ggplot2

Claude REN

10/31/2020

Introduction

We are going to study the library ggplot2, which is one of the most known library for graphical representation. For that we are going to use a dataset from : https://perso.telecom-paristech.fr/eagan/class/igr204/datasets It's a dataset on cars model, with information about the weight, speed and so on. This is a little glimpse of it :

```r
# Path of the file
file <- "cars.csv"

# Load the speed dating dataset
cars2 <- read.csv(file, sep = ";")

# Preview of the table
head(cars2)
```

```
##                           Car MPG Cylinders Displacement Horsepower Weight
## 1 Chevrolet Chevelle Malibu  18         8          307        130   3504
## 2         Buick Skylark 320  15         8          350        165   3693
## 3        Plymouth Satellite  18         8          318        150   3436
## 4             AMC Rebel SST  16         8          304        150   3433
## 5               Ford Torino  17         8          302        140   3449
## 6          Ford Galaxie 500  15         8          429        198   4341
##   Acceleration Model Origin
## 1         12.0    70     US
## 2         11.5    70     US
## 3         11.0    70     US
## 4         12.0    70     US
## 5         10.5    70     US
## 6         10.0    70     US
```

The purpose is to show how the package work, what you can do with it and what are its limit. The relevance of the dataset and what I'm going to plot is not the main focus, it only provides me enough information in order to show some of the function of ggplot2.
Before beginning by talking about why ggplot2 and what about the alternatives, I'd like to precise that it is entirely inspired by ***Selva Prabhakaran's*** tutorial on the subject :
http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html
http://r-statistics.co/Complete-Ggplot2-Tutorial-Part2-Customizing-Theme-With-R-Code.html

Alternatives

Credit to Roger Peng : https://github.com/rdpeng/CourseraLectures/blob/master/ggplot2_part1.pptx

We are going to use data that are already initialized in R :

```r
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

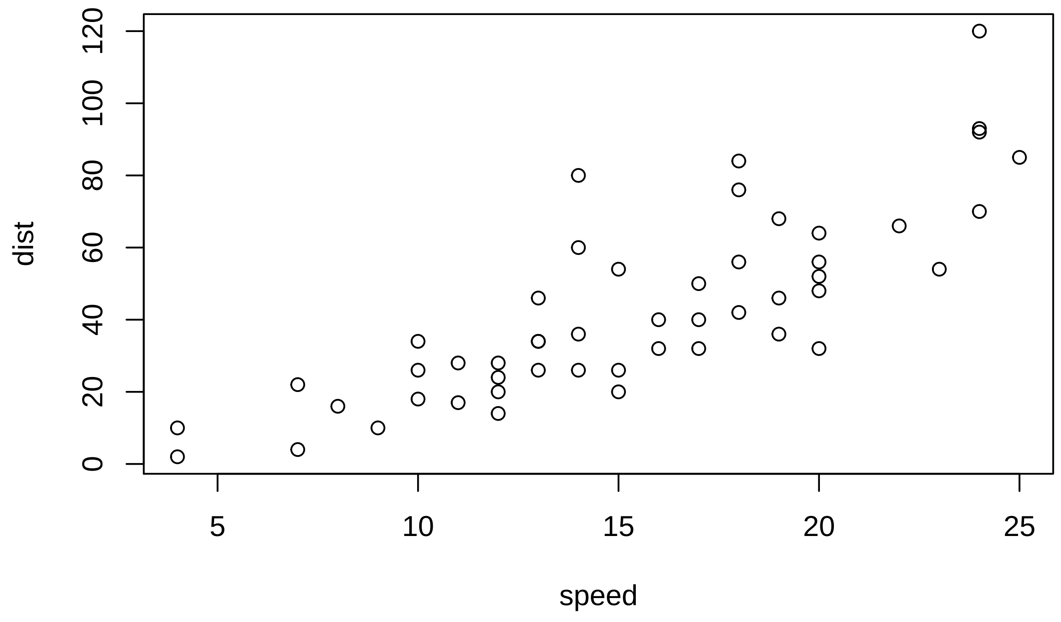
```r
head(EuStockMarkets)
```

```
##          DAX    SMI    CAC   FTSE
## [1,] 1628.75 1678.1 1772.8 2443.6
## [2,] 1613.63 1688.5 1750.5 2460.2
## [3,] 1606.51 1678.6 1718.0 2448.2
## [4,] 1621.04 1684.1 1708.1 2470.4
## [5,] 1618.16 1686.6 1723.1 2484.7
## [6,] 1610.61 1671.6 1714.3 2466.8
```

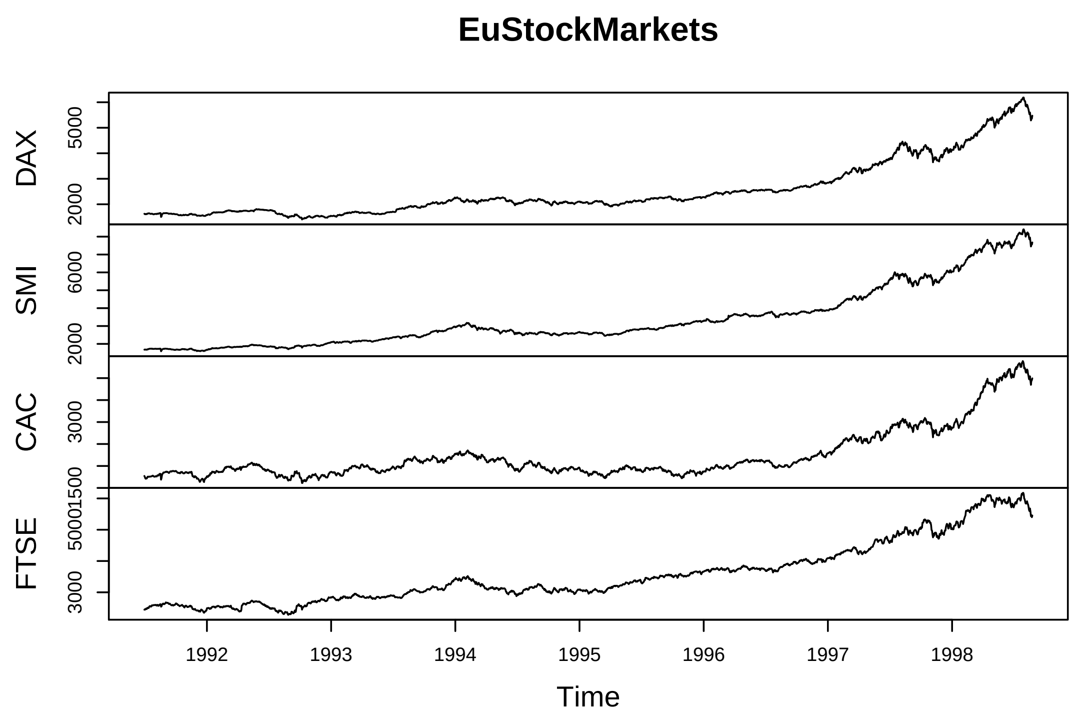
Base plotting system

Let's begin with the basics, the base plotting system, it's the integrated plotting tool of R. Like the many basics tools in R, its syntax is very minimalist and easy. Let's take a look at the example :

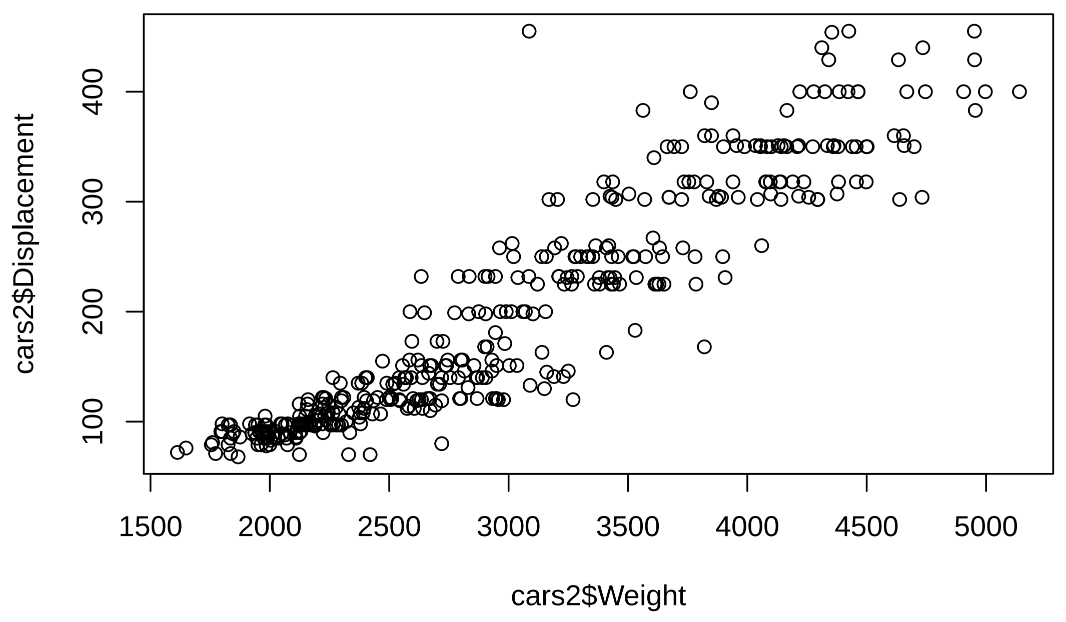
```r
plot(cars)
```



```r
plot(EuStockMarkets)
```



```
plot(cars2$Weight,cars2$Displacement)
```



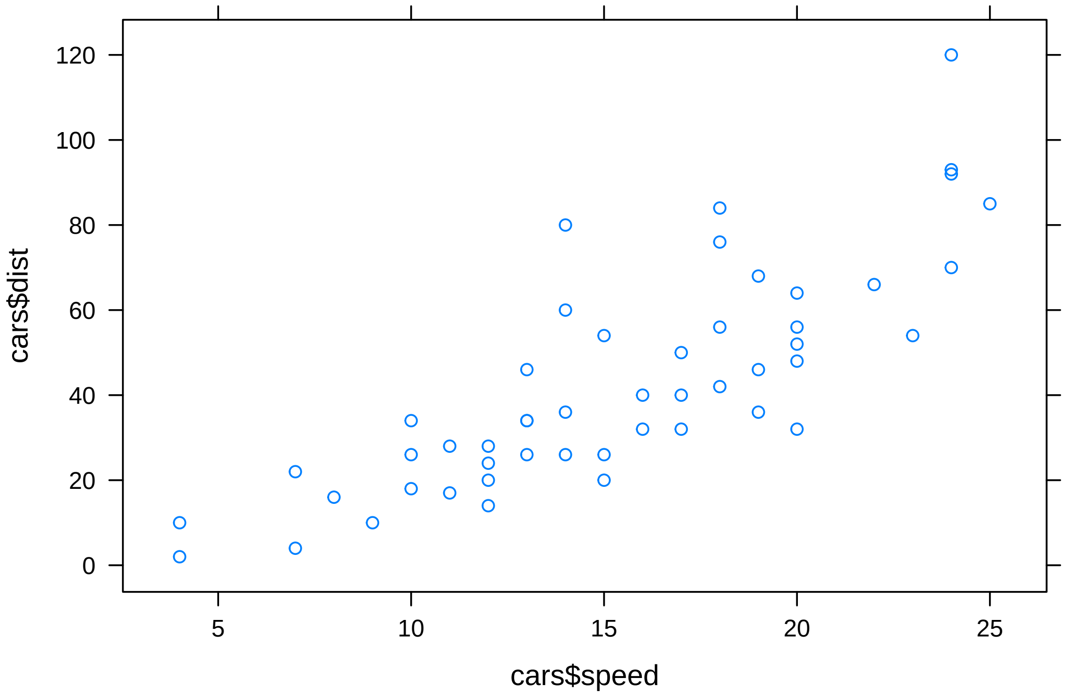
The results is not bad, you get what you look for, which is a representation of your set but in terms of customization and for dataframe type of data this is not what you want to use. The syntax become much more complicated and it is not intuitive either when you try to go further and look for better looking results.

- Pro :
  - Short and simple syntax
  - Useful for quick visualization
- Cons :
  - Hard to customize
  - No update or extensions

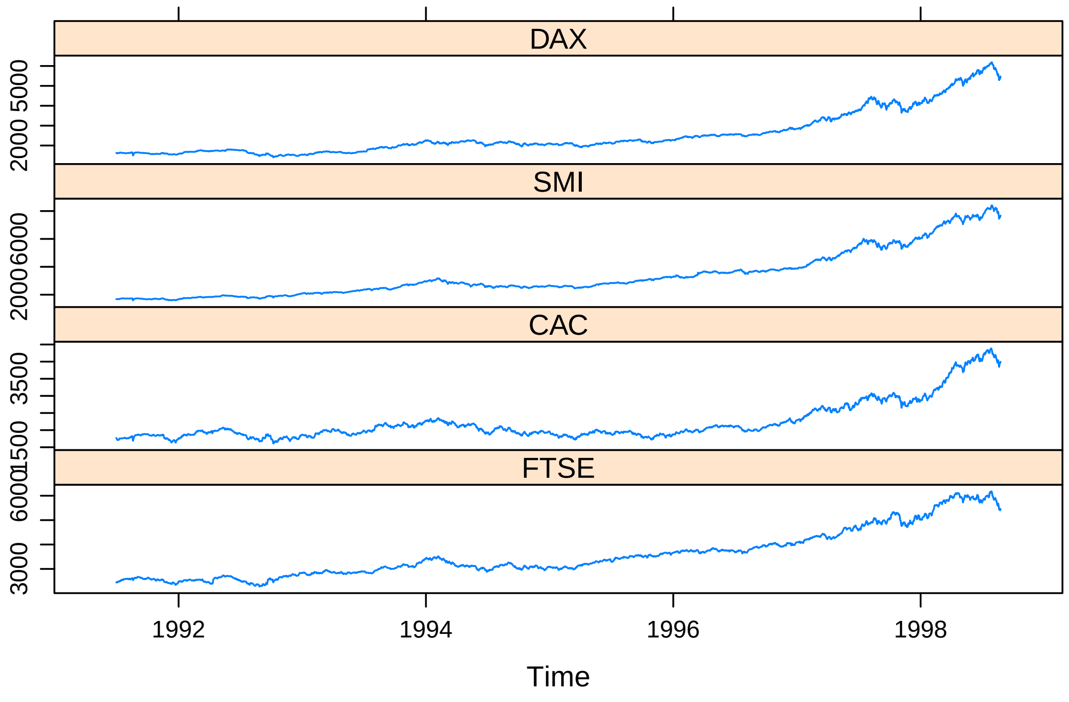
Lattice

Lattice is a far better plotting library than the base plot, you can have much more customization. For basic plot the syntax is not that hard and the basic plot provides you a more colorful result even without adding options.

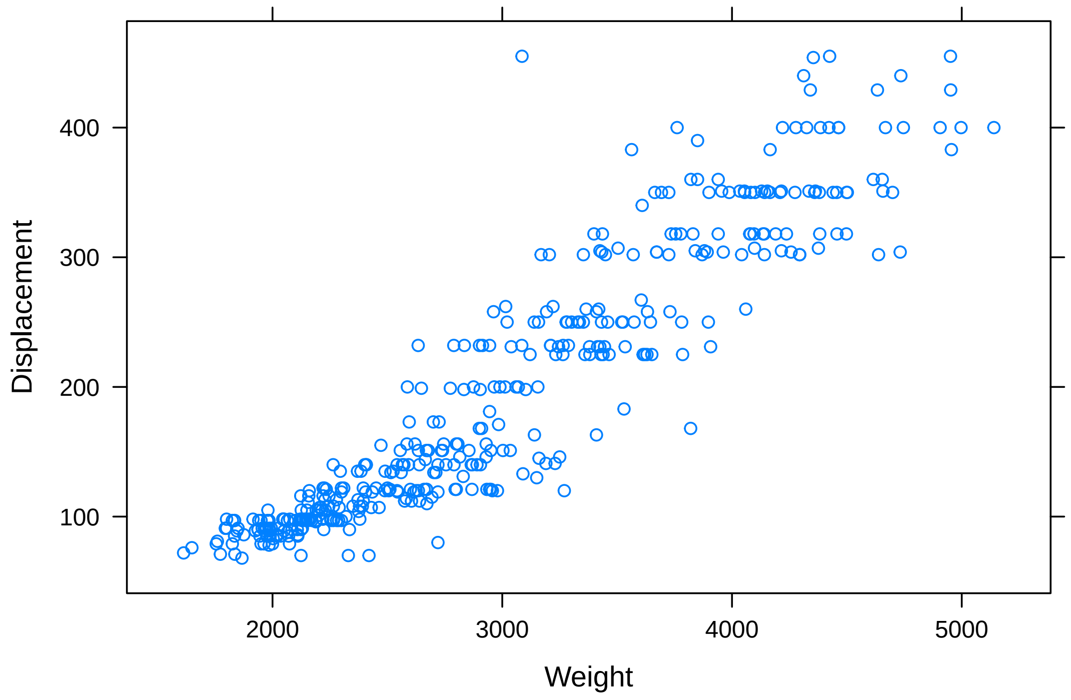
```r
library(lattice)
xyplot(cars$dist ~ cars$speed)
```



```r
xyplot(EuStockMarkets)
```



```r
xyplot(Displacement ~ Weight,
       data = cars2)
```



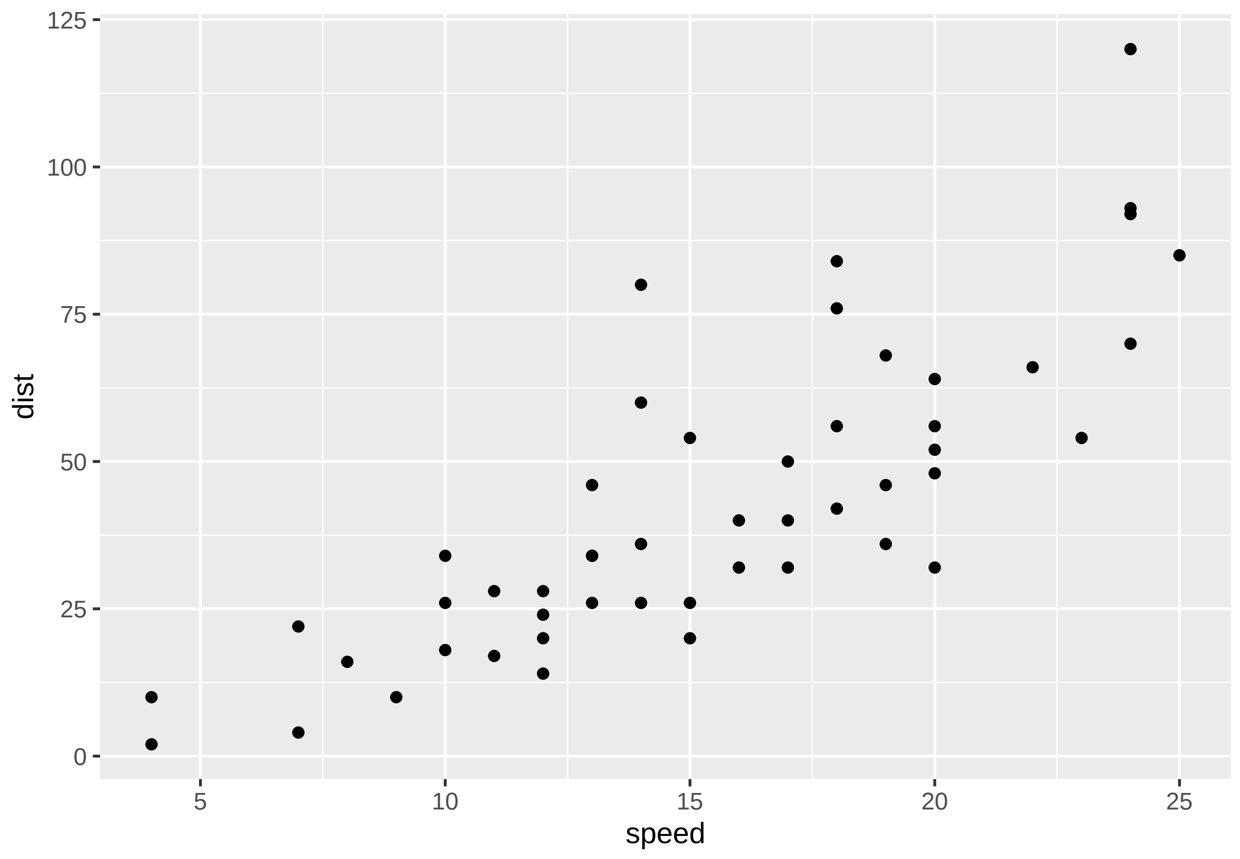
The results are better than the base plotting system, and the syntax is not much more complicated, important to point out that Lattice is very good to visualize time series as we can see with the stock markets one. In terms of customization, Lattice can be quite good and it can do many things that ggplot2. Many would choose one over the other, but both have their strengths and can be used as complement of each other. The only thing that put Ggplot2 at the top is the “adding” system which we’ll see further down.

- Pros :
  - Good for statistical graphics
  - Good for panels representation
  - Good for time series
  - Have extension
- Cons :
  - No major developments
  - Difficult to custom a plot
  - Specify everything in a single function call

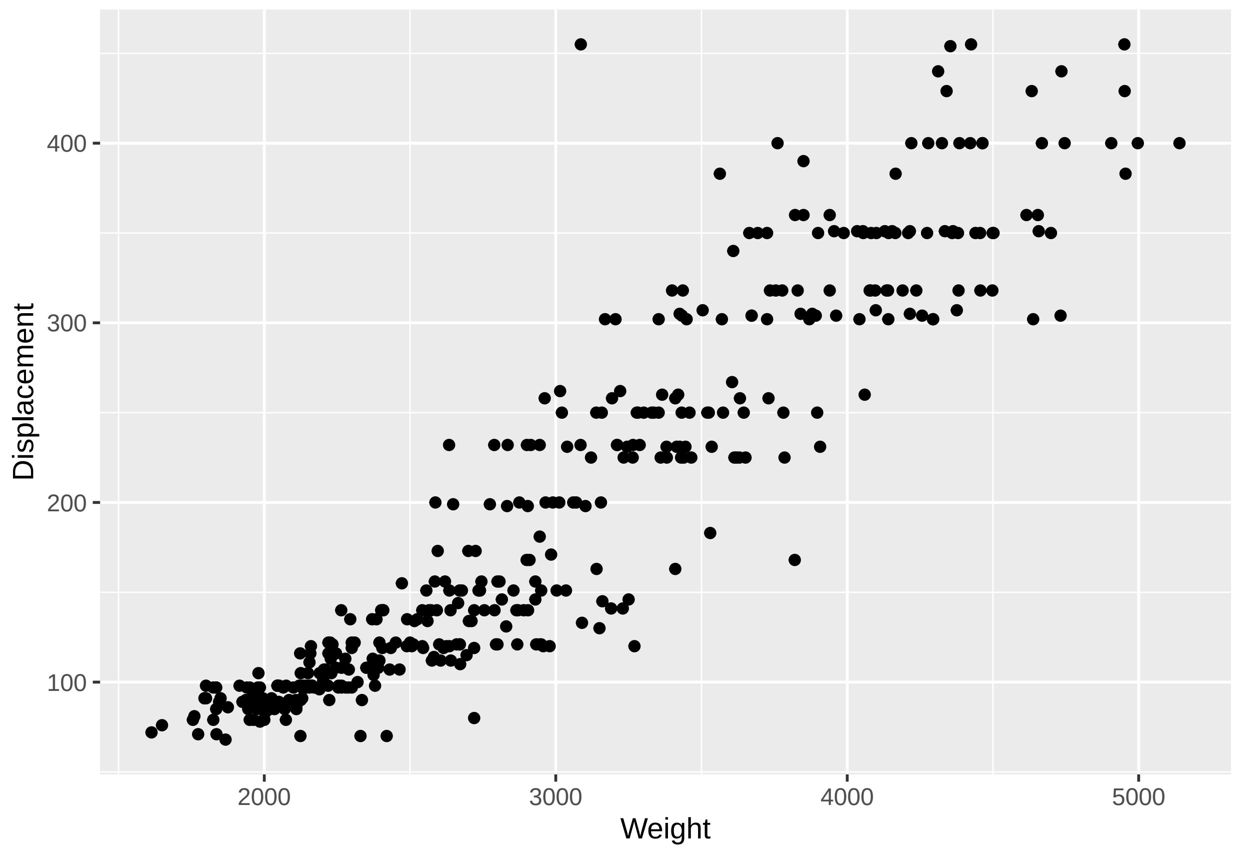
ggplot2

Finally it’s time to talk about Ggplot2, the most used and known graphical library of R. As for the other 2, we will first look at the example, and you’ll see that there is something strange :

```r
library(ggplot2)
ggplot(cars, aes(y=dist, x=speed)) + geom_point()
```



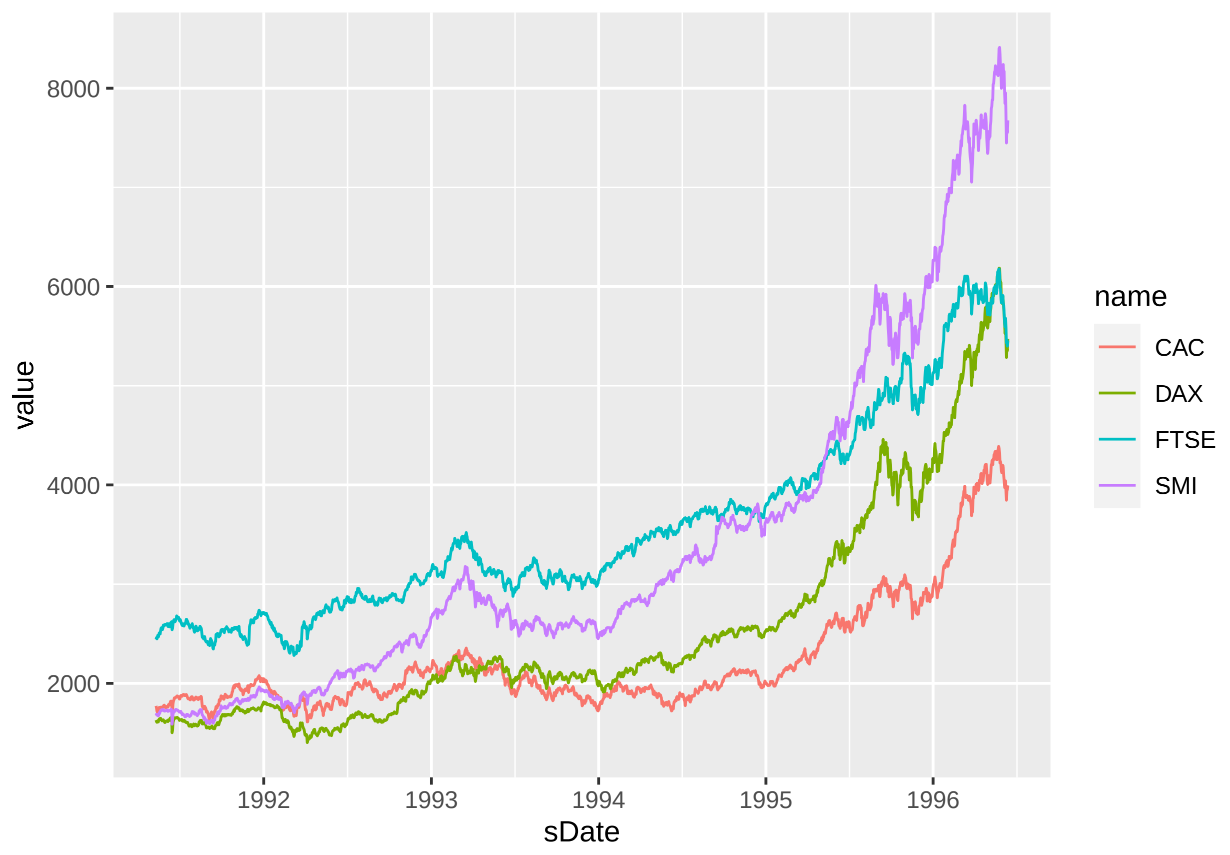
```r
ggplot(cars2, aes(x=Weight, y=Displacement)) + geom_point()
```



What is strange here, is that I didn't plot the time series data of the stock markets. The reason is simple, Ggplot2 is a graphical library dedicated to dataframe, so in order to represent time series, you have to transform it in a dataframe as below :

```
library(dplyr)
library(tidyr)
```

```
EuStockMarkets %>%
  as.data.frame() %>%
  mutate(sDate=as.Date(seq(1,1860,1), origin="1991-05-10")) %>%
  pivot_longer(-sDate) %>%
  ggplot(aes(x=sDate, y=value, color=name)) +
  geom_line()
```



- Little explanation of the syntax here, we used the dplyr and tidyr packages :
  - %>% : This means that you apply the function after to the element before, cars %>% head() = head(cars)
  - mutate() : Create, modify or delete columns of a dataset, here we used it to create a column sDate
  - as.Date() : Convert, here seq(1,1860,1), to a format similar to origin
  - seq() : Generate a sequence from 1 to 1860 by 1, 1860 represent the number of line, and one line is a day so by one. So here the we transform the sequence to a date starting from 1991-05-10
  - pivot_longer() : Transform columns in rows, here sDates is the columns to pivot, and by default it will create a columns "name" and "value"

As you can see, it is doable to plot time series with ggplot2 with not too much trouble but compared to Lattice or the base plotting system it is a bit trickier and it is important to know how to transform your data into dataframe and to know some functions like those above.
Ggplot2 can do much more, even though for those example it's difficult to witness the strength of it you can still see a glimpse of what you can do in terms of customization. Moreover we saw a bit of the "adding" syntax, which is very convenient because you don't have to write everything in a single syntax like in Lattice for example.

- Pros :
  - Based on Grammar of Graphics
  - Provides good defaults
  - Provides controls for almost every aspect of the plot
  - Active development
- Cons :
  - Complex syntax even for easy plot
  - High entry threshold

As showed below you can have a multitude of options so it can be a bit overwhelming at the beginning but in term of usability, the syntax is more intuitive even though it seems complex.

```r
ggplot(data, aes(x = x, y = y, color = color, size = size), alpha = '')+
  geom_()+
  scale_x_()+
  scale_y_()+
  scale_color_()+
  scale_size_()+
  coord_()+
  facet_()+
  labs(title = '', subtitle = '', caption = '')+
  theme_minimal(base_family = '', base_size = '')+
  theme(
    legend.position = '',
    plot.background = element_rect(fill = ''),
    plot.title = element_text(),
    plot.subtitle = element_text(),
    plot.caption = element_text(),
    plot.margin = unit()
  )
```

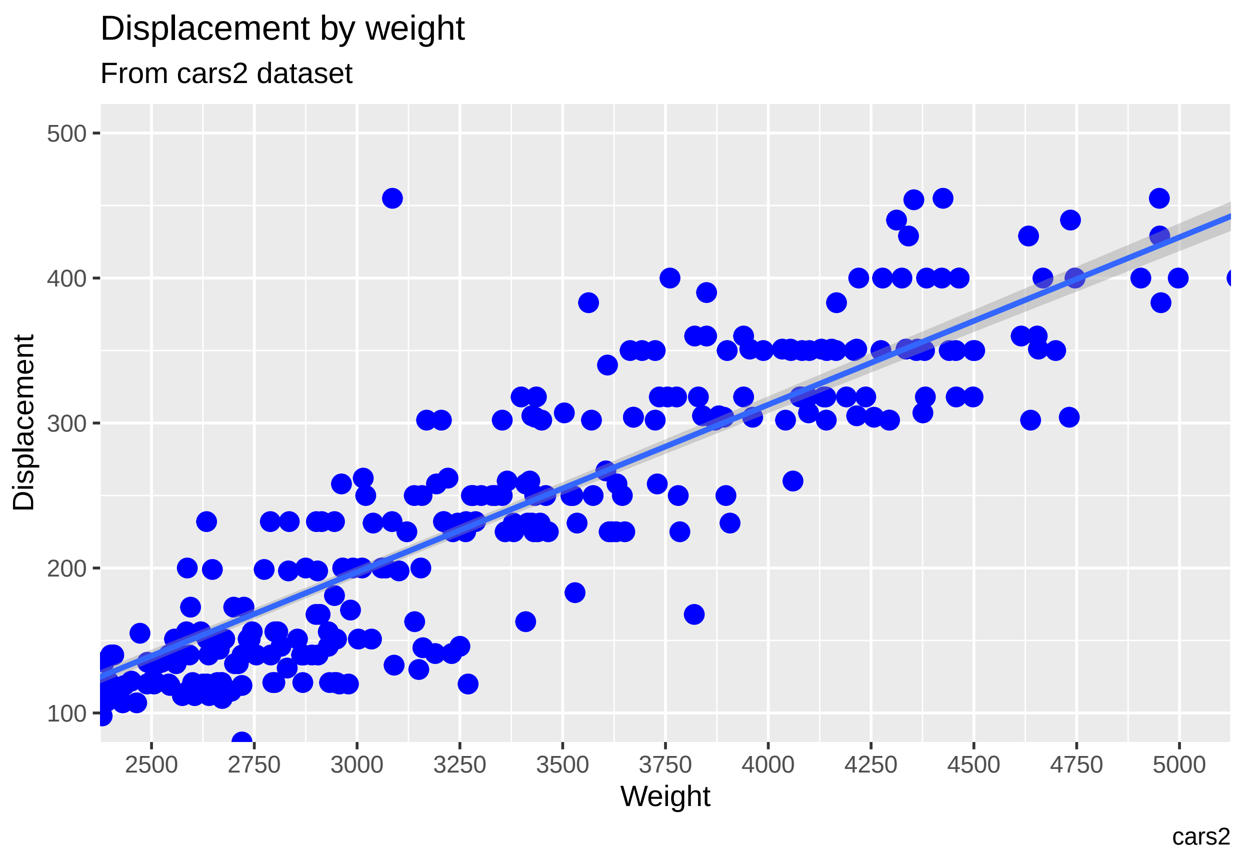
Study case

First of all, one particularity of ggplot2 is how you create the plot, when you don't precise the type of representation you want (points, bars etc…), it will simply plot a blank rectangle and when you bring the adding system it gives the user all the tools to customize your plot as you please.

- In this first chunk, we will see the basic function to plot something with a minimum of customization :
  - ggplot() : Is simply the function which initialize the plot, the first argument is the dataframe;
  - aes() : As stated, this function is about the aesthetic of the plot, you can define the x and y dataset, we will see in another chunk that you can define color and size too;
  - coord_cartesian() : Used to define xlim and ylim;
  - geom_point() : This is one of the function used to precise the type of representation you want, in the argument you can specify many options like color and size;
  - geom_smooth() : Add model, here lm means linear model
  - labs() : Specify title, subtitle, axis label

```r
# Initialization of the plot
g <- ggplot(cars2, aes(x=Weight, y=Displacement))

# Adding points
g <- g + geom_point(col="blue", size=3)

# Changing x and y axis
```

```r
g <- g + coord_cartesian(ylim=c(100, 500), xlim=c(2500, 5000))

# Adding linear model
g <- g + geom_smooth(method="lm")

# Adding title and legends
g <- g + labs(title="Displacement by weight", subtitle="From cars2 dataset",
              y="Displacement", x="Weight", caption="cars2")

# Customize axis
g <- g + scale_x_continuous(breaks=seq(2500, 5000, 250))

# Plotting
plot(g)
```



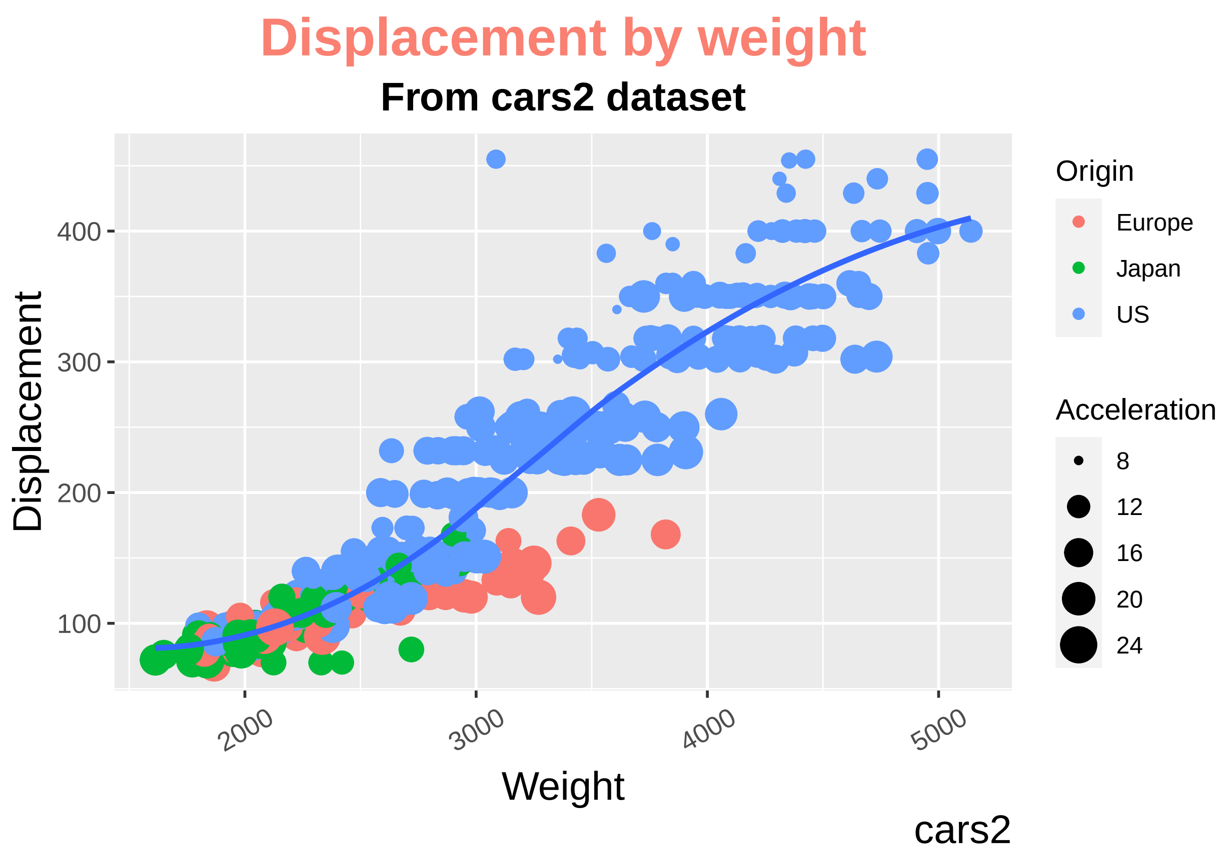
Theme customization syntax

In the following chunks, we will learn more about how to customize the theme of the plot, which includes basicaly everything you can see in the plot.

- In this chunk, we will see how to customize text element, color, size :
  - aes() : Here we used it in geom_point(), and choose to have color and size represented by the dataset, here the Origin and the Acceleration of the cars;

  - geom_smooth() : Add model, here loess for a non linear model, “se” is the confidence interval;
  - theme() : Initialize the theme customization :
    * element_text() : Customize the text size, police, color etc..
    * the vjust and hjust (vertical, horizontal), modify the position of the text

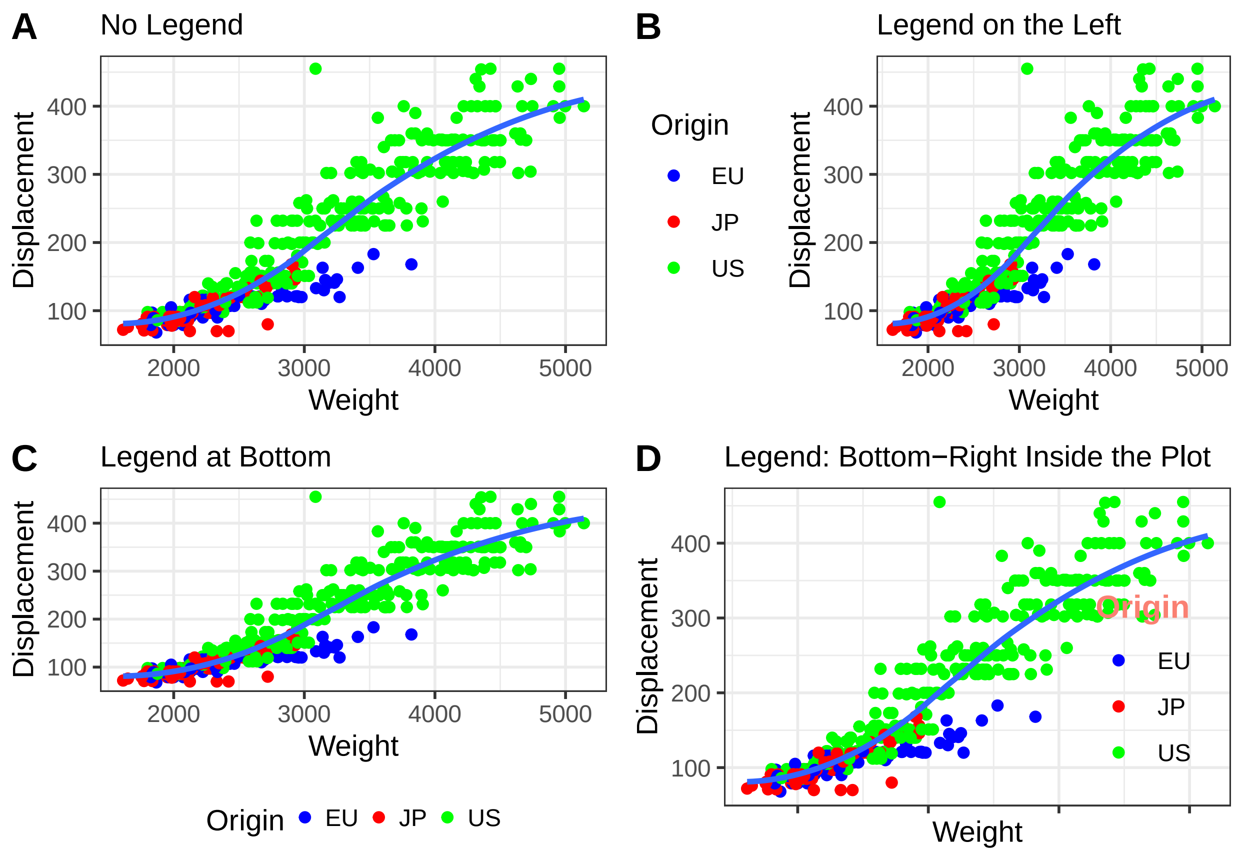
```r
g <- ggplot(cars2, aes(x=Weight, y=Displacement))
g <- g + labs(title="Displacement by weight", subtitle="From cars2 dataset",
              y="Displacement", x="Weight", caption="cars2")

# You can use data to customize the color and the size of your point
g <- g + geom_point(aes(col=Origin, size=Acceleration))

# Adding a non-linear model
g <- g + geom_smooth(method="loess", se=F)

# Customization of the theme
g <- g + theme(plot.title=element_text(size=20,
                                    face="bold",
                                    color="salmon",
                                    hjust=0.5,
                                    lineheight=1.2),  # title
            plot.subtitle=element_text(size=15,
                                       face="bold",
                                       hjust=0.5),  # subtitle
            plot.caption=element_text(size=15),  # caption
            axis.title.x=element_text(vjust=0,
                                      size=15),  # X axis title
            axis.title.y=element_text(size=15),  # Y axis title
            axis.text.x=element_text(size=10,
                                     angle = 30,
                                     vjust=.5),  # X axis text
            axis.text.y=element_text(size=10))  # Y axis text

plot(g)
```



- In this chunk, we will see how to customize the legend :
  - ggpubr : we are going to use this library in order to plot multiple plots;
  - scale_color_manual() : to change labels and colors, name is the legend you want to modify;
  - the legend options are quite self explanatory;
  - ggarange() : create a matrix containing the plot, rremove removed the x axis label.

```r
library(ggpubr)
g <- ggplot(cars2, aes(x=Weight, y=Displacement))
theme_set(theme_bw())
g <- g + geom_point(aes(col=Origin))
g <- g + geom_smooth(method="loess", se=F)

# Legends customization
g <- g + scale_color_manual(name="Origin",
                        labels = c("EU",
                                   "JP",
                                   "US"),
                        values = c("Europe"="blue",
                                   "Japan"="red",
                                   "US"="green"))

# No legend
no_leg <- g + theme(legend.position="None") + labs(subtitle="No Legend")

# Legend to the left
```

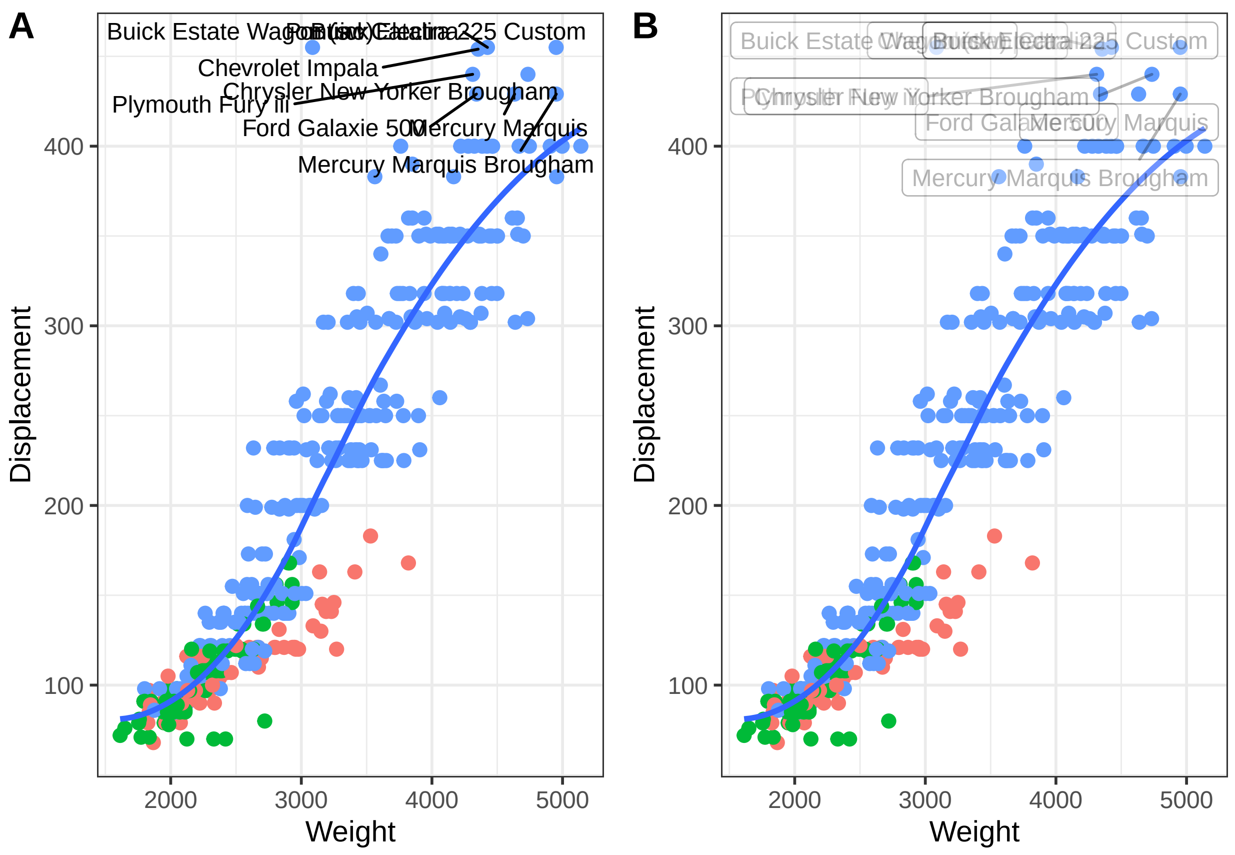
```r
left_leg <- g + theme(legend.position="left") + labs(subtitle="Legend on the Left")

# legend at the bottom and horizontal
bot_leg <- g + theme(legend.key.size = unit(0.05, "cm"),
                     legend.key.width = unit(0.05,"cm"),
                     legend.position="bottom", legend.box = "horizontal") +
  labs(subtitle="Legend at Bottom")

# legend at bottom-right, inside the plot
bot_right <- g + theme(legend.title = element_text(size=12, color = "salmon", face="bold"),
           legend.justification=c(1,0),
           legend.position=c(0.95, 0.05),
           legend.background = element_blank(),
           legend.key = element_blank()) +
  labs(subtitle="Legend: Bottom-Right Inside the Plot")

# ggpubr's function to plot multiple graph in one plot
ggarrange(no_leg, left_leg, bot_leg, bot_right + rremove("x.text"),
          labels = c("A", "B", "C", "D"),
          ncol = 2, nrow = 2)
```



- In this chunk, we will see other geom_() functions :
  - Instead of creating the same plot and just replace the point by text or label, we are going to create a new dataset from cars2 and we well keep only the row where displacement is superior to 400;

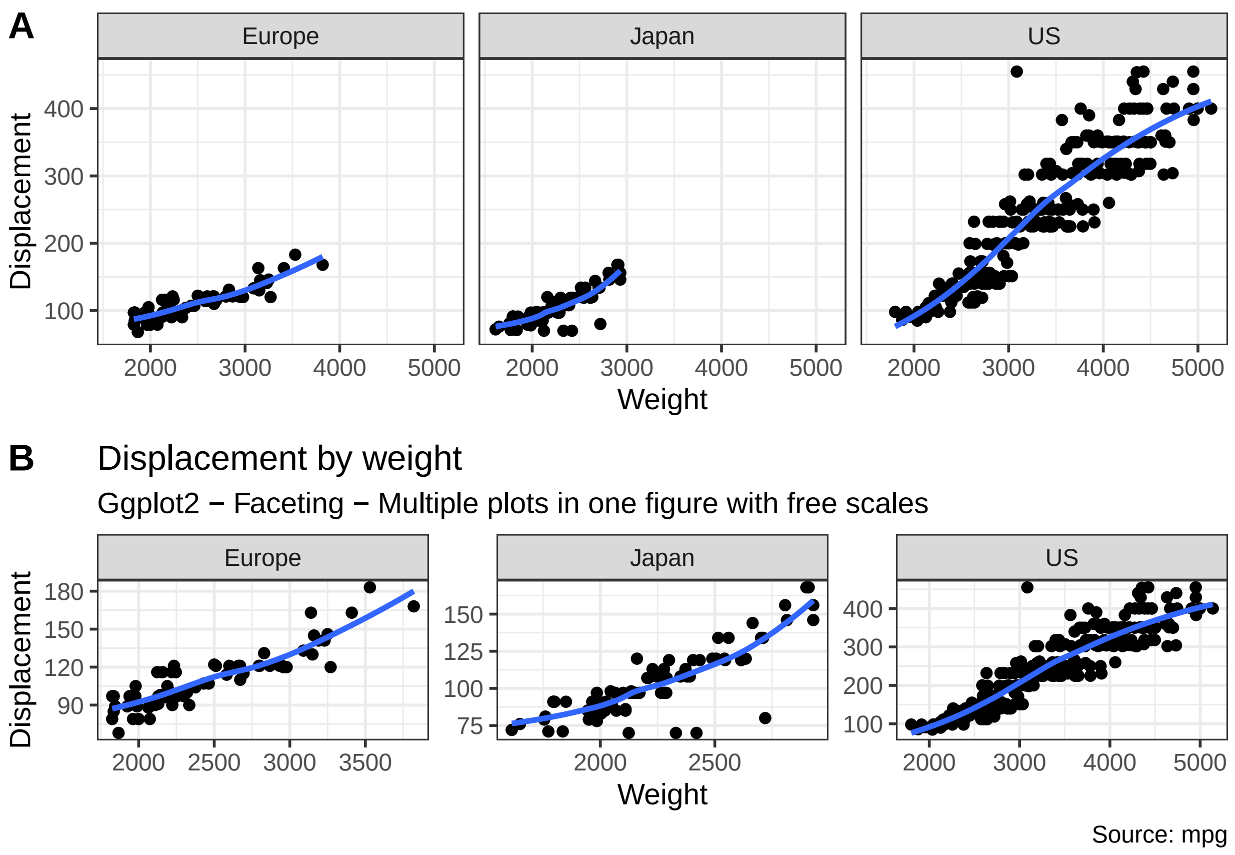
- ggrepel : is the library we used in order to have a more visible plot, the function it contains will make the text and label to repulse each other;
  - geom_text_repel() and geom_label_repel() are the equivalent of geom_point() and label for the ggrepel library;
  - alpha is the transparency option;
- annotation_custom() : you can put annotation on the plot, the library grid contain grid.text() where you can put your annotation.

```r
g <- ggplot(cars2, aes(x=Weight, y=Displacement))
g <- g + geom_point(aes(col=Origin), size=2)
g <- g + scale_color_discrete(name="Origin", guide = FALSE)
g <- g + geom_smooth(method="loess", se=F)

# Sub data
cars2_sub <- cars2[cars2$Displacement > 400, ]
cars2_sub$Car <- ifelse(cars2_sub$Displacement > 400, cars2_sub$Car, "")

# Plot text and label
library(ggrepel)
text_g <- g + geom_text_repel(aes(label=Car), size=3,
                  data=cars2_sub) +
  theme(legend.position = "None")

label_g <-g + geom_label_repel(aes(label=Car), size=3,
                  data=cars2_sub, alpha=0.3) +
  theme(legend.position = "None")

# Define and add annotation
# library(grid)
# note_g <- g + annotation_custom(grid.text("This text is at x=0.2 and y=0.8!",
#        x=0.4,  y=0.8, gp=gpar(col="firebrick", fontsize=10, fontface="bold")))

ggarrange(text_g, label_g,
          labels = c("A", "B"),
          ncol = 2, nrow = 1)
```



- In this chunk, we will see how to divide a plot into multiple other plot :
  - facet_wrap() : you have to choose the data you are going to based your splitting on and you can change the scale to be free because the x and y data apply for every plot;
  - facet_grid() : for this function, the scale is different, the scaling is general and then the results are plotted in the grid according to the dataset you choose.

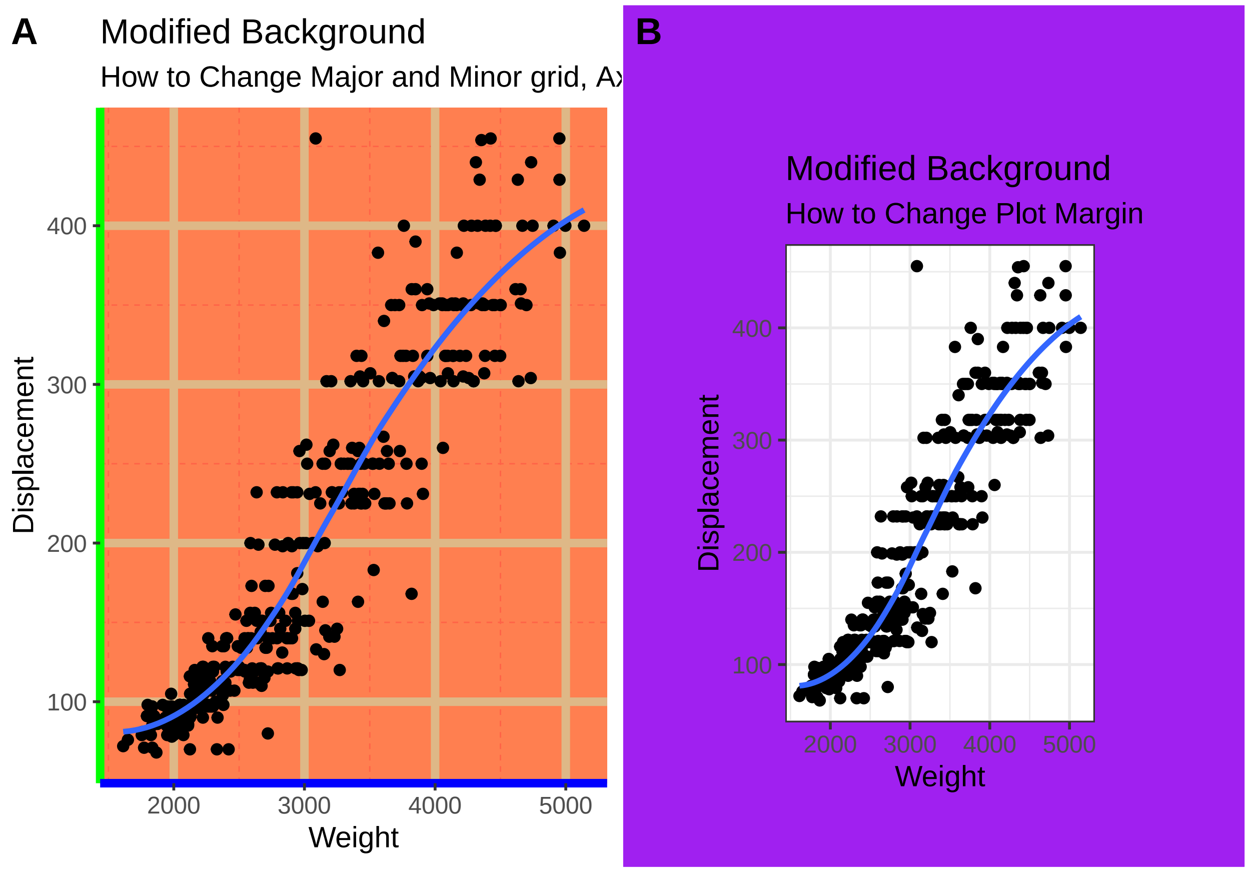
```r
g <- ggplot(cars2, aes(x=Weight, y=Displacement))
g <- g + geom_point()
g <- g + geom_smooth(method="loess", se=F)
theme_set(theme_bw())

# Facet wrap with free scales
# manufacturer in rows and class in columns
g1 <- g + facet_grid( ~ Origin)

g2 <- g + facet_wrap( ~ Origin, scales = "free") +
  labs(title="Displacement by weight", caption = "Source: mpg",
       subtitle="Ggplot2 - Faceting - Multiple plots in one figure with free scales")

ggarrange(g1, g2,
          labels = c("A", "B"),
          ncol = 1, nrow = 2)
```



- In this last chunk, we are going to see the customization of the panel background :
  - The options here are quite clear too, element_line() and element_rect() are the two functions you need to customize plots;

```r
g <- ggplot(cars2, aes(x=Weight, y=Displacement))
g <- g + geom_point()
g <- g + geom_smooth(method="loess", se=F)
theme_set(theme_bw())

# Customize grid, axis and plot
g_panel <- g + theme(panel.background = element_rect(fill = 'coral'),
          panel.grid.major = element_line(colour = "burlywood", size=1.5),
          panel.grid.minor = element_line(colour = "tomato",
                                          size=.25,
                                          linetype = "dashed"),
          panel.border = element_blank(),
          axis.line.x = element_line(colour = "blue",
                                     size=1.5,
                                     lineend = "butt"),
          axis.line.y = element_line(colour = "green",
                                     size=1.5)) +
    labs(title="Modified Background",
         subtitle="How to Change Major and Minor grid, Axis Lines, No Border")

# Change Plot Margins
```

```
g_background <- g + theme(plot.background=element_rect(fill="purple"),
          plot.margin = unit(c(2, 2, 1, 1), "cm")) +  # top, right, bottom, left
    labs(title="Modified Background", subtitle="How to Change Plot Margin")

ggarrange(g_panel, g_background,
          labels = c("A", "B"),
          ncol = 2, nrow = 1)
```



Conclusion

As a conclusion you can see that there is many possibility of customization and you can see that you can pretty much shape the plot the way you want. There is many more different representation, here it's based on geom_point() but you can have lines, bars, texts, labels and so on. I'll let you have a look at the other function in the link below, if you need some more information about this document you can reach me for answers.

If you want to take a loot at all the other functions : https://www.rdocumentation.org