

# 01 GRÁFICOS EN R

*Fernando Villalba*

*25 de enero de 2017*

## Contents

INDICE . . . . .	1
<b>GRÁFICOS EN R</b>	<b>2</b>
INTRODUCCIÓN . . . . .	2
Gráficos simples de 1 variable . . . . .	2
Gráficos 2D . . . . .	6
DISPOSITIVOS GRÁFICOS . . . . .	7
¿Qué es un dispositivo gráfico? . . . . .	7
Cómo se crea un gráfico . . . . .	7
Pintar en ficheros . . . . .	7
Múltiples dispositivos graficos . . . . .	8
Copiar gráficas . . . . .	8
Ejemplos . . . . .	8
SISTEMAS DE TRAZADO DE GRÁFICOS EN R . . . . .	8
Base Plotting system (sistema base) . . . . .	9
Lattice plotting system . . . . .	11
ggplot2 . . . . .	13
ggplot2 part2 . . . . .	13
ggplot part3 . . . . .	15
ggplot2 part 5 . . . . .	16
??? PRACTICAS CON SWIRL . . . . .	16
RECURSOS EXTERNOS . . . . .	17

```
knitr::opts_chunk$set(eval=FALSE)
knitr::opts_chunk$set(echo=TRUE)
```

## INDICE

1. Leccion 1
2. Graficos 1D
3. Graficos 1D
4. Scatterplot
5. Leccion 2
6. Plot

# GRÁFICOS EN R

Esta es la primera lección del curso cuarto de la especialización en DATA SCIENCE. Los apuntes corresponden a la semana 1 del curso Exploratory Data Analysis, donde se explica el uso de gráficos en R.

## INTRODUCCIÓN

Existen unos **Principios del análisis gráfico** sobre los que conviene reflexionar y recordar en todo el proceso de trazado. Recordemos que se trata de un curso de ciencia de datos, por lo que:

1. Siempre compara, busca un estado relativo sobre el que comparar los datos e hipótesis.
2. Muestra la causalidad, el mecanismo, la explicación del sistema y su estructura.
3. El mundo real es multivariable, no muestres solo una cara.
4. Presenta datos de forma integrada, es decir con diferentes puntos de vista o representación.
5. Documenta el proceso de análisis para que sea totalmente reproducible.
6. Cuenta una historia, busca el contenido o fin de lo que presentas.

Al representar los datos en gráficos buscamos muchas cosas: comprenderlos, encontrar patrones, sugerir modelos de comportamiento. Entonces ¿Por qué pintamos los datos en gráficos?, muy sencillo, hay un refrán español que dice: *más vale una imagen que 1000 palabras*.

Aunque existen muchas maneras de pintar datos con R, en estas lecciones vamos a centrarnos en 3 funciones, las características del sistema base de R, la librería lattice y la librería ggplot.

Cuando vamos a representar datos, tenemos también que pensar qué salida queremos para los gráficos:

1. ¿Será en papel? o en una pantalla?
2. ¿Se usará en una web?, para una presentación?
3. ¿Es muy grande el número de datos a pintar?
4. ¿Necesitaremos que cambie de tamaño de forma dinámica? o interactuar con él?
5. ¿Qué sistema usaremos para pintar?... oleo, pastel, digital?

El tipo de salida lo veremos en el apartado de dispositivos gráficos y puede ser importante si piensas usar los gráficos no solo en la pantalla, sino para presentaciones o libros.

En las lecciones del curso web se usa una tabla de datos extraída de la web de la agencia EPA de EEUU. Desde su web EPA Air Quality System podemos descargar los datos de ejemplo, que se corresponden con datos de calidad del aire en cada condado de EEUU.

```
# Lectura del fichero con los datos para los ejemplos
pollution <- read.csv("data/avgpm25.csv", colClasses=c("numeric", "character", "factor", "numeric", "numeric"))
head(pollution)
```

Para estos apuntes, hemos usado los datos de ejemplo incluidos en la librería **datasets**, en concreto la tabla 'cars'.

## Gráficos simples de 1 variable

Para explorar datos simples de una variable de forma rápida y dibujarlos, podemos usar estas características o funciones de R:

1. `summary(data)` que devuelve el min, max, media, mediana y 2 cuantiles, no pinta pero siempre es bueno resumir.

2. `boxplot(data, col="blue")` -> hace un gráfico simple con los datos de summary y una caja color azul en este caso que continene el 50% de los datos (entre el 25 y e 75%)
3. `hist(data, col = "green", breaks=10 )`, hace la grafica del histograma de frecuencias en 10 partes. Se puede omitir breaks. Puedes añadir una marca de los datos con `rug(pollution$pm25)`
4. `barplot(data, col=wheat)` hace gráfico de barras color trigo

Veamos unos ejemplos de código:

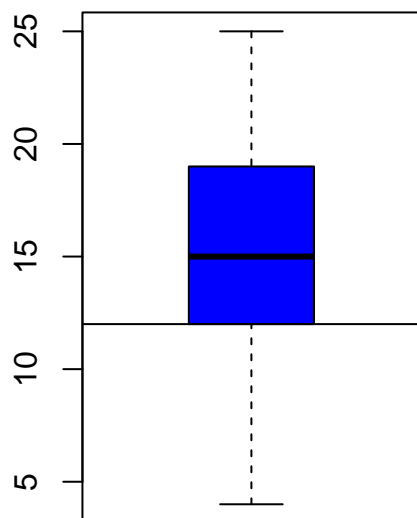
```
#Si no queremos o tenemos el fichero de EPA podemos usar datos de muestra del paquete datasets
library(datasets) # cargamos el paquete de datos de muestra
head(cars)
```

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

```
summary(cars$speed)
```

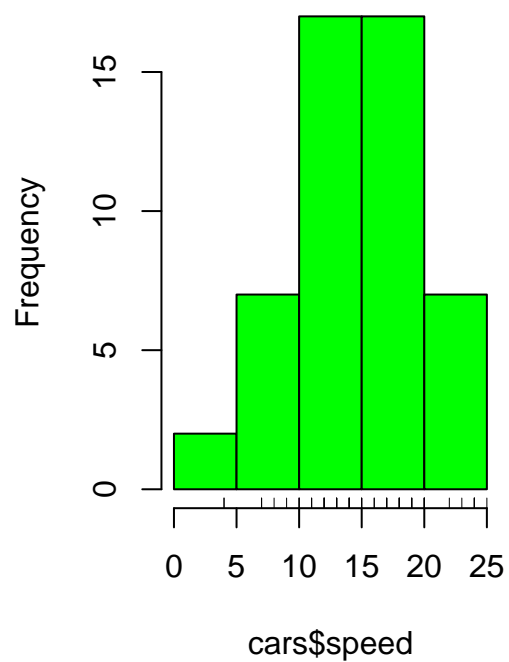
```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      4.0    12.0    15.0    15.4    19.0    25.0
```

```
boxplot(cars$speed, col="blue")
#si queremos añadir una línea al boxplot, por ejemplo para marcar el limite de 12
abline(h=12)
```



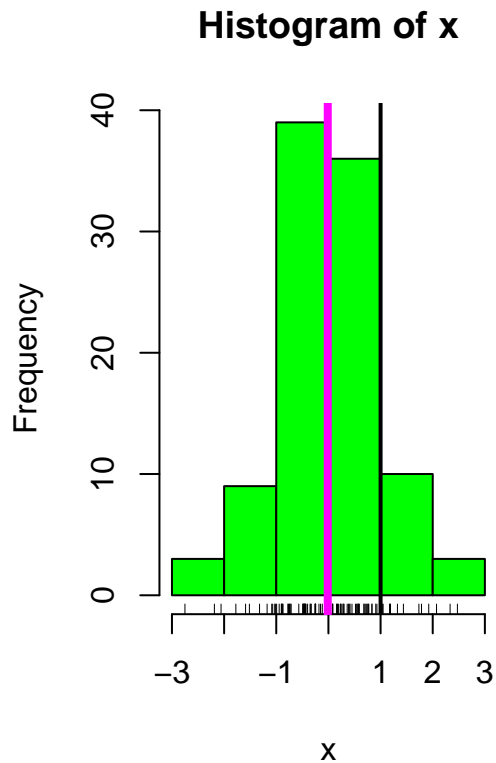
```
hist(cars$speed, col="green")  
  #podemos añadir una regla abajo del hist con todos los puntos  
  # añadiendo rug SOLO VALE PARA 1D. nos pone una marca en cada punto de dato  
rug(cars$speed)
```

## Histogram of cars\$speed



```
#Generamos datos aleatorios
x<-rnorm(100)
# añadimos cortes.
hist(x, col="green", breaks=5)
rug(x)

#añadimos linea vertical
abline(v= 1, lwd=2) # grosor 2
#añadimos linea vertical en la media
abline(v= median(x), col= "magenta", lwd=4)
```



```
# lwd=line_width=grosor 2)
# lty=tipo linea

#barplot(table(pollution$region), col="wheat", main="numero de condados por region")
```

## Gráficos 2D

Si añadimos más variables a la gráfica usaremos funciones como `boxplot` o `with`.

```
# grafico por region de los datos (tecla 4 altgr ~)
boxplot(pm25 ~ region , data= pollution, col="red")
# boxplot(speed ~ dist, cars)
```

Existe la opción de dividir el lienzo en varios gráficos con las opciones de `par()`.

```
# dos histogramas
# dividimos la ventana grafica en 2 filas 1 col
par(mfrow=c(2,1), mar=c(4,4,2,1))
hist(subset(pollution, region=="east")$pm25, col="green")
hist(subset(pollution, region=="west")$pm25, col="blue")
```

## Gráficos de dispersión (SCATTERPLOT)

Otra manera de representar datos de dos en dos es con gráficas de dispersión o scatterplot en inglés.

Para hacerlas en R usamos dos funciones principales `plot` y `with`

1. `plot(y ~ x, data = unatabla)` el símbolo `~` se hace con la tecla `AltGr + 4`.
2. `with(unatabla, plot(y, x))`

```
with(pollution, plot(latitude, pm25))
abline(h=12, lwd=2, lty=2)

# ahora distinguimos por colores las regiones
with(pollution, plot(latitude, pm25, col=region ))
abline(h=12, lwd=2, lty=2)

## multiples scatterplots
par(mfrow=c(2,1), mar=c(5,4,2,1))
with(subset(pollution, region=="west"), plot(latitude, pm25, main="WEST"))
with(subset(pollution, region=="east"), plot(latitude, pm25, main="EAST"))
```

## DISPOSITIVOS GRÁFICOS

### ¿Qué es un dispositivo gráfico?

- Un dispositivo gráfico es el objeto en el que pintamos, el lienzo sobre el que trazamos las gráficas, por ejemplo:
  - una ventana en el PC, es un dispositivo gráfico de pantalla (screen device)
  - un fichero PDF
  - un fichero PNG, JPEG
  - un fichero SVG (scalable vector graphic)
- Cuando generamos un gráfico, tiene que ser enviado a un dispositivo que haga de lienzo.
- Para abrir un dispositivo usamos los comandos:
  - `quartz()` en Mac, `X11()` en Linux, `windows()` en Windows
  - `?Device` par encontrar todos los dispositivos
- `plot`, `xyplot`, `qplot` son funciones que envían el grafico al dispositivo “pantalla”
- los ficheros son tambien muy usados si queremos imprimir o hacer una presentación o guardar nuestro gráfico de forma independiente.

### Cómo se crea un gráfico

- llamando explicitamente al dispositivo

```
# pdf
pdf(file = "myplot.pdf")
```

- llamando a una funcion de dibujo
- sobreescribiendo en un gráfico existente
- *para cerrar específicamente un dispositivo llamamos `dev.off()`*

### Pintar en ficheros

#### Vectoriales

- Pros: bueno para graficos lineales, cambiar tamaño sin perder calidad sin distorsión.
- Contras: no son buenos cuando hay gran cantidad de puntos/datos que pintar.

Los dispositivos usuales de este tipo son: pdf, svg, windows metafile o postscript.

## Raster

- Pros: good for plot with lots of points
- Contras: doesn't resize well

Formatos de este tipo habituales son: png, jpeg, tiff o bmp.

## Múltiples dispositivos graficos

Podemos abrir muchos dispositivos, pero solo podemos pintar desde R en uno cada vez. Cada dispositivo tiene un número entero como identificativo.

- `dev.cur()` -> nos da el numero del dispositivo actual
- `dev.set(integer)` -> establece el dispositivo indicado en integer como actual.

## Copiar gráficas

Lo usamos cuando nos gusta por ejemplo la que hemos hecho en pantalla y la queremos guardar en fichero:

```
library(datasets)
with(faithful,plot(eruptions,waiting)) # Crea un dibujo
title(main="Datos de daños Geyser") # sobreescribimos el título
dev.copy(png, file = "myfile.png") # copia el dibujo a un fichero png
dev.off() # cierra el dispositivo png.
```

Para copiar en pdf existe una funcion especifica `dev.copy2pdf`

**Advertencia:** los gráficos no tienen por qué mostrarse igual que en pantalla.

## Ejemplos

```
# pdf
pdf(file = "myplot.pdf") # abrimos el dispositivo gráfico
#creamos un grafico y lo mandamos al dispositivo pdf
with(faithful,plot(eruptions,waiting))
title(main="Datos de daños Geyser") # sobreescribimos el título
dev.off()# cerramos el dispositivo
```

## SISTEMAS DE TRAZADO DE GRÁFICOS EN R

En estas lecciones vamos a usar los 3 sistemas principales de generar gráficos en R, a los que dedicaremos un apartado de desarrollo específico. No son los únicos, hay cientos de librerías para generar gráficas, pero estas son las más usadas:

1. **base plot system** o sistema base Es el sistema de pintado por defecto incluido en el nucleo de R. La idea es la misma que la de un artista ante un lienzo blanco. Se van añadiendo cosas una a una al lienzo con funciones diferenciadas: lineas, etiquetas, puntos, gráficas... Es una manera muy completa de crear, pero tiene el inconveniente de que si no quieres los valores por defecto tendrás que saber como hacer cada cosa específica.
2. **Lattice plot system** Los gráficos se crean de una vez llaman. Bueno para muchas gráficas juntas en una página o pantalla
3. **ggplot system** Es una mezcla de los anteriores con unas opciones por defecto interesantes y capacidad para personalizar todo



## Base Plotting system (sistema base)

Es el SISTEMA ELEMENTAL PARA PINTAR GRÁFICOS CON R

Los comandos básicos son: `plot`, `hist`, `boxplot`.

`plot` y `hist` lanzarán un objeto gráfico, si no hay ya uno abierto. `plot` tienen muchas opciones de personalización, títulos ejes..gran parte de estas opciones se montan con el comando `par` .

El sistema base está incluido en las librerías del núcleo de R `graphics` y `grDevices`. La primera contiene los comandos `plot`, `hist` etc.. y la segunda todos los comandos relacionados con los dispositivos gráficos de impresión, ya sea la pantalla, un fichero, un pdf etc. . .

### Parámetros globales de ‘plot’ (par):

Usa `par()` para especificar estos parametros, tambien para ver el valor de estos parámetros actualmente `par("col")`.

- `pch`: simbolo del dato (circulos por defecto. ver `points` para más detalles)
- `lty`: tipo de linea
- `lwd`: ancho de linea
- `col`: color, puede ser un numero, o el nombre, hex , `colors()` function gives a vector of color by name
- `xlab`: etiqueta eje x
- `ylab`: etiqueta eje y
- `las`: the orientation of axis labels on the plot
- `bg`: background color
- `mar`: margenes. (abajo, izq, der, arriba). The unit is line of text.
- `oma`: the outer margin
- `mfrow`: number of plots per row and per column, filled row-wise.
- `mfcol`: number of plots per column and per row, filled column-wise.

## Funciones de Base plotting

- `plot`: crea un gráfico dependiendo de los datos hace uno u otro tipo.
- `lines`: añade una linea a una gráfica.
- `points`: añade puntos a una gráfica.
- `with`: crea un gráfico de dispersión. `with(tabla, plot(colA,ColB))`
- `text`: añade texto de etiquetas.
- `title`: añade el título.
- `mtext`: m means margin, add text to margins.
- `axis`: add axis ticks and labels.
- `legend`: add legend. If they are the line, specify `lty`. If they are character, specify `pch`.

## Ejemplos Base Plotting Demonstration

Podemos usar la función `examples` para ver muchos ejemplos gráficos: `examples(points)`

```
library(datasets)
#pinto un histograma
hist(airquality$Ozone) # pinta un gráfico

#pinto un scatterplot
# with(tabla, plot(col,colB))
```

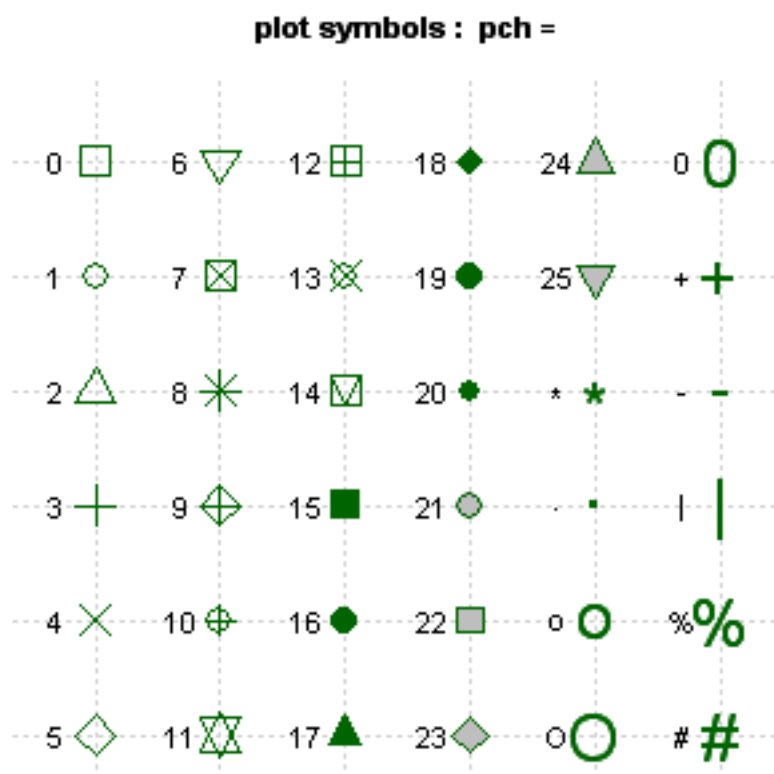


Figure 1: tipos de punto

## Line Types: lty=

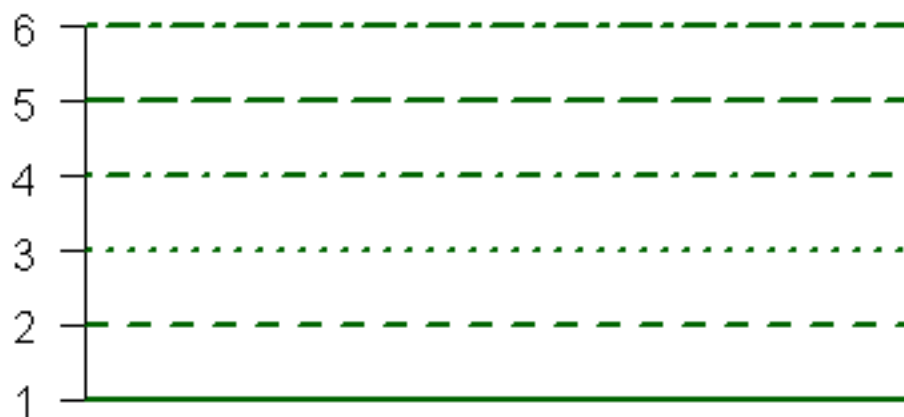


Figure 2: tipos de linea

```

with(airquality, plot(Wind,Ozone, main="Ozono y viento en NY",pch=20))
# Añado que pinte los del mes de mayo en azul
with(subset(airquality, Month==5), points(Wind,Ozone,col="blue"))
# Añadimos una leyenda
legend("topright", pch =1, col=c("blue","black"),legend=c("Mayo","otros"))
#añadimos línea de regresión
model<-lm(Ozone ~ Wind,airquality)
abline(model,lwd=2)

# Grafica de cajas boxplot
airquality<-transform(airquality,Month=factor(Month))
boxplot(Ozone ~ Month, airquality, xlab="mes", ylab="ozono ppb")

#EJEMPLO GRAFICOS MULTIPLES
par(mfrow=c(1,2))# (filas, col)
with(airquality, {
  plot(Wind, Ozone, main="Ozono vs viento")
  model<-lm(Ozone ~ Wind,airquality) #
  abline(model,lwd=2) #
  plot(Solar.R, Ozone, main=" Ozono y radiacion solar")
})

```

## Lattice plotting system

### Introduction

- Lattice contains function to produce trellis graphics, including:
  - xyplot
  - bwplot
  - levelplot
- It builds on grid package, which we seldom use directly
- Doesn't have two phases: plotting and annotation
- All plotting/annotation is done at once with a single function call

### Important functions

- xyplot: scatterplot
- bwplot: boxplot
- histogram: histograms
- stripplot: boxplot with actual points
- dotplot: plot dots like "violin strings"
- splom: scatterplot matrix; like the `pairs` in base system
- levelplot, contourplot: for plotting image data

### xyplot' function

```
xyplot(y ~ x | f * g, data)
```

- Again, we use formula notation, left of ~ is y-axis, right of ~ is x-axis.
- f and g are called conditioning variables, which are optional

- they are categorical variables that we condition on
- it means we want to look at the scatterplot of y and x at every level of f and g
- Don't have to use 2 categorical variables, \* indicates interaction.
- data is the dataframe
  - if no dataframe passed, it will look into parent frame.

## Simple lattice plot

- Basic one

```
library(lattice)
library(datasets)
xyplot(Ozone ~ Wind, data = airquality)
```

- Better one, pay attention how we use **transform** to change the variable in a dataframe

```
library(lattice)
library(datasets)
xyplot(Ozone ~ Wind, data = airquality)
airquality <- transform(airquality, Month = factor(Month))
xyplot(Ozone ~ Wind | Month, data = airquality, layout = c(5, 1))
```

## Lattice behaviour

Fundamental difference between base plot system

- Base system plot to graphic devices.
- Lattice system returns a **trellis** object.
- print methods for lattice functions do the plotting work
- It's better to keep the data and code
- On the command line, trellis objects are auto-printed.

## Lattice panel functions

- Lattice functions have **panel functions** which controls what happens inside each panel
- Can supply our own to customize the panel
- panel functions receive the x/y coordinates of the data points in their panel
- You cannot use the annotation function in base plotting system, you cannot mix the two plotting systems.

```
# Panel functions
xyplot(y ~ x | f, panel = function(x, y, ...) {
  panel.xyplot(x, y, ...)
  panel.abline(h = median(y), lty = 2)
})

xyplot(y ~ x | f, panel = function(x, y, ...) {
  panel.xyplot(x, y, ...)
  panel.lmline(x, y, col = 2)
})
```

## Summary

- Lattice functions are constructed with one single function call
- margins and spacing are handled automatically
- ideal for creating conditional plots where you examine the same kind of plot under many different conditions
- panel functions can be specified and customized to modify what is plotted in each of the plot panels

## ggplot2

### What is ggplot2?

- An implementation of grammar of graphics.
- Written by Hadley Wickham
- A 3rd graphics system in R
- grammar of graphics represents abstraction of graphic ideas and objects
- Think “verb”, “noun”, “adjective” for graphics
- Allow theory of graphics to build new graph and graphic objects
- Shorten the distance from mind to page

### Grammar of Graphics

- Statistic graph is a mapping from data to aesthetic attributes (color, shape, size) and geometric objects (points, lines, bars). The plot may contain statistic transformation of data and is drawn on a specific coordinate system

### The basics qplot

- much like the plot in base system
- must look for a dataframe, if cannot find, look in parent environment.
- Plots are made up of aesthetic and geoms
- Factors are indicating subsets of data. They should be labeled.
- `qplot` hides underneath
- `ggplot` is core function and very flexible

### ggplot2 part2

```
# installation
install.packages("ggplot2")
```

### Hello word for ggplot2

```
library(ggplot2)
str(mpg)
qplot(displ, hwy, data = mpg)
```

### Aesthetic

We map the `drv` variable to different colors, and the plot is automatically labeled.

```
qplot(displ, hwy, data = mpg, color = drv)
```

## Adding geoms

We can add a smooth line here, note that we want 2 geometric objects here, the data points themselves and the a smooth line.

```
qplot(displ, hwy, data = mpg, geom = c("point", "smooth"))
```

## Histogram

Make histogram by just specify single variable. Note that here, we need to use `fill` argument to specify colors.

```
qplot(hwy, data = mpg, fill = drv)
```

## Facets

- Like panels in lattice. We want distinguish different subsets of a dataframe. One option is use different color code, another is to use different panels.
- The `facets` argument takes such format, a variable on the left side and a variable on the right side and they are separated by a `~`.
- The left side is the row of facets and the right side is the column of the facets. If there is nothing to specify, just use `.`

```
qplot(displ, hwy, data = mpg, facets = . ~ drv)
qplot(hwy, data = mpg, facets = drv ~ ., binwidth = 2)
```

## Density smooth

```
qplot(log(eno), data = maacs, geom = "density", color = mopos)
```

## Scatterplot

- In addition to separate subset by color code, we can also use `shape` argument
- We can also add smooth line, the default smooth fitting method is “loose”, we can specify by changing the argument `method`

```
# separate by shape
qplot(log(eno), log(pm25), data = maacs, shape = mopos)
# separate by color
qplot(log(eno), log(pm25), data = maacs, color = mopos)
# Adding linear regression model smooth line
qplot(log(eno), log(pm25), data = maacs, color = mopos, geom = c("point", "smooth"), method = "lm")
# separate by facets argument
qplot(log(eno), log(pm25), data = maacs, facets = . ~ mopos, geom = c("point", "smooth"), method = "lm")
```

## Summary of qplot

- Analog to plot but with many built-in features
- Syntax between base and lattice system
- Nice graphics
- Don't bother to customize it, use `ggplot2` full power

## ggplot part3

### Basic components

- dataframe: the data source
- aesthetic mappings: color, size
- geometric objects: points, lines, bars, tiles
- facets: for conditional graph
- stats: statistical transformation: binning, quantiles, smoothing
- scales: scale aesthetic mapping uses, e.g. male = red, female = blue
- coordinate system

### Building Plots with ggplot2

- Artist's palette model
- Plots are built in layers
  - plot the data
  - Overlay a summary
  - Metadata and annotation

```
ggplot(logpm25, NocturnalSympt, data = maacs, facets = . ~ bmicat, geom = c("point", "smooth"), method =  
  
# Initial call to ggplot, specify dataframe, x, y  
g <- ggplot(maacs, aes(logpm25, NocturnalSympt))  
# Add objects to plot using +  
p <- g + geom_point()  
print(p)  
  
# Can add smooth line  
p <- g + geom_point() + geom_smooth()  
p <- g + geom_point() + geom_smooth(method = "lm")  
# Then add facets  
# The labels are from the variable  
# It's better to make sure to label data properly  
p <- p + facet_grid(. ~ bmicat)
```

### Annotation

- Labels: xlab, ylab, lab, ggtitle
- Each geom function has options to modify
- For things that make sense globally, use theme()
  - theme(legend.position = "none")
- Two standard appearance: theme\_gray(), theme\_bw()

```
geom_point(color = "steelblue", alpha = 1/2, size = 4)  
# Note that if I want to assign color to different data, I have to wrap it in  
# aes() function, thus subsetting it with different colors based on factor variable values  
geom_point(aes(color = bmicat), alpha = 1/2, size = 4)  
# Add labels and title  
+labs(title = "MAACS Cohort")
```

```
+labs(x = expression("log " * PM[2.5]), y = "Nocturnal Symptoms")
# Modify smooth line, se turns off confidence interval
+ geom_smooth(size = 4, linetype = 3, method = "lm", se = FALSE)
# Change the background and font
+ theme_bw(base_family = "Times")
```

## ggplot2 part 5

### A note about axis limit

Sometimes we may not want to look at the outlier and only focus the typical data

```
# if we do this, ggplot will subset the data within the range, outlier is excluded
g <- ggplot(testdat, aes(x, y))
g + geom_line() + ylim(-3, 3)
# We might want to
g + geom_line() + coord_cartesian(ylim(-3, 3))
```

### More complex example

We want to see the NO2 and BMI, but NO2 is continuous variables. We could use `cut()` function to make it categorical variable.

### Making NO2 Tertile

```
# Calculate the deciles of the data
cutpoints <- quantile(maacs$logno2_new, seq(0, 1, length = 4), na.rm = TRUE)
# Cut the data at the deciles and create new
maacs$no2dec <- cut(maacs$logno2_new, cutpoints)
# See the levels of new factor variable
levels(maacs$no2dec)

# The real plotting
g <- ggplot(maacs, aes(logpm25, NocturnalSympt))
g + geom_point(alpha = 1/3)
+ facet_wrap(bmicat ~ no2dec, nrow = 3, ncol = 4)
+ geom_smooth(method = "lm", col = "steelblue", se = FALSE)
+ theme_bw(base_family = "Avenir", base_size = 10)
+ labs(x = expression("log " * PM[2.5]))
+ labs(y = "Nocturnal Symptoms")
+ lebs(title = "MAACS Cohort")
```

### Summary of ggplot

- Very powerful and flexible

## ??? PRACTICAS CON SWIRL

Para realizar las prácticas con swirl, hacemos lo siguiente:



```
install.packages("swirl")  
library(swirl)  
  
install_from_swirl("Exploratory Data Analysis")  
swirl()
```

Hay 5 lecciones para el curso 4.

1. Principles of Analytic Graphs
2. Exploratory Graphs
3. Graphics Devices in R
4. Plotting Systems Base Plotting System

## RECURSOS EXTERNOS

R graph gallery

R Bloggers