```

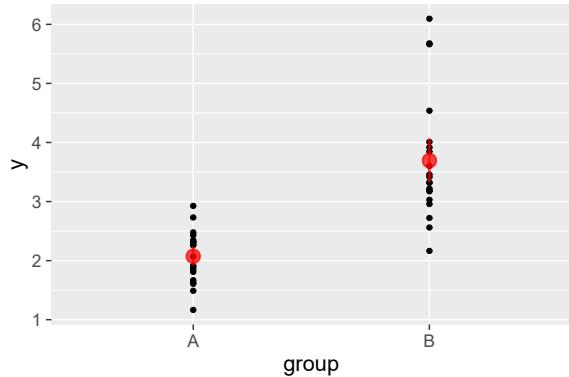
## 1 Plots with ggplot

---



If needed, we can instead add the means and less restrictive confidence intervals, at  $p = 0.90$  in this example, by means of `conf.int = 0.90` passed as a list to the underlying function being called.

```
fig2 + stat_summary(fun.data = "mean_cl_boot",
                     fun.args = list(conf.int = 0.90),
                     colour = "red", size = 1, alpha = 0.7)
```

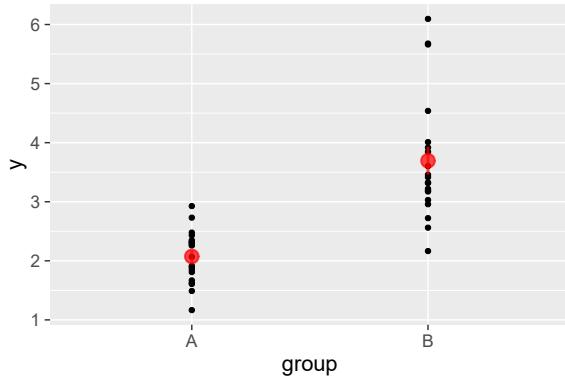


We can plot error bars corresponding to  $\pm$ s.e. (standard errors) with the function "mean\_se" that was added in 'ggplot2' 2.0.0.

```
fig2 + stat_summary(fun.data = "mean_se",
                     colour="red", size=1, alpha=0.7)
```

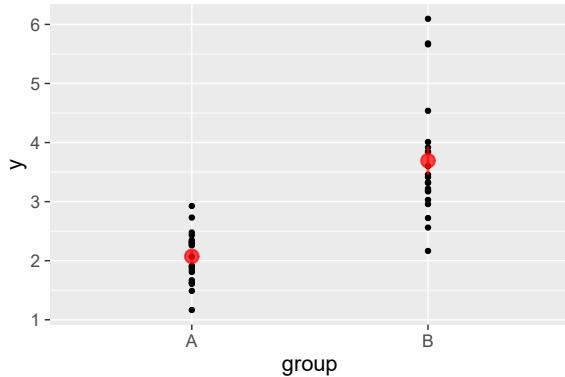
## 1.5 Adding statistical “summaries”

---



As `mult` is the multiplier based on the probability distribution used, by default student's t, by setting it to one, we get also standard errors of the mean.

```
fig2 + stat_summary(fun.data = "mean_cl_normal", fun.args = list(mult = 1),
                    colour="red", size=1, alpha=0.7)
```

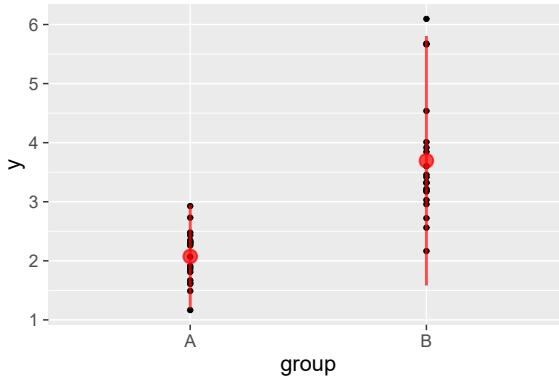


However, be aware that the code below, as used in earlier versions of ‘ggplot2’, needs to be rewritten as above.

```
fig2 + stat_summary(fun.data = "mean_cl_normal", mult = 1,
                    colour="red", size=1, alpha=0.7)
```

Finally we can plot error bars showing  $\pm$ s.d. (standard deviation).

```
fig2 + stat_summary(fun.data = "mean_sdl",
                    colour="red", size=1, alpha=0.7)
```



We do not show it here, but instead of using these functions (from package ‘Hmisc’) it is possible to define one’s own functions, and remember that arguments to all functions, except for the first one containing the actual data should be supplied as a list through formal argument `fun.args`.

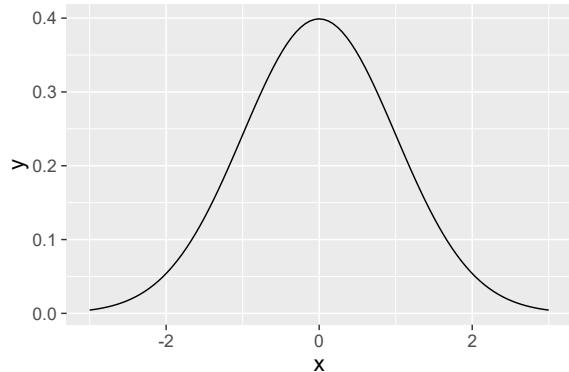
Finally we plot the means in a bar plot, with the observations superimposed and  $p = 0.95$  C.I. (the order in which the geoms are added is important: by having `geom_point` last it is plotted on top of the bars. In this case we set fill, colour and alpha (transparency) to constants, but in more complex data sets mapping them to factors in the data set can be used to distinguish them.

```
ggplot(data=fake.data, aes(y=y, x=group)) +
  stat_summary(fun.y = "mean", geom = "bar",
              fill="yellow", colour="black") +
  stat_summary(fun.data = "mean_cl_normal",
              geom = "errorbar", mult = 1,
              width=0.1, size=1, colour="red") +
  geom_point(size=3, alpha=0.3)
```

## 1.6 Plotting functions

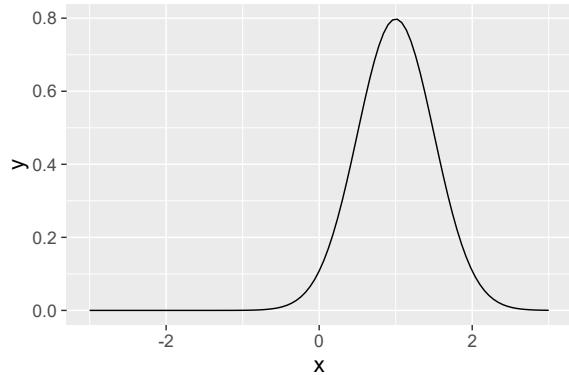
We can also directly plot functions, without need to generate data beforehand (the number of data points to be generated can be also set).

```
ggplot(data.frame(x=-3:3), aes(x=x)) +
  stat_function(fun=dnorm)
```



We can even pass additional arguments to a function.

```
ggplot(data.frame(x=-3:3), aes(x=x)) +  
  stat_function(fun = dnorm, args = list(mean = 1, sd = .5))
```

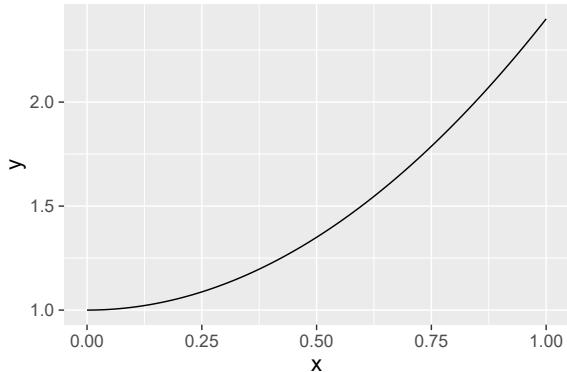


Of course, user-defined functions (not shown), and anonymous functions can also be used.

```
ggplot(data.frame(x=0:1), aes(x=x)) +  
  stat_function(fun = function(x, a, b){a + b * x^2},  
               args = list(a = 1, b = 1.4))
```

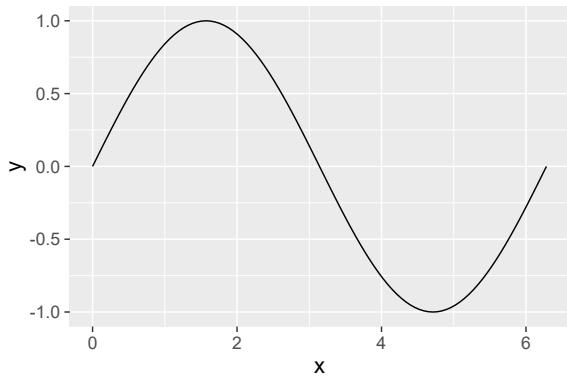
## 1 Plots with ggplot

---



Here is an example of a predefined function, but in this case the default breaks (tick positions) are not the best:

```
ggplot(data.frame(x=c(0, 2 * pi)), aes(x=x)) +  
  stat_function(fun=sin)
```

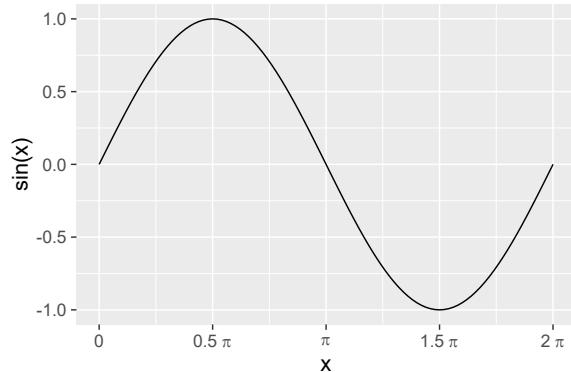


We need to change the  $x$ -axis scale to better suit the sin function and the use of radians as angular units<sup>2</sup>.

```
ggplot(data.frame(x=c(0, 2 * pi)), aes(x=x)) +  
  stat_function(fun=sin) +  
  scale_x_continuous(  
    breaks=c(0, 0.5, 1, 1.5, 2) * pi,  
    labels=c("0", expression(0.5~pi), expression(pi),  
           expression(1.5~pi), expression(2~pi))) +  
  labs(y="sin(x)")
```

---

<sup>2</sup>The use of `expression` is explained in detail in section ??, and the use of `scales` in section ??.



## 1.7 Plotting text and expressions

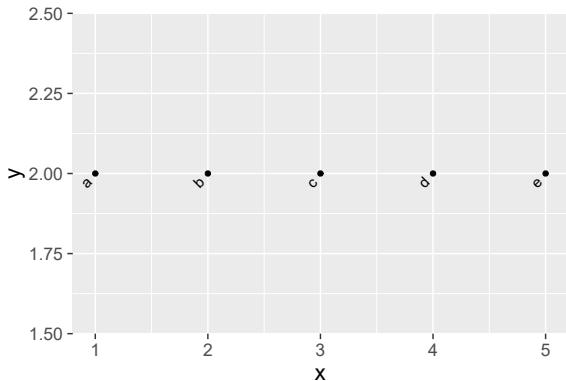
One can use `geom_text` to add text labels to observations. The aesthetic `label` gives text and the usual aesthetics `x` and `y` the location of the labels. As one would expect the `color` aesthetic can be also used for text. In addition `angle` and `vjust` and `hjust` can be used to rotate the label, and adjust its position. The default value of zero for both `hjust` and `vjust` centres the label. The centre of the text is at the supplied `x` and `y` coordinates. ‘Vertical’ and ‘horizontal’ for justification refer to the text, not the plot. This is important when `angle` is different from zero. Negative justification values, shift the label left or down, and positive values right or up. A value of 1 or 0 sets the text so that its edge is at the supplied coordinate. Values outside the range 0...1 sift the text even further away.

```
my.data <-
  data.frame(x=1:5, y=rep(2, 5),
             label=paste(letters[1:5], " "))

ggplot(my.data, aes(x,y,label=label)) +
  geom_text(angle=45, hjust=1) + geom_point()
```

## 1 Plots with ggplot

---

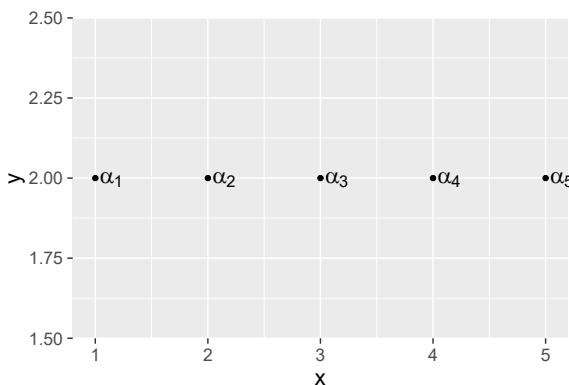


In this example we use `paste()` (which uses recycling here) to add a space at the end of each label. Justification values outside the range  $0 \dots 1$  are allowed, but are relative to the width of the label. As the default font used in this case has variable width characters, the justification would be inconsistent (e.g. try the code above but using `hjust` set to 3 instead of to 1 without pasting a space character to the labels.)

Plotting expressions (mathematical expressions) involves passing as `label` data character strings that can be parsed as expressions, and setting `parse = TRUE`.

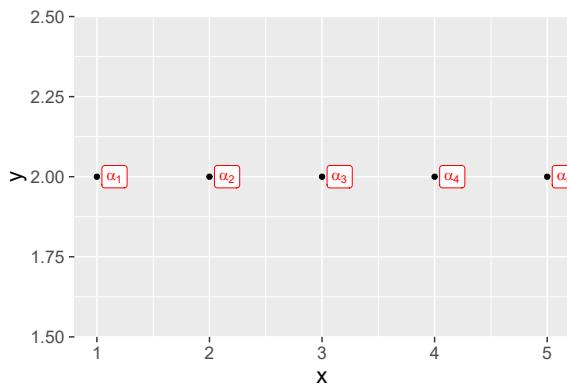
```
my.data <-
  data.frame(x=1:5, y=rep(2, 5),
             label=paste("alpha[", 1:5, "]", sep = ""))

ggplot(my.data, aes(x,y,label=label)) +
  geom_text(hjust=-0.2, parse=TRUE, size = rel(6)) +
  geom_point()
```



A similar example using `geom_label`.

```
ggplot(my.data, aes(x, y, label = label)) +  
  geom_label(hjust = -0.2, parse = TRUE, size = rel(4),  
             fill = "white", colour = "red") +  
  geom_point()
```



See R's 'plotmath' demo for more information on the syntax of expressions.

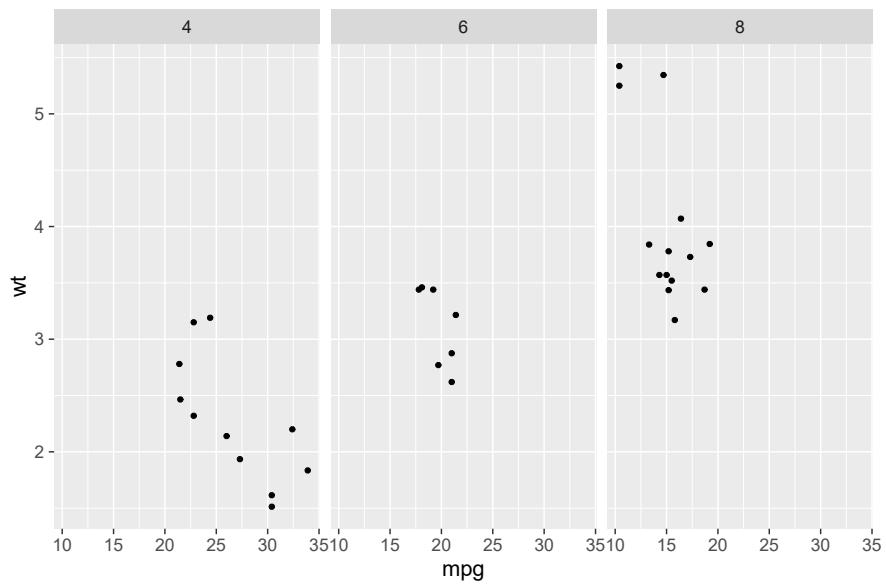
## 1.8 Using facets

Sets of coordinated plots are a very useful tool for visualizing data. These became popular through the `trellis` graphs in S, and the `lattice` package in R. The basic idea is to have row and/or columns of plots with common scales, all plots showing values for the same response variable. This is useful when there are multiple classification factors in a data set. Similarly looking plots but with free scales or with the same scale but a 'floating' intercept are sometimes also useful. In 'ggplot2' there are two possible types of facets: facets organized in a grid, and facets along a single 'axis' but wrapped into several rows. In the examples below we use `geom_point` but facetting can be used with any `ggplot` object (even with maps, spectra and ternary plots produced by functions in packages 'ggmap', 'ggspectra' and 'ggtern').

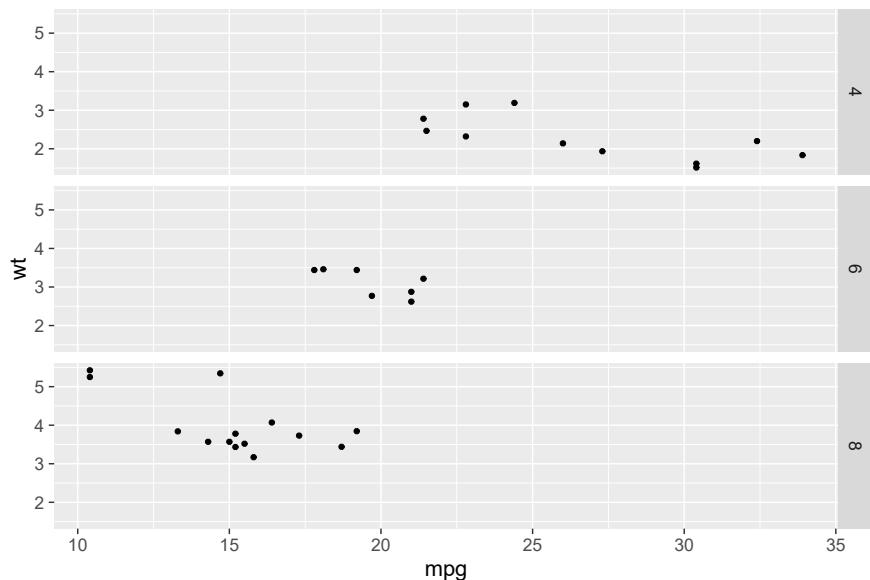
```
p <- ggplot(data = mtcars, aes(mpg, wt)) + geom_point()  
# With one variable  
p + facet_grid(. ~ cyl)
```

## 1 Plots with ggplot

---



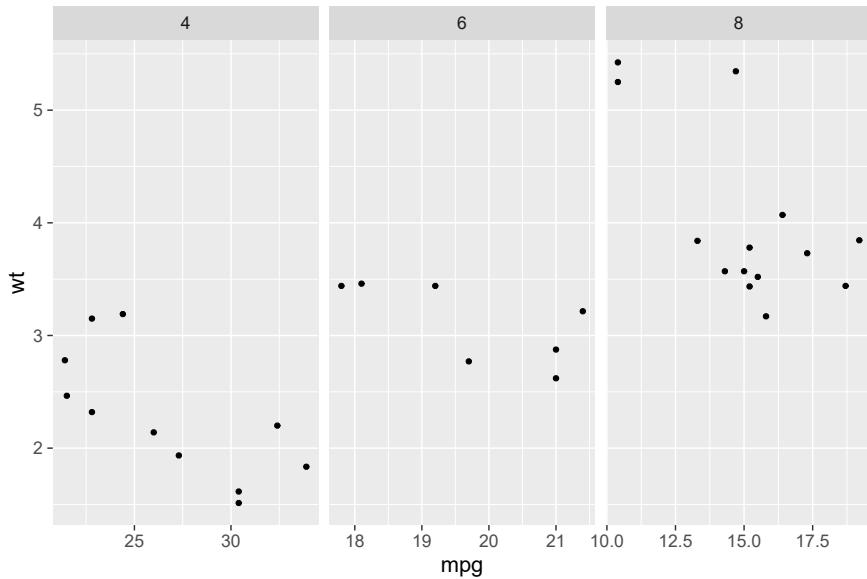
```
p + facet_grid(cyl ~ .)
```



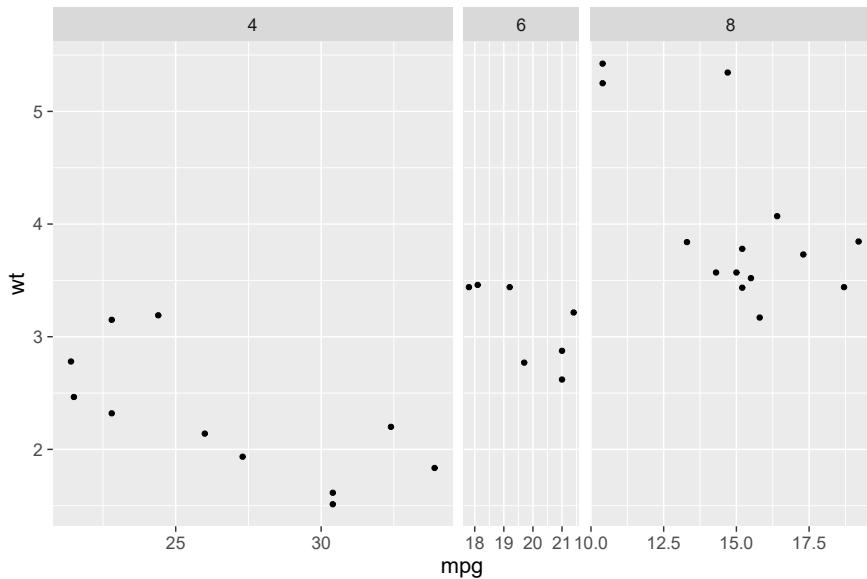
## 1.8 Using facets

---

```
p + facet_grid(. ~ cyl, scales = "free")
```



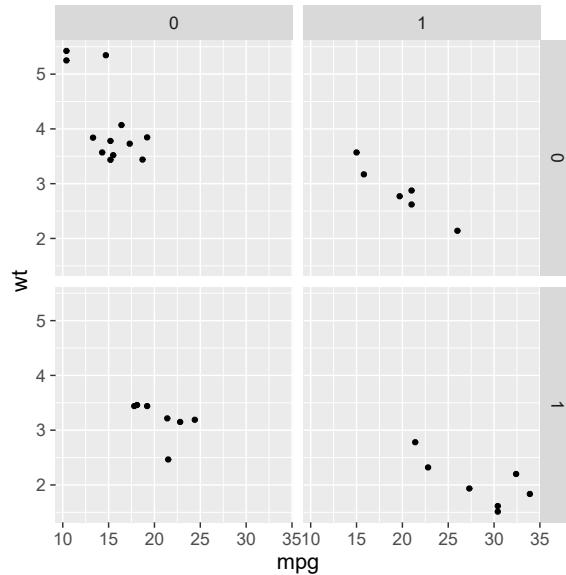
```
p + facet_grid(. ~ cyl, scales = "free", space = "free")
```



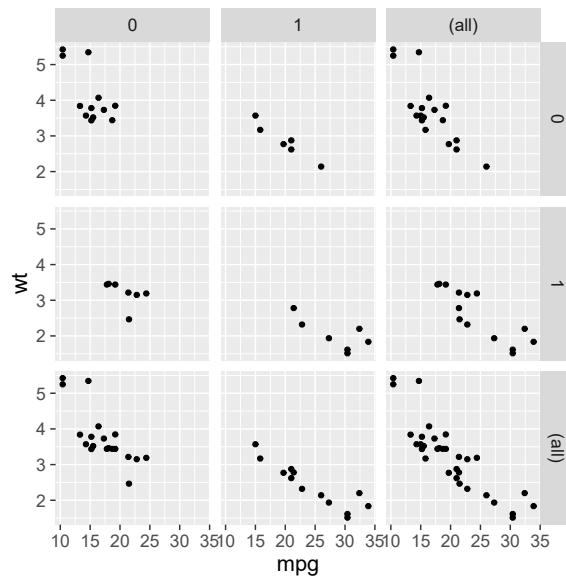
## 1 Plots with ggplot

---

```
p + facet_grid(vs ~ am)
```



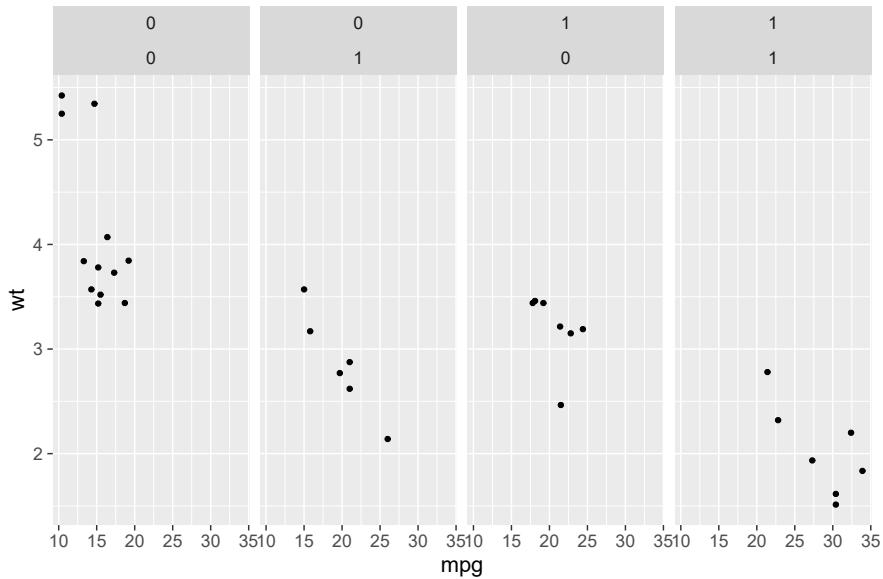
```
p + facet_grid(vs ~ am, margins=TRUE)
```



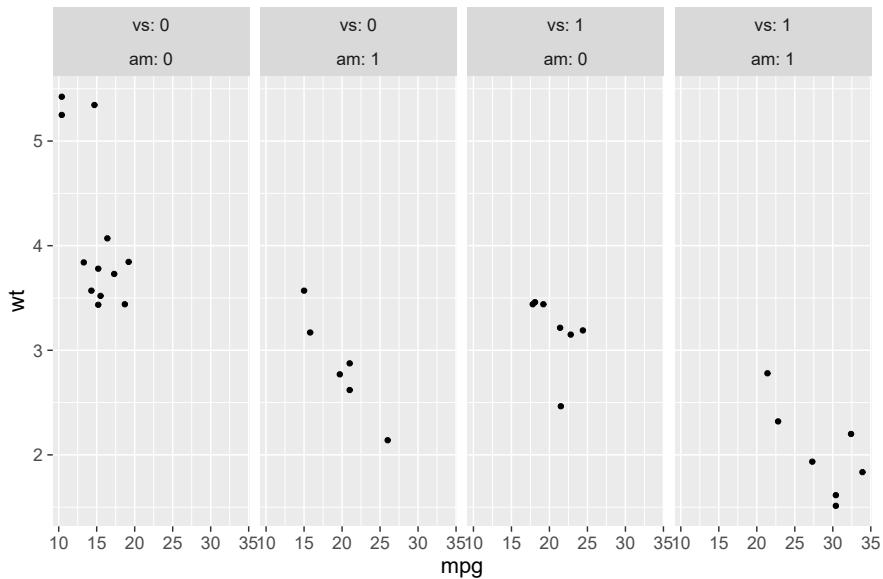
## 1.8 Using facets

---

```
p + facet_grid(. ~ vs + am)
```



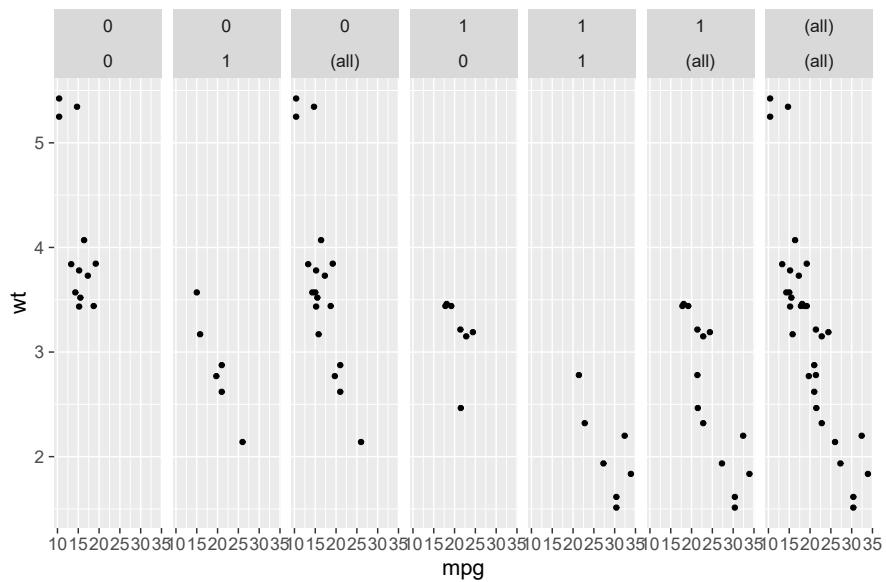
```
p + facet_grid(. ~ vs + am, labeller = label_both)
```



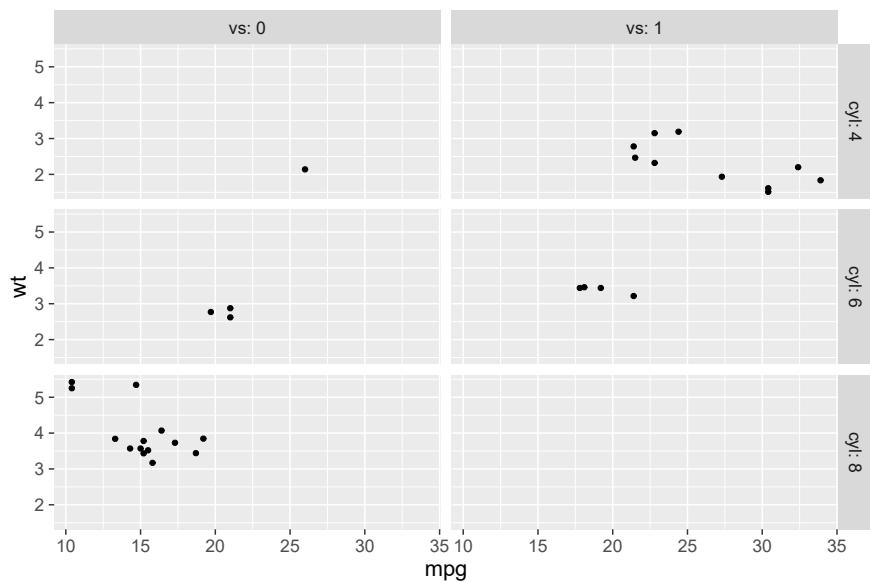
## 1 Plots with ggplot

---

```
p + facet_grid(. ~ vs + am, margins=TRUE)
```



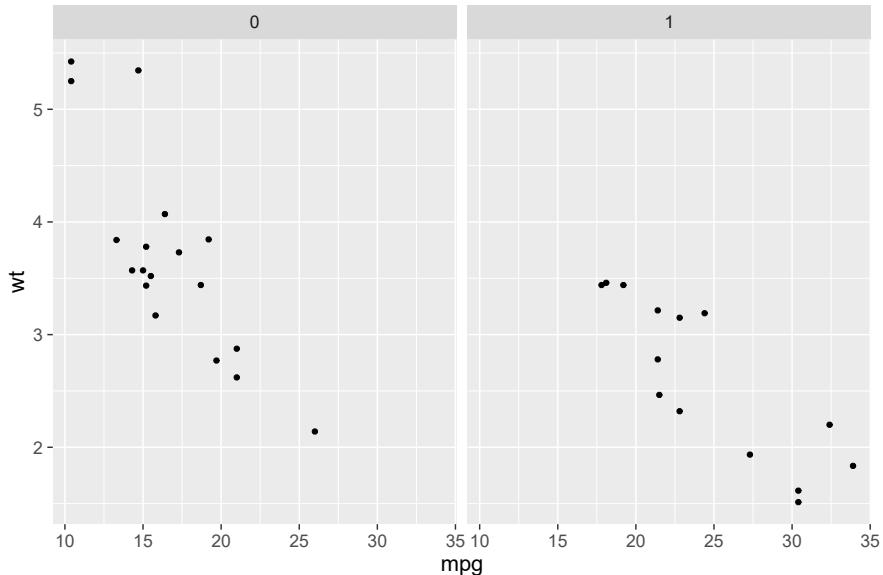
```
p + facet_grid(cyl ~ vs, labeller = label_both)
```



```
mtcars$cyl12 <- factor(mtcars$cyl,
                         labels = c("alpha", "beta", "sqrt(x, y)"))
p1 <- ggplot(data = mtcars, aes(mpg, wt)) +
  geom_point() +
  facet_grid(. ~ cyl12, labeller = label_parsed)
```

Here we use as **labeller** function **label\_bquote()** with a special syntax that allows us to use an expression where replacement based on the facet (panel) data takes place.

```
p + facet_grid(. ~ vs, labeller = label_bquote(alpha ^ .(vs)))
```

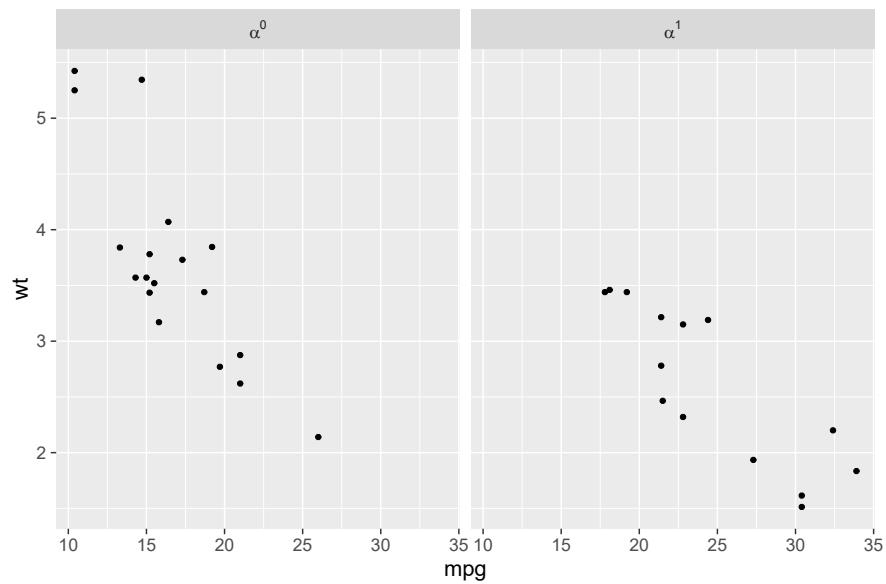


In versions of ‘ggplot2’ 2 before 2.0.0, **labeller** was not implemented for **facet\_wrap()**, it was only available for **facet\_grid()**.

```
p + facet_wrap(~ vs, labeller = label_bquote(alpha ^ .(vs)))
```

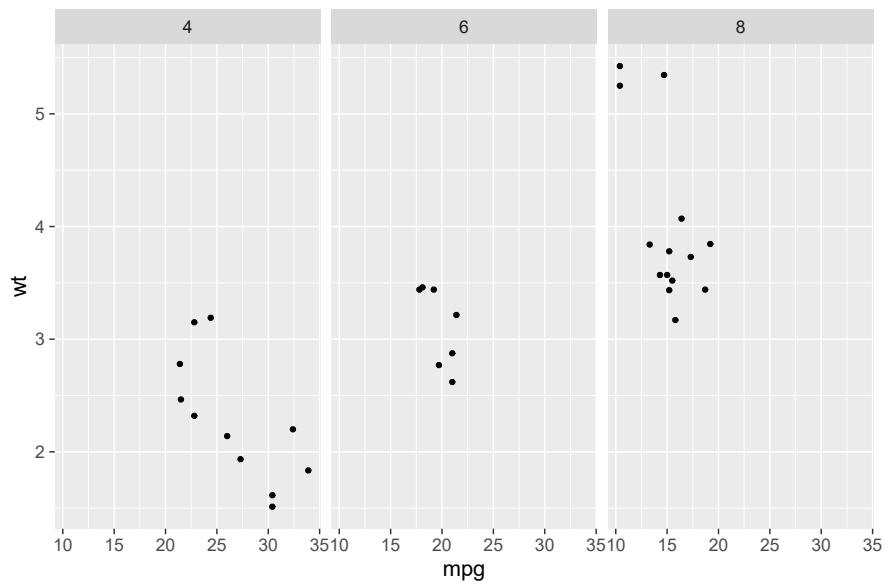
## 1 Plots with ggplot

---



A minimal example of a wrapped facet. In this case the number of levels is small, when they are more the row of plots will be wrapped into two or more continuation rows. When using `facet_wrap()` there is only one dimension, so no ‘.’ is needed before or after the tilde.

```
p + facet_wrap(~ cyl)
```

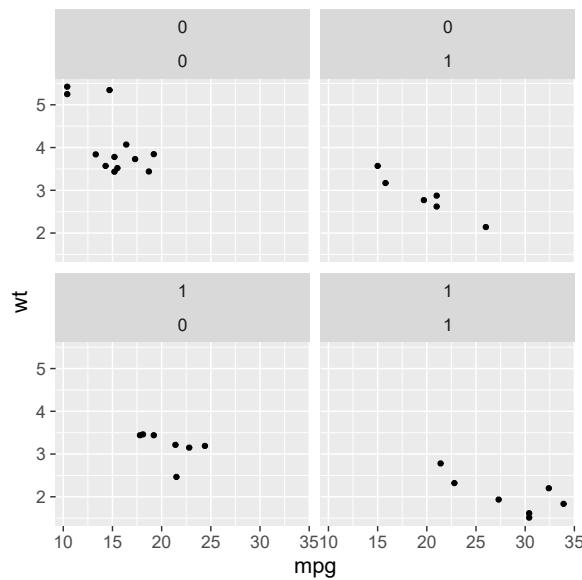


An example showing that even though faceting with `facet_wrap()` is along a single, possibly wrapped, row, it is possible to produce facets based on more than one variable.

```
p + facet_wrap(~ vs + am, ncol=2)
```

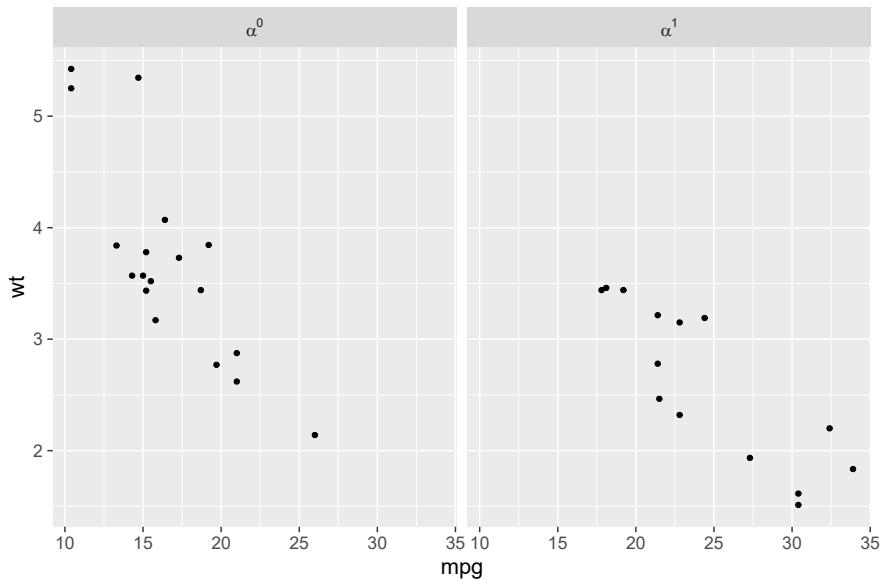
## 1 Plots with ggplot

---



In versions of ‘ggplot2’ before 2.0.0, `labeller` was not implemented for `facet_wrap()`, it was only available for `facet_grid()`. In the current version it is implemented for both.

```
p + facet_wrap(~ vs, labeller = label_bquote(alpha ^ .(vs)))
```



## 1.9 Scales

Scales map data onto aesthetics. There are different types of scales depending on the characteristics of the data being mapped: scales can be continuous or discrete. And of course, there are scales for different attributes of the plotted geometrical object, such as `color`, `size`, position (`x`, `y`, `z`), `alpha` or transparency, `angle`, justification, etc. This means that many properties of, for example, the symbols used in a plot can be either set by a constant, or mapped to data. The most elemental mapping is `identity`, which means that the data is taken at its face value. In a numerical scale, say `scale_x_continuous`, this means that for example a '5' in the data is plotted at a position in the plot corresponding to the value '5' along the x-axis. A simple mapping could be a  $\log_{10}$  transformation, that we can easily achieve with the pre-defined `scale_x_log10` in which case the position on the  $x$ -axis will be based on the logarithm of the original data. A continuous data variable can, if we think it useful for describing our data, be mapped to continuous scale either using an identity mapping or transformation, which for example could be useful if we want to map the value of a variable to the area of the symbol rather than its diameter.

Discrete scales work in a similar way. We can use `scale_colour_identity` and have in our data a variable with values that are valid colour names like "red" or "blue". However we can also map the `colour` aesthetic to a factor

## 1 Plots with ggplot

---

with levels like "control", and "treatment", and these levels will be mapped to colours from the default palette, unless we chose a different palette, or even use `scale_colour_manual` to assign whatever colour we want to each level to be mapped. The same is true for other discrete scales like symbol `shape` and `linetype`. Remember that for example for colour, and 'numbers' there are both discrete and continuous scales available. Mapping colour or fill to `NA` makes such observation invisible.

Advanced scale manipulation requires package `scales` to be loaded, although 'ggplot2' 2.0.0 re-exports many functions from package `scales`. Some simple examples follow.

Some new fake data.

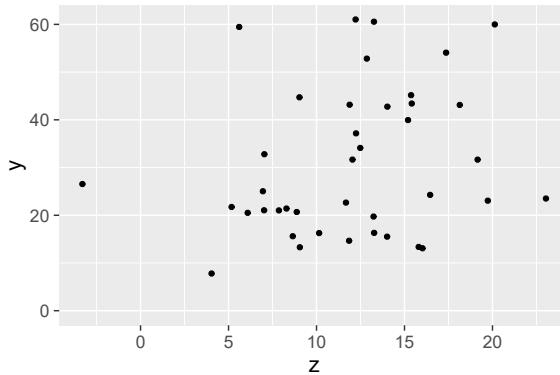
```
fake2.data <-  
  data.frame(y = c(rnorm(20, mean=20, sd=5),  
                  rnorm(20, mean=40, sd=10)),  
             group = factor(c(rep("A", 20), rep("B", 20))),  
             z = rnorm(40, mean=12, sd=6))
```

We save `ggplot` object with the default scales.

```
fig2 <- ggplot(fake2.data, aes(z, y)) + geom_point()
```

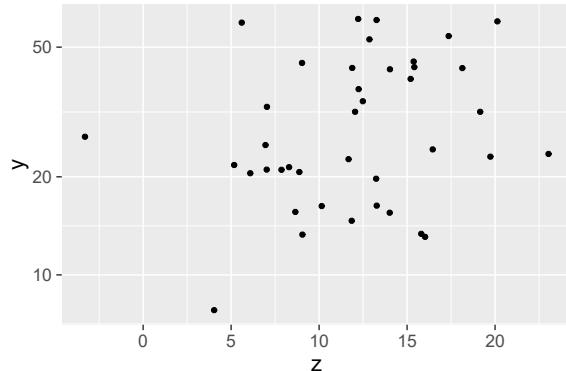
We re-save a `ggplot` object using the default scales, except that we change the limits of the  $y$ -scale. `ylim()` is a convenience function used for modification of the `lims` (limits) of the scale used by the  $y$  aesthetic.

```
fig2 + ylim(0, NA)
```



The default scale used by the  $y$  aesthetic uses `position = "identity"`, but there are predefined for transformed scales. Axis tick labels display the original values before applying the transformation, original numbers. The "breaks" need to be given in the original scale.

```
fig2 + scale_y_log10(breaks=c(10,20,50,100))
```



In contrast, transforming the data, results in tick-labels expressed in the logarithm of the original data.

```
fig2log <- ggplot(fake2.data, aes(z, log10(y))) + geom_point()
```

When combining scale transformations and summaries, one should be aware of which data are used, transformed or not.

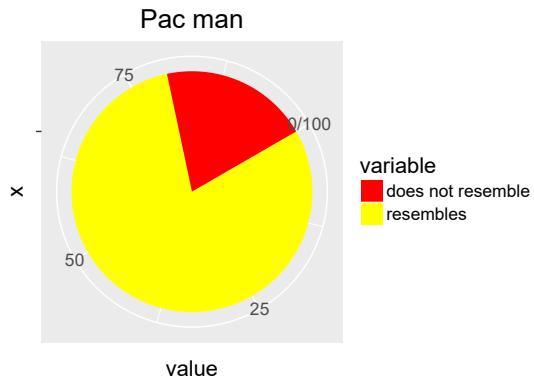
## 1.10 Circular plots

Under circular plots I include pie charts. Here we add a new "word" to the grammar of graphics, *coordinates*, such as `coord_polar()` in the next examples. The default coordinate system for *x* and *y* aesthetics is cartesian.

Pie charts are more difficult to read: our brain is more comfortable at comparing lengths than angles. If used, they should only be used to show composition, or fractional components that add up to a total. In this case only if the number of "pie slices" is small (rule of thumb: less seven).

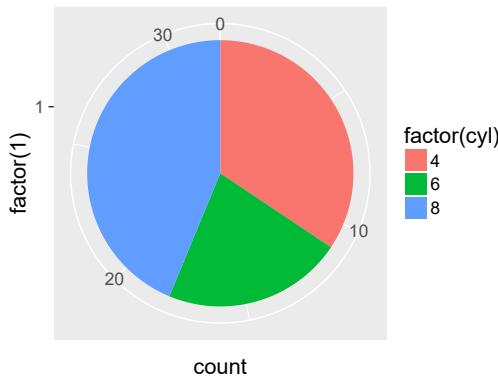
A funny example stolen from the 'ggplot2' website at [http://docs.ggplot2.org/current/coord\\_polar.html](http://docs.ggplot2.org/current/coord_polar.html).

```
# Hadley's favourite pie chart
df <- data.frame(
  variable = c("resembles", "does not resemble"),
  value = c(80, 20)
)
ggplot(df, aes(x = "", y = value, fill = variable)) +
  geom_bar(width = 1, stat = "identity") +
  scale_fill_manual(values = c("red", "yellow")) +
  coord_polar("y", start = pi / 3) +
  labs(title = "Pac man")
```



Something just a bit more useful, also stolen from the same page:

```
# A pie chart = stacked bar chart + polar coordinates
pie <- ggplot(data = mtcars, aes(x = factor(1), fill = factor(cyl))) +
  geom_bar(width = 1)
pie + coord_polar(theta = "y")
```



## 1.11 Bar plots

## 1.12 Pie charts vs. bar plots example

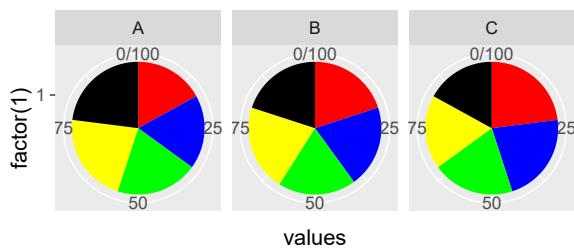
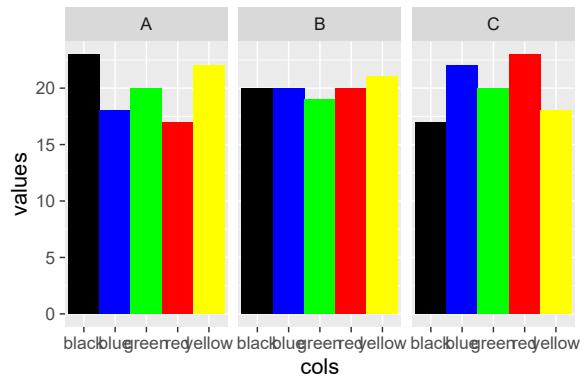
There is an example figure widely used in Wikipedia to show how much easier it is to ‘read’ bar plots than pie charts (<http://commons.wikimedia.org/wiki/File:Piecharts.svg?uselang=en-gb>).

Here is my ‘ggplot2’ version of the same figure, using much simpler code and obtaining almost the same result.

## 1.12 Pie charts vs. bar plots example

```
example.data <-
  data.frame(values = c(17, 18, 20, 22, 23,
                       20, 20, 19, 21, 20,
                       23, 22, 20, 18, 17),
             examples= rep(c("A", "B", "C"), c(5,5,5)),
             cols = rep(c("red", "blue", "green", "yellow", "black"), 3)
  )

ggplot(example.data, aes(x=cols, y=values, fill=cols)) +
  geom_bar(width = 1, stat="identity") +
  facet_grid(.~examples) +
  scale_fill_identity()
ggplot(example.data, aes(x=factor(1), y=values, fill=cols)) +
  geom_bar(width = 1, stat="identity") +
  facet_grid(.~examples) +
  scale_fill_identity() +
  coord_polar(theta="y")
```



## 1.13 Frequencies and densities

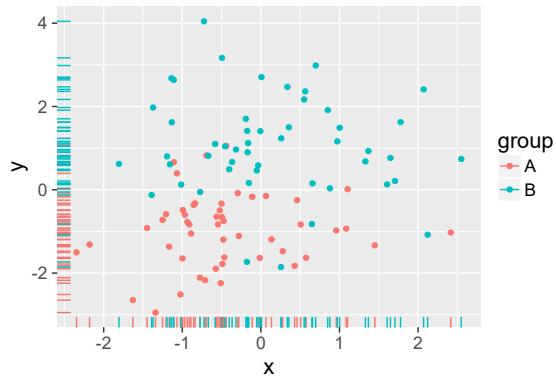
### 1.13.1 Marginal rug plots

Rarely rug-plots are used by themselves. Instead they are usually an addition to scatter plots. An example follows. They make it easier to see the distribution along the  $x$ - and  $y$ -axes.

We generate new fake data by random sampling from the normal distribution. We use `set.seed(1234)` to initialize the pseudo-random number generator so that the same data are generated each time the code is run.

```
set.seed(1234)
my.data <-
  data.frame(x = rnorm(100),
             y = c(rnorm(50, -1, 1), rnorm(50, 1, 1)),
             group = factor(rep(c("A", "B"), c(50, 50))))
```

```
ggplot(my.data, aes(x, y, colour = group)) +
  geom_point() +
  geom_rug()
```

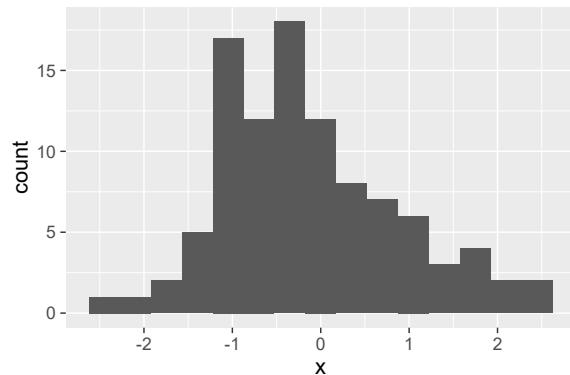


### 1.13.2 Histograms

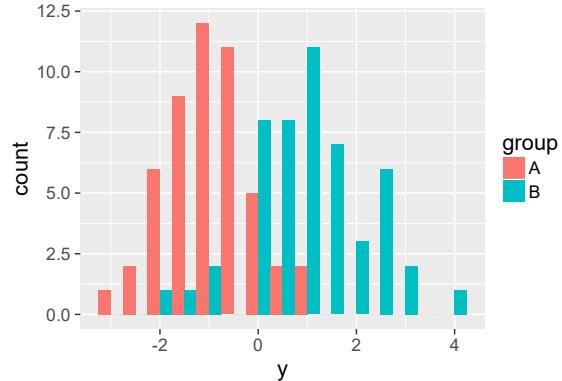
```
ggplot(my.data, aes(x)) +
  geom_histogram(bins = 15)
```

## 1.13 Frequencies and densities

---



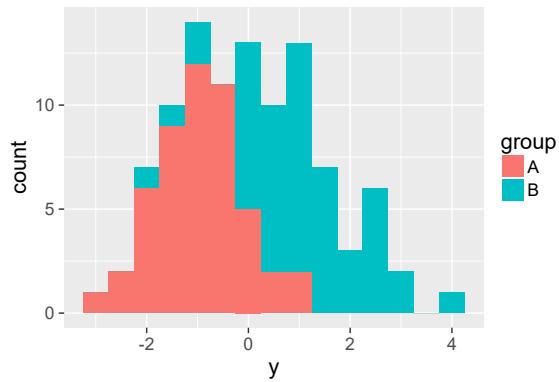
```
ggplot(my.data, aes(y, fill = group)) +  
  geom_histogram(bins = 15, position = "dodge")
```



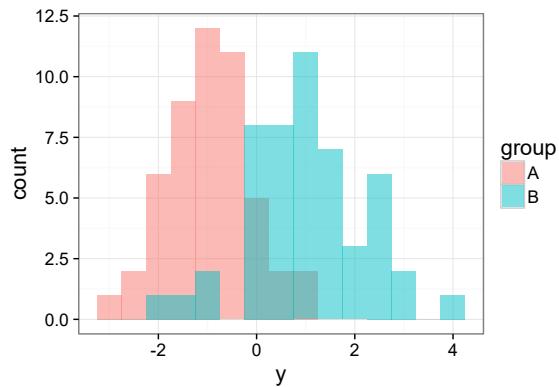
```
ggplot(my.data, aes(y, fill = group)) +  
  geom_histogram(bins = 15, position = "stack")
```

## 1 Plots with ggplot

---



```
ggplot(my.data, aes(y, fill = group)) +  
  geom_histogram(bins = 15, position = "identity", alpha = 0.5) +  
  theme_bw(16)
```

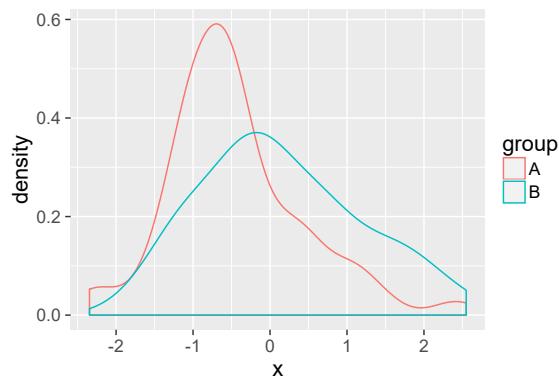


### 1.13.3 Density plots

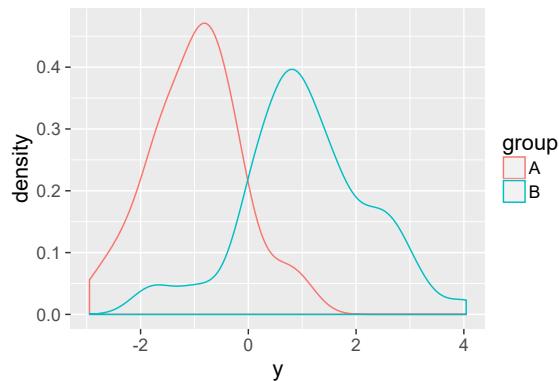
```
ggplot(my.data, aes(x, colour = group)) +  
  geom_density()
```

## 1.13 Frequencies and densities

---



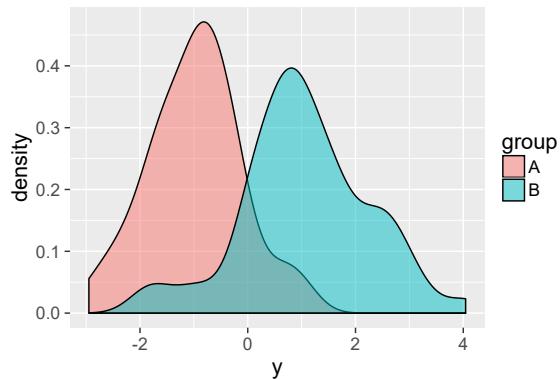
```
ggplot(my.data, aes(y, colour = group)) +  
  geom_density()
```



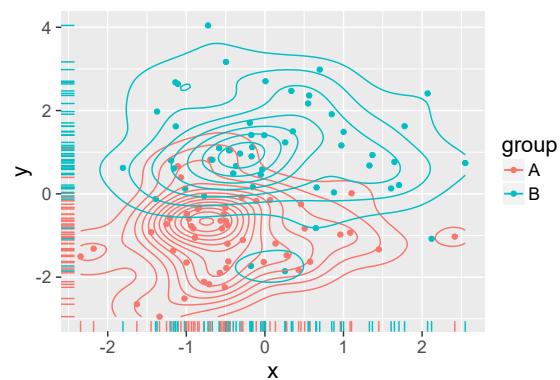
```
ggplot(my.data, aes(y, fill = group)) +  
  geom_density(alpha = 0.5)
```

## 1 Plots with ggplot

---

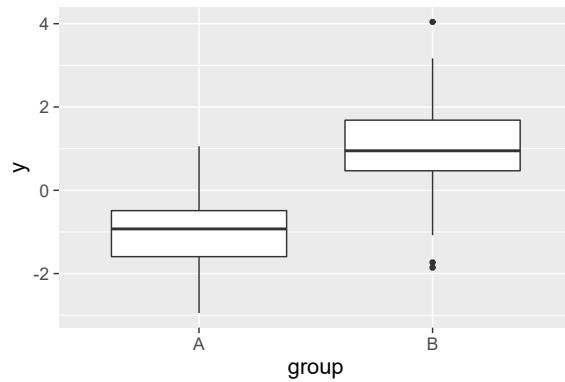


```
ggplot(my.data, aes(x, y, colour = group)) +  
  geom_point() +  
  geom_rug() +  
  geom_density_2d()
```



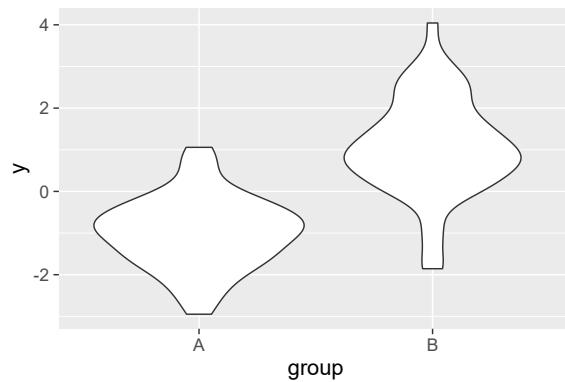
### 1.13.4 Box and whiskers plots

```
ggplot(my.data, aes(group, y)) +  
  geom_boxplot()
```

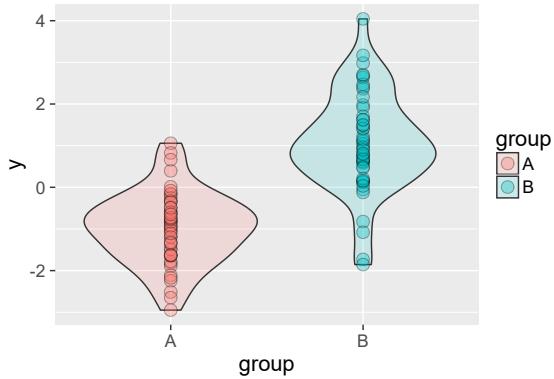


### 1.13.5 Violin plots

```
ggplot(my.data, aes(group, y)) +  
  geom_violin()
```



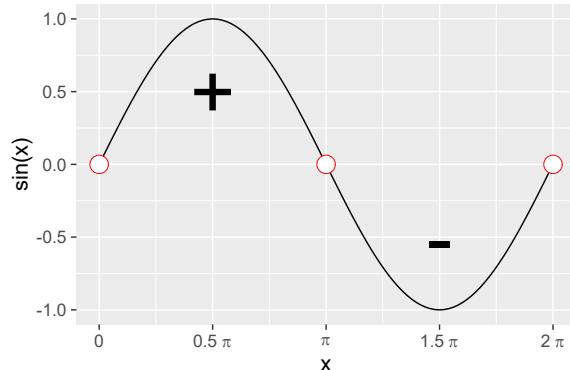
```
ggplot(my.data, aes(group, y, fill = group)) +  
  geom_violin(alpha = 0.16) +  
  geom_point(alpha = 0.33, size = rel(4),  
             colour = "black", shape = 21)
```



## 1.14 Adding annotations

Annotations use the data coordinates of the plot, but do not ‘inherit’ data or aesthetics from the `ggplot` object. In this example we pass directly expressions as tick labels. Do notice that we use recycling for setting the breaks, as `c(0, 0.5, 1, 1.5, 2) * pi` is equivalent to `c(0, 0.5 * pi, pi, 1.5 * pi, 2 * pi)`.

```
ggplot(data.frame(x=c(0, 2 * pi)), aes(x=x)) +
  stat_function(fun=sin) +
  scale_x_continuous(
    breaks=c(0, 0.5, 1, 1.5, 2) * pi,
    labels=c("0", expression(0.5~pi), expression(pi),
            expression(1.5~pi), expression(2~pi))) +
  labs(y="sin(x)") +
  annotate(geom="text",
          label=c("+", "-"),
          x=c(0.5, 1.5) * pi, y=c(0.5, -0.5),
          size=20) +
  annotate(geom="point",
          colour="red",
          shape=21,
          fill="white",
          x=c(0, 1, 2) * pi, y=0,
          size=6)
```



## 1.15 Themes

### 1.15.1 Predefined themes

### 1.15.2 Tweaking a theme

### 1.15.3 Defining a new theme

## 1.16 Advanced topics

## 1.17 Using plotmath expressions

Expressions are very useful but rather tricky to use because the syntax is unusual. In `ggplot` one can either use expressions explicitly, or supply them as character string labels, and tell `ggplot` to parse them. For titles, axis-labels, etc. (anything that is defined with `labs`) the expressions have to be entered explicitly, or saved as such into a variable, and the variable supplied as argument. When plotting expressions using `geom_text` expression arguments should be supplied as character strings and the optional argument `parse=TRUE` used to tell the geom to interpret the labels as expressions. We will go through a few useful examples.

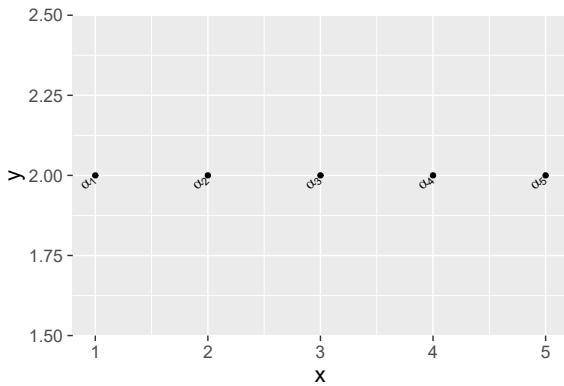
We will revisit the example from the previous section, but now using subscripted Greek  $\alpha$  for labels. In this example we use as subscripts numeric values from another variable in the same data frame.

```
my.data <-  
  data.frame(x = 1:5, y = rep(2, 5),  
             label = paste("alpha[", 1:5, "]", sep = ""))  
my.data$greek.label <- paste("alpha[", my.data$x, "]", sep="")  
(fig <- ggplot(my.data, aes(x,y,label=greek.label)) +
```

## 1 Plots with ggplot

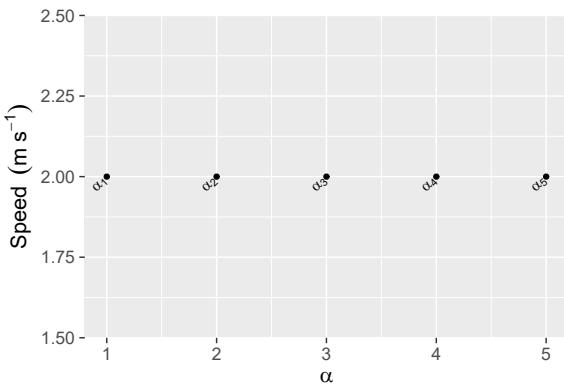
---

```
geom_text(angle=45, hjust=1.2, parse=TRUE) + geom_point()
```



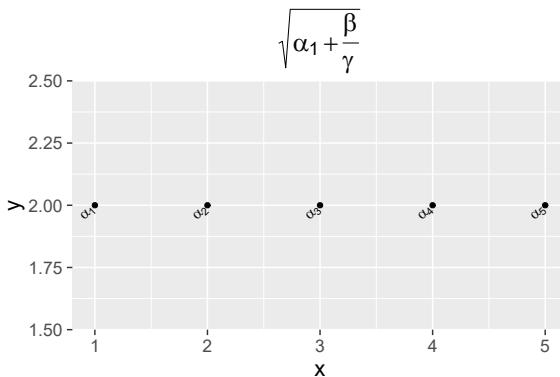
Setting an axis label with superscripts. The easiest way to deal with spaces is to use ‘ ’ or ‘ ’. One can connect pieces that would otherwise cause errors using ‘\*’. If we

```
fig + labs(x=expression(alpha), y=expression(Speed^{m s^{-1}}))
```



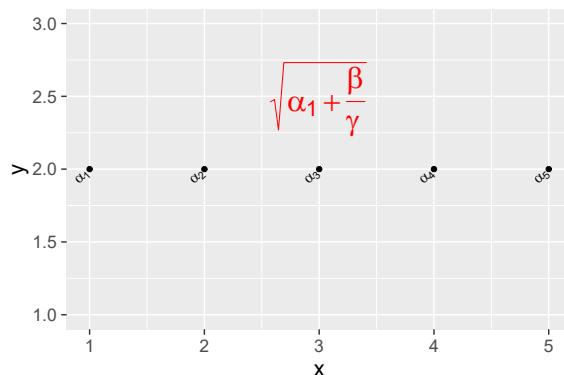
It is possible to store expressions in variables.

```
my.title <- expression(sqrt(alpha[1] + frac(beta, gamma)))
fig + labs(title=my.title)
```



Annotations are plotted ignoring the default aesthetics, but still make use of geoms, so labels for annotations also have to be supplied as character strings and parsed.

```
fig + ylim(1,3) +
  annotate("text", label="sqrt(alpha[1] + frac(beta, gamma))",
          y=2.5, x=3, size=8, colour="red", parse=TRUE)
```



We discuss how to use expressions as facet labels in section ??.

## 1.18 Scales in detail

## 1.19 Generating output files

It is possible, when using RStudio, to directly export the displayed plot to a file. However, if the file will have to be generated again at a later time, or a series

of plots need to be produced with consistent format, it is best to include the commands to export the plot in the script.

In R, files are created by printing to different devices. Printing is directed to a currently open device. Some devices produce screen output, others files. Devices depend on drivers. There are both devices that are part of R, and devices that can be added through packages.

A very simple example of PDF output (width and height in inches):

```
fig1 <- ggplot(data.frame(x=-3:3), aes(x=x)) +
  stat_function(fun=dnorm)
pdf(file="fig1.pdf", width=8, height=6)
print(fig1)
dev.off()
```

Encapsulated Postscript output (width and height in inches):

```
postscript(file="fig1.eps", width=8, height=6)
print(fig1)
dev.off()
```

There are Graphics devices for BMP, JPEG, PNG and TIFF format bitmap files. In this case the default units for width and height is pixels. For example we can generate TIFF output:

```
tiff(file="fig1.tiff", width=1000, height=800)
print(fig1)
dev.off()
```

### 1.19.1 Using L<sup>A</sup>T<sub>E</sub>X instead of plotmath

### 1.19.2 Fonts

## 1.20 Examples

### 1.20.1 Anscombe's regression examples

This is another figure from Wikipedia <http://commons.wikimedia.org/wiki/File:Anscombe.svg?uselang=en-gb>.

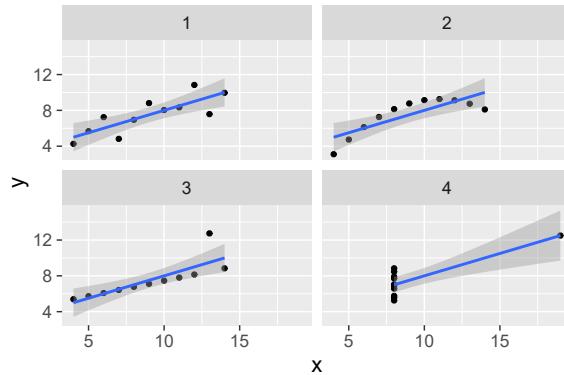
This classical example from Anscombe (1973) demonstrates four very different data sets that yield exactly the same results when a linear regression model is fit to them, including  $R^2 = 0.666$ . It is usually presented as a warning about the need to check model fits beyond looking at  $R^2$  and other parameter's estimates.

I will redraw the Wikipedia figure using 'ggplot2', but first I rearrange the original data.

```
# we rearrange the data
my.mat <- matrix(as.matrix(anscombe), ncol=2)
my.anscombe <- data.frame(x = my.mat[, 1],
                           y = my.mat[, 2],
                           case=factor(rep(1:4, rep(11,4))))
```

Once the data is in a data frame, plotting the observations plus the regression lines is easy.

```
ggplot(my.anscombe, aes(x,y)) +
  geom_point() +
  geom_smooth(method="lm") +
  facet_wrap(~case, ncol=2)
```

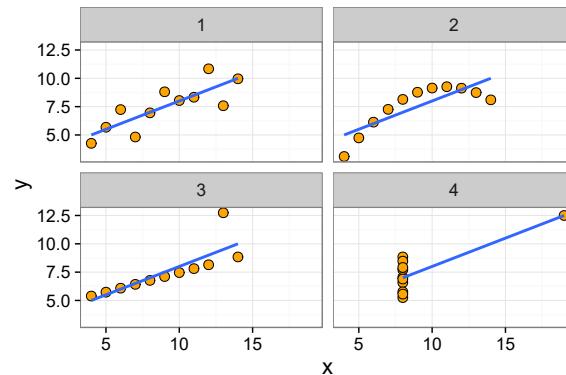


It is not much more difficult to make it look similar to the Wikipedia original.

```
ggplot(my.anscombe, aes(x,y)) +
  geom_point(shape=21, fill="orange", size=3) +
  geom_smooth(method="lm", se=FALSE) +
  facet_wrap(~case, ncol=2) +
  theme_bw(16)
```

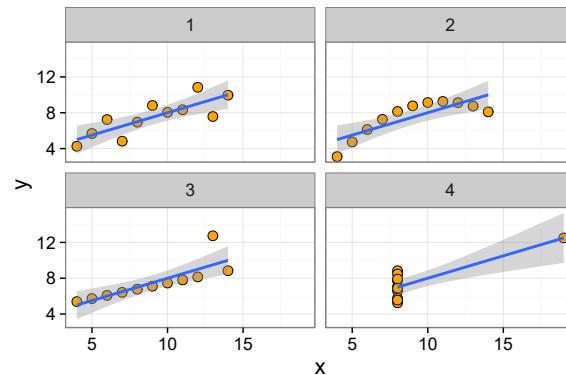
## 1 Plots with ggplot

---



Although I think that the confidence bands make the point of the example much clearer.

```
ggplot(my.anscombe, aes(x,y)) +  
  geom_point(shape=21, fill="orange", size=3) +  
  geom_smooth(method="lm") +  
  facet_wrap(~case, ncol=2) +  
  theme_bw(16)
```



### 1.20.2 Volcano plots

A volcano plot is just an elaborate version of a scatter plot, and can be created with ‘ggplot2’ functions.

```
try(detach(package:ggplot2))
```

## 2 Plots with ‘ggplot’, ‘ggrepel’ and ‘ggpmisc’

### 2.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(ggplot2)
library(ggrepel)
library(ggpmisc)
library(xts)
```

We set a font larger size than the default

```
theme_set(theme_grey(16))
```

### 2.2 ‘ggpmisc’

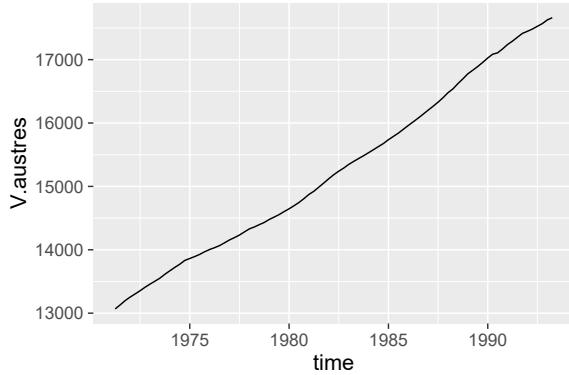
Package ‘ggpmisc’ is a small package developed by myself as a result of questions from work mates and in Stackoverflow, or functionality that I have needed in my own research or for teaching.

#### 2.2.1 Plotting time-series

Instead of creating a new statistics or geometry for plotting time series we provide a function that can be used to convert time series objects into data frames suitable for plotting with ‘ggplot2’. A single function `try_data_frame()` accepts time series objects saved with different packages as well as R’s native `ts` objects. The *magic* is done mainly by package ‘xts’ to which we add a very simple wrapper to obtain a data frame.

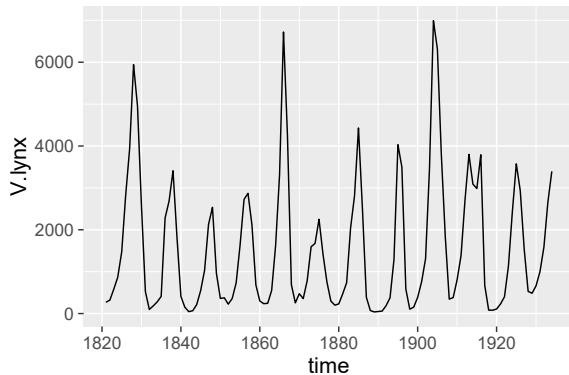
We exemplify this with some of the time series data included in R. In the first example we use the default format for time.

```
ggplot(try_data_frame(austres),
       aes(time, V.austres)) +
  geom_line()
```



In the second example we use years in numeric format for expressing ‘time’.

```
ggplot(try_data_frame(lynx, "year", as.numeric = TRUE),
       aes(x = time, y = V.lynx)) +
  geom_line()
```



Multivariate time series are also supported.

### 2.2.2 New stats

Package ‘*ggpmisc*’ provides new stats: `stat_peaks()`, `stat_valleys()`, and `stat_poly_eq()`. Peaks and valleys are local (or global) maxima and minima. These stats return the *x* and *y* values at the peaks or valleys plus suitable labels,

and default aesthetics that make easy their use with several different geoms, including `geom_point`, `geom_text`, `geom_label`, `geom_vline`, `geom_hline` and `geom_rug`, and also with geoms defined by package ‘`ggrepel`’. Some examples follow.

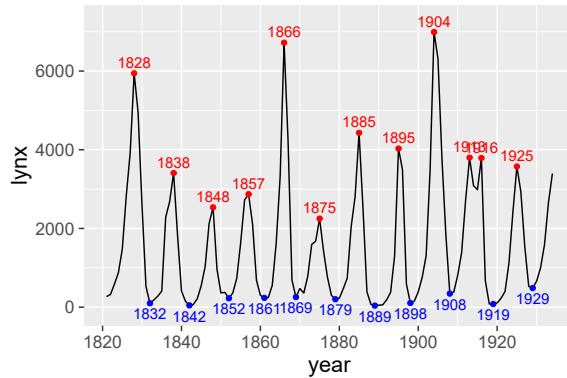
### 2.2.3 Peaks and valleys

There are many cases, for example in physics and chemistry, but also when plotting time-series data when we need to automatically locate and label local maxima (peaks) or local minima (valleys) in curves. The statistics presented here are useful only for dense data as they do not fit a peak function but instead simply search for the local maxima or minima in the observed data. However, they allow flexible generation of labels on both  $x$  and  $y$  peak or valley coordinates.

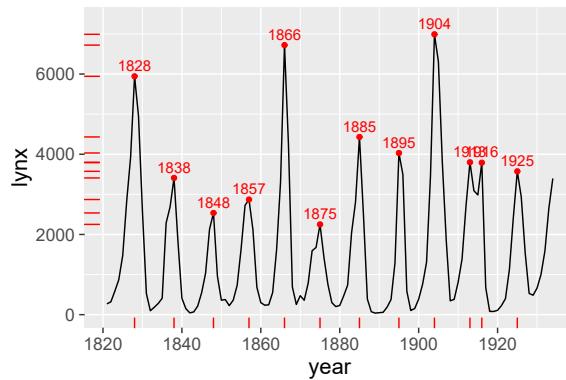
We use as example the same time series as above. In the next several examples we demonstrate some of this flexibility.

```
lynx.df <- data.frame(year = as.numeric(time(lynx)), lynx = as.matrix(lynx))
```

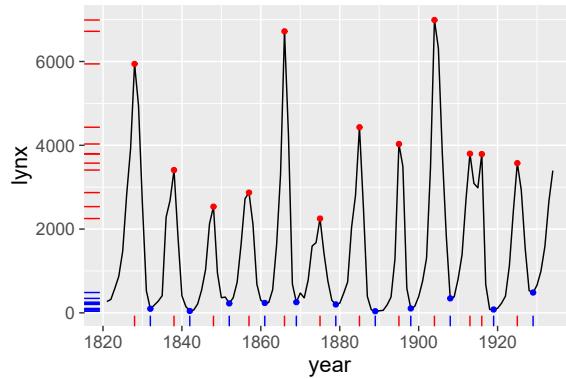
```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "text", colour = "red",
             vjust = -0.5, x.label(fmt = "%4.0f") +
  stat_valleys(colour = "blue") +
  stat_valleys(geom = "text", colour = "blue",
               vjust = 1.5, x.label(fmt = "%4.0f") +
  ylim(-100, 7300)
```



```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "rug", colour = "red") +
  stat_peaks(geom = "text", colour = "red",
             vjust = -0.5, x.label(fmt = "%4.0f") +
  ylim(NA, 7300)
```

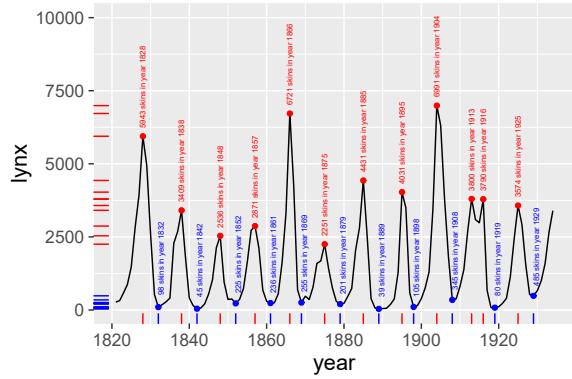


```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "rug", colour = "red") +
  stat_valleys(colour = "blue") +
  stat_valleys(geom = "rug", colour = "blue")
```



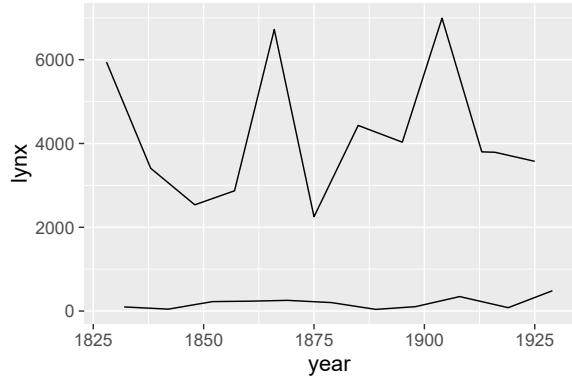
```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "rug", colour = "red") +
  stat_peaks(geom = "text", colour = "red",
             hjust = -0.1, label.fmt = "%4.0f",
             angle = 90, size = rel(2),
             aes(label = paste(..y.label..,
                               "skins in year", ..x.label..))) +
  stat_valleys(colour = "blue") +
  stat_valleys(geom = "rug", colour = "blue") +
  stat_valleys(geom = "text", colour = "blue",
               hjust = -0.1, vjust = 1, label.fmt = "%4.0f",
               angle = 90, size = rel(2)),
```

```
aes(label = paste(..y.label..,
                  "skins in year", ..x.label..)) +
  ylim(NA, 10000)
```



Of course, if one finds use for it, the peaks and/or valleys can be plotted on their own. Here we plot an "envelope" using `geom_line()`.

```
ggplot(lynx.df, aes(year, lynx)) +
  stat_peaks(geom = "line") + stat_valleys(geom = "line")
```



## 2.2.4 Equations as labels in plots

How to add a label with a polynomial equation including coefficient estimates from a model fit seems to be a frequently asked question in Stackoverflow. The parameter estimates are extracted automatically from a fit object corresponding to each *group* or panel in a plot and other aesthetics for the group respected. An aesthetic is provided for this, and only this. Such a statistics needs to be used

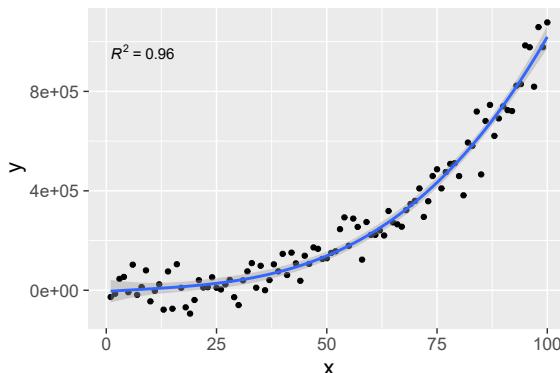
together with another geom or stat like geom smooth to add the fitted line. A different approach, discussed in Stackoverflow, is to write a statistics that does both the plotting of the polynomial and adds the equation label. Package ‘*ggpmisc*’ defines `stat_poly_eq()` using the first approach which follows the ‘rule’ of using one function in the code for a single action. In this case there is a drawback that the users is responsible for ensuring that the model used for the label and the label are the same, and in addition that the same model is fitted twice to the data.

We first generate some artificial data.

```
set.seed(4321)
# generate artificial data
x <- 1:100
y <- (x + x^2 + x^3) +
  rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, y,
  group = c("A", "B"),
  y2 = y * c(0.5, 2))
```

First an example using default arguments.

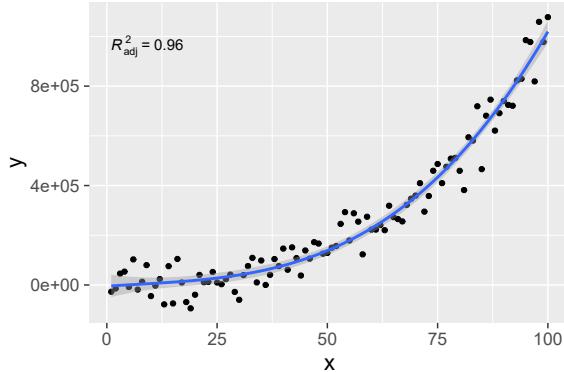
```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(formula = formula, parse = TRUE)
```



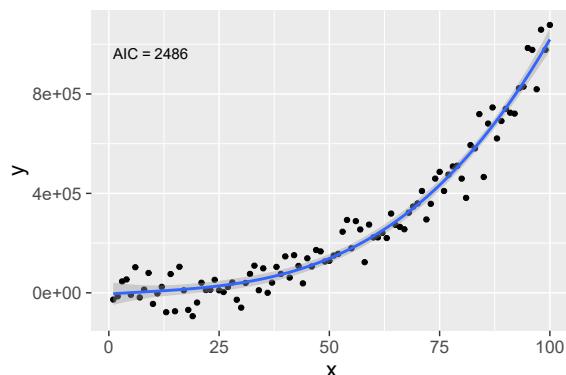
`stat_poly_eq()` makes available five different labels in the returned data frame.  $R^2$ ,  $R_{\text{mathrm{adj}}}^2$ , AIC, BIC and the polynomial equation.  $R^2$  is used by default, but `aes()` can be used to select a different one.

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
```

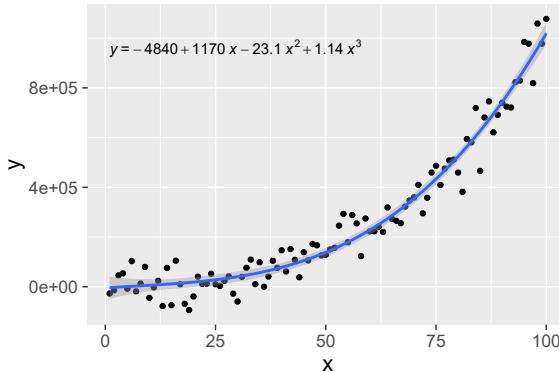
```
geom_smooth(method = "lm", formula = formula) +
stat_poly_eq(aes(label = ..adj.rr.label..),
formula = formula, parse = TRUE)
```



```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
geom_point() +
geom_smooth(method = "lm", formula = formula) +
stat_poly_eq(aes(label = ..AIC.label..),
formula = formula, parse = TRUE)
```

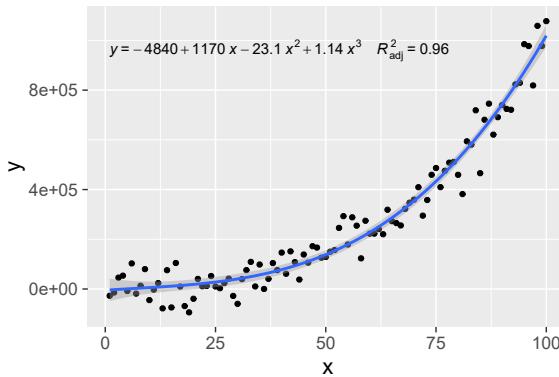


```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
geom_point() +
geom_smooth(method = "lm", formula = formula) +
stat_poly_eq(aes(label = ..eq.label..),
formula = formula, parse = TRUE)
```



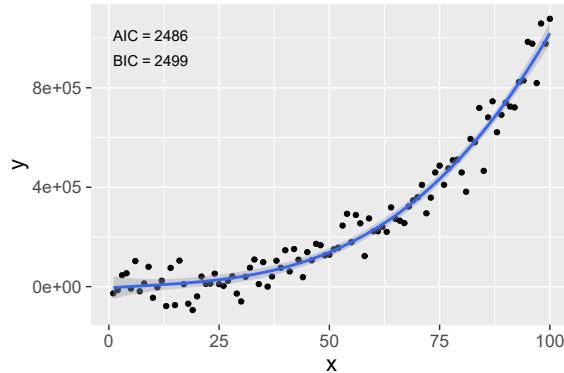
Within `aes()` it is possible to *compute* new labels based on those returned plus “arbitrary” text. The supplied labels are meant to be *parsed* into R expressions, so any text added should be valid for a string that will be parsed.

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste(..eq.label..,
                           ..adj.rr.label..,
                           sep = "~~~~")),
               formula = formula, parse = TRUE)
```



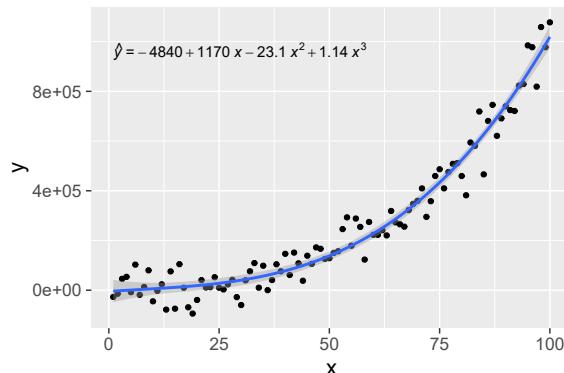
```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste("atop(", ..AIC.label.., ",",
                           ..BIC.label.., ")",
                           sep = "~~~~")),
               formula = formula, parse = TRUE)
```

```
sep = ""),
formula = formula, parse = TRUE)
```



Two examples of removing or changing the *lhs* and/or the *rhs* of the equation. (Be aware that the equals sign must be always enclosed in backticks in a string that will be parsed.)

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               eq.with.lhs = "italic(hat(y))`~`",
               formula = formula, parse = TRUE)
```

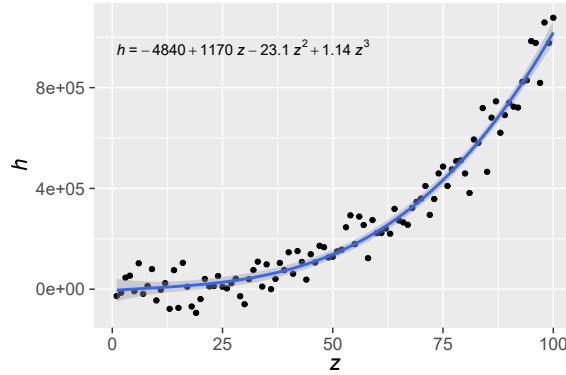


```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
```

## 2 Plots with ggpplot, ggrepel and ggpmisc

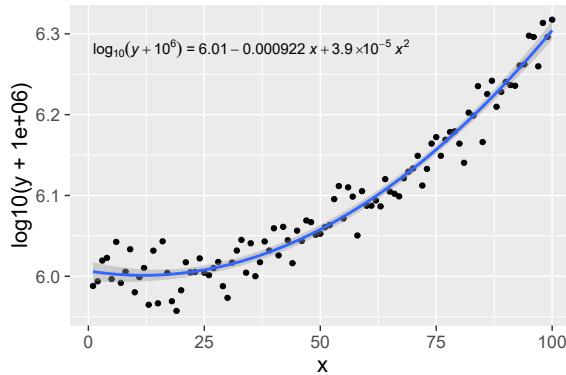
---

```
geom_smooth(method = "lm", formula = formula) +
  labs(x = expression(italic(z)), y = expression(italic(h))) +
  stat_poly_eq(aes(label = ..eq.label..),
               eq.with.lhs = "italic(h)^~~=~~",
               eq.x.rhs = "~italic(z)",
               formula = formula, parse = TRUE)
```



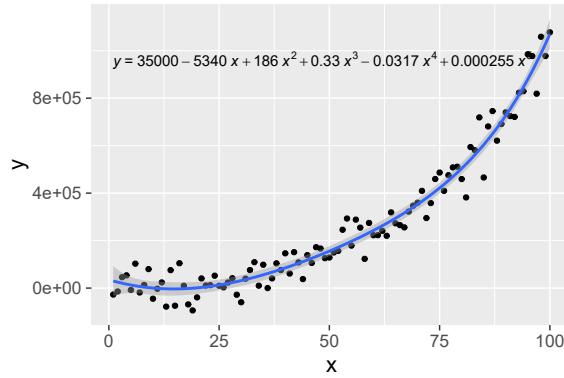
As any valid R expression can be used, Greek letters are also supported, as well as the inclusion in the label of variable transformations used in the model formula.

```
formula <- y ~ poly(x, 2, raw = TRUE)
ggplot(my.data, aes(x, log10(y + 1e6))) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               eq.with.lhs = "plain(log)[10](italic(y)+10^6)^~~=~~",
               formula = formula, parse = TRUE)
```



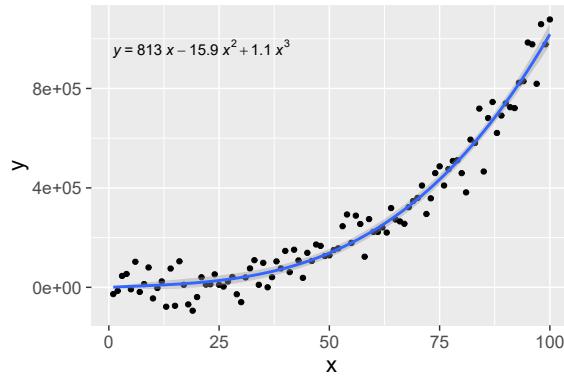
Example of a polynomial of fifth order.

```
formula <- y ~ poly(x, 5, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               formula = formula, parse = TRUE)
```



Intercept forced to zero—line through the origin.

```
formula <- y ~ x + I(x^2) + I(x^3) - 1
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               formula = formula, parse = TRUE)
```



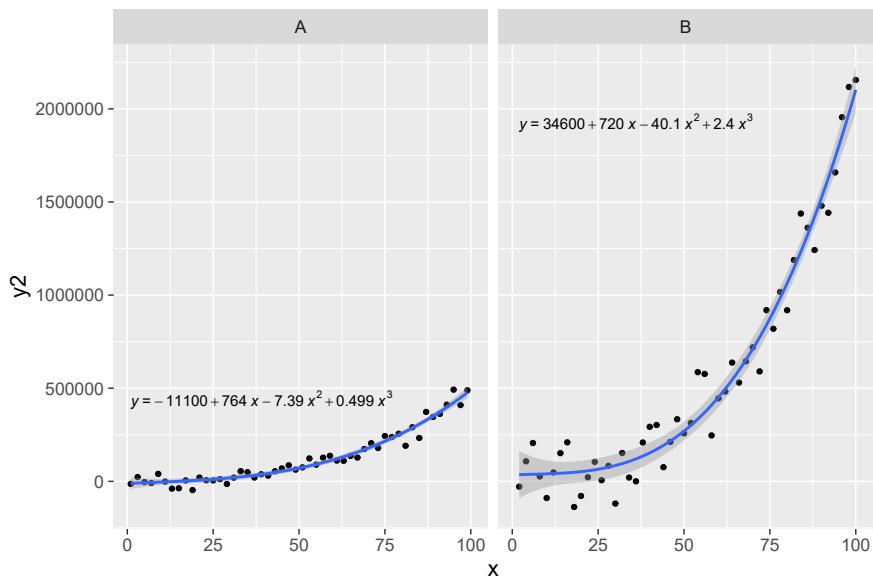
We give some additional examples to demonstrate how other components of the `ggplot` object affect the behaviour of this statistic.

Facets work as expected either with fixed or free scales. Although below we had to adjust the size of the font used for the equation.

## 2 Plots with `ggplot`, `ggrepel` and `ggpmisc`

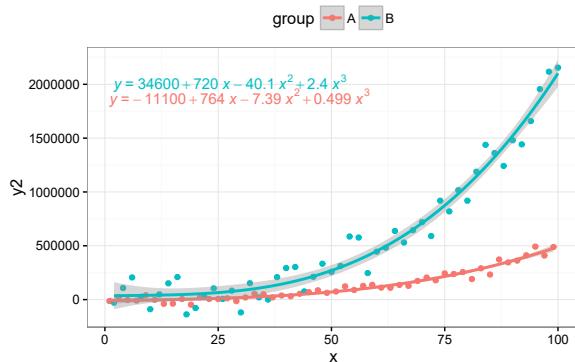
---

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y2)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..), # size = rel(2.8),
               formula = formula, parse = TRUE) +
  facet_wrap(~group)
```



Grouping, in this example using colour aesthetic also works as expected, but we need to use `vjust` to prevent the two equation labels to overlap.

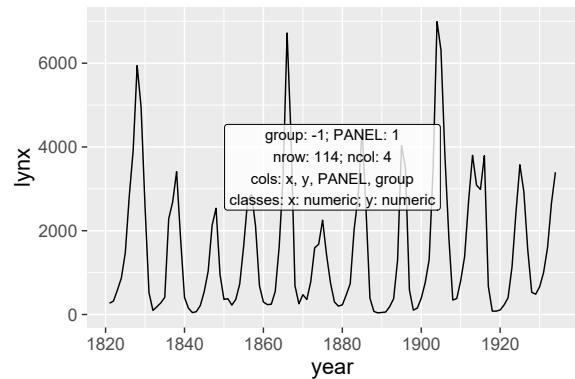
```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y2, colour = group)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..), vjust = c(-8, 0),
               formula = formula, parse = TRUE) +
  theme_bw() +
  theme(legend.position = "top")
```



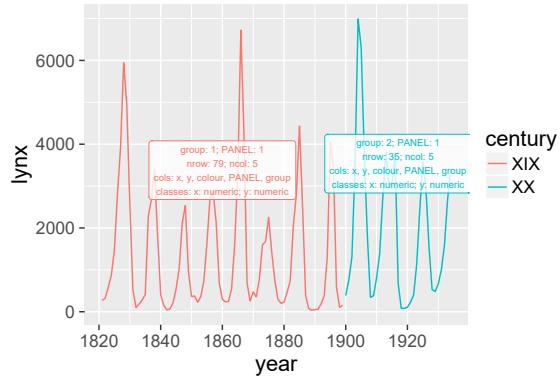
### 2.2.5 Learning and/or debugging

A very simple stat named `stat_debug()` can save the work of adding print statements to the code of stats to get information about what data is being passed to the `compute_group()` function. Because the code of this function is stored in a `ggproto` object, at the moment it is impossible to directly set breakpoints in it. This `stat_debug()` may also help users diagnose problems with the mapping of aesthetics in their code or just get a better idea of how the internals of ‘ggplot2’ work.

```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_debug_group(alpha = 0.8)
```



```
lynx.df$century <- ifelse(lynx.df$year >= 1900, "XX", "XIX")
ggplot(lynx.df, aes(year, lynx, color = century)) +
  geom_line() +
  stat_debug_group(alpha = 0.8, size = rel(2.5))
```



## 2.3 ‘ggrepel’

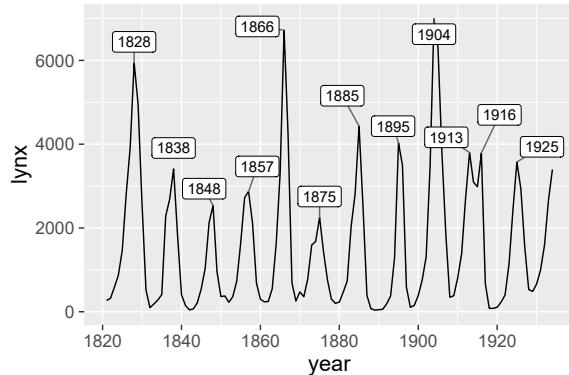
Package ‘`ggrepel`’ is under development by Kamil Slowikowski. It does a single thing, relocates text labels so that they do not overlap. This is achieved through two geometries that work similarly to those provided by ‘`ggplot2`’ except for the relocation. This is incredibly useful both when labeling peaks and valleys and when labeling points in scatter-plots. This is a significant problem in bioinformatics plots and in maps.

### 2.3.1 New geoms

Package ‘`ggrepel`’ provides two new geoms: `geom_text_repel` and `geom_label_repel`. They are used similarly to `geom_text` and `geom_label` but the text or labels “repel” each other so that they rarely overlap unless the plot is very crowded.

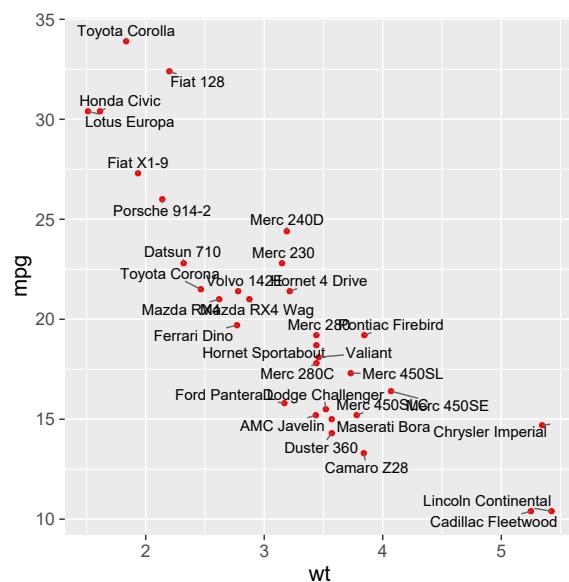
```
ggplot(lynx.df, aes(year, lynx)) +
  geom_line() +
  stat_peaks(geom = "label_repel", nudge_y = 500)
```

## 2.3 ggrepel



For the time being I reproduce here a couple of examples from the package vignette.

```
ggplot(mtcars, aes(wt, mpg)) +  
  geom_point(color = 'red') +  
  geom_text_repel(aes(label = rownames(mtcars)))
```

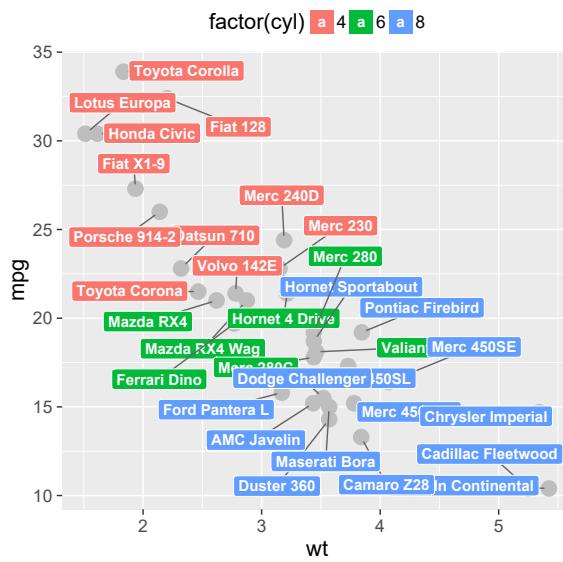


```
set.seed(42)  
ggplot(mtcars) +  
  geom_point(aes(wt, mpg), size = 5, color = 'grey') +  
  geom_label_repel(
```

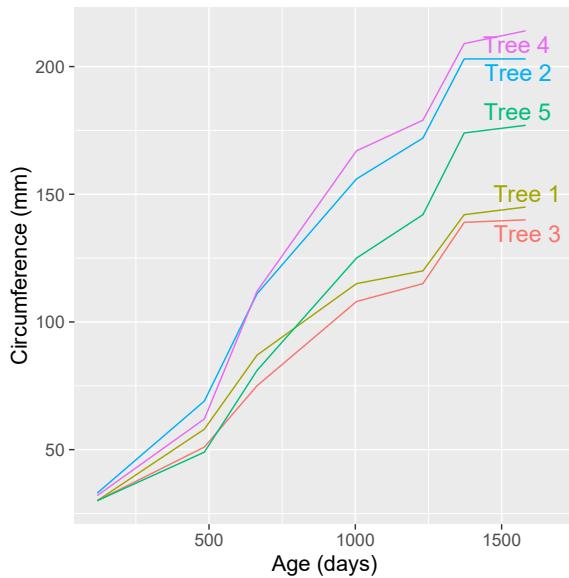
## 2 Plots with ggpplot, ggrepel and ggpmisc

---

```
aes(wt, mpg, fill = factor(cyl), label = rownames(mtcars)),
  fontface = 'bold', color = 'white',
  box.padding = unit(0.25, "lines"),
  point.padding = unit(0.5, "lines")) +
  theme(legend.position = "top")
```



```
set.seed(42)
ggplot(Orange, aes(age, circumference, color = Tree)) +
  geom_line() +
  coord_cartesian(xlim = c(min(Orange$age), max(Orange$age) + 90)) +
  geom_text_repel(data = subset(Orange, age == max(age)),
    aes(label = paste("Tree", Tree)),
    size = 6,
    nudge_x = 45,
    segment.color = NA) +
  theme(legend.position = "none") +
  labs(x = "Age (days)", y = "Circumference (mm)")
```



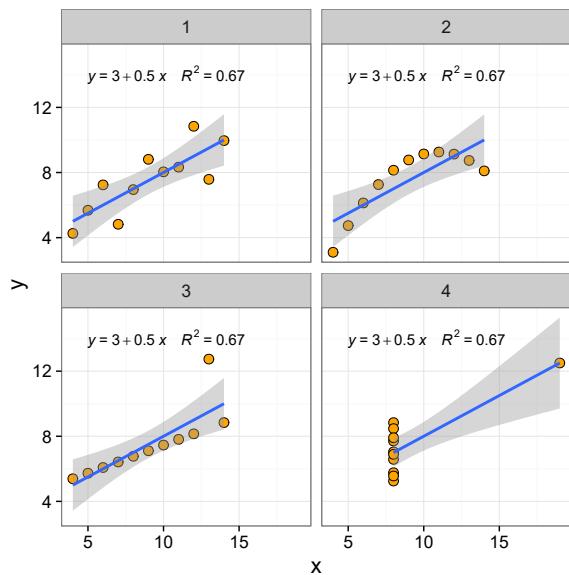
## 2.4 Examples

### 2.4.1 Anscombe's example revisited

To make the chapter self contained we repeat the code from chapter ??.

```
# we rearrange the data
my.mat <- matrix(as.matrix(anscombe), ncol=2)
my.anscombe <- data.frame(x = my.mat[, 1],
                           y = my.mat[, 2],
                           case=factor(rep(1:4, rep(11,4))))
```

```
ggplot(my.anscombe, aes(x,y)) +
  geom_point(shape=21, fill="orange", size=3) +
  geom_smooth(method="lm") +
  stat_poly_eq(formula = y ~ x, parse = TRUE,
               label.x = 5, label.y = 14,
               aes(label = paste(..eq.label.., ..rr.label.., sep = "~~~~")) ) +
  facet_wrap(~case, ncol=2) +
  theme_bw(16)
```



```
try(detach(package:ggpmisc))
try(detach(package:ggrepel))
try(detach(package:ggplot2))
```

# 3 Plotting maps with ggmap

## 3.1 Plotting data onto maps

```
library(ggmap)

## Loading required package: ggplot2

library(rgdal)

## Loading required package: methods
## Loading required package: sp
## rgdal: version: 1.1-3, (SVN revision 594)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
## Path to GDAL shared files: C:/Users/aphalo/Documents/R/win-library/3.2/rgdal/gdal
## GDAL does not use iconv for recoding strings.
## Loaded PROJ.4 runtime: Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]
## Path to PROJ.4 shared files: C:/Users/aphalo/Documents/R/win-library/3.2/rgdal/proj
## Linking to sp version: 1.2-1
```

Another extension to package `ggplot2` is package `ggmap`. Package `ggmap` makes it possible to plot data using normal `ggplot2` syntax on top of a map. Maps can be easily retrieved from the internet through different services. Some of these services require the user to register and obtain a key for access. As Google Maps do not require such a key for normal resolution maps, we use this service in the examples.

The first step is to fetch the desired map. One can fetch the maps base on any valid Google Maps search term, or by giving the coordinates at the center of the map. Although `zoom` defaults to "auto", frequently the best result is obtained by providing this argument. Valid values for `zoom` are integers in the range 1 to 20.

We will fetch maps from Google Maps. We have disabled the messages, to avoid repeated messages about Google's terms of use.

**Google Maps API Terms of Service:** <http://developers.google.com/maps/terms>

**Information from URL:** <http://maps.googleapis.com/maps/api/geocode/json?address=Europe&sensor=false>

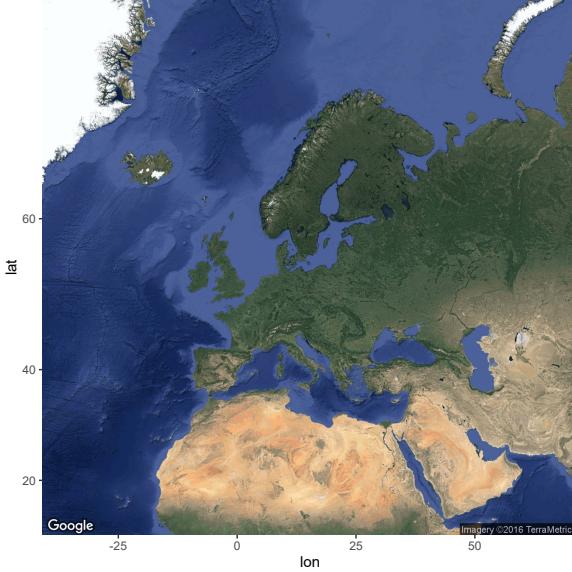
**Map from URL:** <http://maps.googleapis.com/maps/api/staticmap?center=Europe&zoom=3&size=%20640x640&scale=%202&maptyle=terrain&sensor=false>

We start by fetching and plotting a map of Europe of type `satellite`. We use the default extent `panel`, and also the extent `device` and `normal`. The `normal` plot includes axes showing the coordinates, while `device` does not show them, while `panel` shows axes but the map fits tightly into the drawing area:

```
Europe1 <- get_map("Europe", zoom=3, maptype="satellite")
ggmap(Europe1)

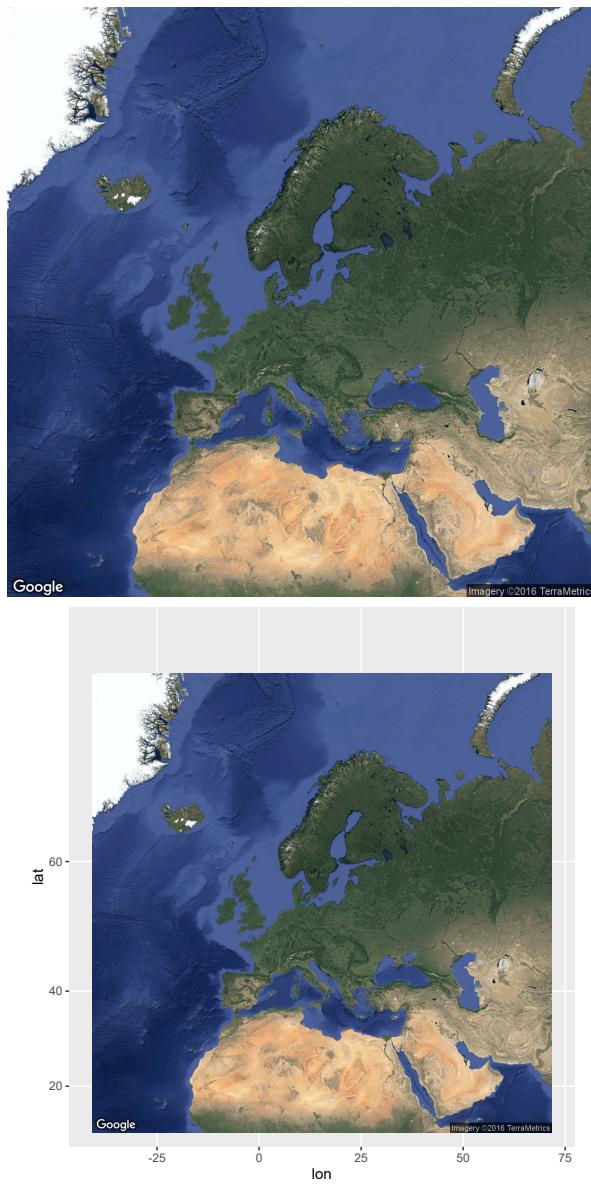
ggmap(Europe1, extent = "device")

ggmap(Europe1, extent = "normal")
```



### 3.1 Plotting data onto maps

---



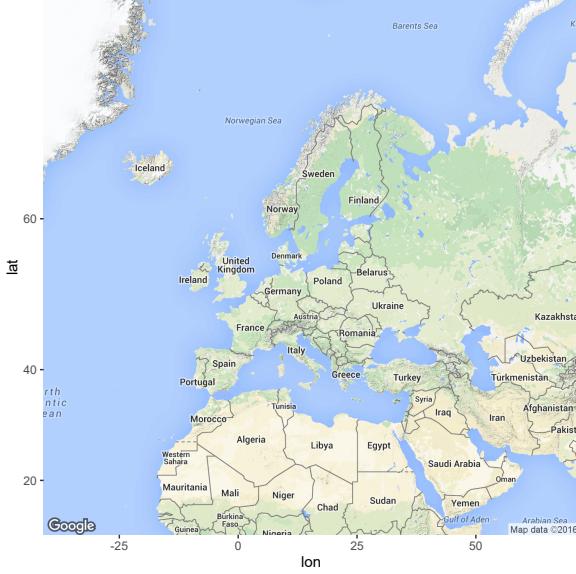
To demonstrate the option to fetch a map in black and white instead of the default colour version, we use a map of Europe of type **terrain**.

```
Europe2 <- get_map("Europe", zoom=3,  
                    maptype="terrain")
```

### 3 Plotting maps with ggmap

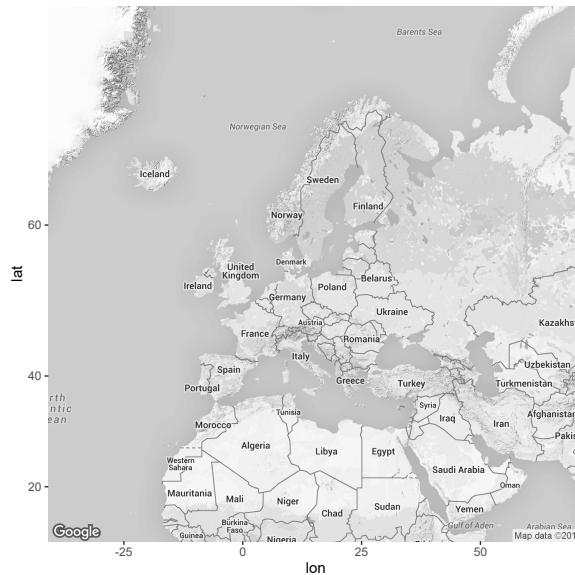
---

```
ggmap(Europe2)  
Europe3 <- get_map("Europe", zoom=3,  
                     maptype="terrain",  
                     color="bw")  
ggmap(Europe3)
```



### 3.1 Plotting data onto maps

---



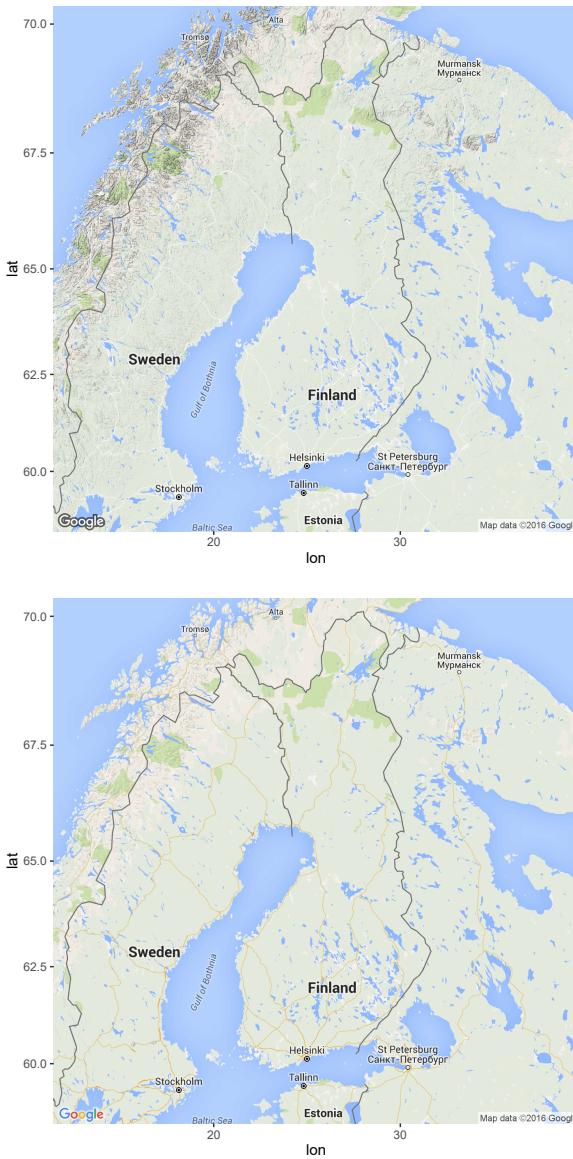
To demonstrate the difference between type `roadmap` and the default type `terrain`, we use the map of Finland. Note that we search for “Oulu” instead of “Finland” as Google Maps takes the position of the label “Finland” as the center of the map, and clips the northern part. By means of `zoom` we override the default automatic zooming onto the city of Oulu.

```
Finland1 <- get_map("Oulu", zoom=5, maptype="terrain")
ggmap(Finland1)

Finland2 <- get_map("Oulu", zoom=5, maptype="roadmap")
ggmap(Finland2)
```

### 3 Plotting maps with ggmap

---



We can even search for a street address, and in this case with high zoom value, we can see the building where one of us works:

```
BIO3 <- get_map("Viikinkaari 1, 00790 Helsinki",
                 zoom=18,
```

### 3.1 Plotting data onto maps

---

```
maptype="satellite")
ggmap(BI03)
```



We will now show a simple example of plotting data on a map, first by explicitly giving the coordinates, and in the second example we show how to fetch from Google Maps coordinate values that can be then plotted. We use function `geocode`. In one example we use `geom_point` and `geom_text`, while in the second example we use `annotate`, but either approach could have been used for both plots:

```
viikki <- get_map("Viikki, 00790 Helsinki",
                   zoom=15,
                   maptype="satellite")

our_location <- data.frame(lat=c(60.225, 60.227),
                            lon=c(25.017, 25.018),
                            label=c("BI03", "field"))

ggmap(viikki, extent = "normal") +
  geom_point(data=our_location, aes(y=lat, x=lon),
             size=4, colour="yellow") +
  geom_text(data=our_location, aes(y=lat, x=lon, label=label),
            hjust=-0.3, colour="yellow")

our_geocode <- geocode("Viikinkaari 1, 00790 Helsinki")
ggmap(viikki, extent = "normal") +
  annotate(geom="point",
          y=our_geocode[ 1, "lat"], x=our_geocode[ 1, "lon"],
          size=4, colour="yellow") +
```

### 3 Plotting maps with ggmap

---

```
annotate(geom="text",
        y=our_geocode[ 1, "lat"], x=our_geocode[ 1, "lon"],
        label="BIO3", hjust=-0.3, colour="yellow")
```



Using `get_map` from package `ggmap` for drawing a world map is not possible at the

### 3.1 Plotting data onto maps

---

time of writing. In addition a worked out example of how to plot shape files, and how to download them from a repository is suitable as our final example. We also show how to change the map projection. The example is adapted from a blog post at <http://rpsychologist.com/working-with-shapefiles-projections-and-world-maps-in-ggplot>.

We start by downloading the map data archive files from <http://www.naturalearthdata.com> which is available in different layers. We only use three of the available layers: ‘physical’ which describes the coastlines and a grid and bounding box, and ‘cultural’ which gives country borders. We save them in a folder with name ‘maps’, which is expected to already exist. After downloading each file, we unzip it.

```
getwd()
## [1] "L:/Own_manuscripts/Books/using-r"

oldwd <- setwd("./maps")

url_path <-
# "http://www.naturalearthdata.com/download/110m/"
"http://www.naturalearthdata.com/http://www.naturalearthdata.com/download/110m/"

download.file(paste(url_path,
                     "physical/ne_110m_land.zip",
                     sep=""), "ne_110m_land.zip")
unzip("ne_110m_land.zip")

download.file(paste(url_path,
                     "cultural/ne_110m_admin_0_countries.zip",
                     sep=""), "ne_110m_admin_0_countries.zip")
unzip("ne_110m_admin_0_countries.zip")

download.file(paste(url_path,
                     "physical/ne_110m_graticules_all.zip",
                     sep=""), "ne_110m_graticules_all.zip")
unzip("ne_110m_graticules_all.zip")

setwd(oldwd)
```

We list the layers that we have downloaded.

```
ogrListLayers(dsn="./maps")

## [1] "ne_110m_admin_0_countries"
## [2] "ne_110m_graticules_1"
## [3] "ne_110m_graticules_10"
## [4] "ne_110m_graticules_15"
## [5] "ne_110m_graticules_20"
## [6] "ne_110m_graticules_30"
## [7] "ne_110m_graticules_5"
## [8] "ne_110m_land"
## [9] "ne_110m_wgs84_bounding_box"
```

### 3 Plotting maps with ggmap

---

```
## attr(),"driver")
## [1] "ESRI Shapefile"
## attr(),"nlayers")
## [1] 9
```

Next we read the layer for the coastline, and use `fortify` to convert it into a data frame. We also create a second version of the data using the Robinson projection.

```
wmap <- readOGR(dsn=".maps", layer="ne_110m_land")

## OGR data source with driver: ESRI Shapefile
## Source: ".maps", layer: "ne_110m_land"
## with 127 features
## It has 2 fields

wmap.data <- fortify(wmap)

## Regions defined for each Polygons

wmap_robin <- spTransform(wmap, CRS("+proj=robin"))
wmap_robin.data <- fortify(wmap_robin)

## Regions defined for each Polygons
```

We do the same for country borders,

```
countries <- readOGR("./maps", layer="ne_110m_admin_0_countries")

## OGR data source with driver: ESRI Shapefile
## Source: "./maps", layer: "ne_110m_admin_0_countries"
## with 177 features
## It has 63 fields

countries.data <- fortify(countries)

## Regions defined for each Polygons

countries_robin <- spTransform(countries, CRS("+init=ESRI:54030"))
countries_robin.data <- fortify(countries_robin)

## Regions defined for each Polygons
```

and for the graticule at 15° intervals, and the bounding box.

```
grat <- readOGR("./maps", layer="ne_110m_graticules_15")

## OGR data source with driver: ESRI Shapefile
## Source: "./maps", layer: "ne_110m_graticules_15"
## with 35 features
## It has 5 fields
```

### 3.1 Plotting data onto maps

---

```
grat.data <- fortify(grat)
grat_robin <- spTransform(grat, CRS("+proj=robin"))
grat_robin.data <- fortify(grat_robin)

bbox <- readOGR("./maps", layer="ne_110m_wgs84_bounding_box")

## OGR data source with driver: ESRI Shapefile
## Source: "./maps", layer: "ne_110m_wgs84_bounding_box"
## with 1 features
## It has 2 fields

bbox.data <- fortify(bbox)

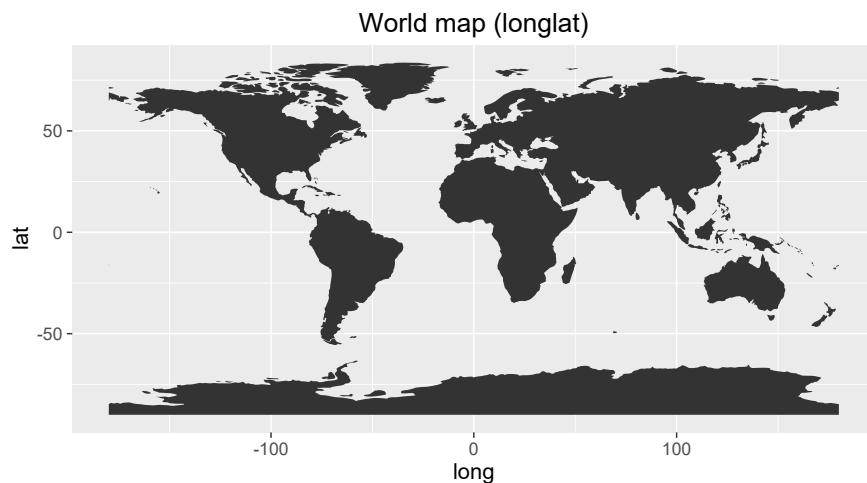
## Regions defined for each Polygons

bbox_robin <- spTransform(bbox, CRS("+proj=robin"))
bbox_robin.data <- fortify(bbox_robin)

## Regions defined for each Polygons
```

Now we plot the world map of the coastlines, on a longitude and latitude scale, as a ggplot using geom\_polygon.

```
ggplot(wmap.data, aes(long,lat, group=group)) +
  geom_polygon() +
  labs(title="World map (longlat)") +
  coord_equal()
```

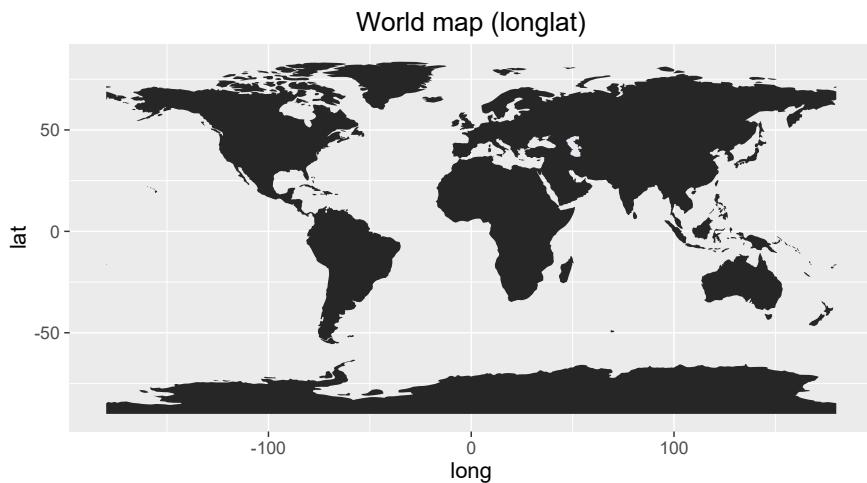


### 3 Plotting maps with ggmap

---

There is one noticeable problem in the map shown above: the Caspian sea is missing. We need to use aesthetic `fill` and a manual scale to correct this.

```
ggplot(wmap.data, aes(long,lat, group=group, fill=hole)) +  
  geom_polygon() +  
  labs(title="World map (longlat)") +  
  scale_fill_manual(values=c("#262626", "#e6e8ed"),  
                    guide="none") +  
  coord_equal()
```



When plotting a map using a projection, many default elements of the `ggplot` theme need to be removed, as the data is no longer in units of degrees of latitude and longitude and axes and their labels are no longer meaningful.

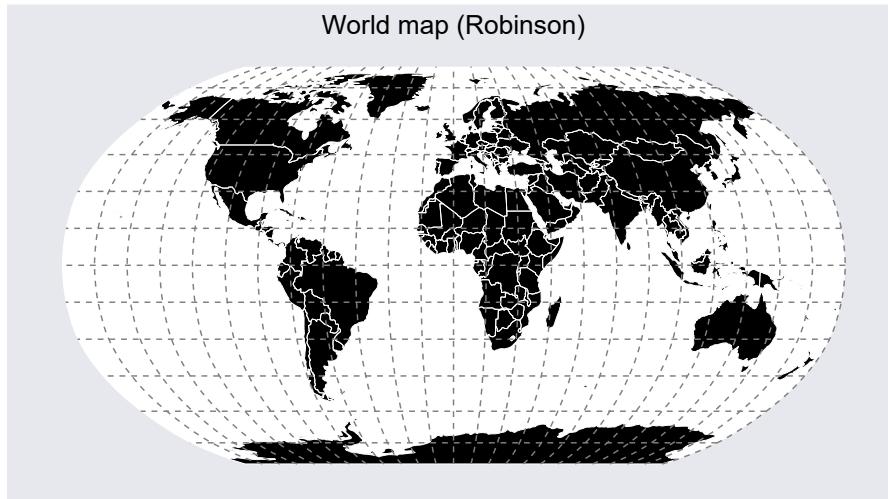
```
theme_map_opts <-  
  list(theme(panel.grid.minor = element_blank(),  
            panel.grid.major = element_blank(),  
            panel.background = element_blank(),  
            plot.background = element_rect(fill="#e6e8ed"),  
            panel.border = element_blank(),  
            axis.line = element_blank(),  
            axis.text.x = element_blank(),  
            axis.text.y = element_blank(),  
            axis.ticks = element_blank(),  
            axis.title.x = element_blank(),  
            axis.title.y = element_blank()))
```

### 3.1 Plotting data onto maps

---

Finally we plot all the layers using the Robinson projection. This is still a `ggplot` and consequently one can plot data on top of the map, being aware of the transformation of the scale needed to make the data location match locations in a map using a certain projection.

```
ggplot(bbox_robin.data, aes(long,lat, group=group)) +
  geom_polygon(fill="white") +
  geom_polygon(data=countries_robin.data,
    aes(long,lat, group=group,
        fill=hole)) +
  geom_path(data=countries_robin.data,
    aes(long,lat, group=group, fill=hole),
    color="white",
    size=0.3) +
  geom_path(data=grat_robin.data,
    aes(long, lat, group=group, fill=NULL),
    linetype="dashed",
    color="grey50") +
  labs(title="World map (Robinson)") +
  coord_equal() +
  theme_map_opts +
  scale_fill_manual(values=c("black", "white"),
    guide="none")
```



```
try(detach(package:ggmap))
try(detach(package:rgdal))
```



## **4 Further reading about R**

**4.1 Introductory texts**

**4.2 Texts on specific aspects**

**4.3 Advanced texts**



# Bibliography

- Anscombe, F. J. (1973). ‘Graphs in Statistical Analysis’. In: *The American Statistician* 27.1, p. 17. DOI: 10.2307/2682899. URL: <http://dx.doi.org/10.2307/2682899> (cit. on p. 56).
- Chang, W. (2013). *R Graphics Cookbook*. 1-2. Sebastopol: O'Reilly Media, p. 413. ISBN: 9781449316952. URL: <http://medcontent.metapress.com/index/A65RM03P4874243N.pdf> (cit. on p. 7).
- Dalgaard, P. (2008). *Introductory Statistics with R*. Springer, p. 380. ISBN: 0387790543.
- Everitt, B. and T. Hothorn (2011). *An Introduction to Applied Multivariate Analysis with R*. Springer, p. 288. ISBN: 1441996494. URL: <http://www.amazon.co.uk/Introduction-Applied-Multivariate-Analysis-Use/dp/1441996494>.
- Faraway, J. J. (2004). *Linear Models with R*. Boca Raton, FL: Chapman & Hall/CRC, p. 240. URL: <http://www.maths.bath.ac.uk/~jjf23/LMR/>.
- Faraway, J. J. (2006). *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. Chapman & Hall/CRC Taylor & Francis Group, p. 345. ISBN: 158488424X.
- Fox, J. (2002). *An {R} and {S-Plus} Companion to Applied Regression*. Thousand Oaks, CA, USA: Sage Publications. URL: <http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html>.
- Fox, J. and H. S. Weisberg (2010). *An R Companion to Applied Regression*. SAGE Publications, Inc, p. 472. ISBN: 141297514X. URL: <http://www.amazon.com/An-R-Companion-Applied-Regression/dp/141297514X>.
- Ihaka, R. and R. Gentleman (1996). ‘R: A Language for Data Analysis and Graphics’. In: *J. Comput. Graph. Stat.* 5, pp. 299–314.
- Matloff, N. (2011). *The Art of R Programming: A Tour of Statistical Software Design*. No Starch Press, p. 400. ISBN: 1593273843. URL: <http://www.amazon.com/The-Art-Programming-Statistical-Software/dp/1593273843>.
- Murrell, P. (2011). *R Graphics, Second Edition (Chapman & Hall/CRC The R Series)*. CRC Press, p. 546. ISBN: 1439831769. URL: <http://www.amazon.com/Graphics-Second-Edition-Chapman-Series/dp/1439831769>.
- Pinheiro, J. C. and D. M. Bates (2000). *Mixed-Effects Models in S and S-Plus*. New York: Springer.
- Teator, P. (2011). *R Cookbook*. 1st ed. Sebastopol: O'Reilly Media, p. 436. ISBN: 9780596809157.

## Bibliography

---

- Wickham, H. (2014). *Advanced R*. Chapman & Hall/CRC The R Series. CRC Press. ISBN: 9781466586970. URL: <https://books.google.fi/books?id=G5PNBQAAQBAJ>.
- (2015). *R Packages*. O'Reilly Media. ISBN: 9781491910542. URL: <https://books.google.fi/books?id=eq0xBwAAQBAJ>.
- Xie, Y. (2013). *Dynamic Documents with R and knitr (Chapman & Hall/CRC The R Series)*. Chapman and Hall/CRC, p. 216. ISBN: 1482203537. URL: <http://www.amazon.com/Dynamic-Documents-knitr-Chapman-Series/dp/1482203537>.
- Zuur, A. F., E. N. Ieno and E. Meesters (2009). *A Beginner's Guide to R*. 1st ed. Springer, p. 236. ISBN: 0387938362. URL: <http://www.amazon.com/Beginners-Guide-Use-Alain-Zuur/dp/0387938362>.