Learn R

…as you learnt your mother tongue

Learn R

…as you learnt your mother tongue

Git hash: 719ae12; Git date: 2016-10-22 23:50:49 +0300

Pedro J. Aphalo

Helsinki, 5 November 2016

Draft, 75% done
Available through Leanpub

© 2001–2016 by Pedro J. Aphalo
Licensed under one of the Creative Commons licenses as indicated, or when not explicitly indicated, under the CC BY-SA 4.0 license.

Typeset with XeLaTeX in Lucida Bright and Lucida Sans using the KOMA-Script book class.
The manuscript was written using R with package knitr. The manuscript was edited in WinEdt and RStudio. The source files for the whole book are available at https://bitbucket.org/aphalo/using-r.

Contents

Preface

> *Do not struggle, just play! If going gets difficult and frustrating, take a break! If you get a new insight, take a break to enjoy the victory!*
>
> — Learning like a child

This book covers different aspects of the use of R. They are meant to be used possibly as a complement to a course or book, as explanations are rather short and terse. I do not discuss here statistics, just Ras a tool and language for data manipulation and display. The idea is for you to learn the Rlanguage like children learn a language: they work-out what the rules are, simply by listening to people speak and trying to utter what they want to tell their parents. I do give some explanations and comments, but the idea of these notes is mainly for you to use the numerous examples to find-out by yourself the overall patterns and coding philosophy behind the Rlanguage. Instead of parents being the sound board for your first utterances in R, the computer will play this role. You should look and try to repeat the examples, and then try your own hand and see how the computer responds, does it understand you or not?

When teaching I tend to lean towards challenging students rather than telling a simple story. I do the same here, because it is what I prefer as a student, and how I learn best myself. Not everybody learns best with the same approach, for me the most limiting factor is for what I listen to, or read, to be in a way or another challenging or entertaining enough to keep my thoughts focused. This I achieve best when making an effort to understand the contents or to follow the thread of teh plot of a story. So, be warned, reading this book will be about exploring a new world, this book aims to be a travel guide, neither a traveler's account, nor a cookbook of R recipes.

Do not expect to ever know everything about R! Rin a broad sense is vast because its capabilities can be expanded with independently developed packages. Currently there are close to ten thousand packages available for free. You just need to learn what you need. Being Rvery popular there is nowadays lots of information available, plus a helpful and open minded on-line community willing to help with those difficult problems for which Goggle will not be of help.

How to read this book? My idea is that you will run all the code ex-

amples and try as many other variations as needed until you are sure to understand the basic 'rules' of the R language and how each function or command described works. In R for each function, data set, etc. there is a help page available. In addition, if you use a front-end like RStudio, auto-completion is available as well as balloon help on the arguments accepted by functions. For scripts, there is syntax checking of the source code before its execution: *possible* mistakes and even formatting style problems are highlighted in the editor window. Error messages tend to be terse in R, and may require some lateral thinking and/or 'experimentation' to understand the real cause behind problems. When you are not sure to understand how some command works, it is useful in many cases to try simple examples for which you know the correct answer and see if you can reproduce them in R.

I recommend you to use as an editor or IDE (integrated development environment) RStudio. RStudio is user friendly, actively maintained, and available both in desktop and server versions. The desktop version runs on Windows, Linux, and OS X and other Unixes. In addition it is available for free! R itself also runs under all these operating systems and a few more. Being R a command line application in its simplest incarnation, it can be made to work on what nowadays are frugal computing resources equivalent to a personal computer of a couple of decades ago. Nowadays R can be made to run even on the Raspberry Pi, a Linux micro-controller board with the processing power of a modest smartphone. At the other end of the spectrum on really powerful servers it can be used for the analysis of big data sets with millions of observations. How powerful a computer you will need will depend on the size of the data sets to analyze and on how patient you are.

When searching for answers, asking for advice or reading books you will be confronted with different ways of doing the same tasks. Do not this overwhelm you, in most cases it will not matter as many computations can be done in R, as in any language, in several different ways, still obtaining the same result. The different approaches may differ mainly in two aspects: 1) how readable to humans are the instructions given to the computer as part of a script or program, and 2) how fast the code will run. Unless performance is an important bottleneck in your work, just concentrate on writing code that is easy to understand to you and to others, and consequently easy to check and reuse. Of course do always check any code you write for mistakes, preferably using actual numerical test cases for any complex calculation or even relatively simple scripts. Testing and validation are extremely important steps in data analysis, so get into this habit while reading this book. Testing how every function works as I will challenge you to do in this book, is at the core of any robust data analysis

or computing programming. In addition, when writing computer code, as for any other text intended for humans to read, consistent writing style and formatting go a long way in making your intentions clear.

These notes are work-in-progress. I will appreciate suggestions for further examples, notification of errors and unclear sections and also any larger 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. I warmly thank the students that over the years have asked the questions and posed the problems that have helped me write this text and correct the mistakes and voids of previous versions. I have also received help on on-line forums and in person from numerous people, learnt from archived e-mail list messages, blog posts, books, articles, tutorials, webinars, and by struggling to solve some new problems on my own. In many ways this text owes much more to people who are not authors than to myself. However, as I am the one who has written this version and decided what to include and exclude, as author, I take full responsibility for any errors and inaccuracies.

I have been using R since around 1998 or 1999, but I am still constantly learning new things about R itself and R packages. With time it has replaced in my work as a researcher and teacher several other pieces of software: SPSS, Systat, Origin, Excel, and it has become a central piece of the tool set I use for producing lecture slides, notes, books and even web pages. This is to say that it is the most useful piece of software and programming language I have ever learnt to use. Of course, in time it will be replaced by something better, but at the moment it is the "hot" thing to learn for anybody with a need to analyse and display data.

Status as of 2016-10-23. I have updated the manuscript to track package updates since the previous version uploaded nearly five months ago, and added some examples of the new functionality added to packages 'ggpmisc', 'ggrepel', and to some extent 'ggplot2'. There is a new section on package 'viridis'.

With respect to the chapter *Storing and manipulating data with R* I have put it on hold until I can see a soon to be published book covering the same subject. Hadley Wickham has named the set of tools developed by him and his collaborators as *tidyverse* to be described in the book titled *R for Data Science* by Grolemund and Wickham (O'Reilly).

An important update to 'ggplot2'is scheduled to be released in a few

weeks time, and will change to some extent the behavior of existing functions, specially faceting is to become extensible through other packages. I will write the sections on faceting based on this new version of 'ggplot2'.

I expect the next update to take place in one or two months from now, and to be a more substantial one. The present one, adds only a few tens of pages.

1 Extensions to '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)
library(viridis)
library(ggrepel)
library(ggpmisc)
library(gganimate)
library(xts)
library(MASS)
```

We set a font larger size than the default

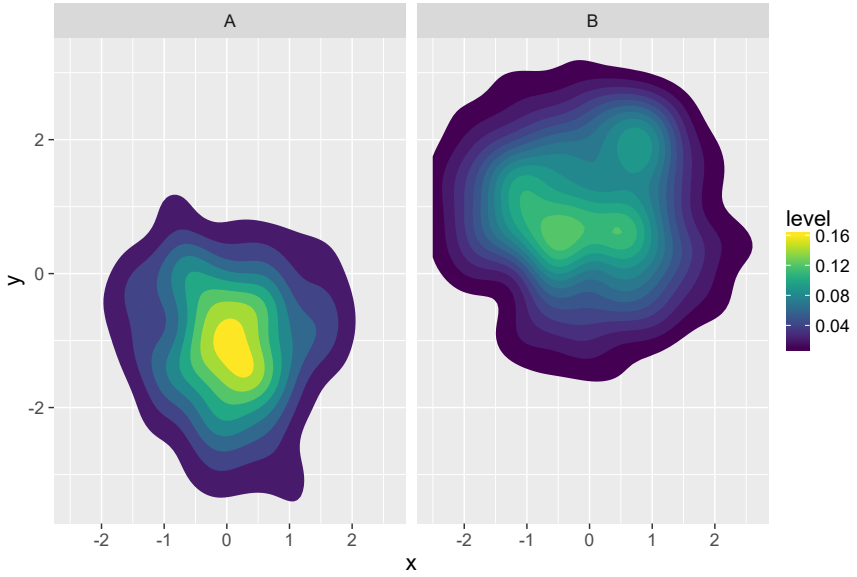
```
theme_set(theme_grey(16))
```

1.2 'viridis'

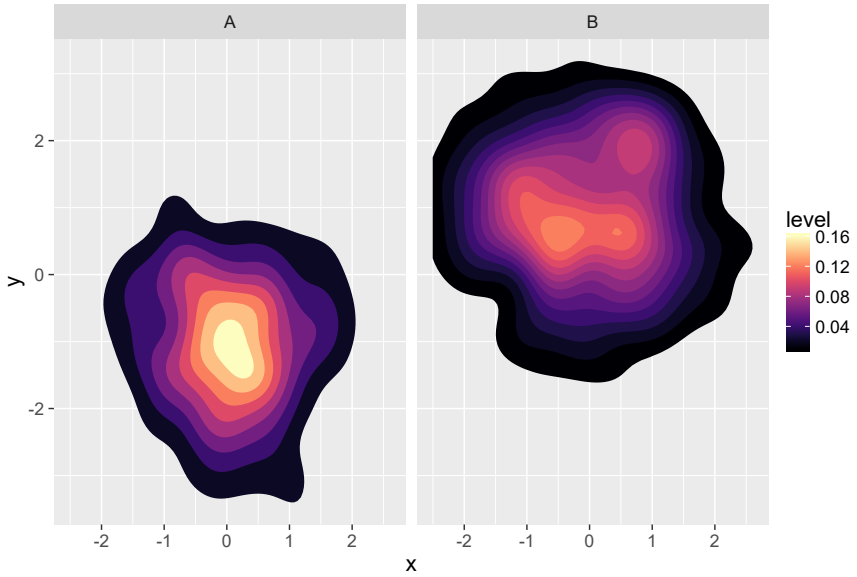
Package 'viridis'defines color palettes and fill and color scales with colour selected based on human perception, with special consideration of visibility for those with different kinds of color blindness and well as in grey-scale reproduction.

```
set.seed(56231)
my.data <-
  data.frame(x = rnorm(500),
             y = c(rnorm(250, -1, 1), rnorm(250, 1, 1)),
             group = factor(rep(c("A", "B"), c(250, 250))) )
```

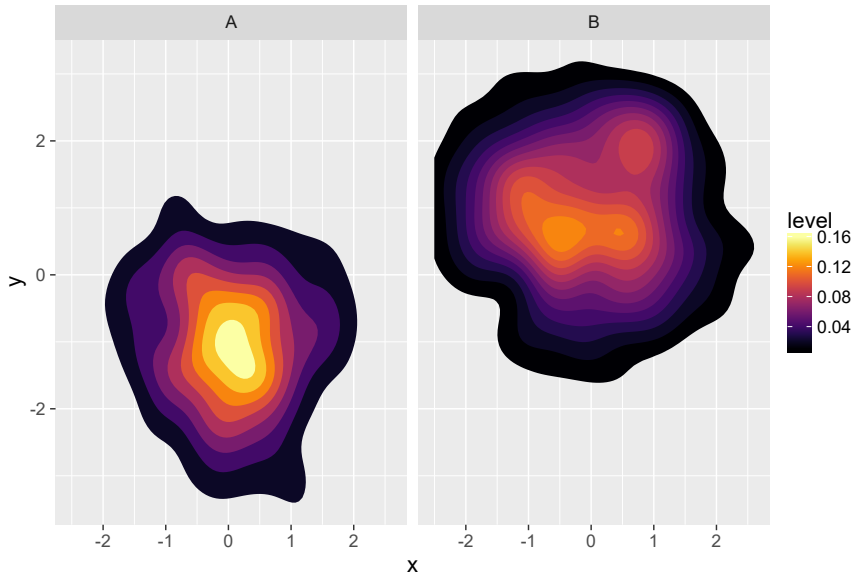
```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_viridis()
```



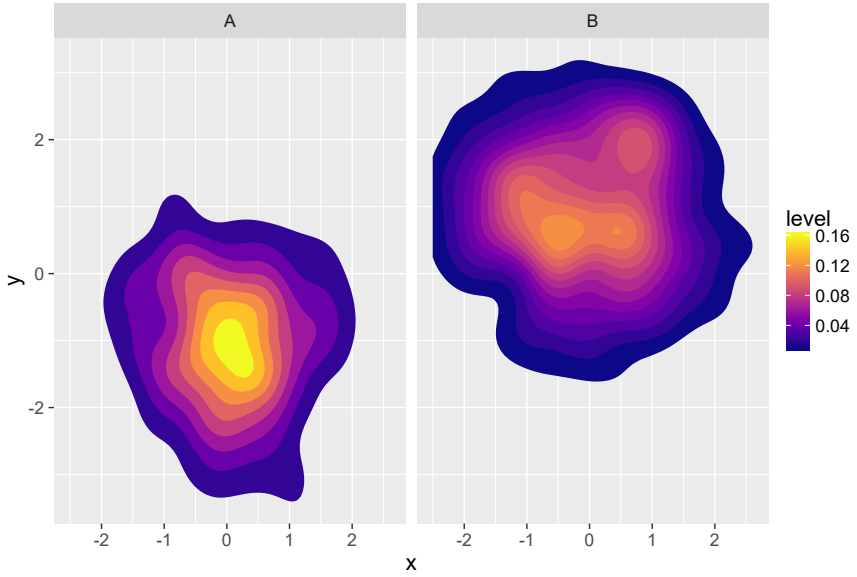

```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_viridis(option = "magma")
```



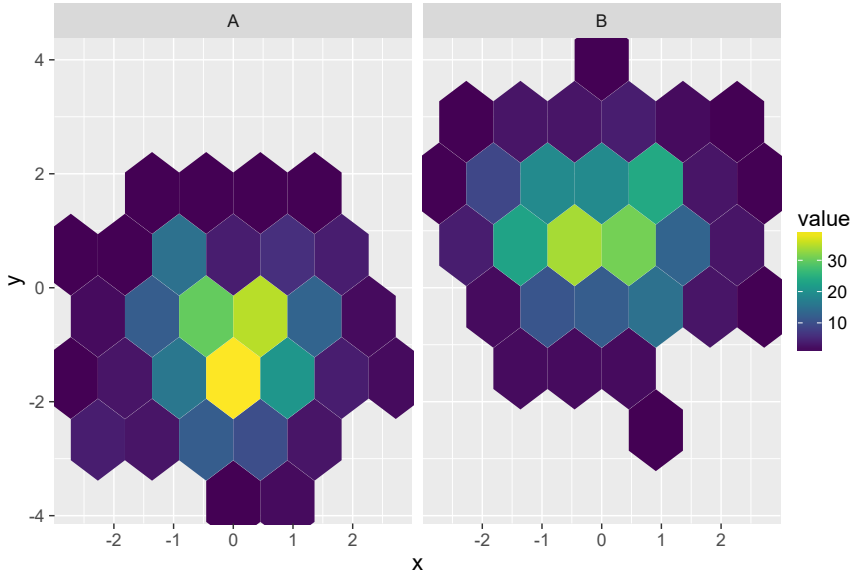
```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_viridis(option = "inferno")
```




```
ggplot(my.data, aes(x, y)) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon") +
  facet_wrap(~group) +
  scale_fill_viridis(option = "plasma")
```




```
ggplot(my.data, aes(x, y)) +
  geom_hex(bins = 6) +
  facet_wrap(~group) +
  scale_fill_viridis()

## Loading required package: methods
```



1.3 'gganimate'

Package 'gganimate' allows the use of package 'animation' in ggplots with a syntax consistent with the grammar of graphics. It adds a new aesthetic `frame`, which can be used to map *groups* of data to *frames* in the animation.

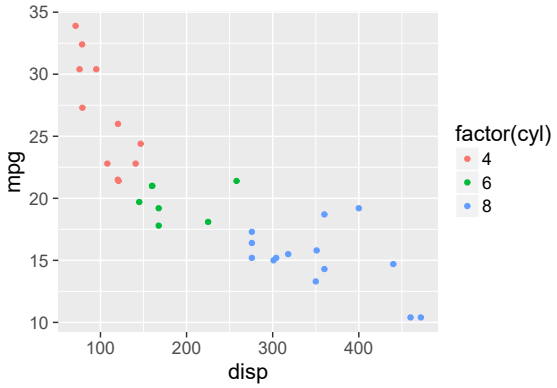
Use of the package is extremely easy, but installation can be somehow tricky because of system requirements. Just, make sure to have ImageMagic installed and included in the search `PATH`.

We modify an example from section ?? on page ??. We add the `frame` aesthetic to the earlier figure.

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

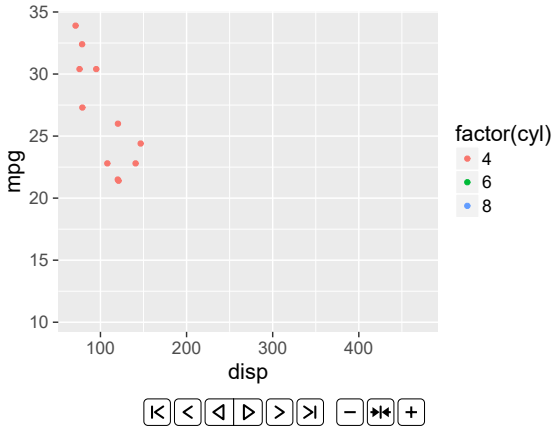
Now we can print `p` as a normal plot,

```
p
```



Or display an animation. The animation will look differently depending on the output format, and the program used for viewing it. For example, in this PDF files, the animation will work when viewed with Adobe Viewer or Adobe Acrobat but not in Sumatra PDF viewer. We add `title_frame = FALSE` as a title does not seem useful in this simple animation.

```
gg_animate(p, title_frame = FALSE)
```



Or save it to a file.

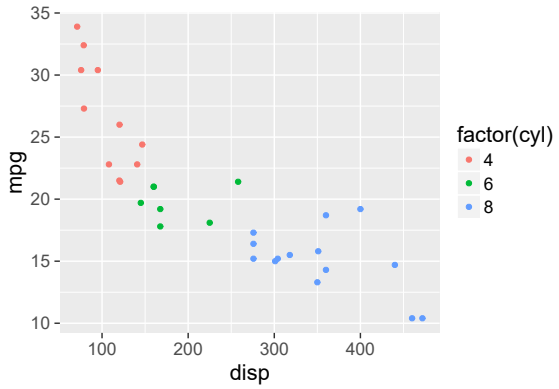
```
gg_animate(p, "p-animation.gif")
```

Cumulative animations are also supported. We here use the same example with three frames, but this type of animation is particularly effective for time series data. To achieve this we only need to add `cumulative = TRUE` to the aesthetics mappings.

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

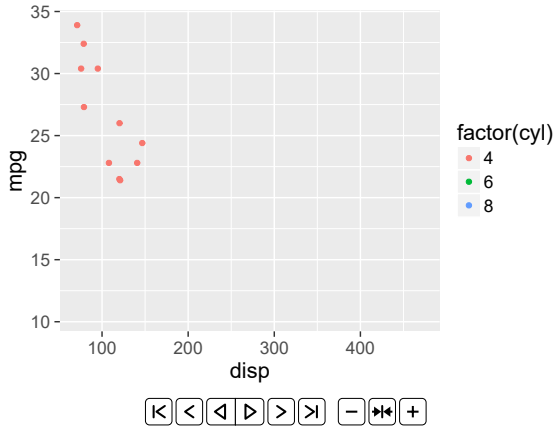
Now we can print `p` as a normal plot,

```
p
```



Or display an animation. The animation will look differently depending on the output format, and the program used for viewing it. For example, in this PDF files, the animation will work when viewed with Adobe Viewer or Adobe Acrobat but in Sumatra PDF viewer.

```
gg_animate(p, title_frame = FALSE)
```



1.4 'ggstance'

coming soon.

1.5 'ggbiplot'

coming soon.

1.6 'ggalt'

coming soon.

1.7 'ggExtra'

coming soon.

1.8 'ggfortify'

coming soon.

1.9 'ggradar'

coming soon.

1.10 'ggseas'

coming soon.

1.11 'ggsci'

coming soon.

1.12 'ggthemes'

coming soon.

1.13 'ggnetwork'

coming soon.

1.14 'ggpmisc'

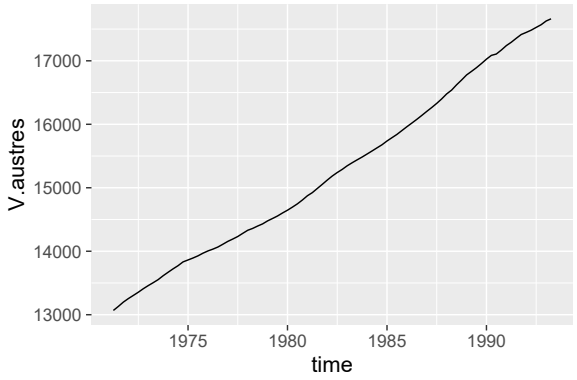
Package 'ggpmisc'is a package developed by myself as a result of questions from work mates and in Stackoverflow, or functionality that I have needed in my own research or for teaching. It provides new stats for everyday use: `stat_peaks()`, `stat_valleys()`, `stat_poly_eq()`, `stat_fit_glance()`, `stat_fit_deviations()`, and `stat_fit_augment()`. A function for converting time-series data to a data frame that can be easily plotted with 'ggplot2'. It also provides some debugging tools that echo the data received as input: `stat_debug_group()`, `stat_debug_panel()`, and codegeom_debug(), and `geom_null()` that does not plot its input.

1.14.1 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.

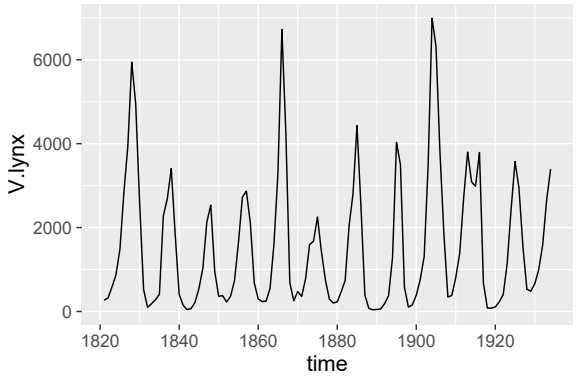
We exemplify this with some of the time series data included in R. In the first example we use the default format for time.

```
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.

1.14.2 Peaks and valleys

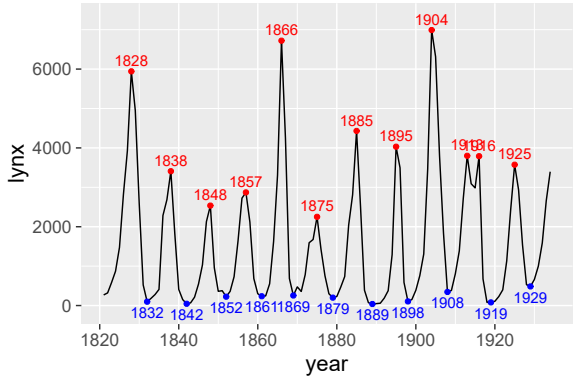
Peaks and valleys are local (or global) maxima and minima. These stats return the x and y values at the peaks or valleys plus suitable labels, and default aesthetics that make easy their use with several different geoms, including `geom_point`, `geom_text`, `geom_label`, `geom_vline`, `geom_hline` and `geom_rug`, and also with geoms defined by package 'ggrepel'. Some examples follow.

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.

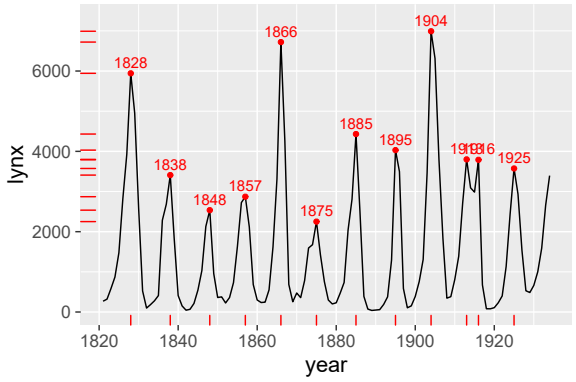
We use as example the same time series as above. In the next several examples we demonstrate some of this flexibility.

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

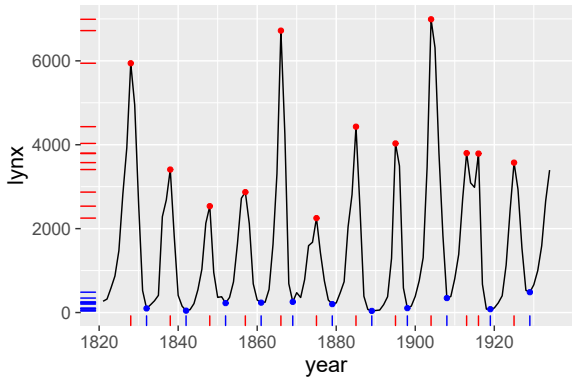
```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "text", colour = "red",
             vjust = -0.5, x.label.fmt = "%4.0f") +
  stat_valleys(colour = "blue") +
  stat_valleys(geom = "text", colour = "blue",
               vjust = 1.5, x.label.fmt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

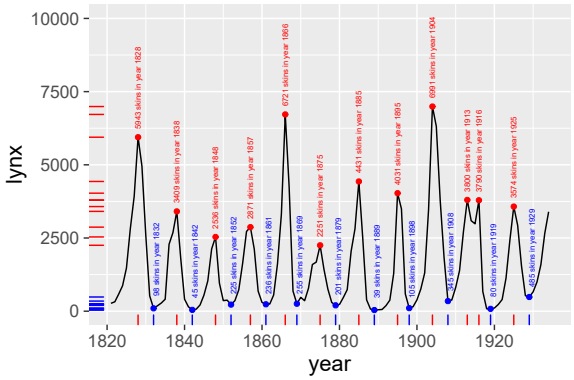



```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "rug", colour = "red") +
  stat_valleys(colour = "blue") +
  stat_valleys(geom = "rug", colour = "blue")
```



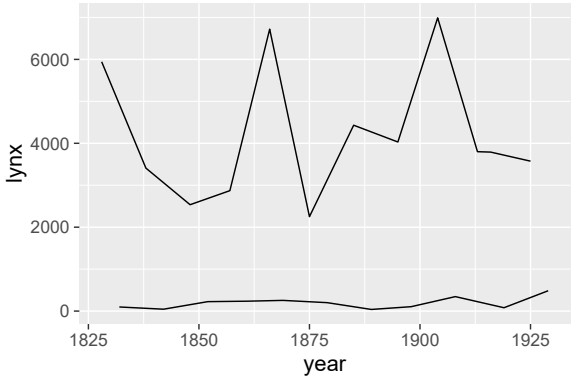

```
ggplot(lynx.df, aes(year, lynx)) + geom_line() +
  stat_peaks(colour = "red") +
  stat_peaks(geom = "rug", colour = "red") +
  stat_peaks(geom = "text", colour = "red",
             hjust = -0.1, label.fmt = "%4.0f",
             angle = 90, size = rel(2),
             aes(label = paste(..y.label..,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



1.14.3 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.

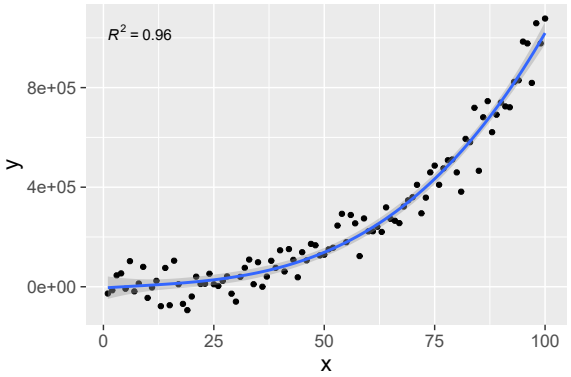
We first generate some artificial data.

```
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))
```

Linear models

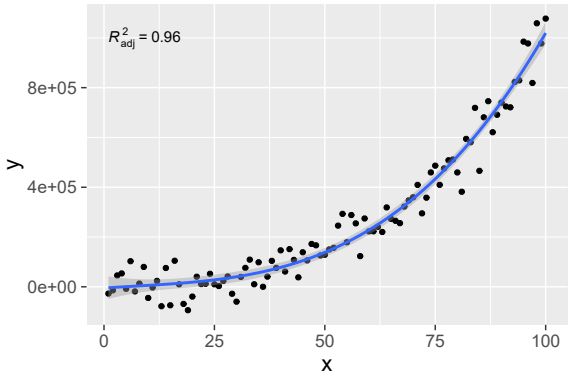
This section shows examples of linear models with one independent variables, including different polynomials. We first give an example using default arguments.

```
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)
```

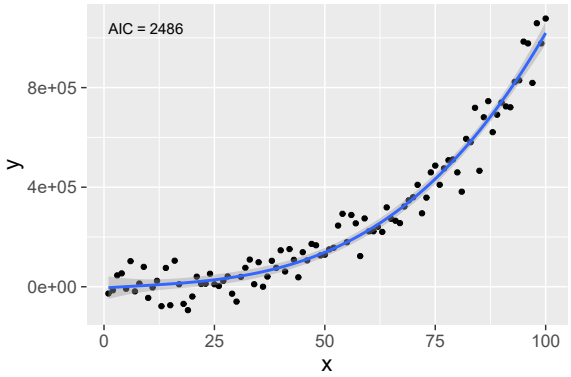


`stat_poly_eq()` makes available five different labels in the returned data frame. R^2, $R_mathrmadj^2$, AIC, BIC and the polynomial equation. R^2 is used by default, but `aes()` can be used to select a different one.

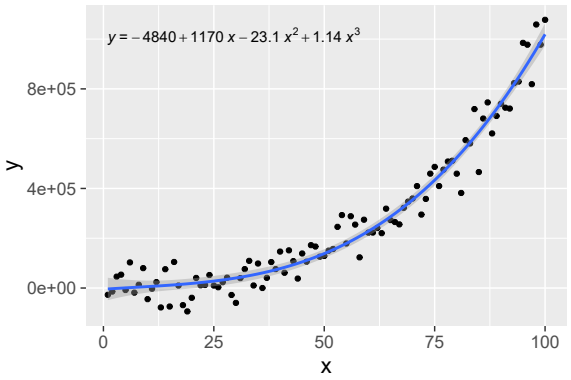
```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..adj.rr.label..),
               formula = formula, parse = TRUE)
```




```
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)
```

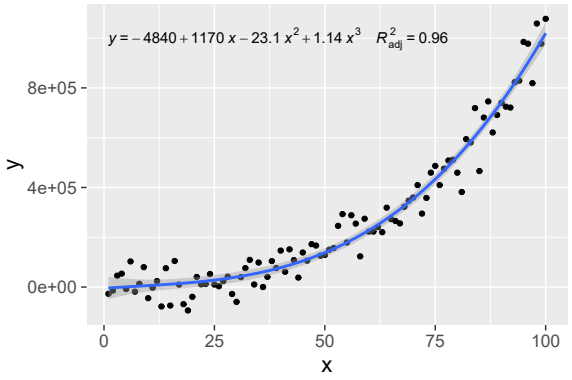


```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               formula = formula, parse = TRUE)
```

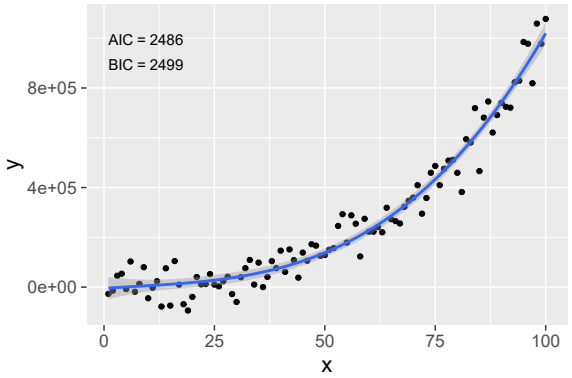


Within `aes()` it is possible to *compute* new labels based on those returned plus "arbitrary" text. The supplied labels are meant to be *parsed* into R expressions, so any text added should be valid for a string that will be parsed.

```
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)
```



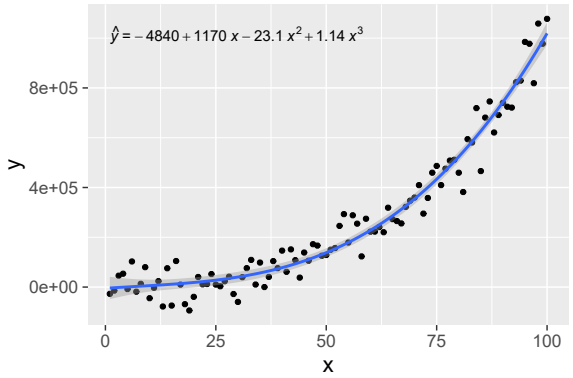

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = paste("atop(", ..AIC.label.., ",",
                                 ..BIC.label.., ")",

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



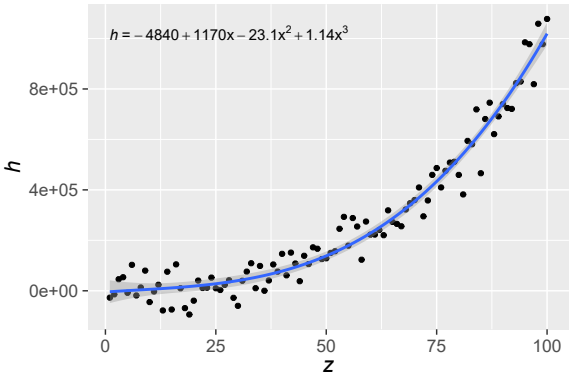
Two examples of removing or changing the *lhs* and/or the *rhs* of the equation. (Be aware that the equals sign must be always enclosed in backticks in a string that will be parsed.)

```
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)
```

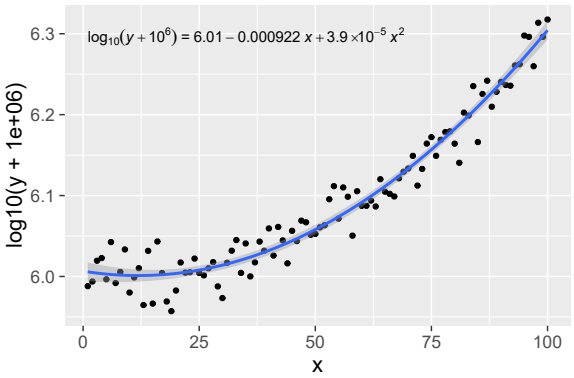



```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  labs(x = expression(italic(z)), y = expression(italic(h)) ) +
  stat_poly_eq(aes(label = ..eq.label..),
               eq.with.lhs = "italic(h)~`=`~",
         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



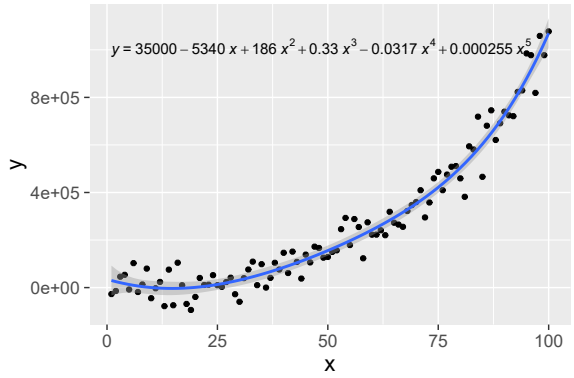
As any valid R expression can be used, Greek letters are also supported, as well as the inclusion in the label of variable transformations used in the model formula.

```
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)
```



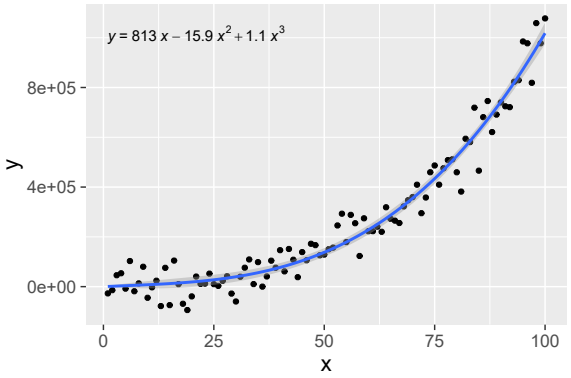
Example of a polynomial of fifth order.

```
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.

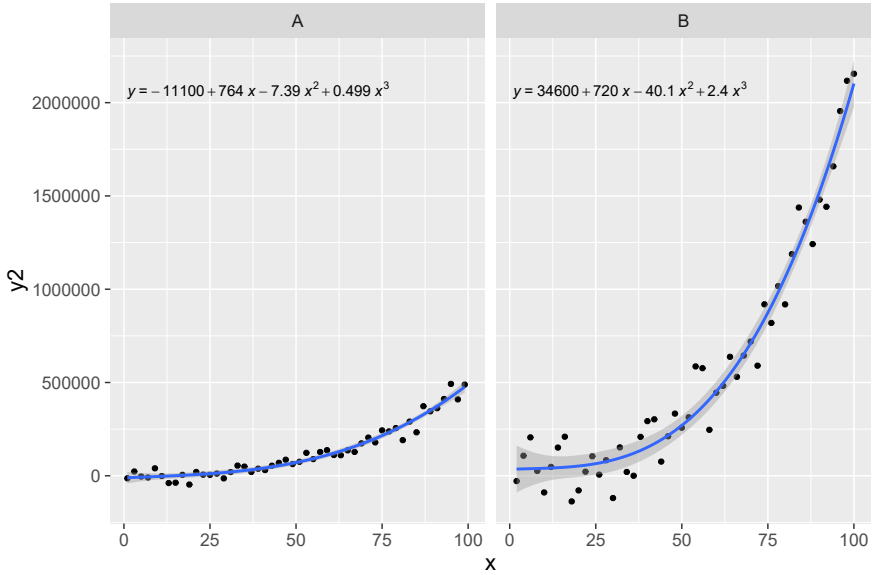
```
formula <- y ~ x + I(x^2) + I(x^3) - 1
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_poly_eq(aes(label = ..eq.label..),
               formula = formula, parse = TRUE)
```



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

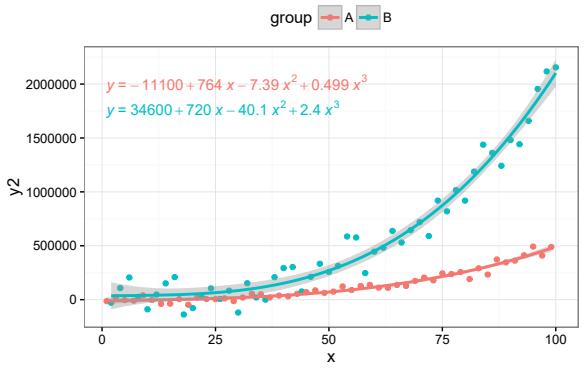
Facets work as expected either with fixed or free scales. Although bellow we had to adjust the size of the font used for the equation.

```
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 = 2.8,
               formula = formula, parse = TRUE) +
  facet_wrap(~group)
```



Grouping, in this example using colour aesthetic also works as expected.

```
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..),
               formula = formula, parse = TRUE) +
  theme_bw() +
  theme(legend.position = "top")
```



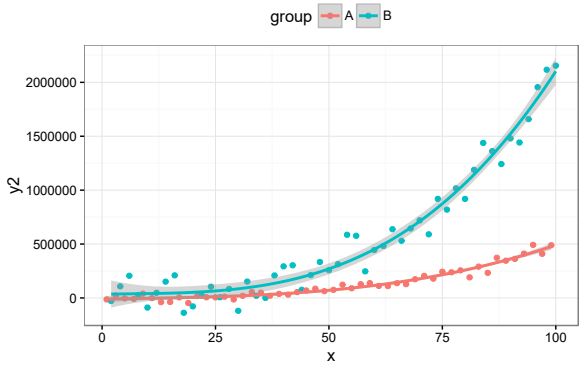
Other types of models

Another statistic, `stat_fit_glance()` allows lots of flexibility, but at the moment there is no equivalently flexible version of `stat_smooth()`.

We give an example with a linear model, showing a P-value (a frequent request for which I do not find much use).

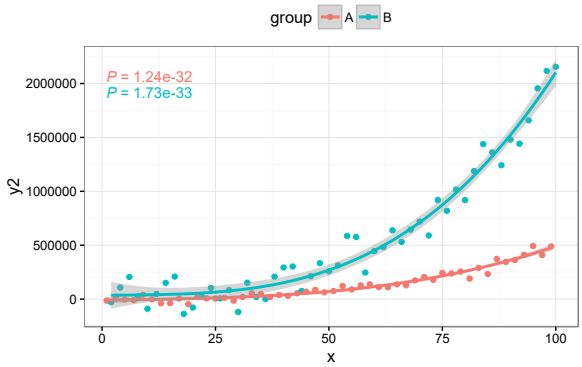
We use `geom_debug()` to find out what values `stat_glance()` returns for our linear model, and add labels with P-values for the fits.

```
formula <- y ~ x + I(x^2) + I(x^3)
ggplot(my.data, aes(x, y2, colour = group)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_fit_glance(method.args = list(formula = formula),
                  geom = "debug",
                  summary.fun = print,
                  summary.fun.args = list()) +
  theme_bw() +
  theme(legend.position = "top")

## Input 'data' to 'geom_debug()':

##    colour hjust vjust r.squared adj.r.squared
## 1 #F8766D     0   1.4 0.9619032     0.9594187
## 2 #00BFC4     0   2.8 0.9650270     0.9627461
##       sigma statistic      p.value df    logLik
## 1  29045.57  387.1505 1.237801e-32  4 -582.6934
## 2 118993.86  423.0996 1.732868e-33  4 -653.2037
##        AIC      BIC     deviance df.residual x
## 1 1175.387 1184.947  38807664340          46 1
## 2 1316.407 1325.968 651338752799          46 1
##         y PANEL group
## 1 2154937     1     1
## 2 2154937     1     2
##    colour hjust vjust r.squared adj.r.squared
## 1 #F8766D     0   1.4 0.9619032     0.9594187
## 2 #00BFC4     0   2.8 0.9650270     0.9627461
##       sigma statistic      p.value df    logLik
## 1  29045.57  387.1505 1.237801e-32  4 -582.6934
## 2 118993.86  423.0996 1.732868e-33  4 -653.2037
##        AIC      BIC     deviance df.residual x
## 1 1175.387 1184.947  38807664340          46 1
## 2 1316.407 1325.968 651338752799          46 1
##         y PANEL group
## 1 2154937     1     1
## 2 2154937     1     2
```



Using the information now at hand we create some labels.

```
formula <- y ~ x + I(x^2) + I(x^3)
ggplot(my.data, aes(x, y2, colour = group)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
        stat_fit_glance(aes(label     =     paste('italic(P)~`=`~',     sig-
nif(..p.value.., 3)), sep = ""),
                  parse = TRUE,
                  method.args = list(formula = formula),
                  geom = "text") +
  theme_bw() +
  theme(legend.position = "top")
```

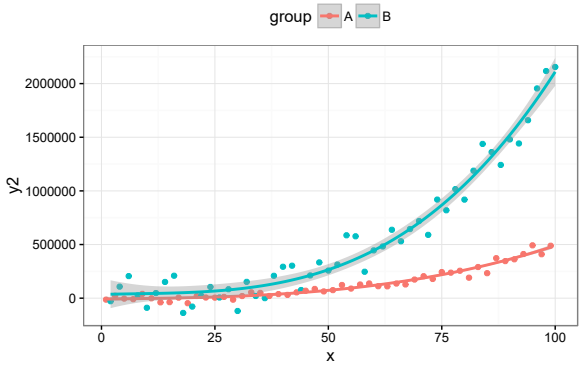


We use `geom_debug()` to find out values `stat_glance()` returns for our resistant linear model fitted with `MASS:rlm()`.

```
formula <- y ~ x + I(x^2) + I(x^3)
ggplot(my.data, aes(x, y2, colour = group)) +
  geom_point() +
```

```
  geom_smooth(method = "rlm", formula = formula) +
  stat_fit_glance(method.args = list(formula = formula),
                  geom = "debug",
                  method = "rlm",
                  summary.fun = print,
                  summary.fun.args = list()) +
  theme_bw() +
  theme(legend.position = "top")
```

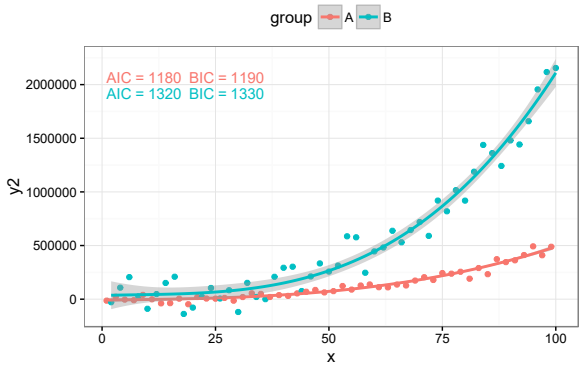
```
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
## Input 'data' to 'geom_debug()':

##    colour hjust vjust     sigma converged
## 1 #F8766D     0   1.4  20078.62      TRUE
## 2 #00BFC4     0   2.8 126111.74      TRUE
##      logLik      AIC      BIC     deviance x
## 1 -582.8362 1175.672 1185.232  39029842201 1
## 2 -653.2392 1316.478 1326.039 652263183741 1
##         y PANEL group
## 1 2154937     1     1
## 2 2154937     1     2
##    colour hjust vjust     sigma converged
## 1 #F8766D     0   1.4  20078.62      TRUE
## 2 #00BFC4     0   2.8 126111.74      TRUE
##      logLik      AIC      BIC     deviance x
## 1 -582.8362 1175.672 1185.232  39029842201 1
## 2 -653.2392 1316.478 1326.039 652263183741 1
##         y PANEL group
## 1 2154937     1     1
## 2 2154937     1     2
```



Using the information now at hand we create some labels.

```
formula <- y ~ x + I(x^2) + I(x^3)
ggplot(my.data, aes(x, y2, colour = group)) +
  geom_point() +
  geom_smooth(method = "rlm", formula = formula) +
  stat_fit_glance(aes(label = paste('AIC~`=`~', signif(..AIC.., 3),
                      "~~", 'BIC~`=`~', signif(..BIC.., 3), sep = "")),
                  parse = TRUE,
                  method = "rlm",
                  method.args = list(formula = formula),
                  geom = "text") +
  theme_bw() +
  theme(legend.position = "top")
```

```
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
## Warning: partial match of 'coefficient' to 'coefficients'
```

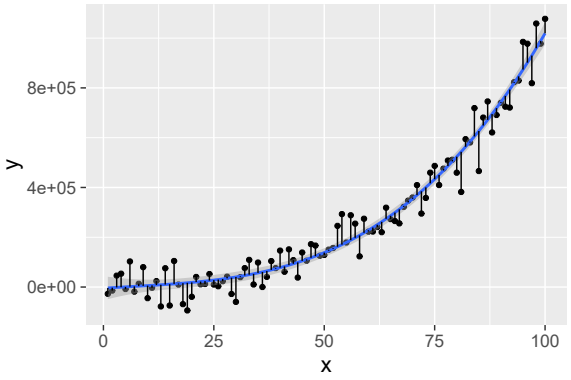


In a similar way one can generate labels for any fit supported by package 'broom'.

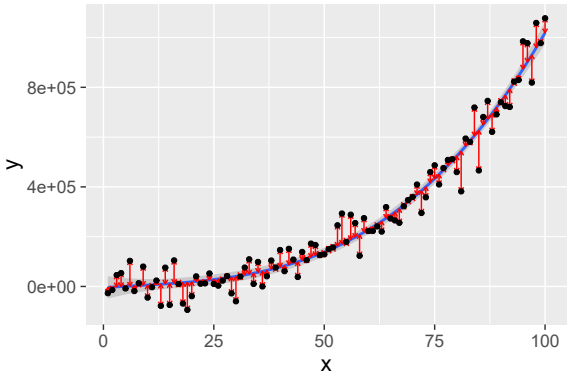
1.14.4 Highlighting deviations from fitted line

First an example using default arguments.

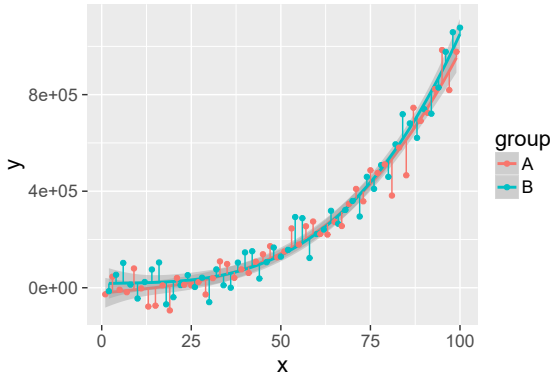
```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = formula) +
  stat_fit_deviations(formula = formula)
```




```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y)) +
  geom_smooth(method = "lm", formula = formula) +
  stat_fit_deviations(formula = formula, color = "red",
                      arrow = arrow(length = unit(0.015, "npc"),
                                    ends = "both")) +
  geom_point()
```

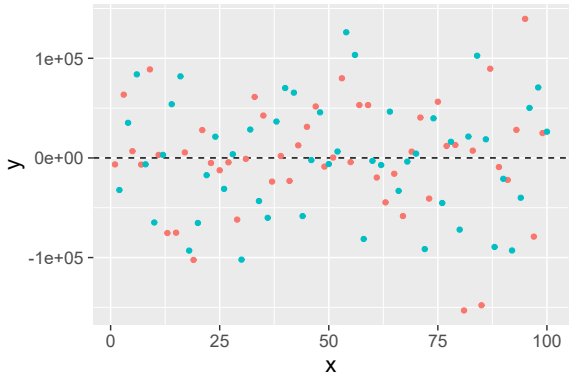



```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y, colour = group)) +
  geom_smooth(method = "lm", formula = formula) +
  stat_fit_deviations(formula = formula) +
  geom_point()
```



1.14.5 Plotting residuals from linear fit

```
formula <- y ~ poly(x, 3, raw = TRUE)
ggplot(my.data, aes(x, y, colour = group)) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  stat_fit_residuals(formula = formula)
```



1.14.6 Filtering observations based on local density

Statistics `stat_dens2d_filter` works best with clouds of observations, so we generate some random data.

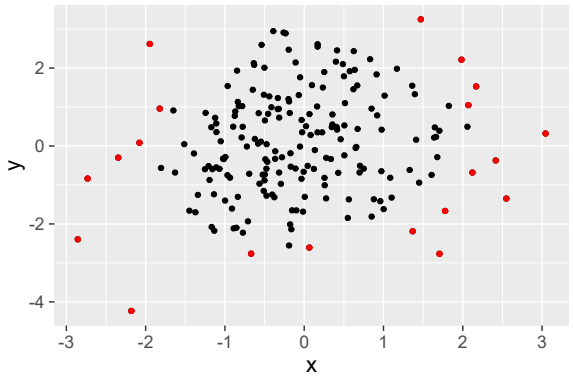
```
set.seed(1234)
nrow <- 200
my.2d.data <-
```

```
  data.frame(
    x = rnorm(nrow),
    y = rnorm(nrow) + rep(c(-1, +1), rep(nrow / 2, 2)),
    group = rep(c("A", "B"), rep(nrow / 2, 2))
   )
```

In most recipes in the section we use `stat_dens2d_filter` to highlight observations with the `color` aesthetic. Other aesthetics can also be used.

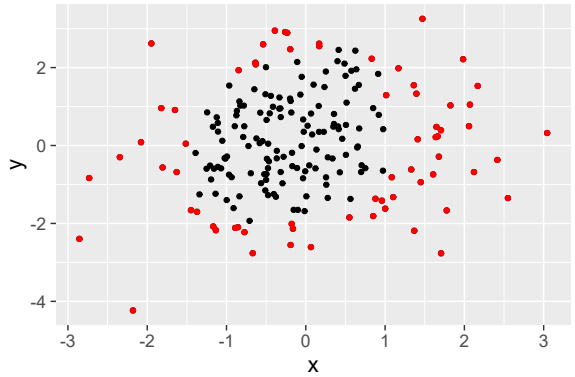
By default 1/10 of the observations are kept from regions of lowest density.

```
ggplot(my.2d.data, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red")
```



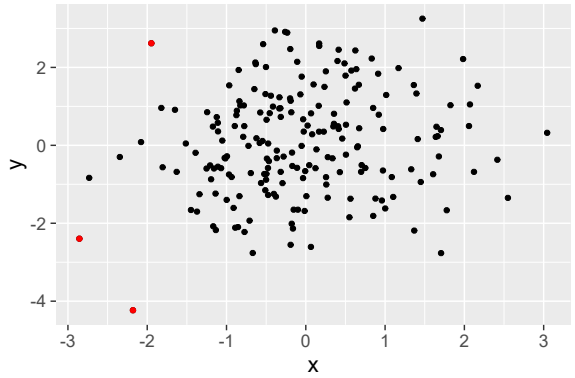
Here we change the fraction to 1/3.

```
ggplot(my.2d.data, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red",
                     keep.fraction = 1/3)
```



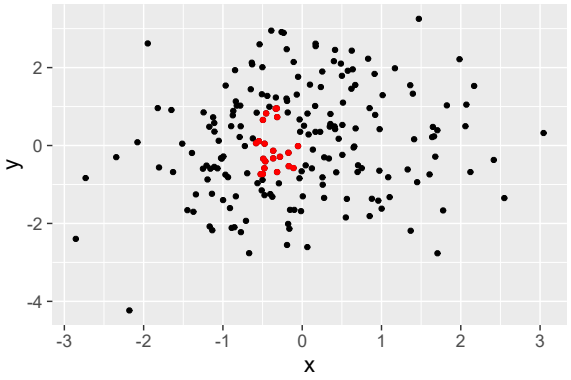
We can also set a maximum number of observations to keep.

```
ggplot(my.2d.data, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red",
                     keep.number = 3)
```

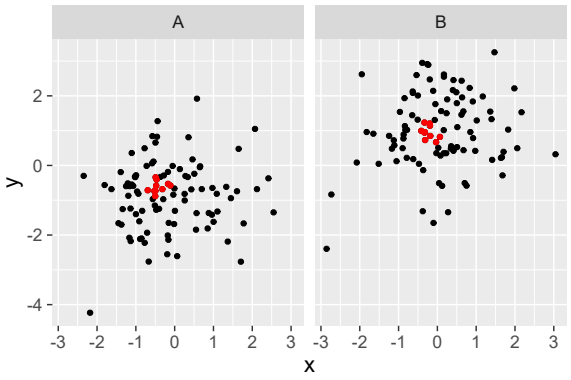


We can also keep the observations from the densest areas instead of the from the sparsest.

```
ggplot(my.2d.data, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red",
                     keep.sparse = FALSE)
```

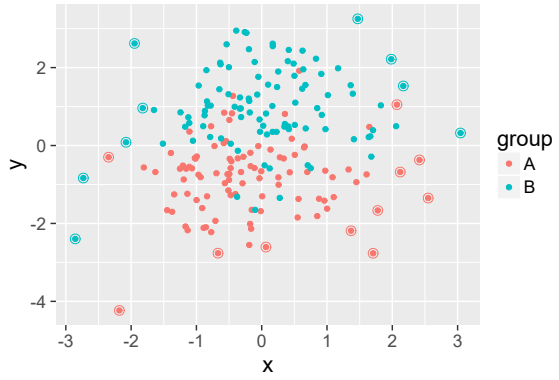



```
ggplot(my.2d.data, aes(x, y)) +
  geom_point() +
  stat_dens2d_filter(color = "red",
                     keep.sparse = FALSE) +
  facet_grid(~group)
```

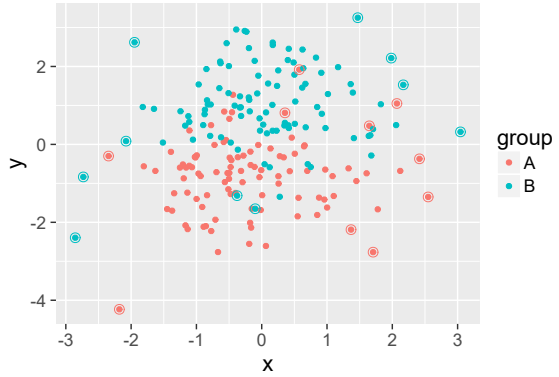


In addition to `stat_dens2d_filter` there is `stat_dens2d_filter_g`. The difference is in that the first one computes the density on a plot-panel basis while the second one does it on a group basis. This makes a difference only when observations are grouped based on another aesthetic within each panel.

```
ggplot(my.2d.data, aes(x, y, color = group)) +
  geom_point() +
  stat_dens2d_filter(shape = 1, size = 3)
```




```
ggplot(my.2d.data, aes(x, y, color = group)) +
  geom_point() +
  stat_dens2d_filter_g(shape = 1, size = 3)
```

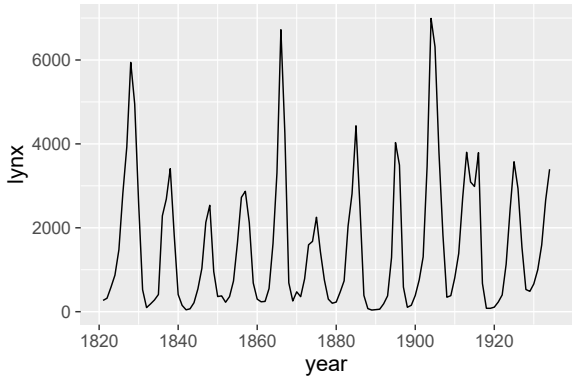


A related stat `stat_dens2d_label`, also defined in package 'ggpmisc'is described in section 1.15.2 on page 38.

1.14.7 Learning and/or debugging

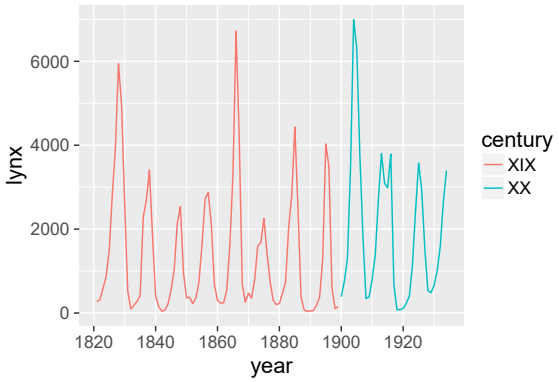
A very simple stat named `stat_debug()` can save the work of adding print statements to the code of stats to get information about what data is being passed to the `compute_group()` function. Because the code of this function is stored in a `ggproto` object, at the moment it is impossible to directly 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()

## [1] "Input 'data' to 'compute_group()':"
## # A tibble: 114 × 4
##        x     y PANEL group
## *  <dbl> <dbl> <int> <int>
## 1   1821   269     1    -1
## 2   1822   321     1    -1
## 3   1823   585     1    -1
## 4   1824   871     1    -1
## 5   1825  1475     1    -1
## 6   1826  2821     1    -1
## 7   1827  3928     1    -1
## 8   1828  5943     1    -1
## 9   1829  4950     1    -1
## 10  1830  2577     1    -1
## # ... with 104 more rows
```



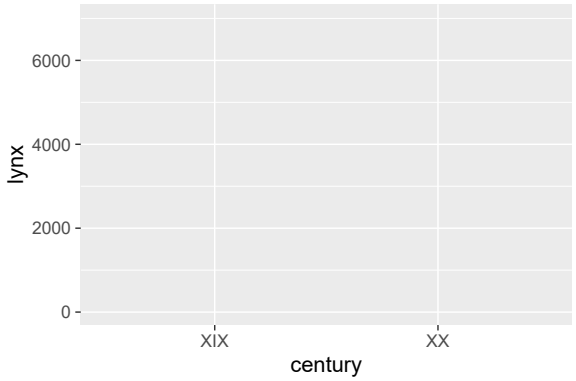

```
lynx.df$century <- ifelse(lynx.df$year >= 1900, "XX", "XIX")
ggplot(lynx.df, aes(year, lynx, color = century)) +
  geom_line() +
  stat_debug_group()

## [1] "Input 'data' to 'compute_group()':"
## # A tibble: 79 × 5
##        x     y colour PANEL group
## *  <dbl> <dbl>  <chr> <int> <int>
## 1   1821   269    XIX     1     1
## 2   1822   321    XIX     1     1
## 3   1823   585    XIX     1     1
## 4   1824   871    XIX     1     1
## 5   1825  1475    XIX     1     1
## 6   1826  2821    XIX     1     1
## 7   1827  3928    XIX     1     1
```

```
## 8   1828  5943    XIX     1     1
## 9   1829  4950    XIX     1     1
## 10  1830  2577    XIX     1     1
## # ... with 69 more rows
## [1] "Input 'data' to 'compute_group()':"
## # A tibble: 35 □ 5
##        x     y colour PANEL group
## *  <dbl> <dbl>  <chr> <int> <int>
## 1   1900   387     XX     1     2
## 2   1901   758     XX     1     2
## 3   1902  1307     XX     1     2
## 4   1903  3465     XX     1     2
## 5   1904  6991     XX     1     2
## 6   1905  6313     XX     1     2
## 7   1906  3794     XX     1     2
## 8   1907  1836     XX     1     2
## 9   1908   345     XX     1     2
## 10  1909   382     XX     1     2
## # ... with 25 more rows
```



By means of `geom_debug` it is possible to ”print” to the console the data returned by a ggplot statistic.

```
ggplot(lynx.df, aes(century, lynx)) +
  geom_blank() +
  stat_summary(fun.y = median,
               geom = "debug", summary.fun = head, summary.fun.args = list())

## Input 'data' to 'geom_debug()':

##   x group ymin    y ymax PANEL
## 1 1     1   NA  684   NA     1
## 2 2     2   NA 1388   NA     1
```



1.15 '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.

1.15.1 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. The vignette *ggrepel Usage Examples* provides very nice examples of the power and flexibility of these geoms. The algorithm used for avoiding overlaps through repulsion is iterative, and can be slow when the number of labels or observations are in the thousands.

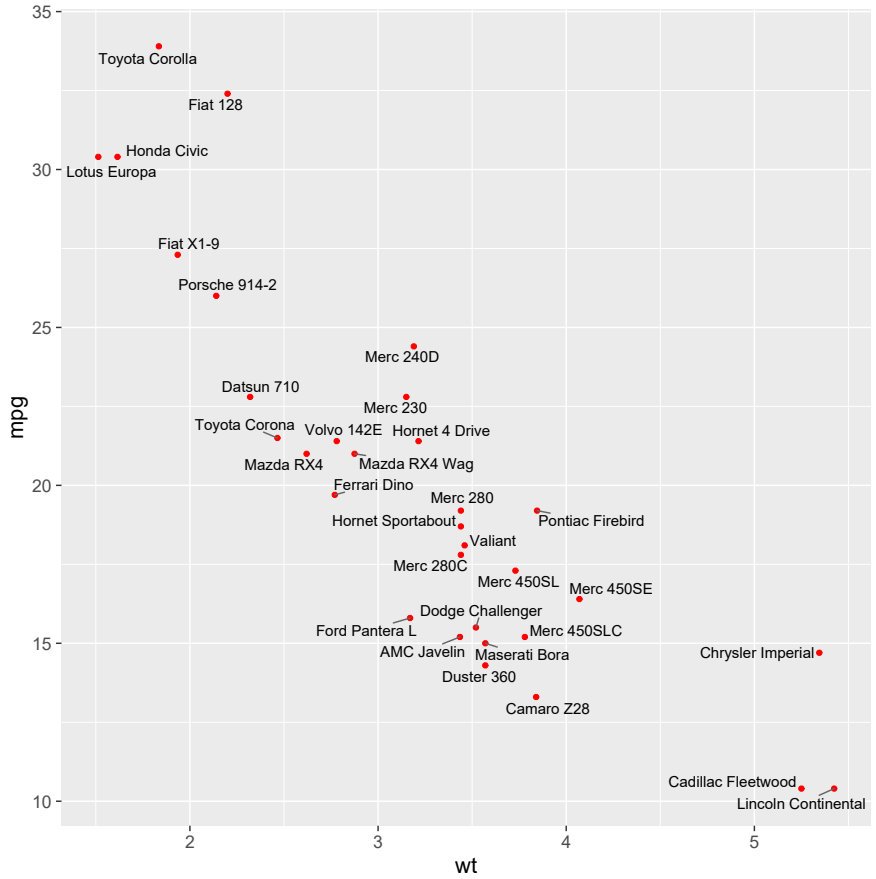
I reproduce here some simple examples from the 'ggrepel'vignette.

```
opts_chunk$set(opts_fig_wide_square)
```

Just using defaults, we avoid overlaps among text items on the plot. `geom_text_repel` has some parameters matching those in `geom_text`, but those related to manual positioning are missing except for `angle`.

Several new parameters control both the appearance of text and the function of the repulsion algorithm.

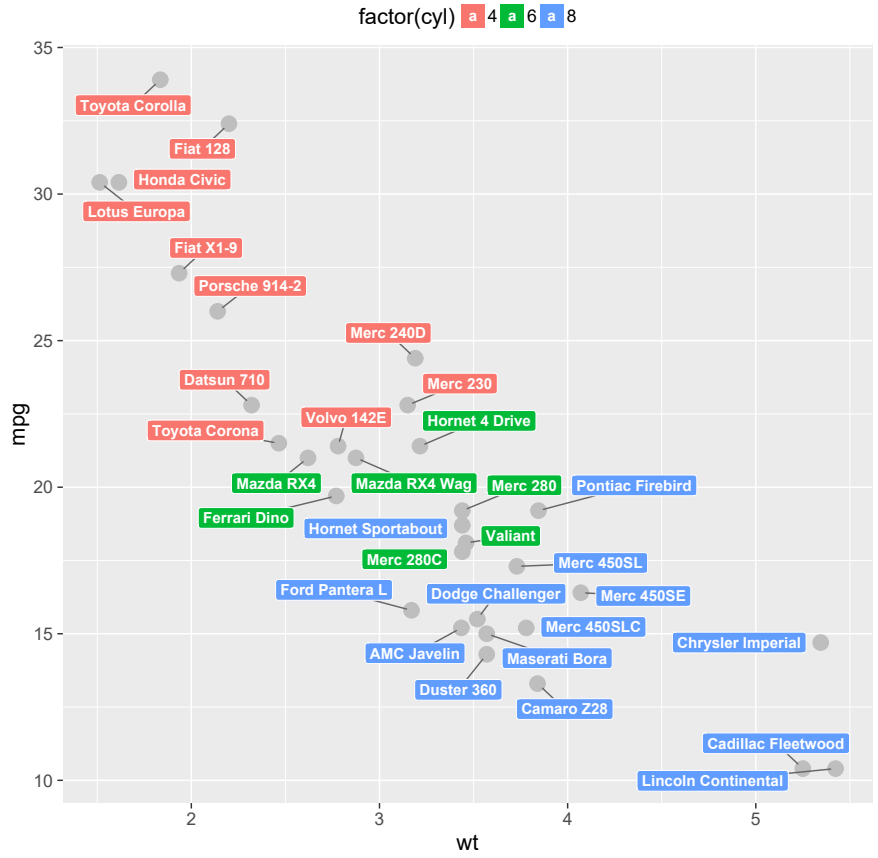
```
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(color = 'red') +
  geom_text_repel(aes(label = rownames(mtcars)))
```



The chunk below shows how to change the appearance of labels. `geom_label_repel` is comparable to `geom_label`, but with repulsion.

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

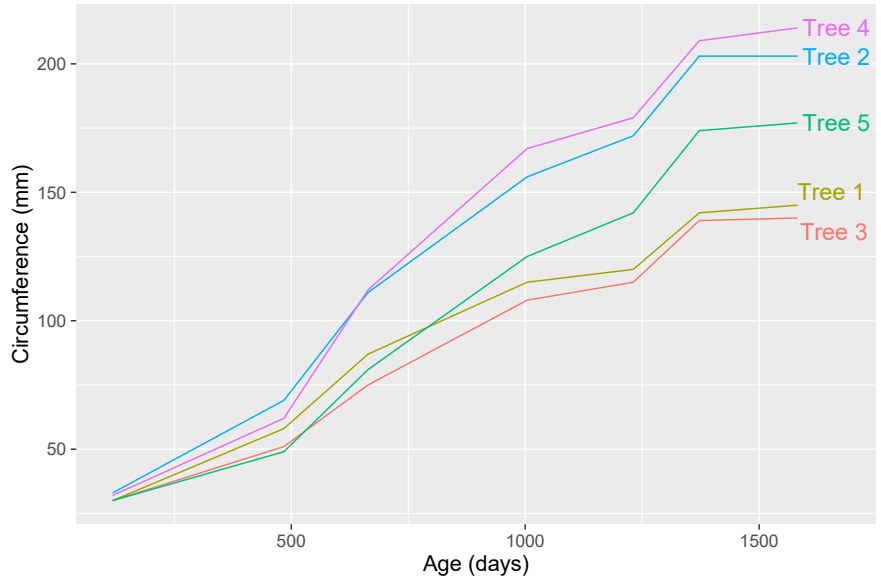


The parameters `nudge_x` and `nudge_y` allow strengthening or weakening the repulsion force, or favouring a certain direction.

```
opts_chunk$set(opts_fig_wide)
```

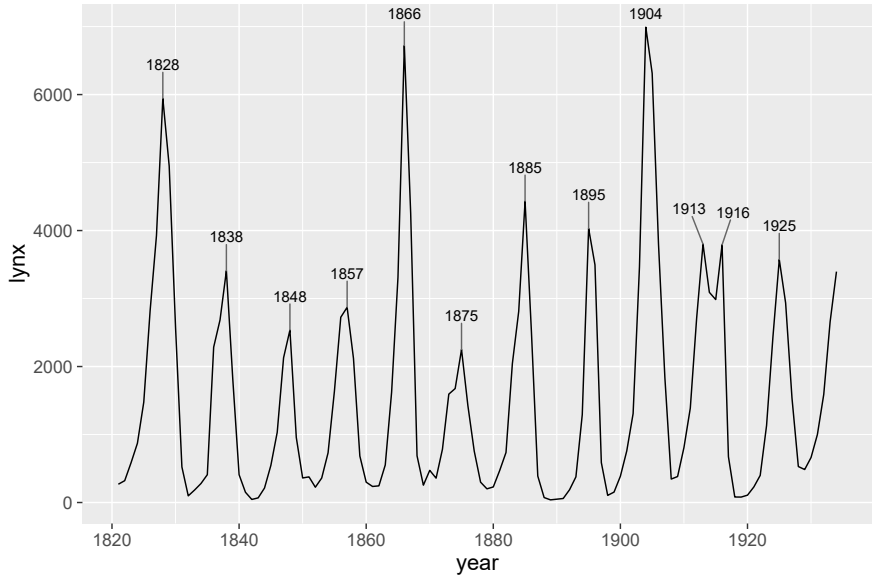
```
set.seed(42)
ggplot(Orange, aes(age, circumference, color = Tree)) +
  geom_line() +
  coord_cartesian(xlim = c(min(Orange$age), max(Orange$age) + 90)) +
```

```
  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)")
```



We can combine `stat_peaks` from package 'ggpmisc'with the use of repulsive text to avoid overlaps between text items. We use `nudge_y = 500` to push the text upwards.

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



1.15.2 Selectively plotting repulsive labels

To repel text or labels so that they do not overlap unlabelled observations, one can set the labels to an empty character string `""`. Setting labels to `NA` skips the observation completely, as is the usual behavior in 'ggplot2'2 geoms, and can result in text or labels overlapping those observations. Labels can be set manually to `""`, but in those cases where all observations have labels in the data, but we would like to plot only those in low density regions, this can be automated. Geoms `geom_text_repel` and `geom_label_repel` from package 'ggrepel'can be used together with `stat_dens2d_label` from package 'ggpmisc'.

To demonstrate this we first generate suitable data and labels.

```
# Make random labels
random_string <- function(len = 6) {
paste(sample(letters, len, replace = TRUE), collapse = "")
}
```

```
# Make random data.
set.seed(1001)
myl.data <- data.frame(
  x = rnorm(100),
```

```
  y = rnorm(100),
  group = rep(c("A", "B"), c(50, 50)),
  lab = replicate(100, { random_string() })
)
head(myl.data)

##            x           y group    lab
## 1  2.1886481  0.07862339     A emhufi
## 2 -0.1775473 -0.98708727     A yrvrlo
## 3 -0.1852753 -1.17523226     A wrpfpp
## 4 -2.5065362  1.68140888     A ogrqsc
## 5 -0.5573113  0.75623228     A wfxezk
## 6 -0.1435595  0.30309733     A zjccnn
```

The first example uses defaults.

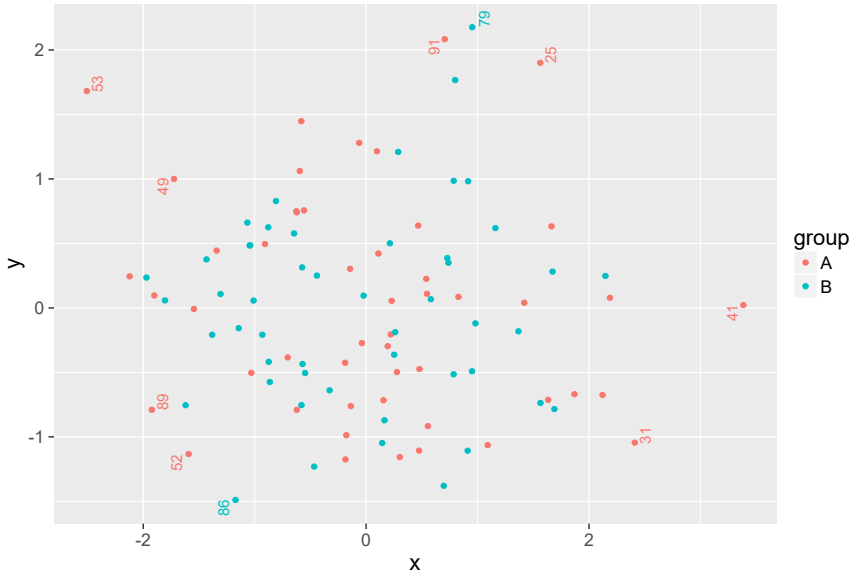
```
ggplot(data = myl.data, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel")
```

The fraction of observations can be plotted, as well as the maximum number can be both set through parameters, as shown in section 1.14.6 on page 27.

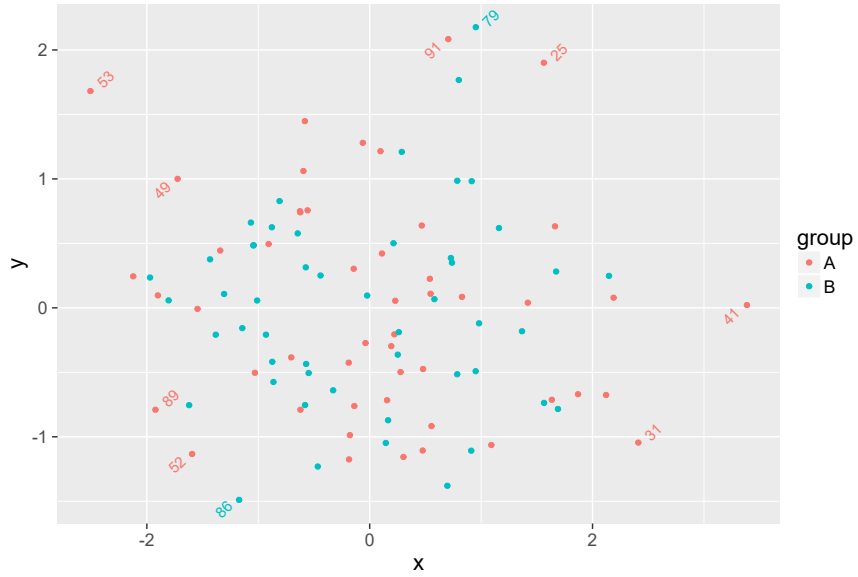
Something to be aware of when rotating labels is that repulsion is always based on bounding box that does not rotate, which for long labels and angles that are not multiples of 90 degrees, reserves too much space

and leaves gaps between segments and text. Compare the next two figures.

```
ggplot(data = myl.data, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel", angle = 90)
```

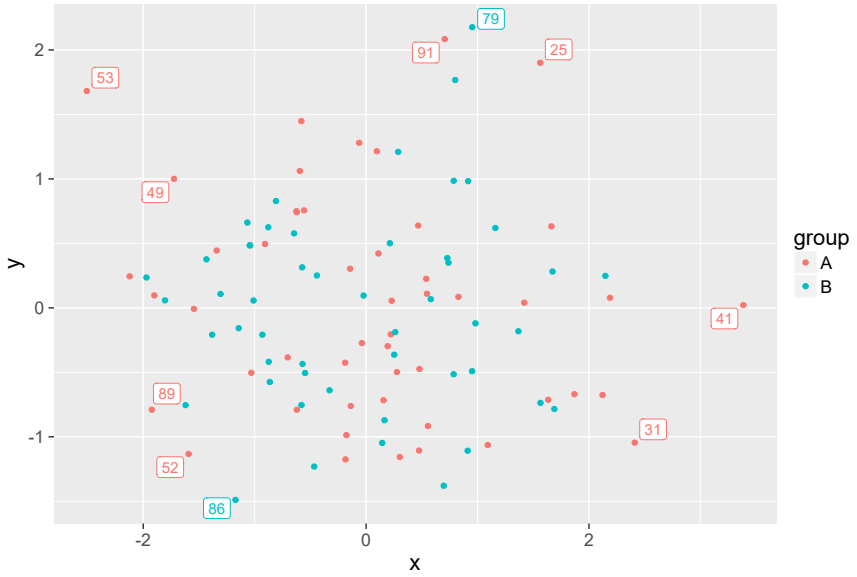



```
ggplot(data = myl.data, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "text_repel", angle = 45)
```



Labels cannot be rotated.

```
ggplot(data = myl.data, aes(x, y, label = lab, color = group)) +
  geom_point() +
  stat_dens2d_labels(geom = "label_repel")
```



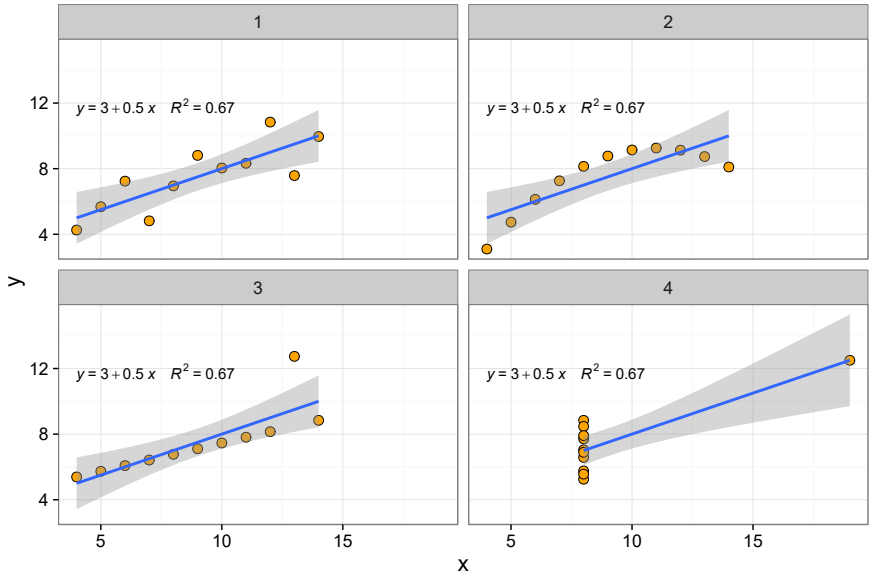
1.16 Examples

1.16.1 Anscombe's example revisited

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



1.16.2 Heatmaps

1.16.3 Volcano plots

1.16.4 Quadrat plots

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

2 Further reading about R

2.1 Introductory texts

Dalgaard2008; Zuur2009; Teetor2011

2.2 Texts on specific aspects

Chang2013; Fox2002; Fox2010; Faraway2004; Faraway2006; Everitt2011

2.3 Advanced texts

Xie2013; wickham2015; wickham2014advanced; Pinheiro2000; Murrell2011; Matloff2011; Ihaka1996

Index