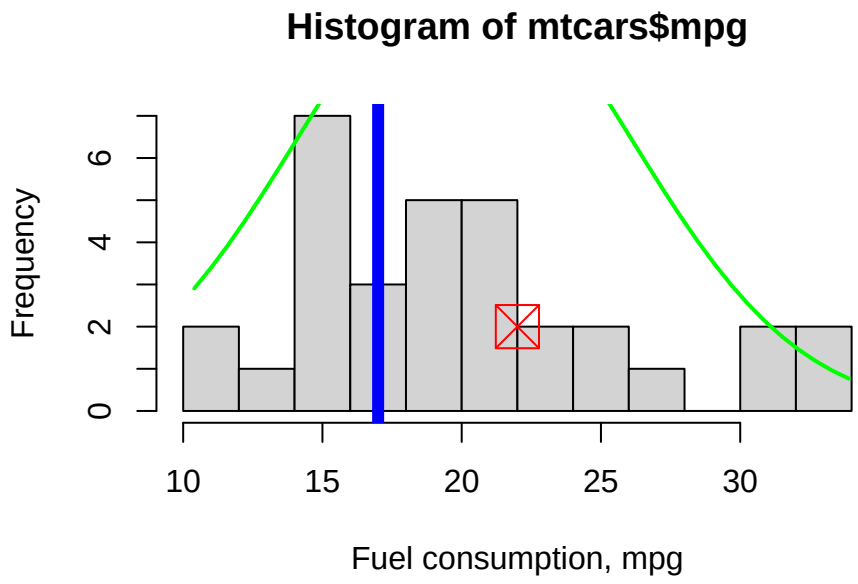
Statistical programming using R

Lecture 4 – Data visualisation

Basic functions for data visualisation

You have already seen "basic" plotting functionality of R and it is very impressive. Below there is an example from the past lecture with minor changes

```
# creates a plotting window and put a histogram there
hist(mtcars$mpg, breaks = 10, xlab = "Fuel consumption, mpg")

abline(h=15, col="red", lty=2) # add a horizontal line - it does not appear on the graph!!!

abline(v=17, col="blue", lwd=6)  # add a vertical line

points(x=c(13, 22), y=c(8,2), pch=7, cex=3, col="red") # add 2 points - one point disappeared!!!

# prepare data for normal distribution density curve
xfit <- seq(min(mtcars$mpg), max(mtcars$mpg), length = 40)
yfit <- dnorm(xfit, mean = mean(mtcars$mpg), sd = sd(mtcars$mpg))
yfit <- yfit * length(mtcars$mpg) * 5

lines(xfit, yfit, col="green", lwd=2) # add a custom line - line is chopped on the top!!!
```



The reason for the problems with disappearing elements is simple. The very first function `hist()` set an overall size of the graph – 10 to 35 units horizontally and 0 to 7 units vertically. Horizontal line at level 15, point at coordinates (13,8) and the top of the custom line happen to be above the graph.

It is possible to fix all these problems – you just need to adjust the size of the first graph. Parameter `ylim=c(0,16)` for the function `hist()` will do the job by setting explicit limits for the vertical size. However, you might want plotting functionality that is smarter and looks better at the same time.

Package ggplot2

While standard plotting in R is very good and it made R popular among professionals working with data in different areas, the package `ggplot2` lifted the game up enormously and it stays a standard for data visualisation for all statistical applications. Many of them try to copy `ggplot2` but no one made it better.

Package `ggplot2` is a part of `tidyverse` collection. So, you can load it as a stand-alone package:

```
library(ggplot2)
```

or as a part of the `tidyverse` and get `ggplot2`, `dplyr`, `readr`, `purrr` and some other packages at the same time.

```
# there are a lot of warnings and messages, we can suppress them for now
suppressMessages(suppressWarnings(library(tidyverse)))
```

Similar to the concept of `dplyr`, package `ggplot2` works (mostly) with data frames. This is important to stress – with correctly prepared data frames.

```
# original data to play with
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

For example, data set `mtcars` can be analyzed and plotted successfully. However, some of its variables are not really numerical but categorical and should be stored as factors to allow the right analysis and plotting.

```
# create a new data frame with updated variables
# package "dplyr" will be handy here
df <- mtcars %>%
  mutate(model = rownames(.)) %>%                # store model names as a variable
  mutate(cyl = factor(cyl, levels = c(4, 6, 8),
                      labels = c("4 cylinders", "6 cylinders", "8 cylinders"))) %>%
  mutate(gear = factor(gear, levels = c(3, 4, 5),
                       labels = c("3 gears", "4 gears", "5 gears"))) %>%
  mutate(transmission = factor(am, levels = c(0, 1),
                               labels = c("Automatic", "Manual"))) %>%
  select(-c("am", "carb", "drat", "vs"))         # remove variables you don't need

head(df)
```

```
##                    mpg         cyl disp  hp    wt  qsec    gear
## Mazda RX4         21.0 6 cylinders  160 110 2.620 16.46 4 gears
## Mazda RX4 Wag     21.0 6 cylinders  160 110 2.875 17.02 4 gears
## Datsun 710        22.8 4 cylinders  108  93 2.320 18.61 4 gears
## Hornet 4 Drive    21.4 6 cylinders  258 110 3.215 19.44 3 gears
## Hornet Sportabout 18.7 8 cylinders  360 175 3.440 17.02 3 gears
## Valiant           18.1 6 cylinders  225 105 3.460 20.22 3 gears
##                               model transmission
## Mazda RX4                 Mazda RX4       Manual
## Mazda RX4 Wag         Mazda RX4 Wag       Manual
## Datsun 710               Datsun 710       Manual
## Hornet 4 Drive       Hornet 4 Drive    Automatic
## Hornet Sportabout Hornet Sportabout    Automatic
## Valiant                     Valiant    Automatic
```

New data frame `df` (very poor name) has the right format for all variables – double, factor or character – whatever is appropriate for the data.

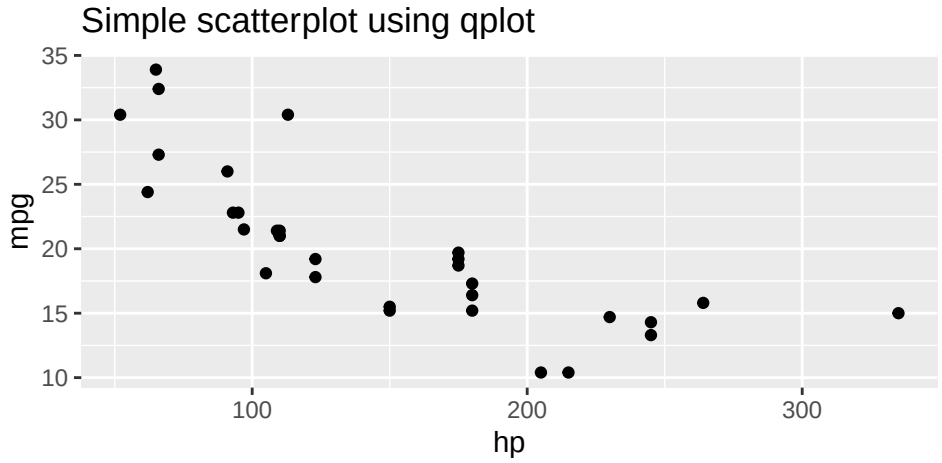
There are two main functions for plotting in `ggplot2` – `qplot()` and `ggplot()`.

Function `qplot()`

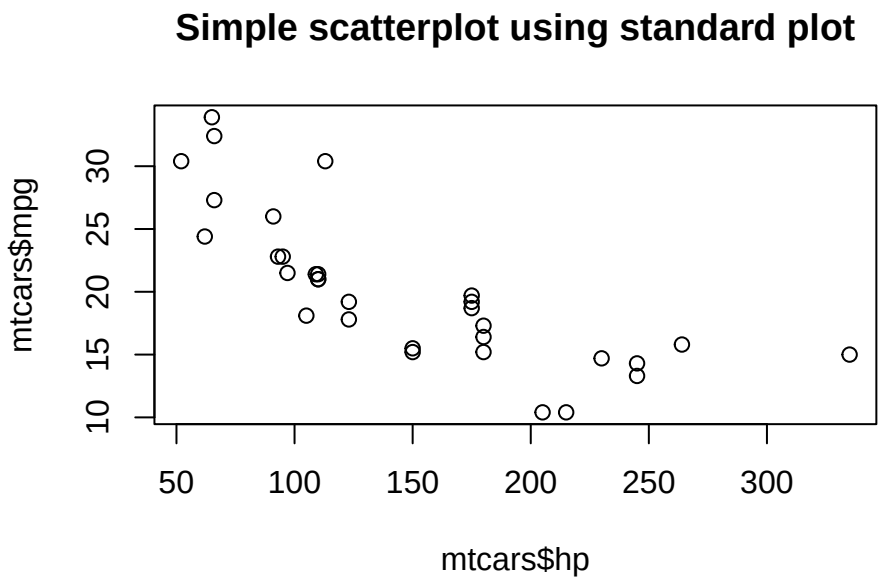
Function `qplot()` or `quickplot()` is a substitute for a base function `plot()`. It has a similar syntax of putting all possible arguments inside round brackets of the function. This makes a transition from basic plotting into `ggplot2` somewhat easier for people already comfortable with `plot()` function. It is possible to make good data visualisations using `qplot()`. At the same time, it is highly recommended to make an extra effort and learn `ggplot()` function as it allows to create complex graphs much easier.

Here are some examples of using `qplot()`

```
# Scatterplot is a default settings
qplot(x = hp, y = mpg, data = df, main = "Simple scatterplot using qplot")
```

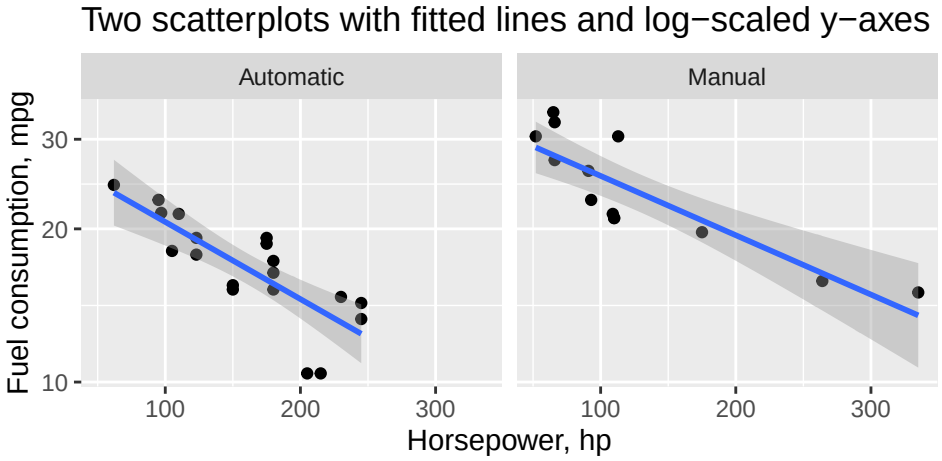



```
# compare it to a "standard" plot in R
plot(x = mtcars$hp, y = mtcars$mpg, main = "Simple scatterplot using standard plot")
```

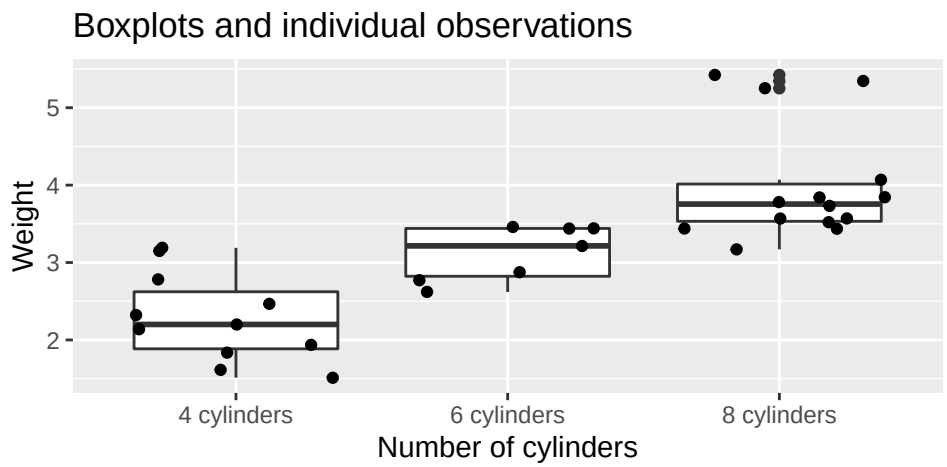


More examples of `qplot()`

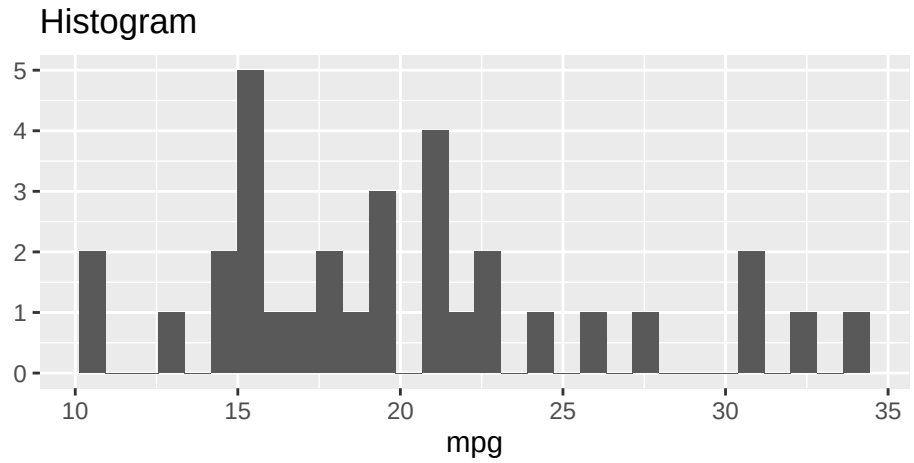
```
qplot(x = hp, y = mpg, data = df, facets = . ~ transmission,
      log = "y",
      main = "Two scatterplots with fitted lines and log-scaled y-axes",
      xlab = "Horsepower, hp", ylab = "Fuel consumption, mpg",
      geom = c("point", "smooth"), method = "lm")
```




```
qplot(x = cyl, y = wt, data = df, geom = c("boxplot", "jitter"),
      main = "Boxplots and individual observations",
      xlab = "Number of cylinders", ylab = "Weight")
```




```
# default settings for qplot if there is only x variable provided
qplot(x = mpg, data = df, main = "Histogram")
```

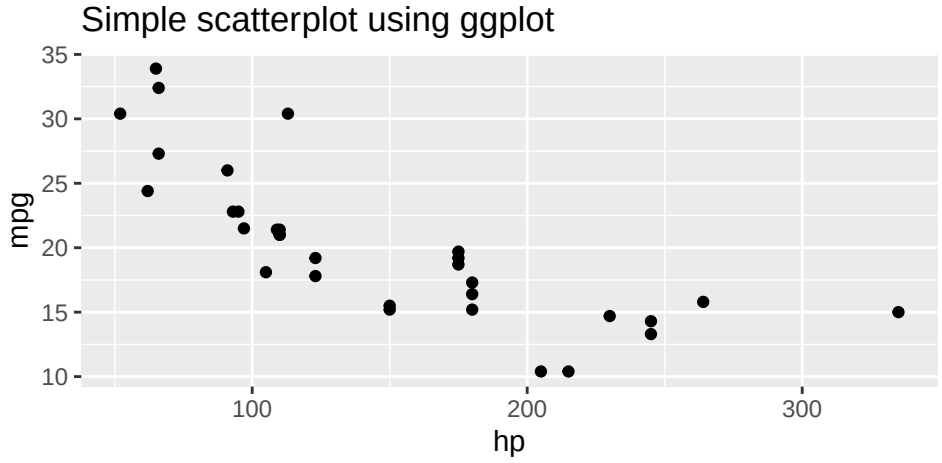


Most probably, it is possible to plot almost any graph using function `qplot()`. This function is relatively easy to use for relatively simple graphs. However, there is another function that is more complex but it is a way more powerful and much more popular.

Function ggplot()

Function `ggplot()` works on layers, you start with the function `ggplot()` itself and then add as many layers as you need. You might not need too many layers and your code can be very simple

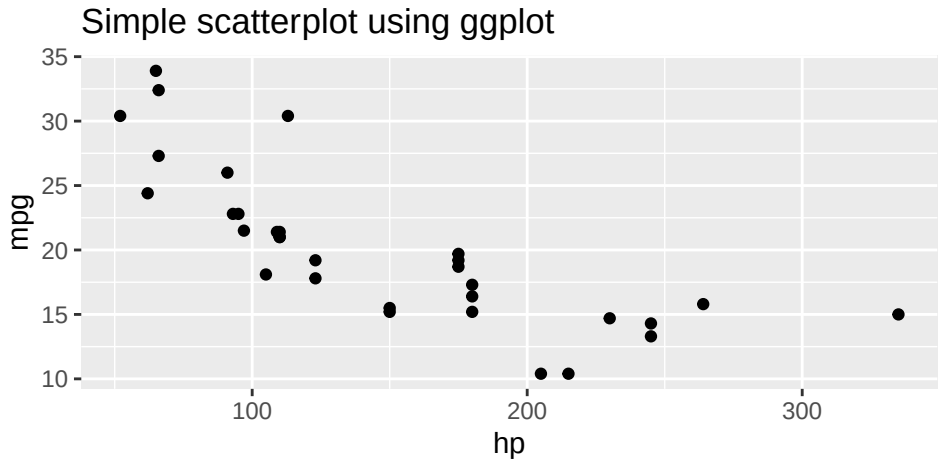
```
ggplot(data = df, mapping = aes(x = hp, y = mpg)) + # nominate data frame and variables
  geom_point() +                                    # define the type of visualisation
  ggtitle("Simple scatterplot using ggplot")        # add a layer with a title
```



Resulted graph is identical to what you had before with `qplot()` function as both functions use the same engine for plotting. However, there is an opportunity to add more layers and create more advanced data visualisation.

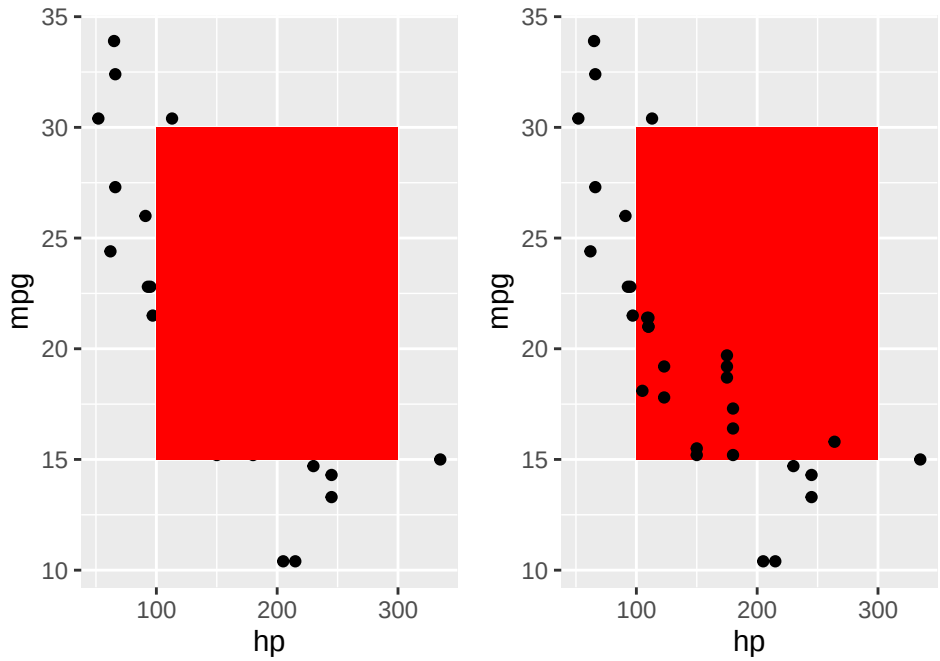
Function `ggplot()` returns ggplot object. If you run the code in the interactive mode, then you get the graph. If you run the same code as a source file, then you get no output. To get a graph you have to use function `print()` for ggplot object.

```
gr1 <- ggplot(data = df, mapping = aes(x = hp, y = mpg))  # get ggplot object
gr1 <- gr1 + geom_point() + ggtitle("Simple scatterplot using ggplot") # add two layers

print(gr1) # code to output the latest version of the graph (ggplot object)
```



Symbol + adds a new layer to ggplot object. There can be as many layers as you want. Order of layers might be important. In the example above, title was on top of the graph itself. Doing graph on top of the title would make not difference. But in other situations, the order of layers would make a really big difference.

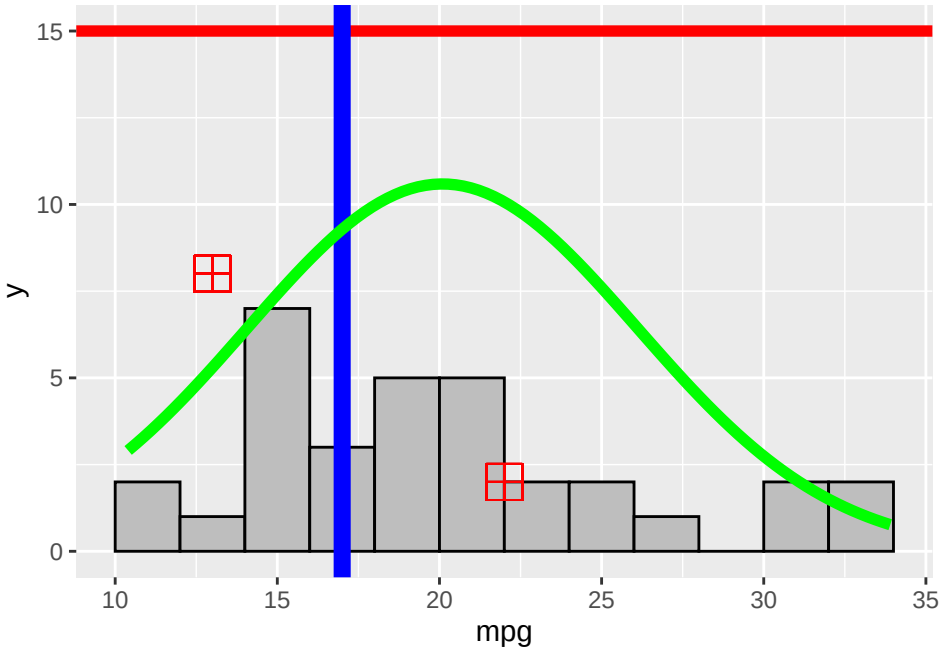
```
# get ggplot object
gr1 <- ggplot(data = df, mapping = aes(x = hp, y = mpg))

# create a new graph by adding a scatterplot visualisation
# and then a rectangular box
gr2 <- gr1 +           # existing ggplot object
  geom_point() +       # add point graph visualisation
  geom_rect(aes(xmin=100, xmax=300, ymin=15, ymax=30), fill = "red") # add a rectangular

# create another graph by adding a rectangular box first
# and scatterplot visualisation after that
gr3 <- gr1 +         # existing ggplot object
  geom_rect(aes(xmin=100, xmax=300, ymin=15, ymax=30), fill = "red") +  # add a rectangular
  geom_point()       # add point graph visualisation

# plot two new ggplot objects side by side
# be aware of the extra package for that functionality
ggpubr::ggarrange(gr2, gr3)
```



The difference is clear and the importance of the order of the layers is obvious from the above graphs.

Now, let's try to replicate the plot from the very beginning of the lecture.

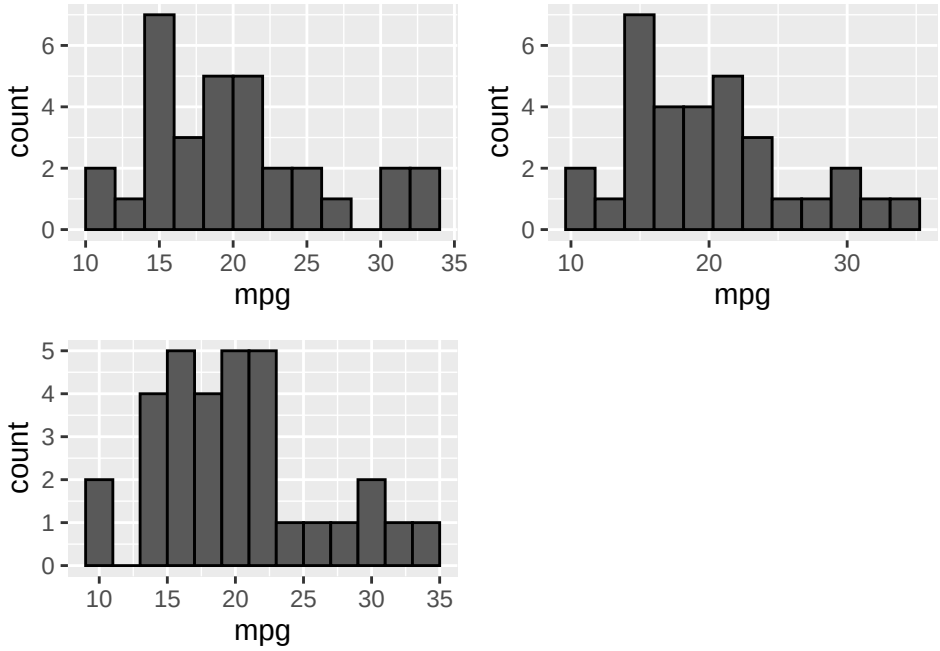
```
# step 1 create ggplot object
ggplot(data = mtcars, aes(mpg)) +

  # step 2 add a histogram with custom breaks to match old graph
  geom_histogram(breaks=seq(10, 34, by=2), colour = "black", fill = "grey") +

  # step 3 add horizontal line
  geom_hline(yintercept = 15, colour = "red", size = 2) +

  # step 4 add vertical line
  geom_vline(xintercept = 17, colour = "blue", size = 3) +

  # steps 5 and 6 add custom dots
  # this part is ugly as ggplot prefers to work with data frames
  geom_point(aes(x=13, y=8), colour = "red", size = 6, shape = 12) +
  geom_point(aes(x=22, y=2), colour = "red", size = 6, shape = 12) +

  # add custom scaled density function for normal distribution
  stat_function(fun = function(x) 160*dnorm(x, mean = mean(mtcars$mpg), sd = sd(mtcars$mpg)),
                colour = "green", size = 2)
```



Horizontal line is really high – well above the histogram, but ggplot automatically adjusted the scale to fit everything and as a result histogram looks squashed. Fitting scaled density curve was much easier than in the original example as `ggplot()` is a very smart function.

Function `ggplot()` and all related functions `geom_...()` have a huge number of parameters that give a flexibility but make everything really complex. For example, here are three different ways to define bin size for a histogram.

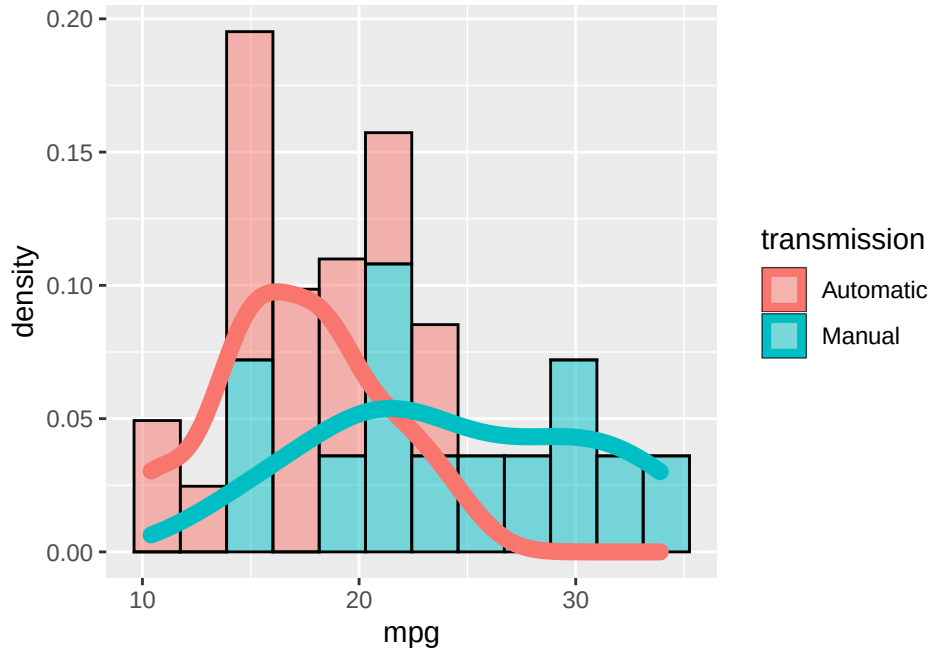
```
gr1 <- ggplot(data = mtcars, aes(mpg))

# set custom breaks for each bin
gr2 <- gr1 + geom_histogram(breaks = seq(10, 34, by=2), colour = "black")

# set a total number of bins
gr3 <- gr1 + geom_histogram(bins = 12, colour = "black")

# set bin width
gr4 <- gr1 + geom_histogram(binwidth = 2, colour = "black")

# plot three new ggplot objects together
# be aware of the extra package for that functionality
ggpubr::ggarrange(gr2, gr3, gr4)
```



Three histograms – the same data, very close or the same bin sizes but somewhat different shapes.

More examples for a histogram

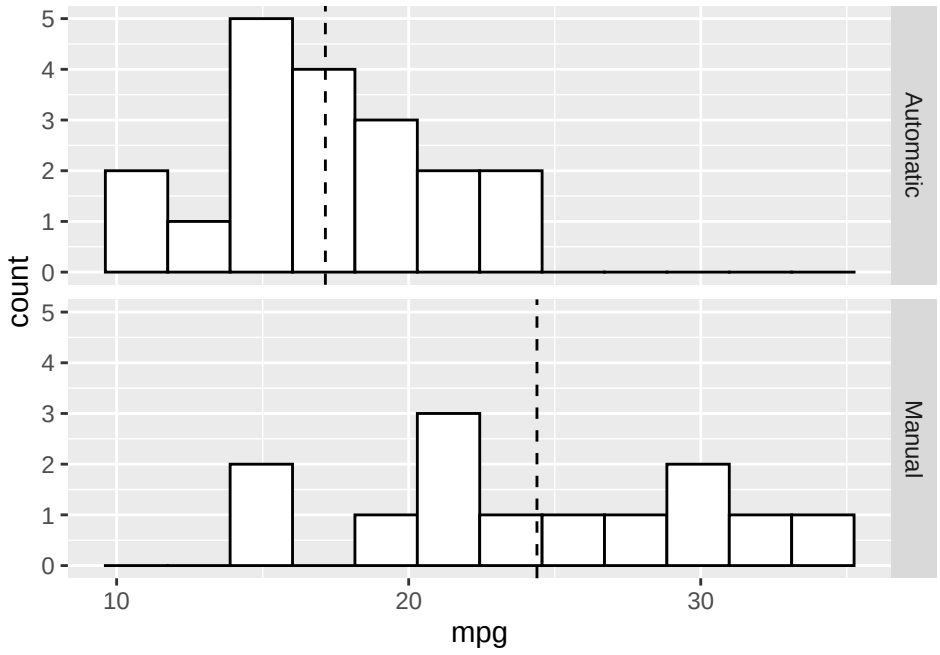
```
df %>%    # you can use piping to pipe data frame into ggplot function

  # get a basic ggplot object, set colour to factor variable with two levels
  ggplot(aes(x = mpg, fill = transmission)) +

  # add histogram visualisation and set y axes to density instead of counts
  # alpha makes visualisation semi-transparent
  geom_histogram(aes(y=..density..), bins = 12, alpha = 0.5, colour = "black") +

  # add density curve with its own colour
  geom_density(aes(colour = transmission), alpha = 0, size = 3)
```



```r
# prepare a small data frame with extra data
# calculate means of mpg for two groups
df_mu <- df %>% group_by(transmission) %>%
  summarise(ave_mpg = mean(mpg), .groups = "drop")

df_mu # check results
```

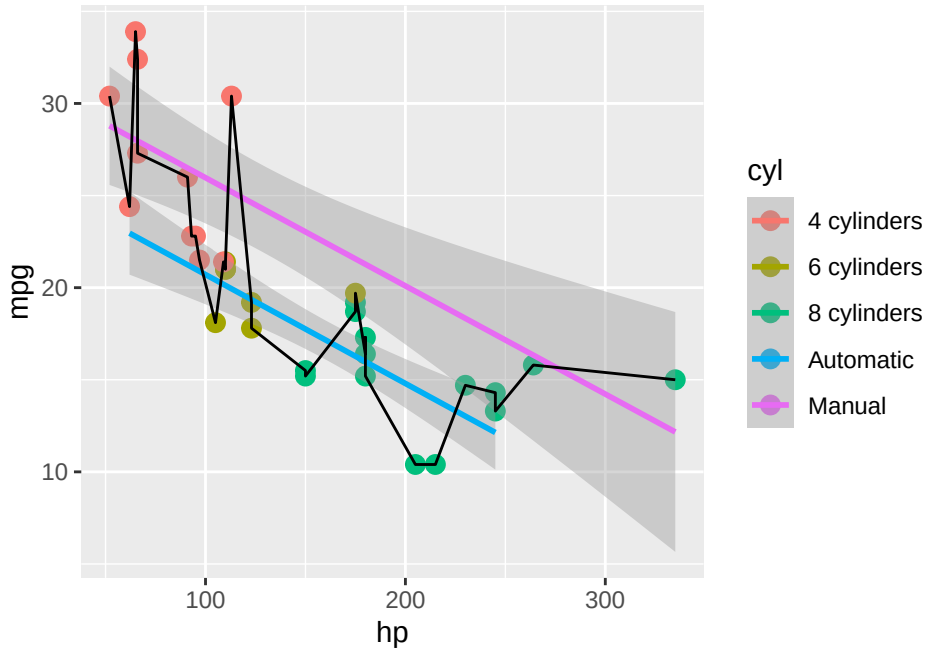
```
## # A tibble: 2 x 2
##   transmission ave_mpg
##   <fct>          <dbl>
## 1 Automatic       17.1
## 2 Manual          24.4
```

```r
# get a ggplot object
ggplot(df, aes(x = mpg)) +

  # add histogram visualisation
  geom_histogram(color = "black", fill="white", bins = 12) +

  # add a split by categorical variable
  facet_grid(transmission ~ .) +

  # add a vertical line at mean value for each group
  # use new data for mapping
  geom_vline(data = df_mu, aes(xintercept = ave_mpg), linetype = "dashed")
```



Now let's have a look on some other types of `geom_...` functions.

```
# this example is a bit crazy statistic-wise but it is a good illustration of ggplot functionality

# load data into ggplot function and map only x and y axis
df %>% ggplot(aes(x = hp, y = mpg)) +

  # add a layer of dots with colour defined by a number of cylinders - three groups
  # there is its own aesthetic
  geom_point(aes(colour = cyl), size = 3) +

  # add a layer with regression lines for two groups defined by transmissions
  # again, there is its own aesthetic separated from other layers
  geom_smooth(aes(colour = transmission), method = "lm") +

  # add a layer with the line connecting all points without any grouping
  geom_line()
```

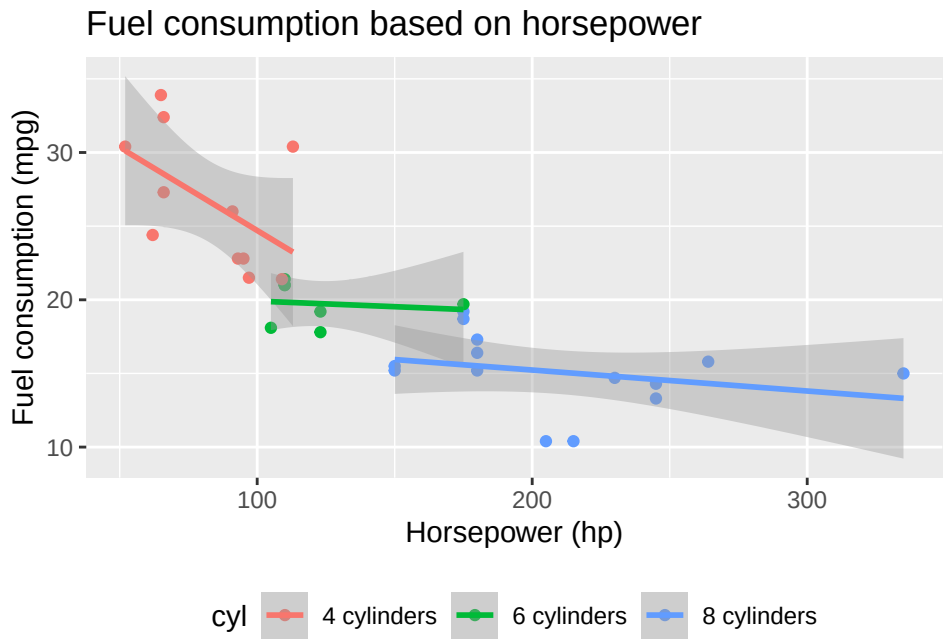


Above graph is "strange" it selects three groups of cars by a number of cylinders but then runs regression analysis for two groups defined by the type of transmission. This is very odd. However, this example shows that each `geom_...` function can have its own aesthetic.

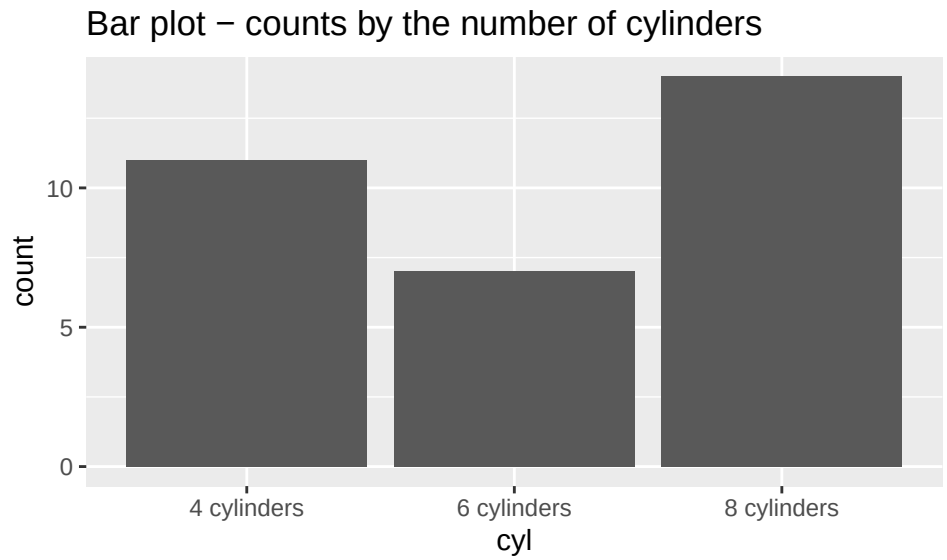
Here is a more "reasonable" version of the above graph. It shows that 4, 6 and 8 cylinder cars make three well defined clusters in terms of power and fuel consuption.

There is a strong relationship between horsepower and miles per gallon for 4 cylinder cars – higher horsepower means less miles per gallon. Similar relationship for 6 and 8 cylinder cars is not so obvious and possibly not statistically significant. Increase in horsepower does not result in a decrease in miles per gallon – both regression lines are almost horizontal.

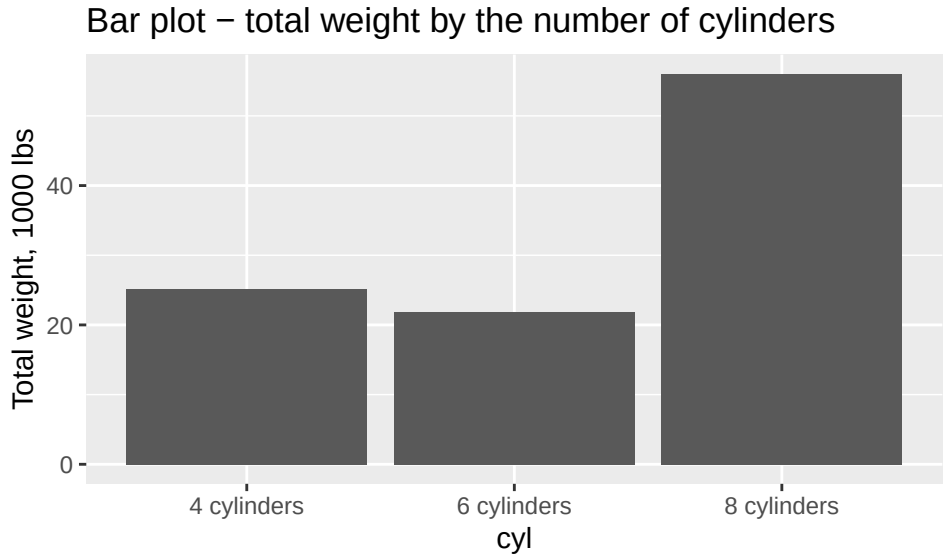
```
# load data into ggplot function
# map x and y axis and grouping colour
df %>% ggplot(aes(x = hp, y = mpg, colour = cyl)) +

  # add a layer of dots
  geom_point() +

  # add a layer with regression lines
  geom_smooth(method = "lm") +

  # adjust legend position
  theme(legend.position="bottom") +

  # add titles for graph and axis
  labs(title = "Fuel consumption based on horsepower",x = "Horsepower (hp)", y = "Fuel consumption (mpg
```



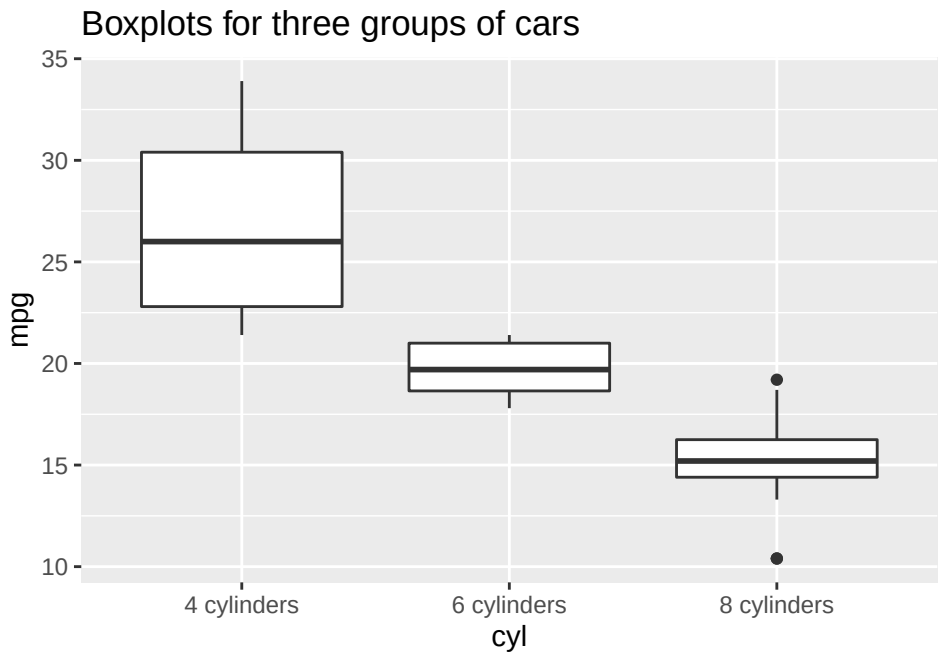
```
# load data into ggplot function and then
# add bars for counts in each group defined by variable cyl
df %>% ggplot(aes(x = cyl)) + geom_bar() +
  ggtitle("Bar plot - counts by the number of cylinders")
```



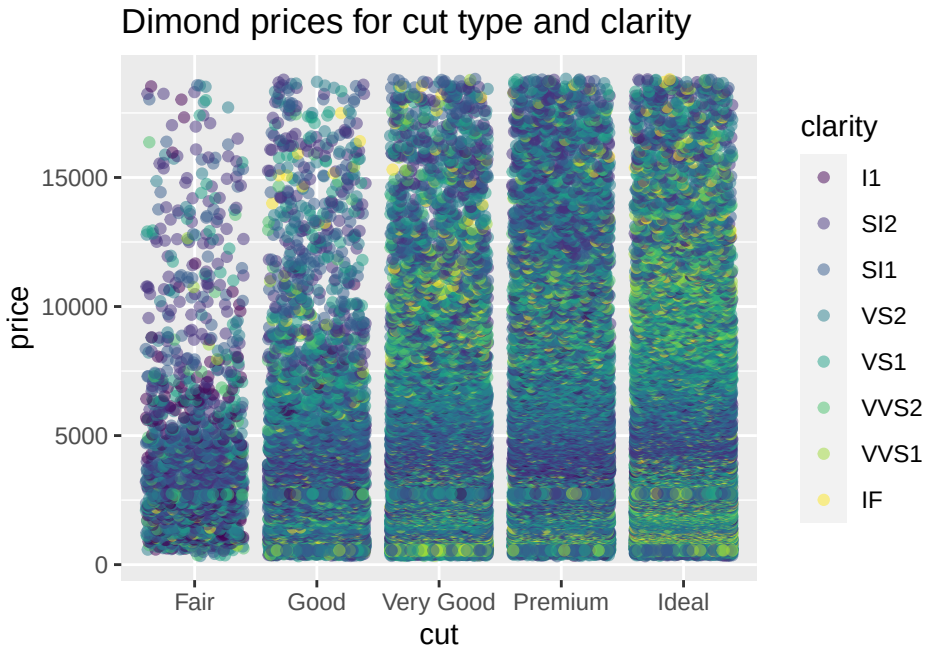

```
# load data into ggplot function and then
# add bars for counts in each group defined by variable cyl
df %>% ggplot(aes(x = cyl)) + geom_bar(aes(weight = wt)) +
  ggtitle("Bar plot - total weight by the number of cylinders") +

  # default y axes label is incorrect, now it is a summation
  ylab("Total weight, 1000 lbs")
```



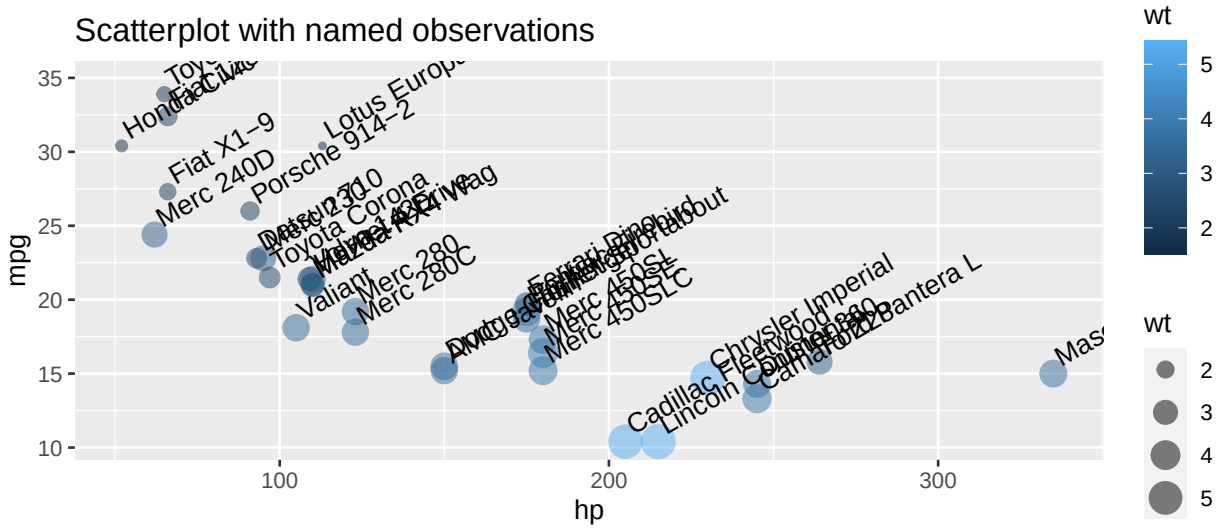
```
# load data into ggplot function and then
# add bars for counts in each group defined by variable cyl
df %>% ggplot(aes(x = cyl, y = mpg)) + geom_boxplot() +
  ggtitle("Boxplots for three groups of cars")
```




```
# data set on dimonds
diamonds %>% ggplot(aes(x = cut, y = price, colour = clarity)) + geom_jitter(alpha = 0.5) +
  ggtitle("Dimond prices for cut type and clarity")
```

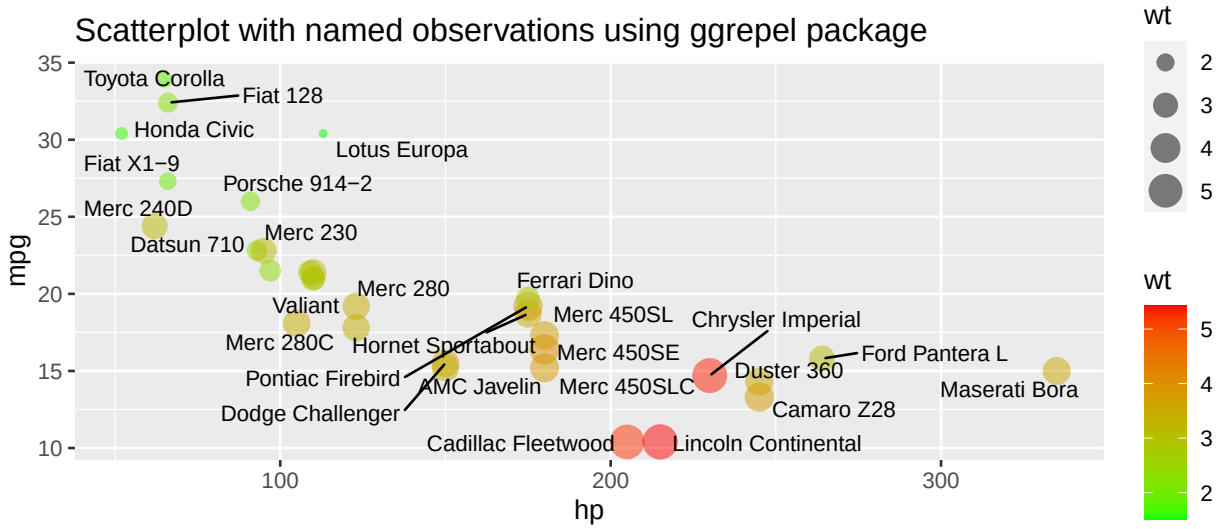


```
df %>% ggplot(aes(x = hp, y = mpg)) +
  # add scatterplot layer
  geom_point(aes(size = wt, color = wt), alpha = 0.5) +
  # add text labels layer
  geom_text(aes(label = model, x = hp + 1, y = mpg + 1), hjust = 0, angle = 30, size = 4) +
  ggtitle("Scatterplot with named observations")
```



Despite all efforts, it is difficult to read car models on the above graph. But we can fix it by using extra package `ggrepel`.

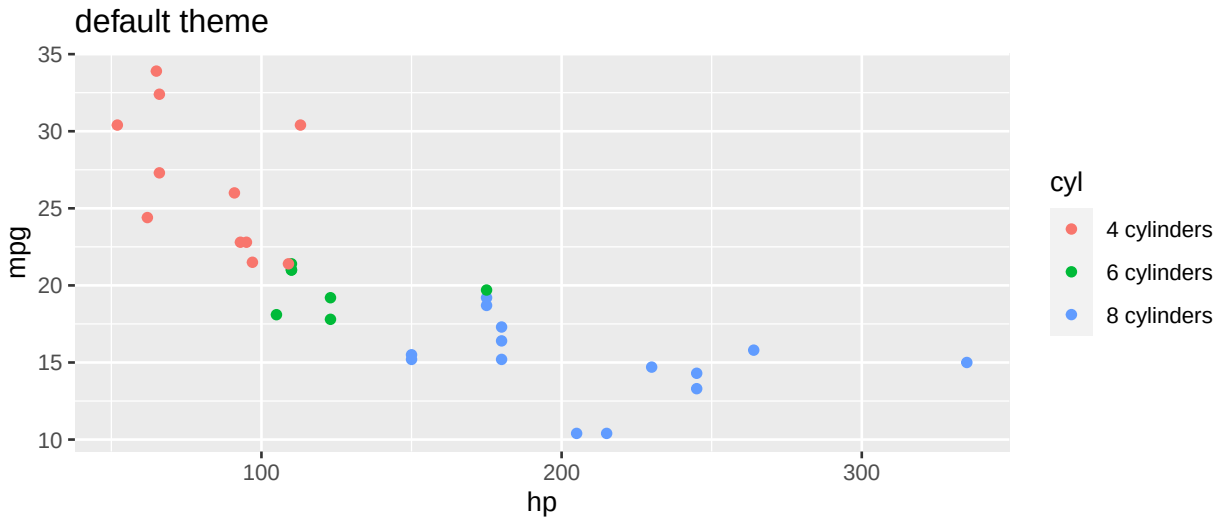
```
df %>% ggplot(aes(x = hp, y = mpg, label = model)) +
  geom_point(aes(size = wt, color = wt), alpha = 0.5) +
  scale_color_gradient(low="green", high="red") +  # custom colours for weight
  # add geom function from another package
  ggrepel::geom_text_repel(size = 3) +
  ggtitle("Scatterplot with named observations using ggrepel package")
```



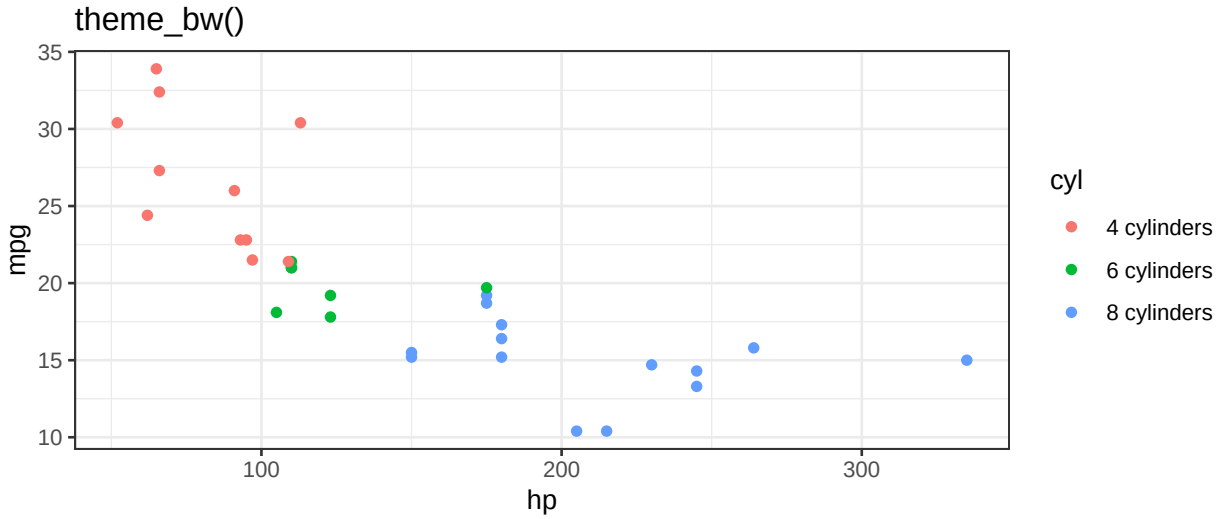
There are a lot of parameters you can change to improve visual appearance of the graph. You can change fonts, colours, sizes. You can do it manually or you can use pre-defined themes available in `ggplot` and some other packages. Theme is a function with an information about visualisation style and it can be added to any `ggplot` object as an extra layer.

Check the help file for themes inside `ggplot` package. Here are some examples:

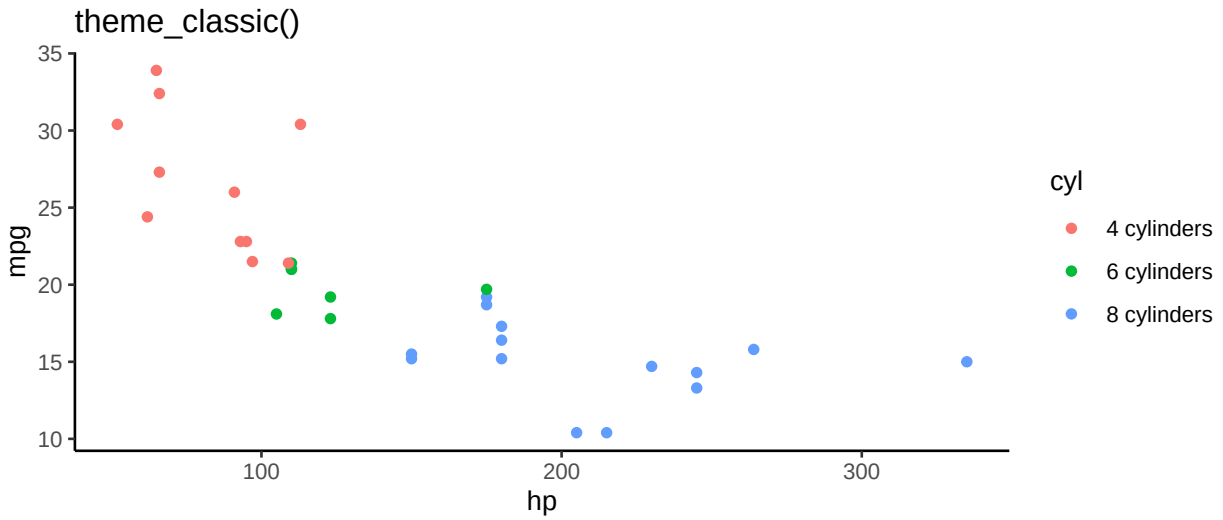
```
# our basic scatterplot
gr <- ggplot(data = df, aes(y = mpg, x = hp, colour = cyl)) + geom_point()

gr + ggtitle("default theme")
```



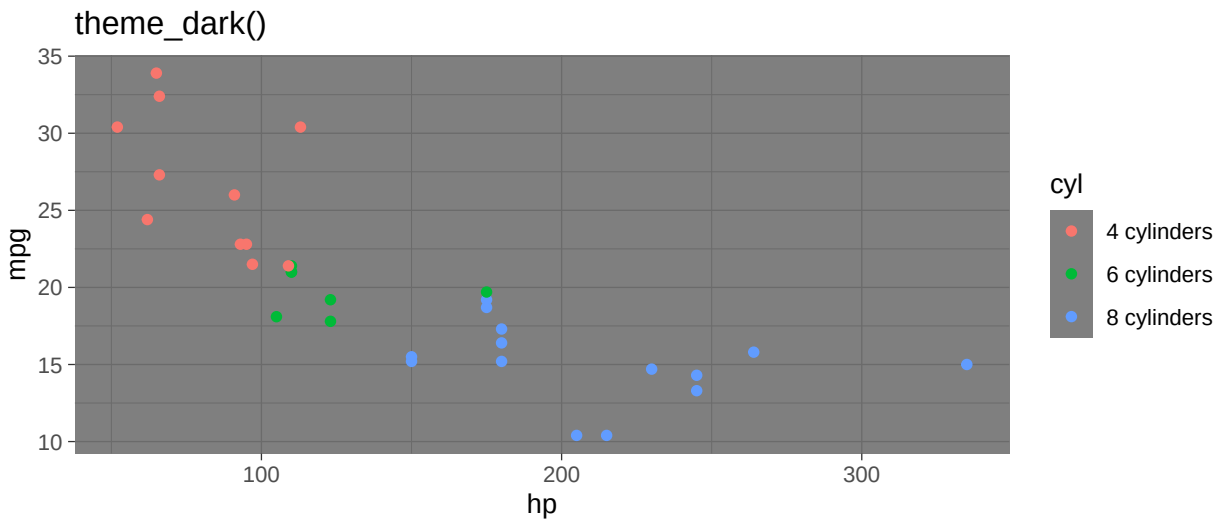

```
gr + theme_bw() + ggtitle("theme_bw()")
```



```
gr + theme_classic() + ggtitle("theme_classic()")
```

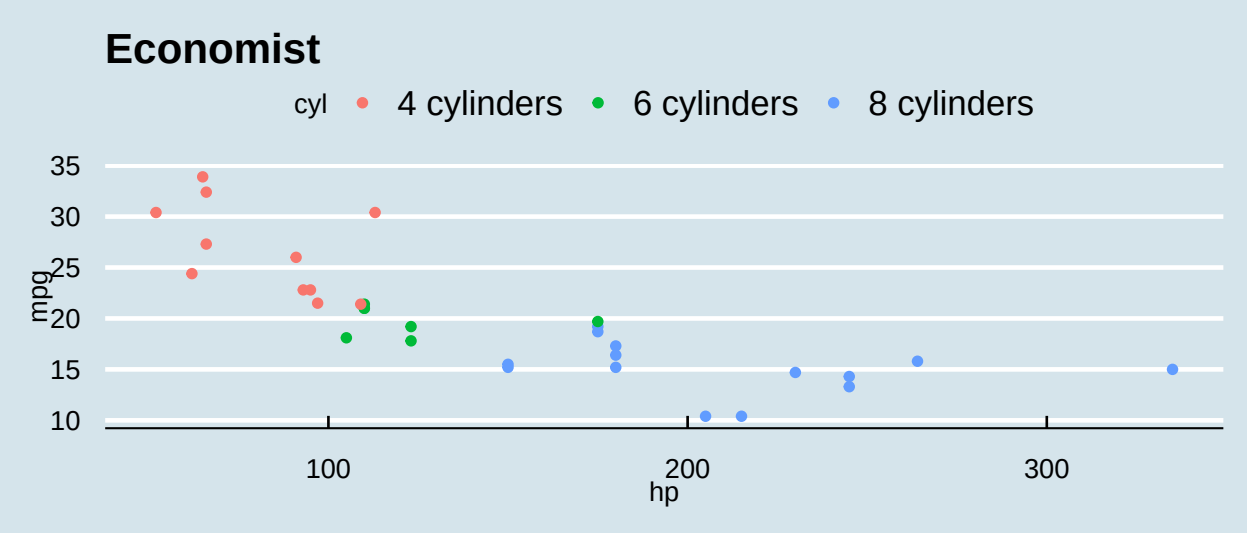



```
gr + theme_dark() + ggtitle("theme_dark()")
```

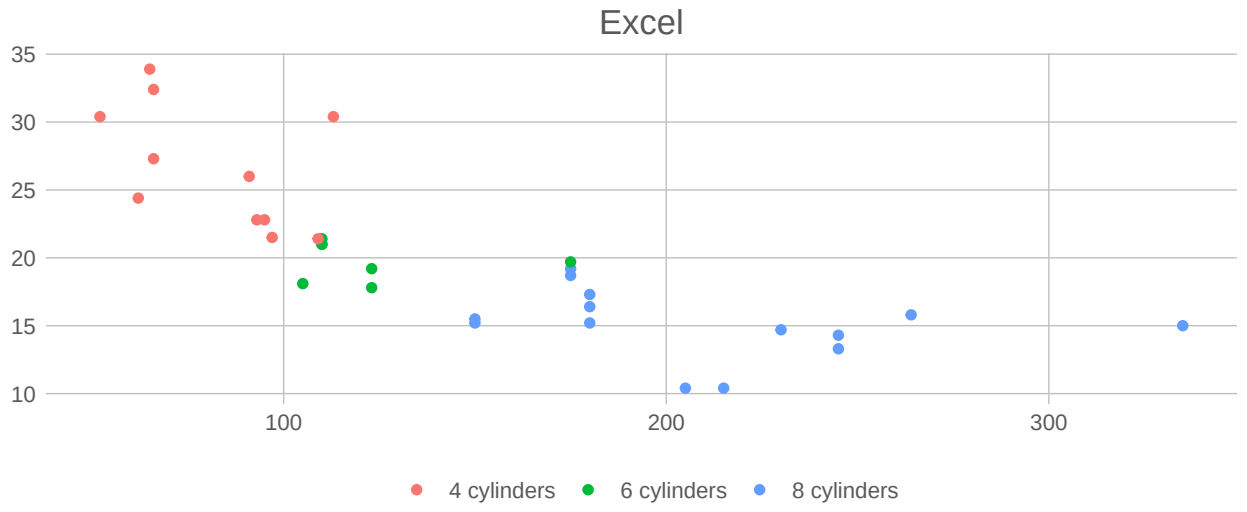


There is a package `ggthemes` that brings extra themes. These themes refer to styles and colour palette of some populat journals or software packages.

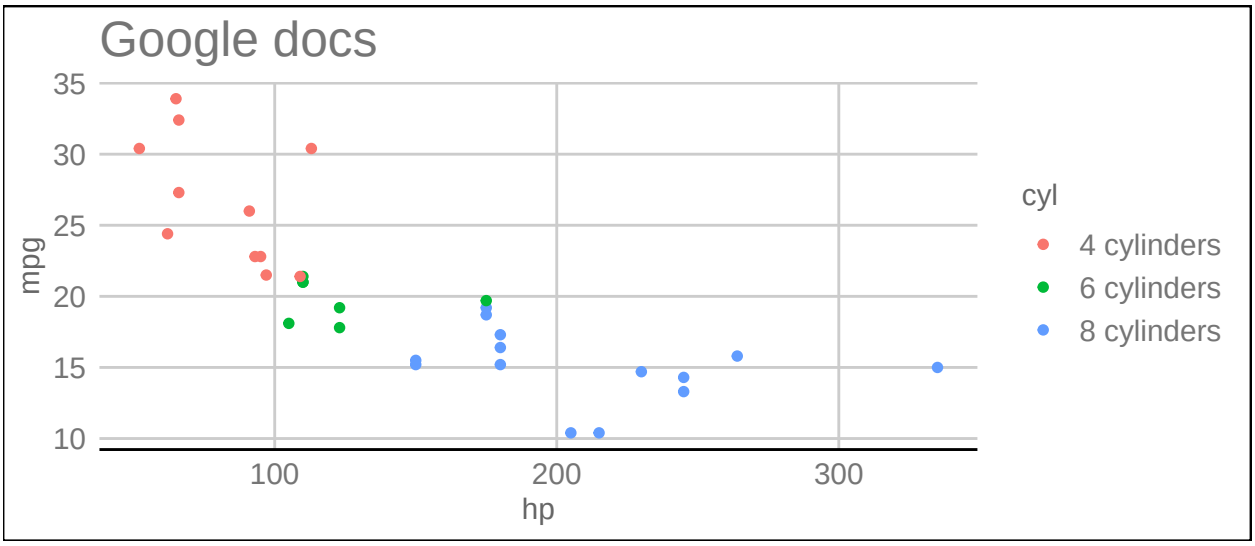
```
# load the package
library(ggthemes)

# the same basic scatterplot
gr <- ggplot(data = df, aes(y = mpg, x = hp, colour = cyl)) + geom_point()

gr + theme_economist() + ggtitle("Economist")
```



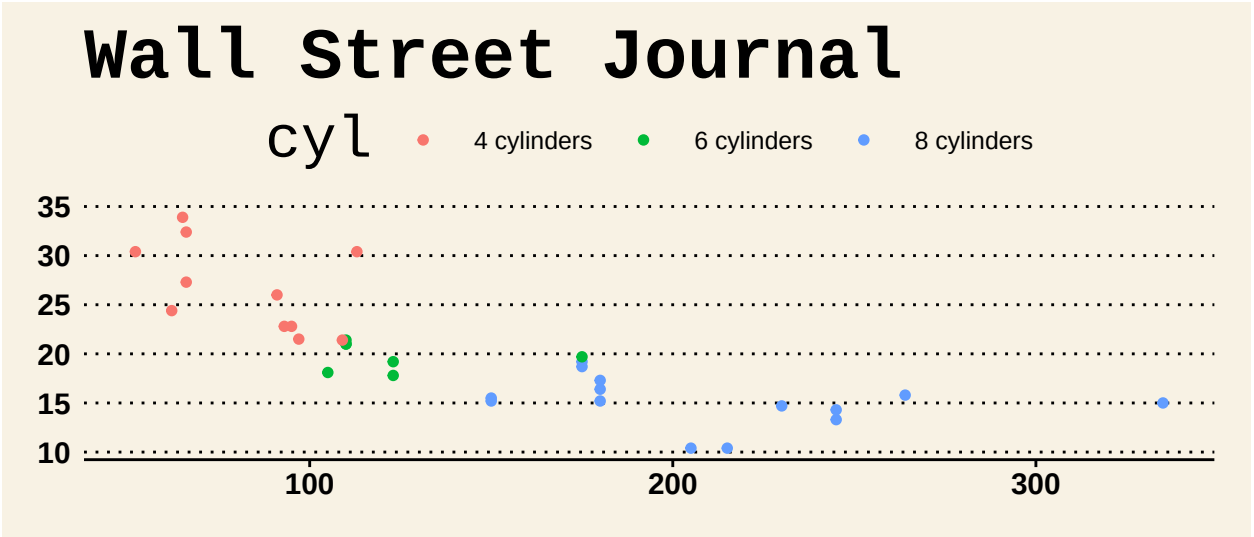

```
gr + theme_excel_new() + ggtitle("Excel")
```



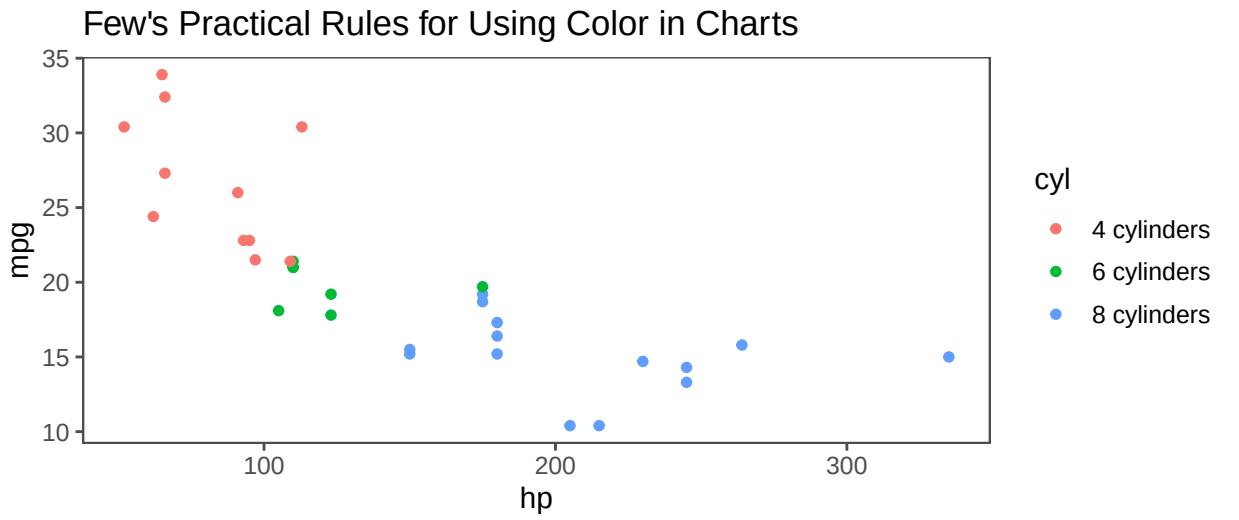
```
gr + theme_gdocs() + ggtitle("Google docs")
```



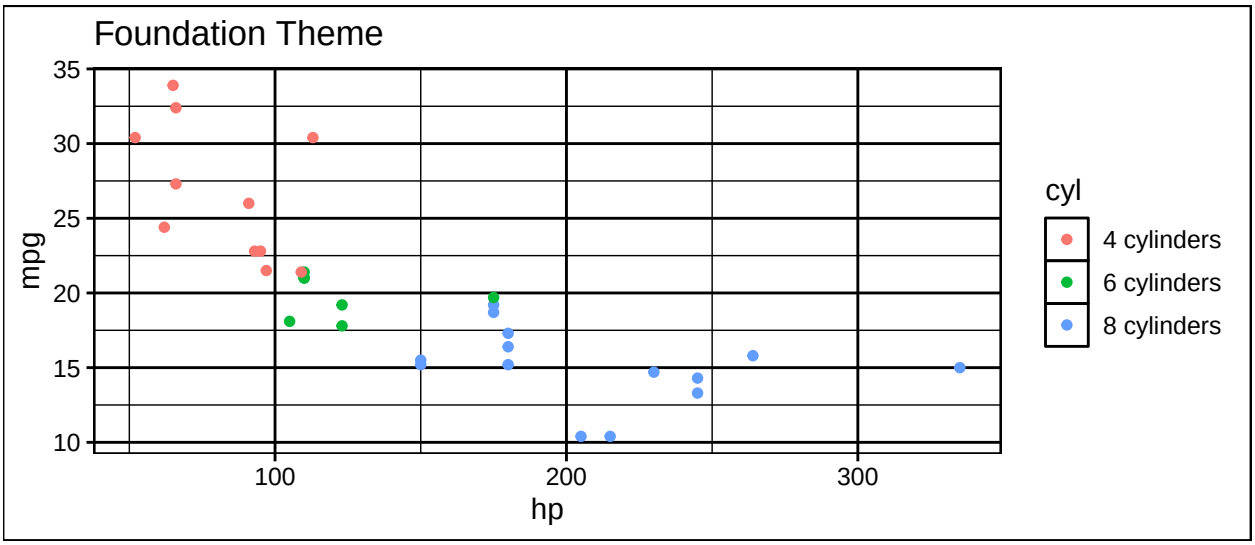

```
gr + theme_wsj() + ggtitle("Wall Street Journal")
```



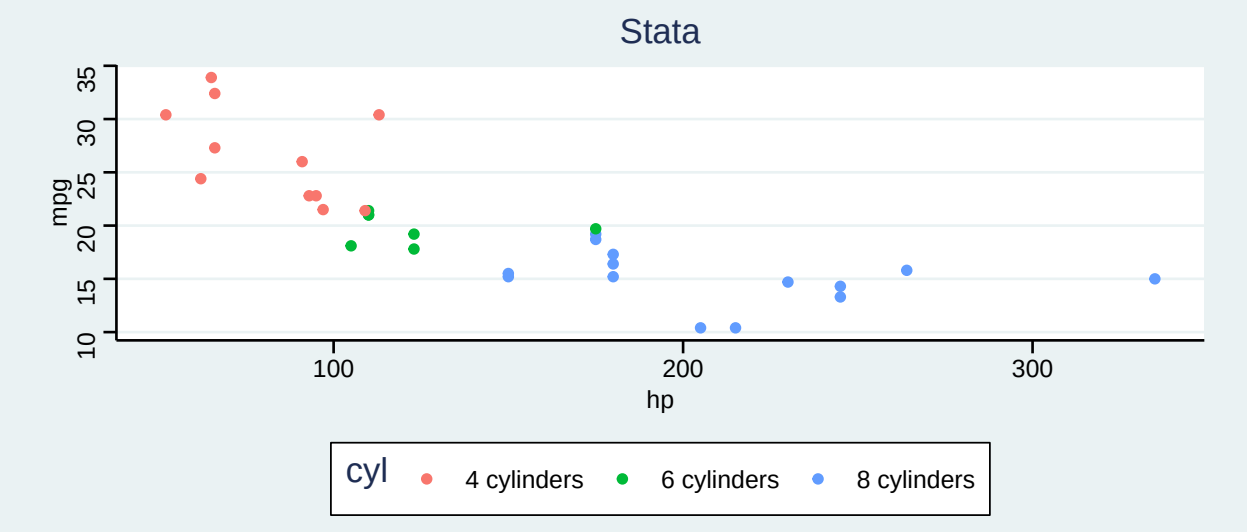
```
gr + theme_few() + ggtitle("Few's Practical Rules for Using Color in Charts")
```



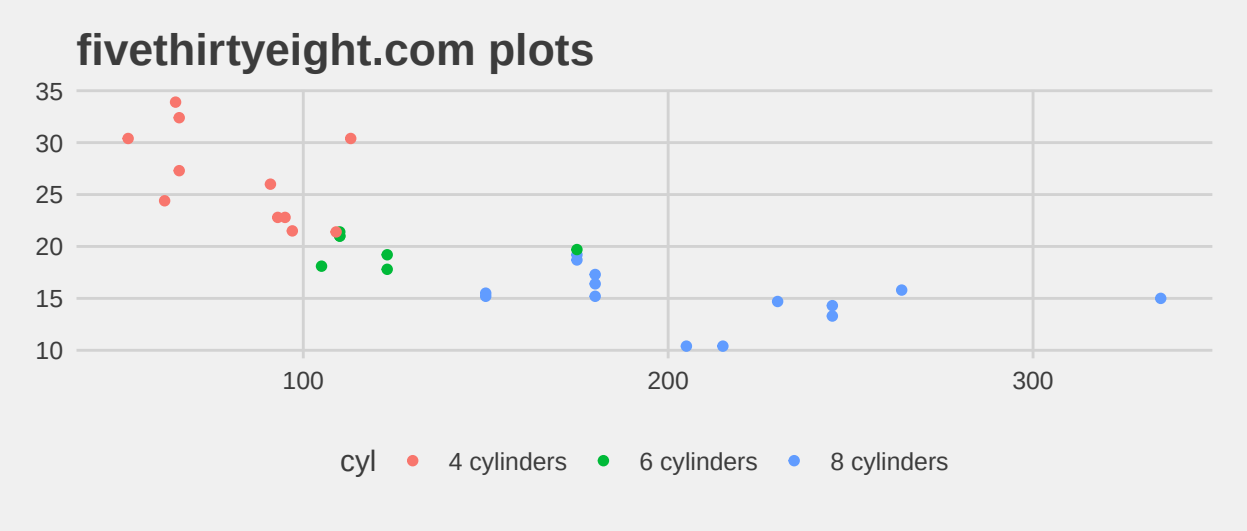

```
gr + theme_foundation() + ggtitle("Foundation Theme")
```



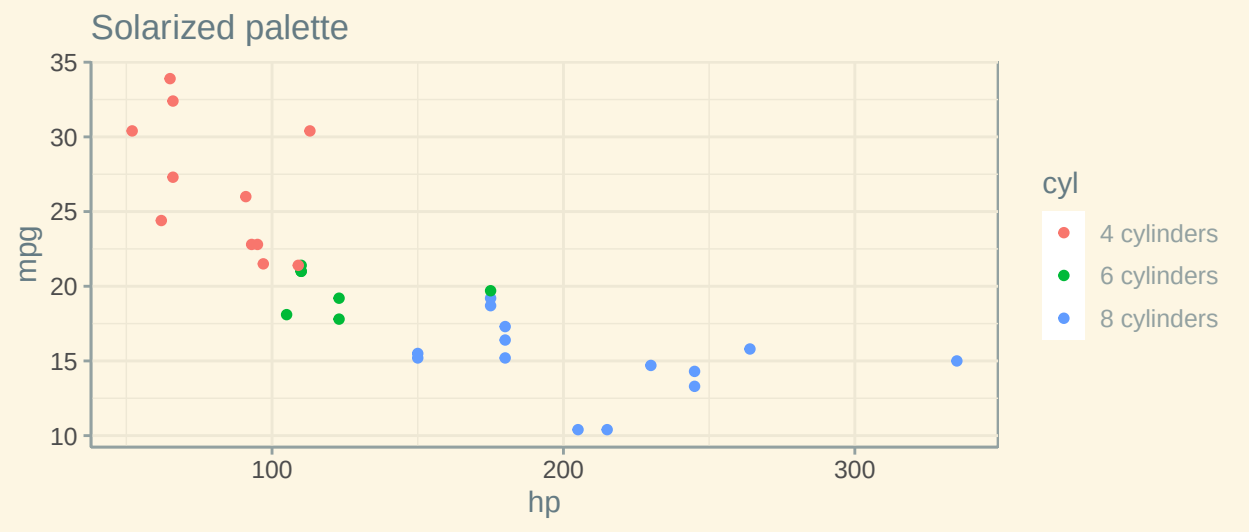
```
gr + theme_stata() + ggtitle("Stata")
```



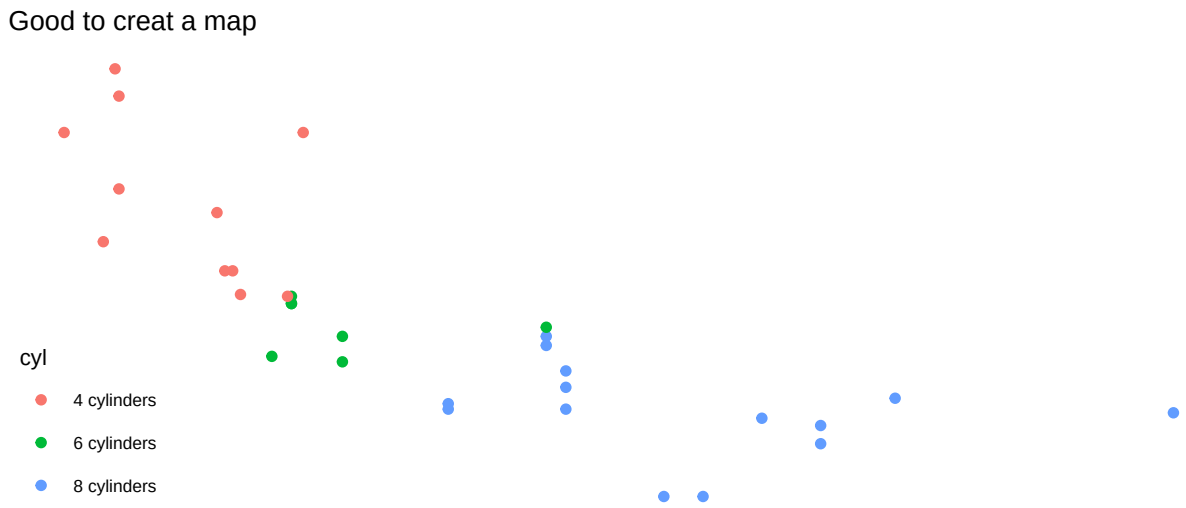

```
gr + theme_fivethirtyeight() + ggtitle("fivethirtyeight.com plots")
```



```
gr + theme_solarized() + ggtitle("Solarized palette")
```




```
gr + theme_map() + ggtitle("Good to creat a map")
```



You can change any settings and adjust existing themes or you can greate your own theme.

Specialised data visualisations

There are some packages with somewhat narrow specialisation. For example, package `geofacet` help you to arrange multiple graphs in an order that represents geographical locations. Only US states are supported at the moment, still it is an impressive example.

```
# extra library to load
library(geofacet)

# check the data set for the example, look at variables
# recall conversion to long table in week 3
head(state_ranks, 10)
```

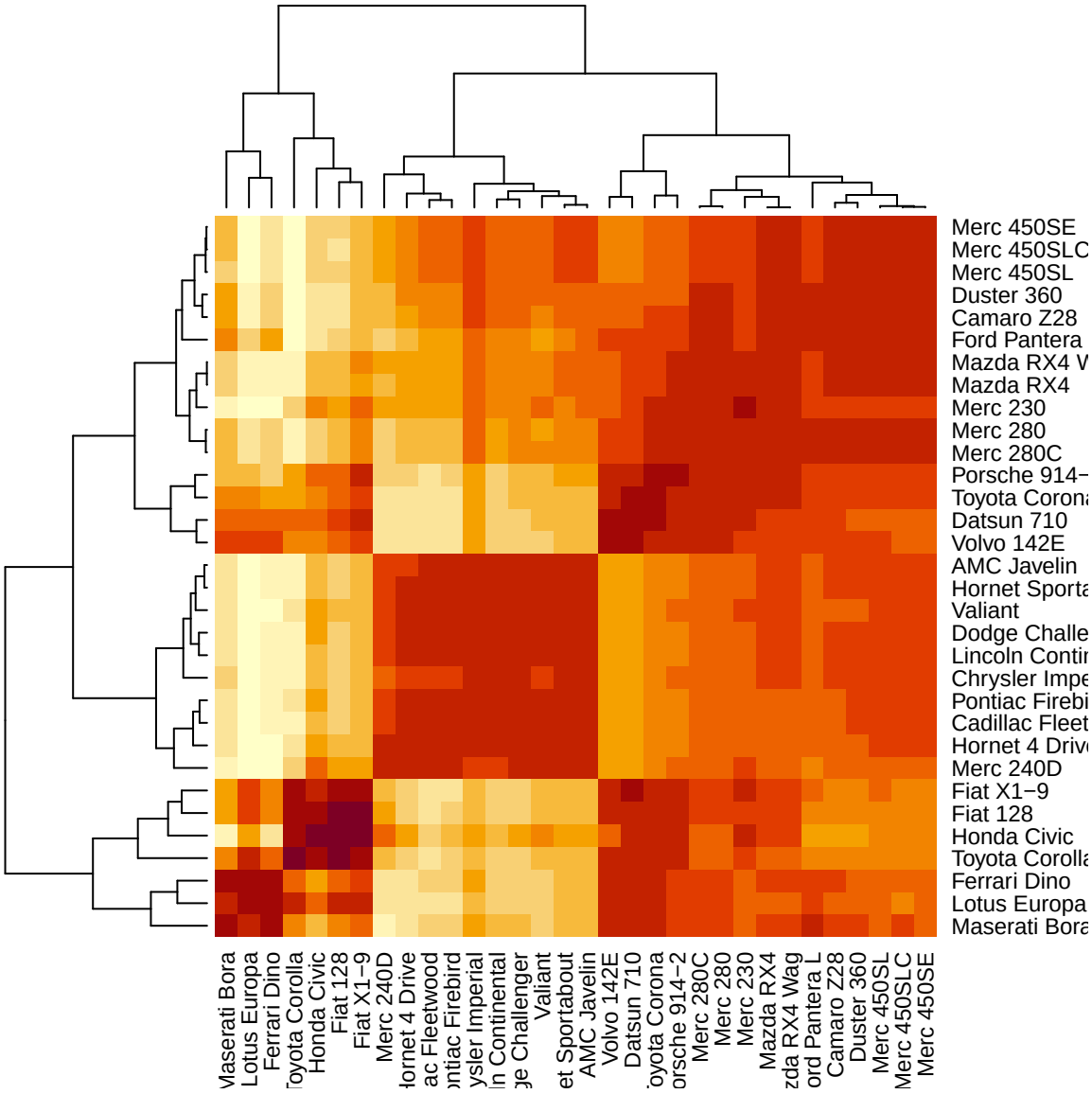
```
##    state    name   variable rank
## 1     AK  Alaska  education   28
## 2     AK  Alaska employment   50
## 3     AK  Alaska     health   25
## 4     AK  Alaska     wealth    5
## 5     AK  Alaska      sleep   27
## 6     AK  Alaska    insured   50
## 7     AL Alabama  education   45
## 8     AL Alabama employment   49
## 9     AL Alabama     health   48
## 10    AL Alabama     wealth   47
```

```
# you can check the help file if you want to know more about data
# ?geofacet::state_ranks

# define ggplot object
ggplot(data = state_ranks, aes(x = variable, y = rank, fill = variable)) +
  # add a layer of columns
  geom_col() +

  # make columns horizontal
  coord_flip() +

  # add state-based locations for each graph
  facet_geo(~ state) +

  # adjust location of the legend and reduce font size as there are too much information
  theme(legend.position="bottom", text = element_text(size=8))
```


WI
VT
NH
ME
WA
ID
MT
ND
MN
IL
MI
NY
MA
OR
NV
WY
SD
IA
IN
OH
PA
NJ
CT
RI
CA
UT
CO
NE
MO
KY
WV
VA
MD
DE
AZ
NM
KS
AR
TN
NC
SC
DC
OK
LA
MS
AL
GA
HI
AK
TX
FL
wealth
sleep
insured
health
employment
education
0 10 20 30 40 50
variable
rank
education
employment
health
insured
sleep
wealth

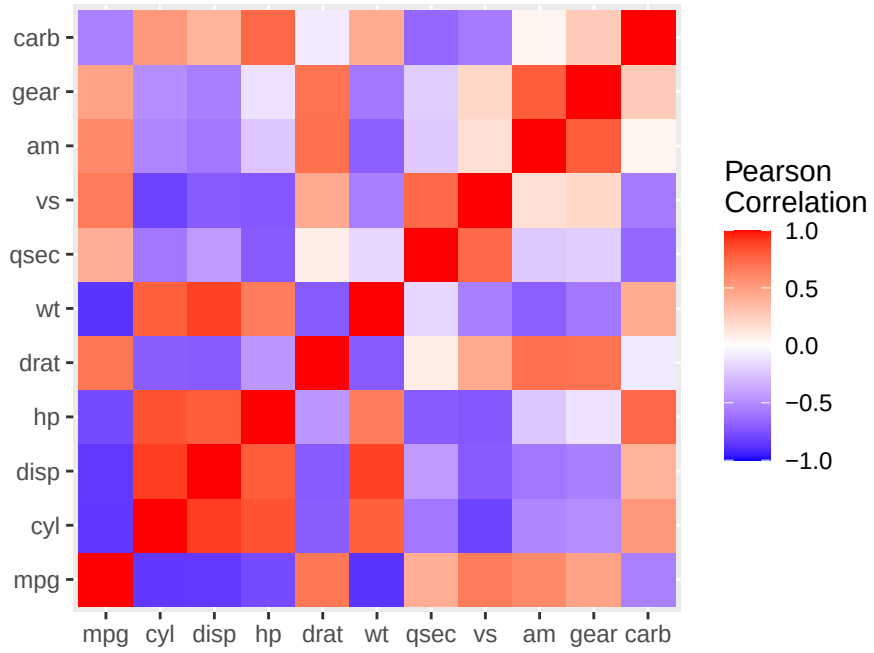
Heatmap is a very powerful visualisation to study relationships between variables.

```
# make a heatmap for mtcars data set
# this is a first step toward next week topic - clustering
heatmap(cor(t(mtcars)))
```



Simpler version of a heatmap with ggplot. It is useful for plotting correlations matrices.

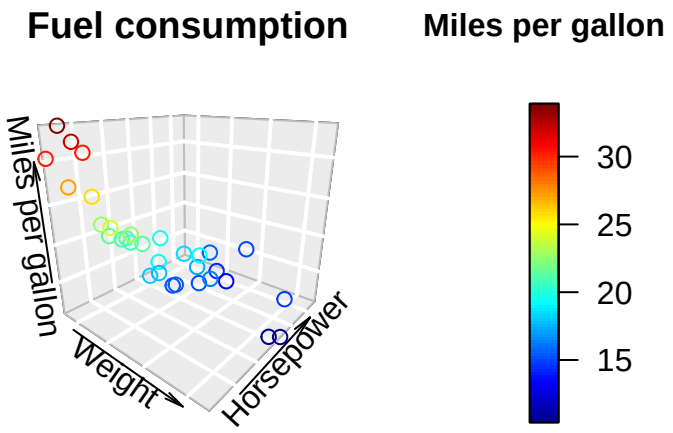
```
# prepare data first
# beware of extra package "reshape2" used to convert correlation matrix
# from wide format to long table format required for "ggplot2"
mtcars %>% cor %>% reshape2::melt() %>%

  # create ggplot object
  ggplot(aes(x = Var1, y = Var2, fill = value)) +

  # make a heatmap
  geom_tile() +

  # adjust fill colours
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson\nCorrelation") +

  # make the graph square
  coord_fixed() +

  # remove axis names as they are not really informative
  labs(x = NULL, y = NULL)
```



3D graphs

With R you can make 3D data visualisations. Your print out will be on the flat screen or flat piece of paper, that is 2D. As a result, the output might look messy. However, sometimes and some particular situations 3D data visualisations might be useful. Here are several examples.

```
library(plot3D)

scatter3D(z = mtcars$mpg, y = mtcars$hp, x = mtcars$wt, # values for axes
          colvar = mtcars$mpg,                          # values for colour
          zlab = "Miles per gallon",
          ylab = "Horsepower", xlab = "Weight",         # names for axis
          clab = "Miles per gallon",                    # name for colour
          theta = 40, phi = 20,     # the angles defining the viewing direction
          bty = "g",                                    # type of the box
          main = "Fuel consumption")                    # main heading
```

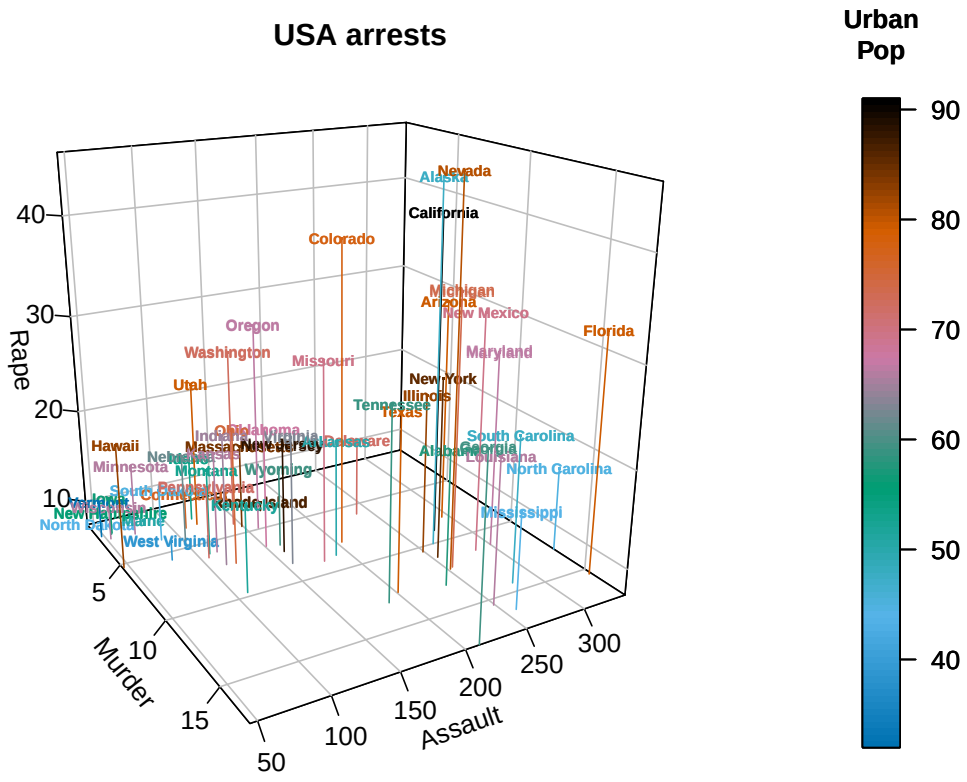



```
# Check new data set
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
with(USArrests, text3D(Murder, Assault, Rape,
    colvar = UrbanPop, col = gg.col(100), theta = 60, phi = 20,
```

```
    xlab = "Murder", ylab = "Assault", zlab = "Rape",
    main = "USA arrests",
    labels = rownames(USArrests), cex = 0.6,
    bty = "b2", ticktype = "detailed", d = 2,
    clab = c("Urban","Pop"), adj = 0.5, font = 2))

with(USArrests, scatter3D(Murder, Assault, Rape,
    colvar = UrbanPop, col = gg.col(100),
    type = "h", pch = ".", add = TRUE))
```



Graphs above are static. You have to choose the viewing angle to plot them. However, if you run the code below after creating the graph with `plot3D` package and you get an extra window with dynamic 3D graph. You can rotate it in 3 dimensions.

```
library(plot3Drgl)
plotrgl()
```

... and when you are happy with the perspective, you can save it as a graphical file

```
snapshot3d("my3Dplot.png")
```

3D data visualisations are useful to show some surfaces where one axis is a function of two other axes. To get such graph you need to have a lot of observations. Ideally, they have to cover all possible values of axes x and y.

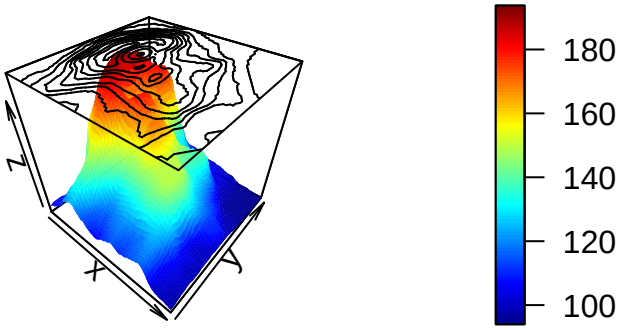
```
# new data set
volcano[1:5, 1:10]  # top 5 row for first 10 columns
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  100  100  101  101  101  101  101  100  100   100
## [2,]  101  101  102  102  102  102  102  101  101   101
## [3,]  102  102  103  103  103  103  103  102  102   102
## [4,]  103  103  104  104  104  104  104  103  103   103
## [5,]  104  104  105  105  105  105  105  104  104   103
```

```
dim(volcano)    # it is a matrix 87 rows and 61 columns of 10x10 meters grid
```

```
## [1] 87 61
```

```
persp3D(z = volcano, contour = list(side = "zmax"))
```



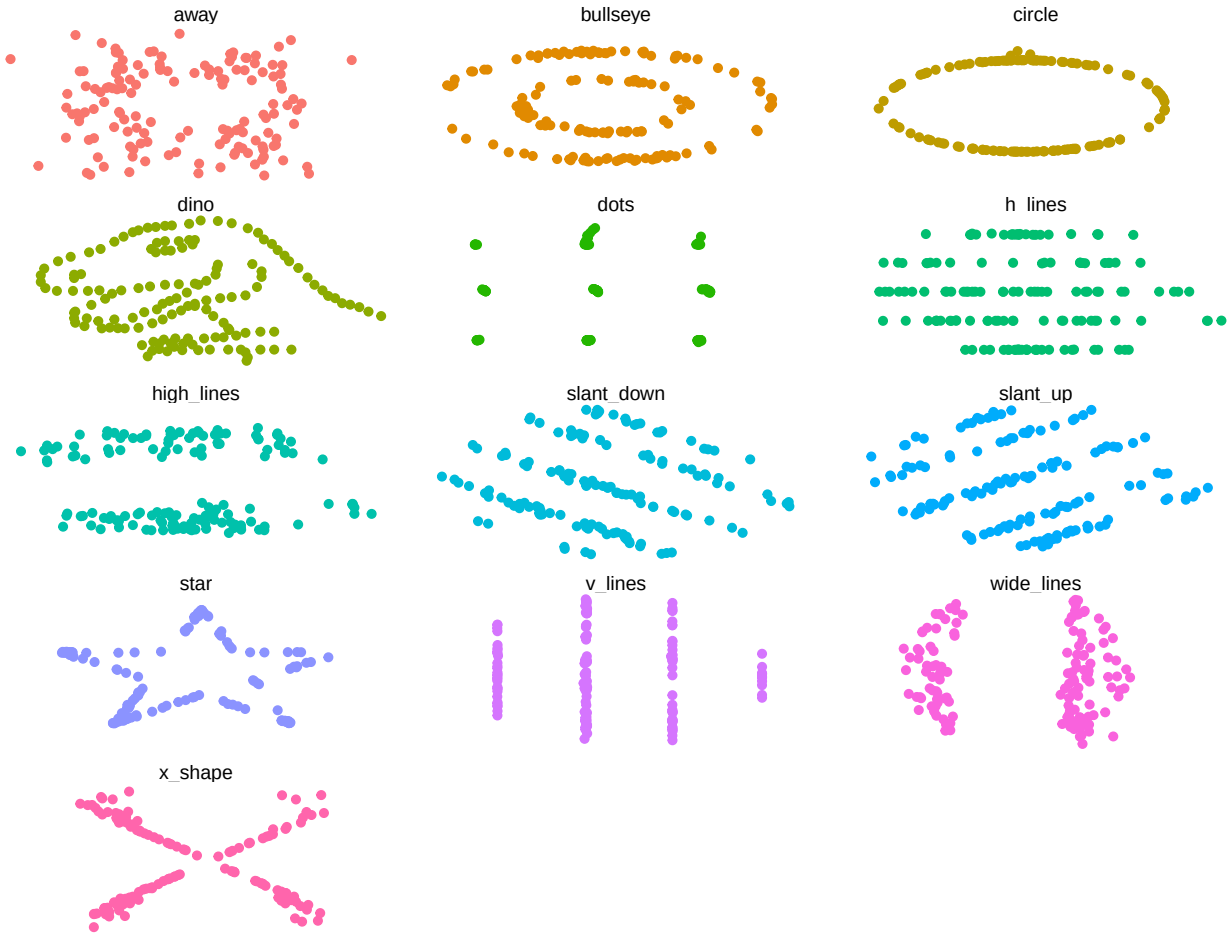

```
# try the command below for a dynamic version of the graph
# persp3Drgl(z = volcano, contour = list(side = "zmax"))
```

Animation

When you want to show the process that changes over time, you might want to employ an animation. Animation is a very simple concept – it is a set of graphs or images that changes every second or half a second or even quicker. Human eye can not distinguish individual frame if they change quicker than 24 frames per second.

And R can do it for you.

```
# collection of data sets with the same summary statistics
library(datasauRus)

# examples of visualisations of these data sets
ggplot(datasaurus_dozen, aes(x = x, y = y, colour = dataset)) +
    geom_point() +
    theme_void() +
    theme(legend.position = "none") +
    facet_wrap( ~ dataset, ncol = 3)
```




```
# package for ggplot-based animation
library(gganimate)

# prepare a set of ggplots that will be frames
```

```
anim <- ggplot(datasaurus_dozen, aes(x=x, y=y))+
  geom_point()+
  theme_minimal() +
  transition_states(dataset, 3, 1) +
  ease_aes('cubic-in-out')

# you need to install "magick" library then uncomment
# and run the line below to create animation

# animate(anim, renderer = magick_renderer())
```

The package `gganimate` creates animation but to store it you need other packages. For example, package `magick` above to see animation in RStudio; package `gifski` to store frames as one animated gif-file; `ffmpeg` or `av` for video files. Please check the help file.

An alternative package for animation is `animation`. It can create animation that will be embedded into a web page or flash file or even PDF.

```
library(animation)

saveHTML({
  par(mar = c(4, 4, 0.5, 0.5))
  for (i in 1:20) {
    plot(runif(20), ylim = c(0, 1))
    ani.pause()
  }
}, title = "Demo of 20 uniform random numbers")
```

Run the above code in R. There will be an HTML file created for you and opened in web browser. HTML page will show data animation and controls to make it slower or quicker, run it step-by-step or pause. You can find more examples on the developers' web page – https://yihui.org/animation/examples/

Extra resources

Excellent guide on `ggplot` package: http://www.sthda.com/english/wiki/ggplot2-essentials

Ggplot colours tricks and tips: https://www.datanovia.com/en/blog/ggplot-colors-best-tricks-you-will-love/

Animations using R: https://towardsdatascience.com/animating-your-data-visualizations-like-a-boss-using-r-f94ae20843e3