Introduction to advance plotting and maps with R

Olalla Díaz-Yáñez

4/5/2018

Contents

Intro

In the previous session we saw how to plot a simple graph using the function plot() and how using graphical parameters we can improve our graph. With R you can also plot maps and even make complex spatial analysis. In this session we will looked at how to plot maps but if you are over excited and want to directly move to the second part I recomend the book Spatial Point Patterns: Methodology and Applications with R.

There are several R packages to plot, work with maps and make nice graphs and visualizations and packages that would help you with spatial analysis and can substitute (if you want) traditional GIS software. Some of these packages are ggplot2, latice, sp, rgdal and rgeos.

Today's goals

- Introduce more advance packages for ploting
- Use GIS packages to plot maps and do basic modifications

Prerequisites

You are going to need to install few extra packages for this session:

```
#Install the packages (if you do not have them already)
#install.packages(c("ggplot2", "maps", "mapproj", "sp", "rgeos", "maptools", "rgdal"))
#install.packages(c("lattice"))

# Load the packages
library("ggplot2")
```

```
library("lattice")
library(mapproj)       # maps
```

```
## Loading required package: maps
```

```
library(sp)            # maps
library(rgeos)         # maps
```

```
## rgeos version: 0.3-26, (SVN revision 560)
##  GEOS runtime version: 3.6.2-CAPI-1.10.2 4d2925d6
##  Linking to sp version: 1.2-6
##  Polygon checking: TRUE
```

```
library(maptools)      # for geospatial services; also loads foreign and sp
```

```
## Checking rgeos availability: TRUE
```

```
library(rgdal)         # for map projection work; also loads sp
```

```
## rgdal: version: 1.2-18, (SVN revision 718)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
##  Path to GDAL shared files: /usr/local/Cellar/gdal2/2.2.3/share/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 5.0.0, March 1st, 2018, [PJ_VERSION: 500]
##  Path to PROJ.4 shared files: (autodetected)
##  Linking to sp version: 1.2-7
```

Plotting with ggplot2

ggplot2 is based on the grammar of graphics, the idea is that you build the graph by components. These componenets follow the fundamental parts of every data graph: Aesthetics (color, shape, etc), geoms (bars, points, lines), statistics (e.g. draw a linear model line), scale (legend) and facets (groups in your dat).There is a Data visualization with ggplot2 cheat sheet where you can find all the details of this package.

In my opinion the positive thing of using ggplot2 is that is rather easy to find a pre-defined cool plot and build on it to get what you want. The downside is that is kind of a language in itself and if you want to use it you will have to learn it; it also give graphs that are often a bit overcrowded (too many lines or unecessary elements) and thats another thing you will have to fix (specially for a Tufte fun like me).

Lets see some examples of ploting something with ggplot2:

A scatterplot

First we load the dataset iris as in previous sessions

```
 library(datasets)
  data (iris)
  head (iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
```

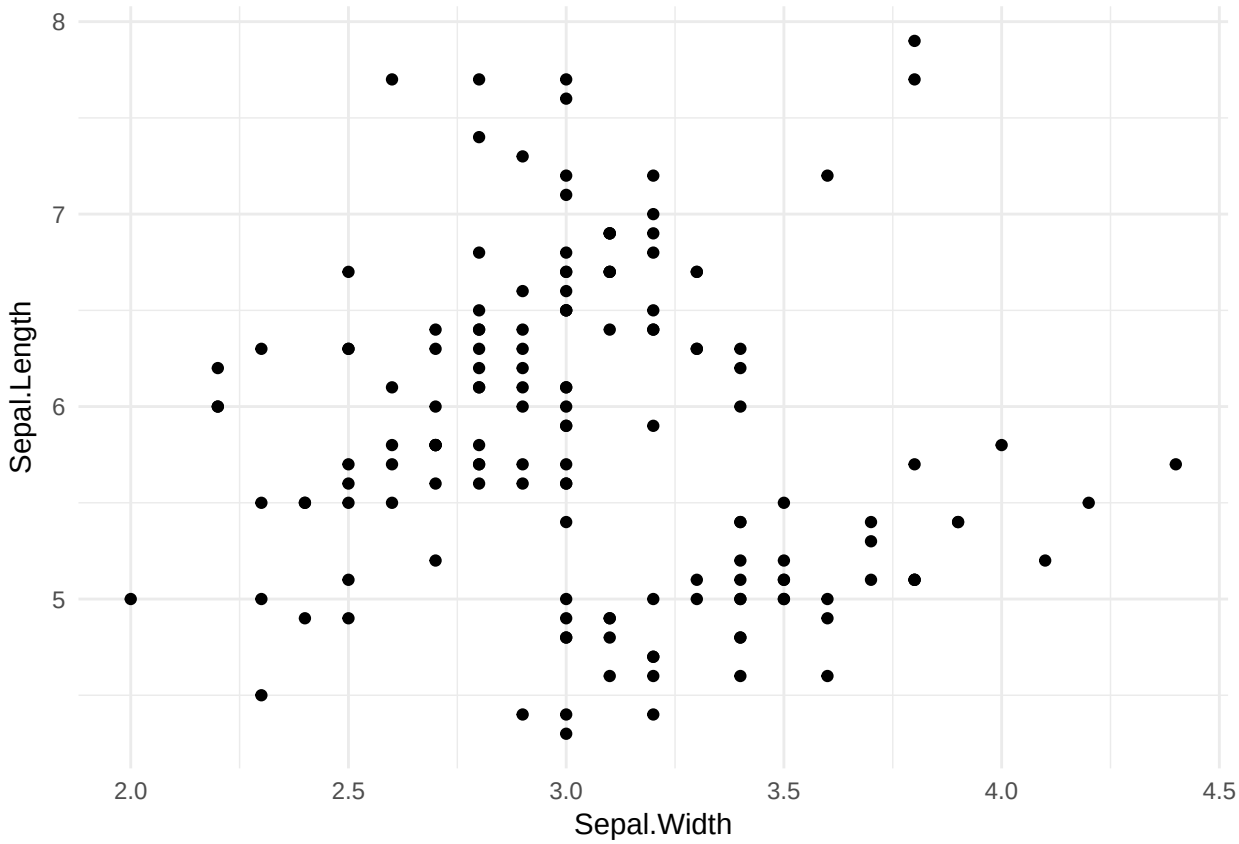
```
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Then we plot a simple scatterplot

```
ggplot(data = iris,
         aes (x = Sepal.Width, y = Sepal.Length)) + # data to plot
    geom_point () # scatter plot
```

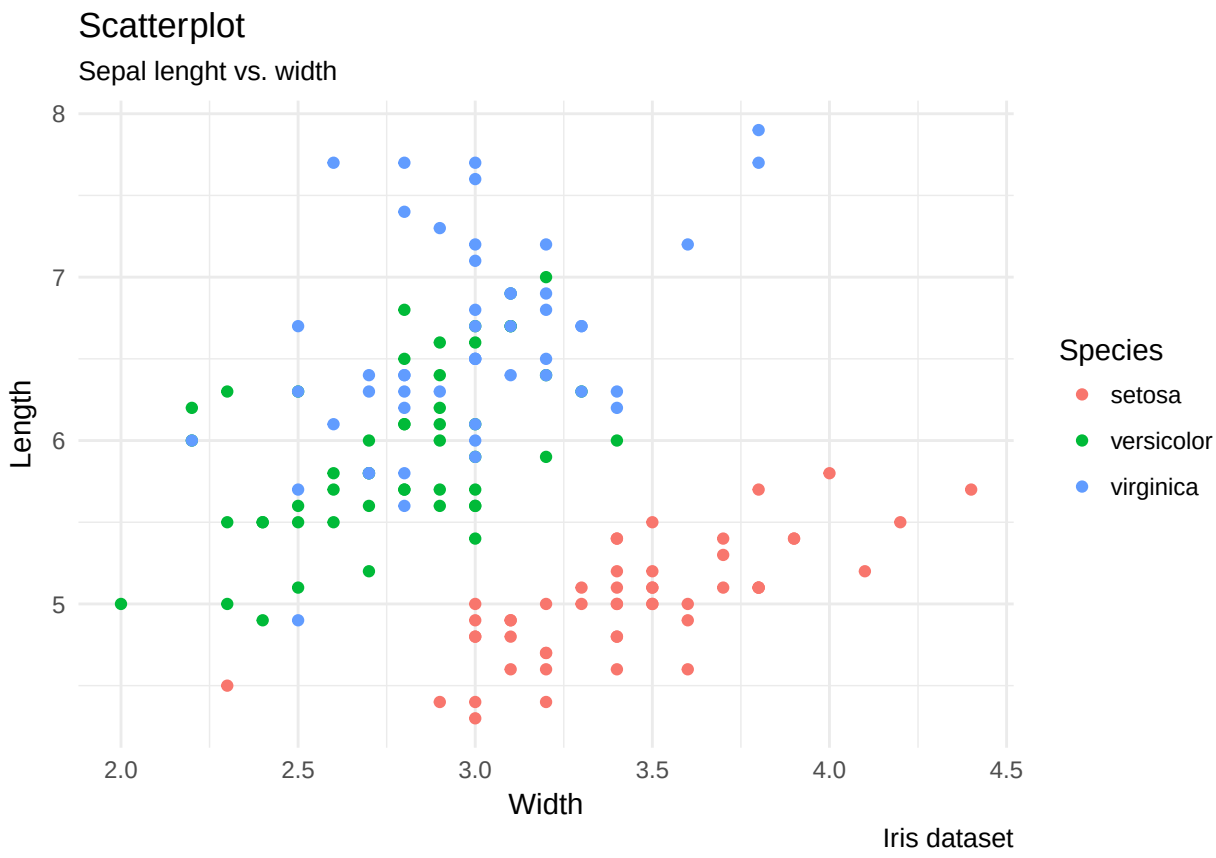
We can use a simplified version of it

```
ggplot (data = iris,
         aes (x = Sepal.Width, y = Sepal.Length)) +
    geom_point () +
  theme_minimal() #simplified version
```



We can get some fancy adding some colors to see wich species the dots corresnpodns to and titles, and axes names.

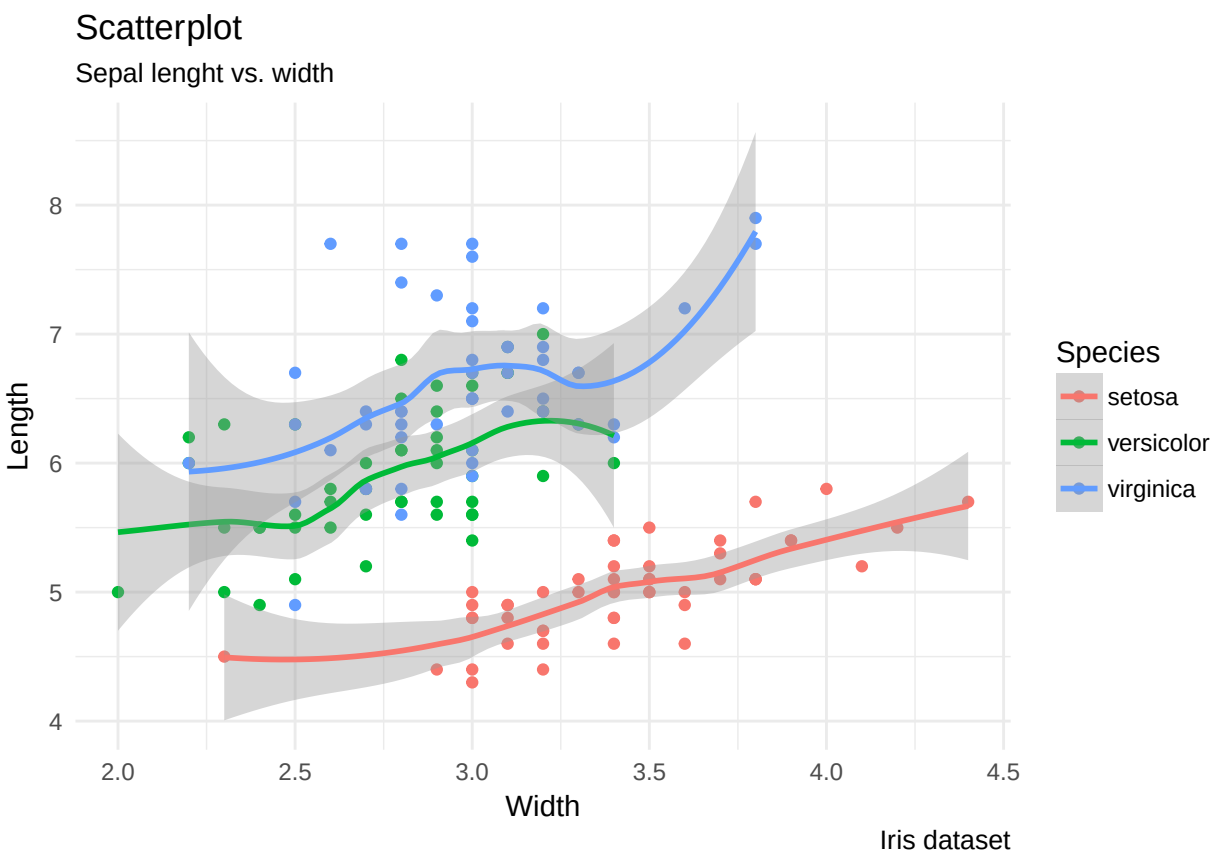
```
ggplot (data = iris,
         aes (x = Sepal.Width, y = Sepal.Length,
              color=Species)) + # we add color by spp
  geom_point () +
  theme_minimal() +
  labs(subtitle="Sepal lenght vs. width", # we add axes names and titles
       y = "Length",
       x = "Width",
       title = "Scatterplot",
       caption = "Iris dataset")
```



There seems to be a difference between species, lets try to plot a trend line to see

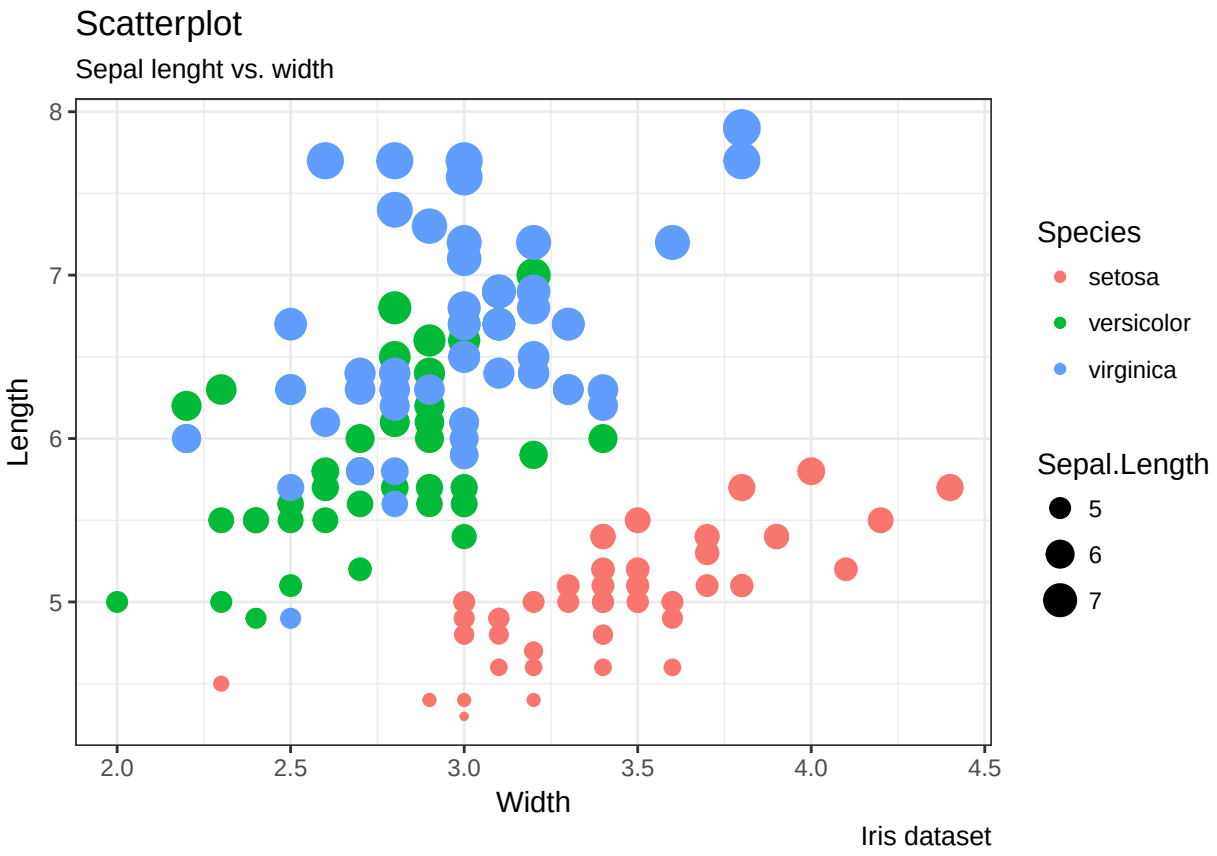
```
ggplot (data = iris,
         aes (x = Sepal.Width, y = Sepal.Length, color=Species)) +
    geom_point (aes(col=Species)) +
  geom_smooth() + #add a trend line
  theme_minimal() +
  labs(subtitle="Sepal lenght vs. width",
       y = "Length",
       x = "Width",
       title = "Scatterplot",
       caption = "Iris dataset")
```

```
## `geom_smooth()` using method = 'loess'
```



And I can even add more information into the plot by adding a dot size according to its sepal length

```
ggplot (data = iris,
         aes (x = Sepal.Width, y = Sepal.Length)) +
    geom_point (aes(col=Species, size=Sepal.Length )) +
  theme_bw()  + # pre-set the bw theme.
  labs(subtitle="Sepal lenght vs. width",
       y = "Length",
       x = "Width",
       title = "Scatterplot",
       caption = "Iris dataset")
```



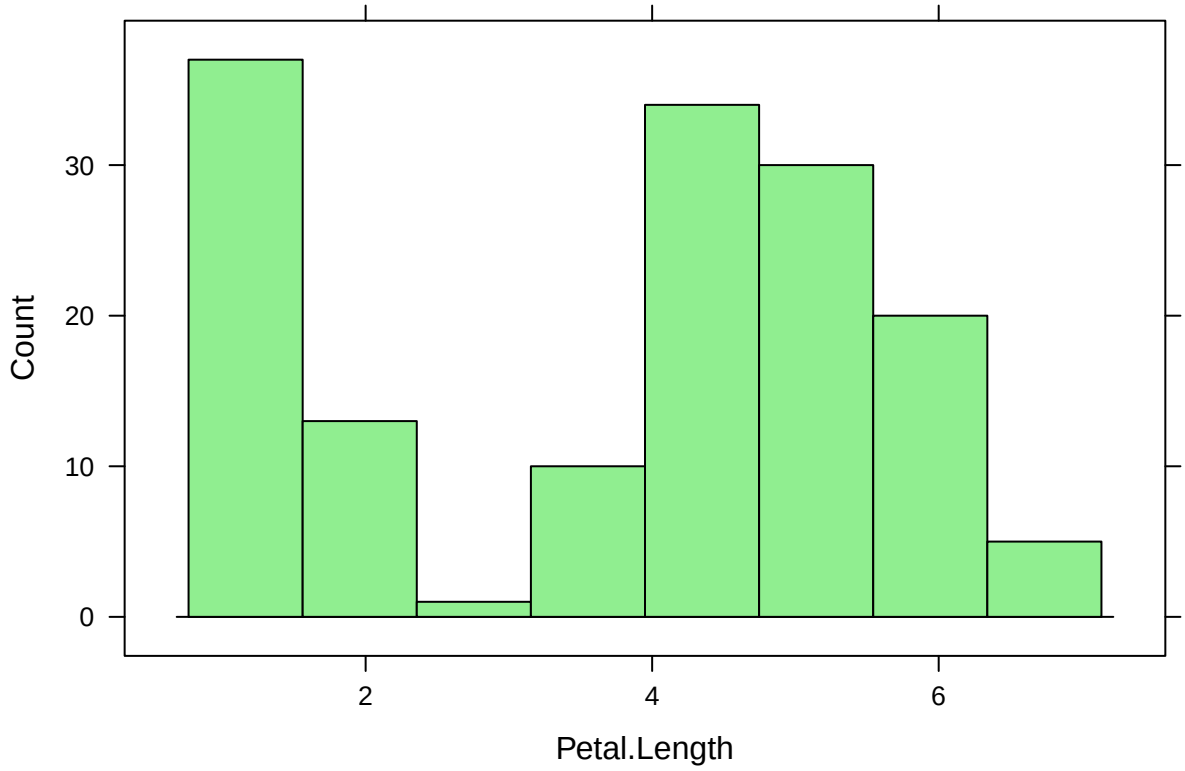
There are many other fancy plots that you can do with ggplot2, here there is a collection of them.

Ploting with lattice

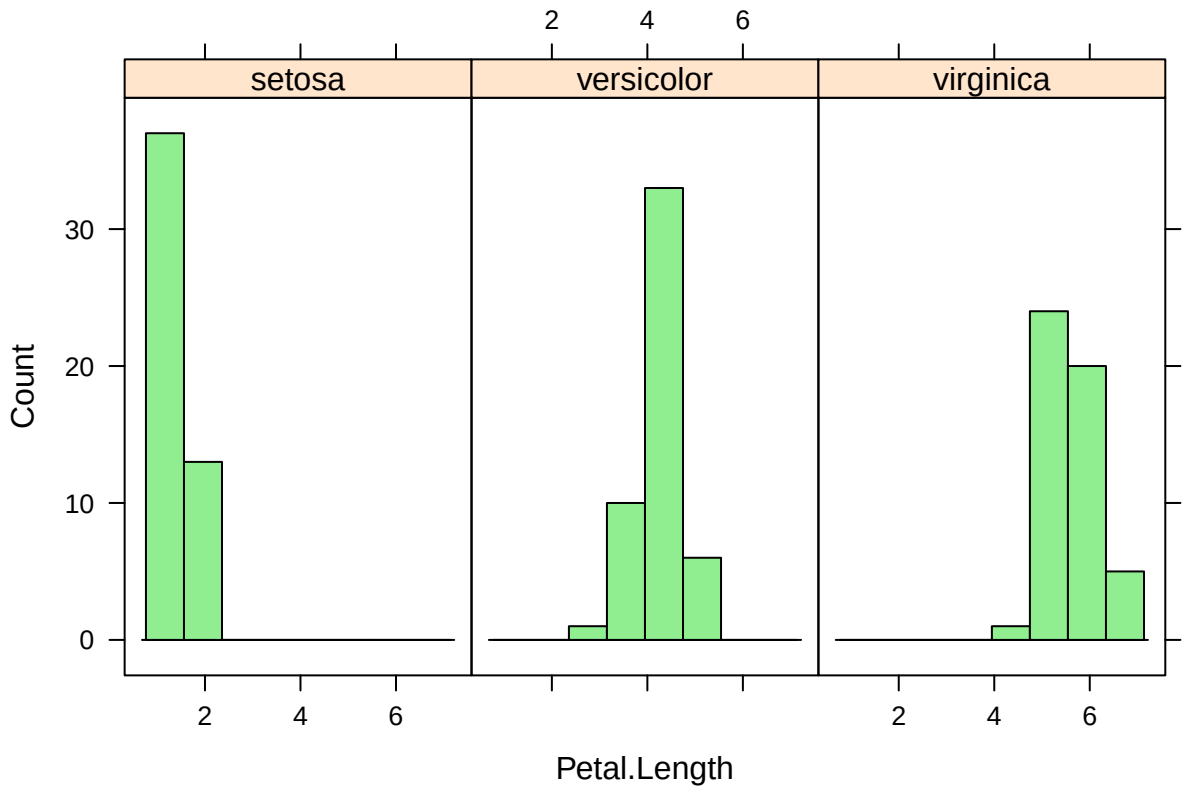
Lattice graphics makes some plots of multivariate data easy. It can make a lattice of multiple panels and plot with varying colors (or symbols) via a grouping variable.

Histogram

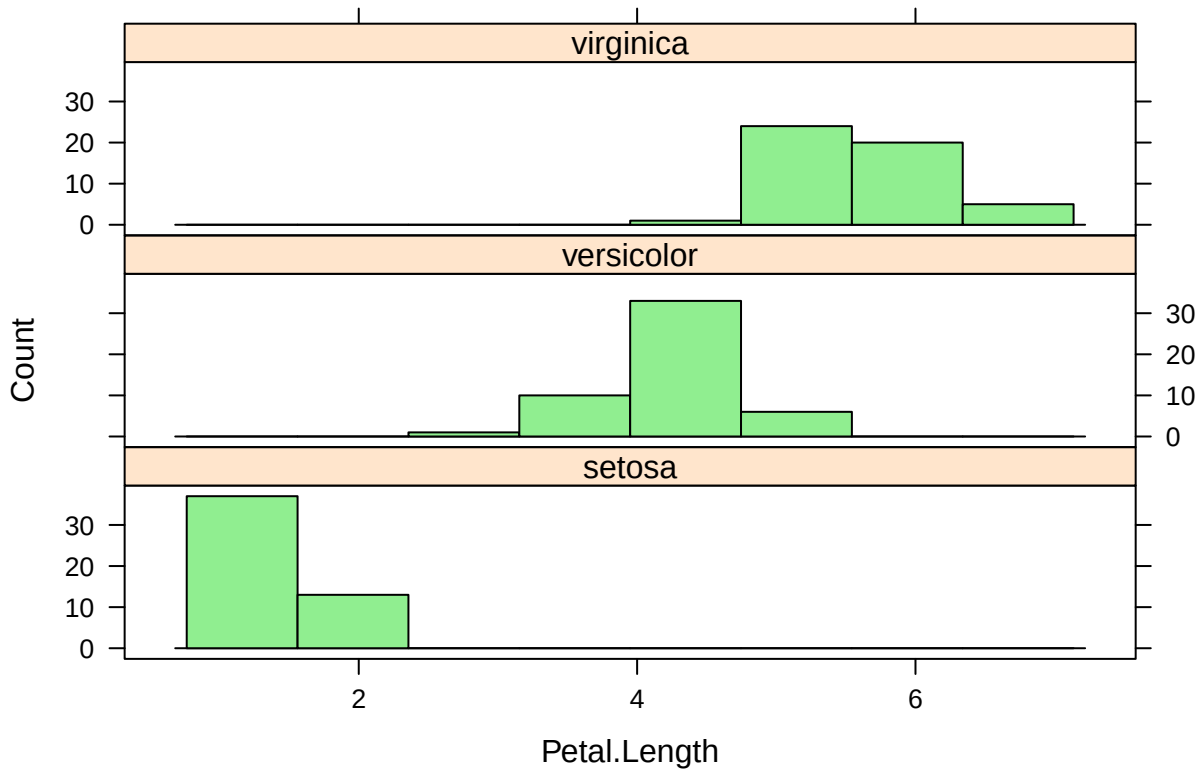
```
histogram(~Petal.Length, data=iris, type="count", col="lightgreen")
```



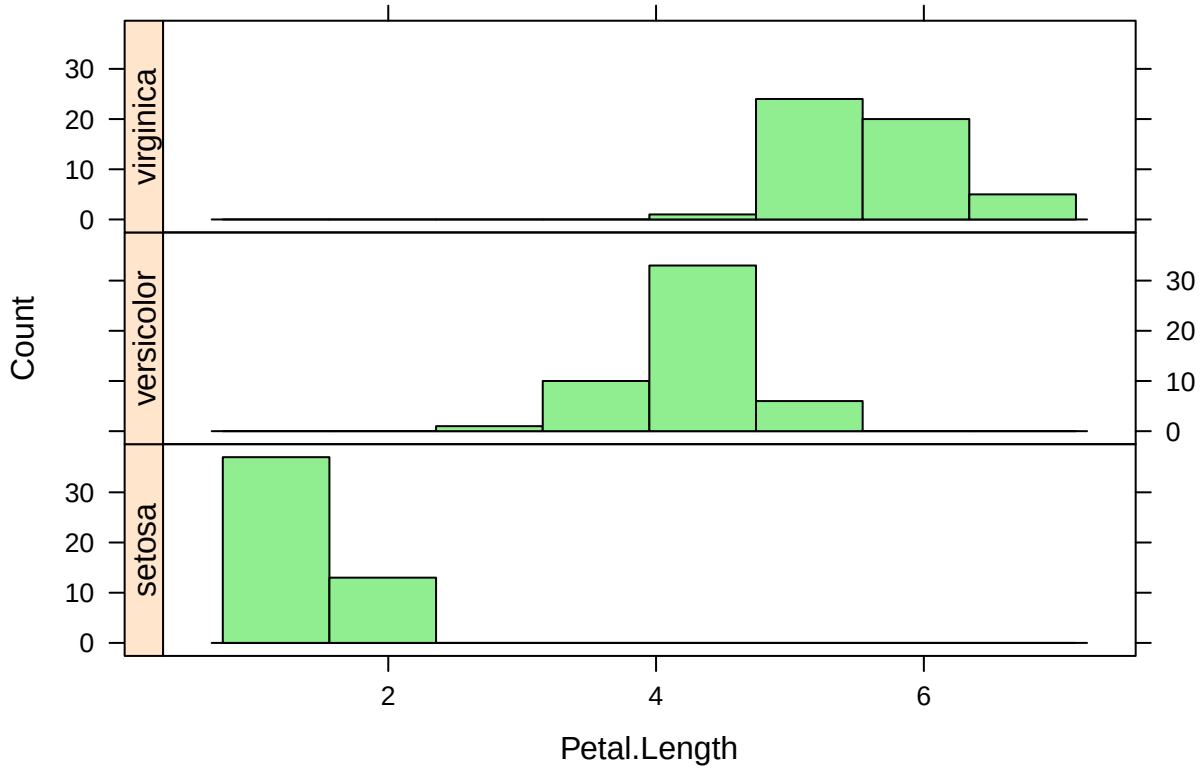

```
histogram(~Petal.Length | Species, data=iris, type="count", col="lightgreen")
```




```
histogram(~Petal.Length | Species, data=iris, type="count", layout=c(1,3), col="lightgreen")
```

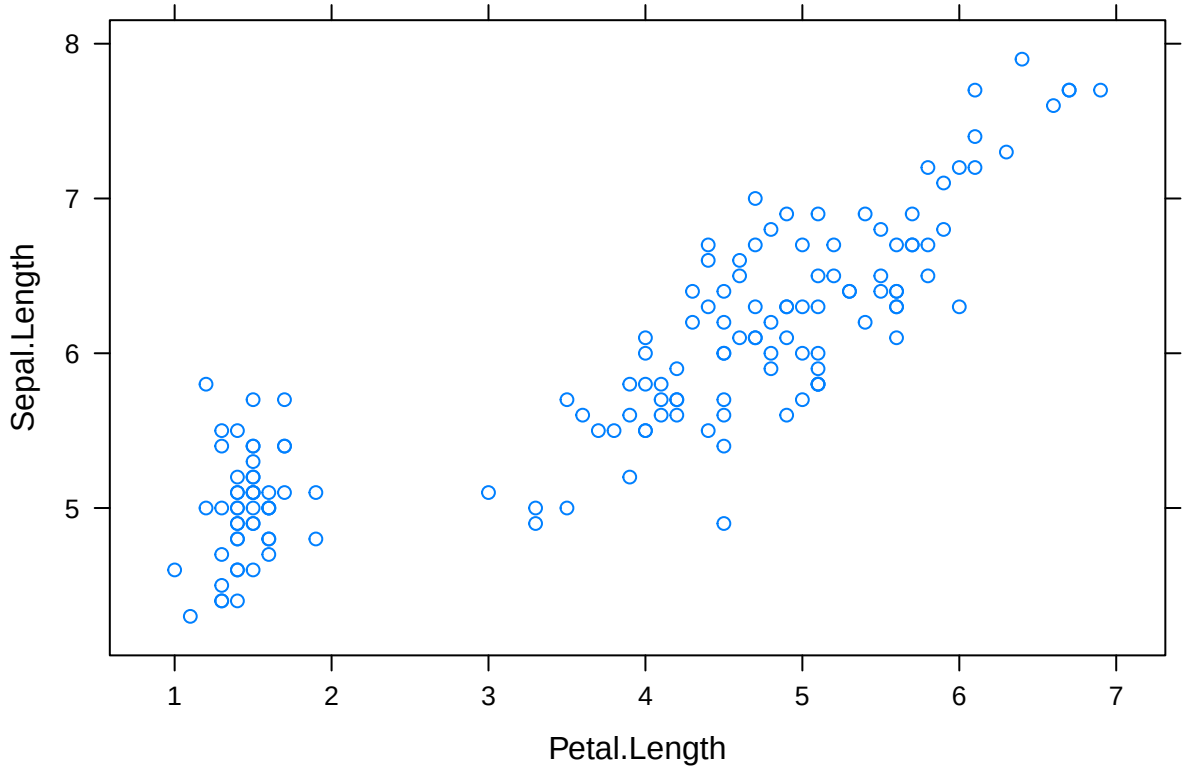



```
histogram(~Petal.Length | Species, data=iris, type="count", layout=c(1,3),
strip=FALSE, strip.left=TRUE, col="lightgreen")
```

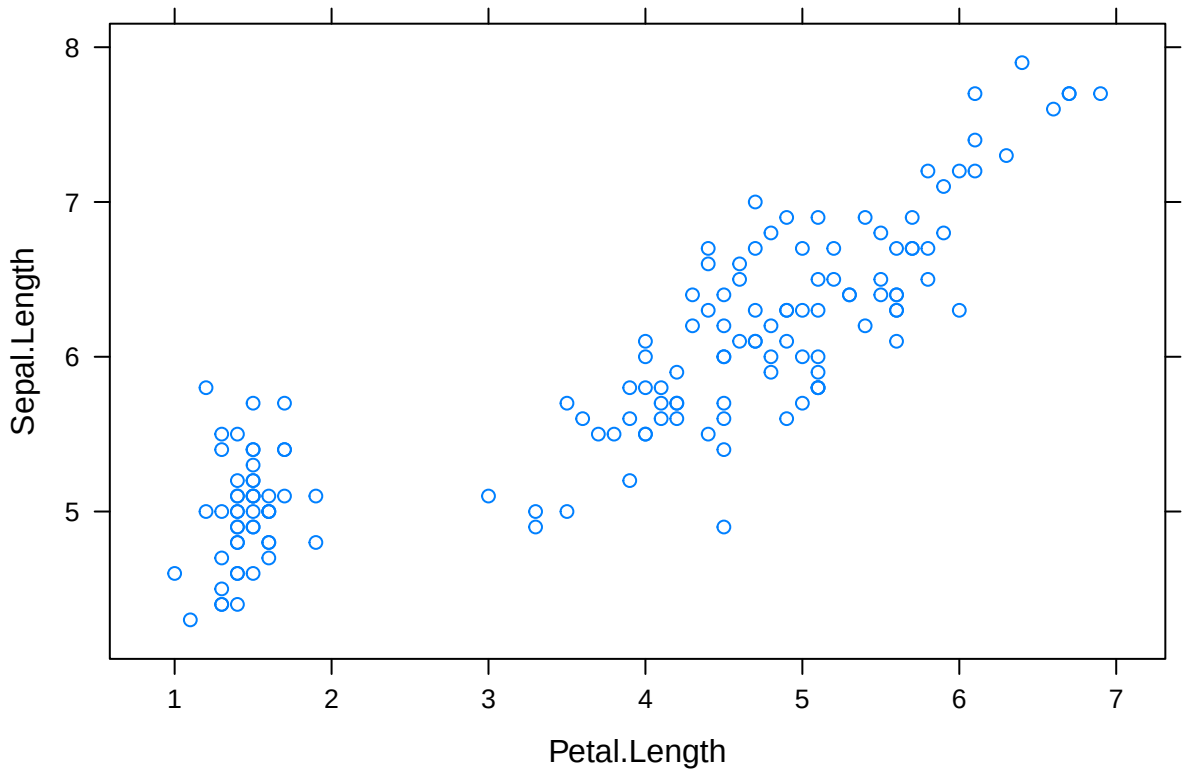


Scatterplots

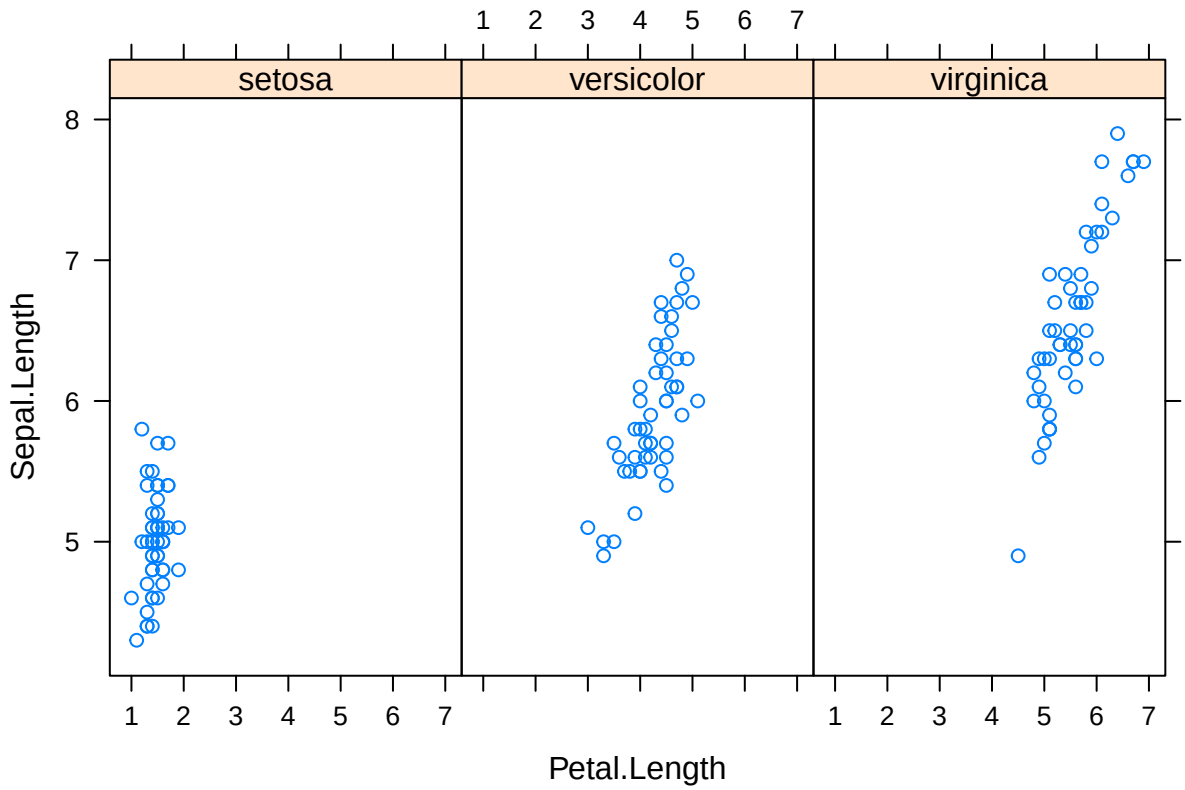
```
xyplot(Sepal.Length ~ Petal.Length, data=iris)
```




```
# Adding legend
xyplot(Sepal.Length ~ Petal.Length, data=iris, auto.key=TRUE)
```




```
# Plot them in separate plots
xyplot(Sepal.Length ~ Petal.Length | Species, data=iris, auto.key=TRUE)
```



Plot a map

First we need a map to plot, lets download a map of the world, I have download the map without boundary lakes from http://www.naturalearthdata.com/ version 4.0.0, the direct link to the download is here. I have added the whole folder to a new folder called Maps located in my working file (where my Rproject is), remember that for accesing it you need to write the exact adress as you have it in the folders!

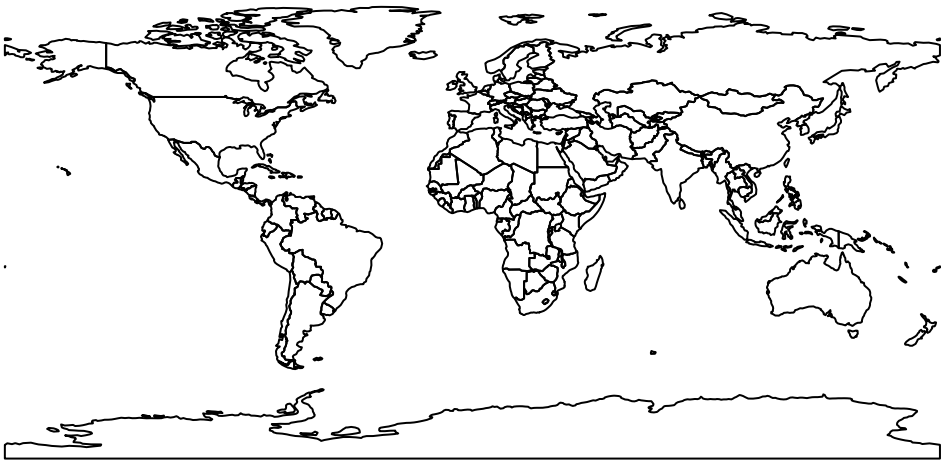
```
library(rgdal)
myworldmap <- readOGR("Maps/ne_110m_admin_0_countries_lakes/ne_110m_admin_0_countries_lakes.shp",
                      verbose = FALSE)

# We can check its structure by using
#str(myworldmap)

# We can access their data by
#head(myworldmap@data)
```

We can know plot the map and add a color red to one country

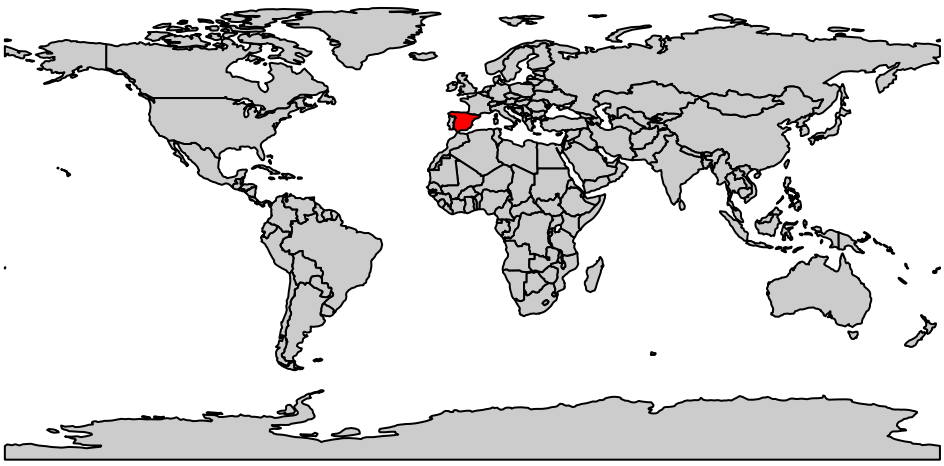
```
library(rgdal)
myworldmap <- readOGR("Maps/ne_110m_admin_0_countries_lakes/ne_110m_admin_0_countries_lakes.shp",
                      verbose = FALSE)
plot(myworldmap)
```



Now we can color in red the countries where we were born

```
library(rgdal)
myworldmap <- readOGR("Maps/ne_110m_admin_0_countries_lakes/ne_110m_admin_0_countries_lakes.shp",
                       verbose = FALSE)

#You can check the whole list of country names
# myworldmap@data$NAME
myCountries = myworldmap@data$NAME %in% c("Spain")
plot(myworldmap, col = c(gray(.80), "red")[myCountries+1])
```

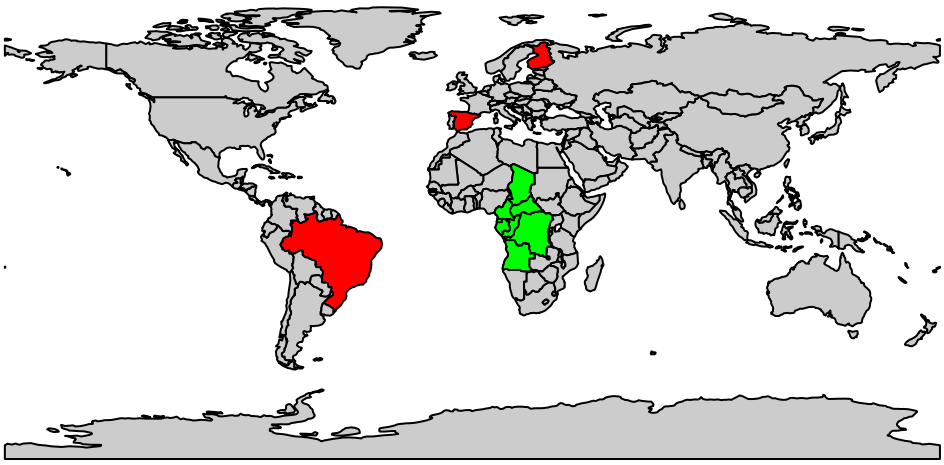


If you want to add come extra colors you can subset some of the contries, color them and then add it to the previous plot, something like this

```
library(rgdal)
myworldmap <- readOGR("Maps/ne_110m_admin_0_countries_lakes/ne_110m_admin_0_countries_lakes.shp",
                      verbose = FALSE)

#You can check the whole list of country names
# myworldmap@data$NAME
myCountries = myworldmap@data$NAME %in% c("Spain", "Finland", "Brazil")
plot(myworldmap, col = c(gray(.80), "red")[myCountries+1])

MiddleAfrica <- myworldmap[myworldmap@data$SUBREGION == "Middle Africa", ]
plot(MiddleAfrica, col = "green", add = TRUE)
```



You can also plot maps using ggplot2, here there is a tutorial that you can follow.