Ready for R

A language and environment for statistical computing and graphics

Eugenio Thieme

Universitá degli Studi di Milano

Table of contents

But before we begin, I want to start with the caveats. And the bad news is, whenever you're learning a new tool, for a long time you're going to suck.

But the good news is that that is typical of something that happens to everyone and it's only temporary. Unfortunately there's no way to go from knowing nothing about a subject to knowing something about a subject and being an expert in it without going through a period of great frustration and much suckiness.

Hadley Wickham, dplyr workshop @UCLA, useR 2014

Introduzione

Cos'è R?

R is a language and environment for statistical computing and graphics

(Fonte: https://www.r-project.org/about.html)

Per essere un po’ piú precisi...

- é un progetto GNU
- é un ”dialetto” di S
- é un linguaggio di programmazione interpretato

Perchè non usare Python?

Ci sono tanti motivi che spingono ad usare R al posto di Python per analisi dati e in data science:

- R é un linguaggio di programmazione per statistici **scritto da statistici** (eg. John Chambers, Hadley Wickham)
- CRAN

Ma vedremo che si puó usare con profitto anche all'infuori della statistica...

CRAN

CRAN (*The Comprehensive R Archive Network*) é il repository dei pacchetti di R. Al 01/04/17 sono contenuti 10365 pacchetti.

A differenza degli oltre 100000 pacchetti presenti su PyPI (il repo ufficiale per Python), la quasi totalitá dei pacchetti di CRAN sono per calcolo statistico e visualizzazione dati, rendendolo di fatto piú completo per la data analysis.

Un'analisi imparziale delle differenze tra R e Python può essere trovata qui: https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis#gs.SyDGnY8

Strutture dati

Vector

La struttura dati fondamentale in R é il **vector**. Esistono due tipi di vector, atomic vectors e lists, accomunati dal fatto di possedere tre proprietá: **typeof()**, **length()**, **attributes()**.

Mentre le liste possono contenere elementi di qualsiasi tipo, gli elementi degli atomic vectors devono essere tutti dello stesso tipo:

- logical
- integer
- numeric
- character
- (complex)
- (raw)

```
> # An atomic vector is declared via c()
> # (c() means "combine")
> temp <- c(0.2,0.5,1.47,3.82, 0.2, 0.78)
> temp
[1] 0.20 0.50 1.47 3.82 0.20 0.78
> # Testing type
> is.atomic(temp)
[1] TRUE
> is.list(temp)
[1] FALSE
> # Testing length
> length(temp)
[1] 6
```

```
> # Declaring an atomic vector of logical
> logi <- c(T,F,F)
> # A vector can also be subsetted:
> temp_subsetted <- temp[logi]
> temp_subsetted
[1] 0.20 3.82
> temp_subsetted[2]
[1] 3.82
```

Nota: a differenza di C, R ”comincia a contare” da 1. Questo per facilitare il subsetting e la rimozione di elementi da strutture dati.

```
> # Subset is also useful for removing
> # elements from an array:
> atomic <- c('first', 'second', NA, 'fourth', 5)
> is.atomic(atomic)
[1] TRUE
> # That's not an atomic! Elements aren't of the
> # same type... are they?
> # Here's where coercion comes out:
> atomic
[1] "first"  "second" NA       "fourth" "5"
> # We want to remove 3rd and 5th element:
> atomic <- atomic[c(-3,-5)]
> atomic
[1] "first"  "second" "fourth"
```

Dataframes

Il **dataframe** é la struttura dati più comune in cui immagazzinare dati in R, e consiste in una lista di vettori di uguale lunghezza.

A differenza di una matrice, permette di immagazzinare dati di tipo diverso, rendendo il dataframe molto piú versatile.

Un dataframe si dichiara attraverso la chiamata a *data.frame()*:

```
> foo <- data.frame(x=c('a','b','c','d'),
                    y=1:4, z=c(T,F))
> foo
  x y     z
1 a 1  TRUE
2 b 2 FALSE
3 c 3  TRUE
4 d 4 FALSE
```

Funzioni

É possibile (ed é fortemente consigliato) dichiarare funzioni: le funzioni si possono dichiarare nello script o includere da un file/pacchetto. Un esempio di dichiarazione:

```
#Declaring a function
> our_sum <- function(a,b){
+    ans <- a + b
+    ans
+ }
#Calling our function
> our_sum(100,200)
[1] 300
```

Figure 1: Questa é una parte del vocabolario essenziale...

```
library(wordcloud)

words <- c('str', 'which', 'predict',
           ..., 'read.csv')
b=sample(seq(0,1,0.01), length(words), replace=TRUE)
png(file='wordcloud.png', width=600, height = 600)
par(bg='white', cex=1.2)
wordcloud(words, b, rot.per=0.5)
dev.off()
```

Questo é il codice per riprodurre la figura precedente.
I puntini di sospensione nella dichiarazione del vettore sono stati messi per concisione, non sono parte del codice originario.

Pro Tip:

?png()
?sample()

Vettorizzazione

Failing to Vectorize

We arrive at the third Circle, filled with cold, unending rain. Here stands Cerberus barking out of his three throats. Within the Circle were the blasphemous wearing golden, dazzling cloaks that inside were all of lead—weighing them down for all of eternity. This is where Virgil said to me, "Remember your science—the more perfect a thing, the more its pain or pleasure."

(Tratto da *R Inferno*, Patrick Burns)

Sempre prendendo ispirazione da Patrick Burns, un esempio di codice scritto con forte accento di C

```
x <- c(1:10000000)
lsum <- 0
for(i in 1:length(x)) {
  lsum <- lsum + log(x[i])
}
```

e uno scritto in puro R

```
x <- c(1:10000000)
lsum <- sum(log(x))
```

Il codice scritto in puro R é piú leggibile, piú compatto e piú efficiente.

R non é un linguaggio particolarmente veloce (non é il suo scopo), ma diffidate dei benchmark che trovate in giro: spesso non sfruttano le peculiaritá proprie del linguaggio, ottenendo un risultato peggiore di quello ottenibile scrivendo in un buon R.

Usiamo la funzione **system.time()** per fare un benchmark tra i due frammenti di codice sopra riportati.

```
> system.time(
    for(i in 1:length(x))
      lsum <- lsum + log(x[i])
  )
   user  system elapsed
  9.306   0.020   9.363

> system.time(
    lsum <- sum(log(x))
  )
   user  system elapsed
  0.449   0.022   0.474
```

Scrivendo in R puro il codice é 20 volte piú veloce!

Hands on: interpolazione lineare

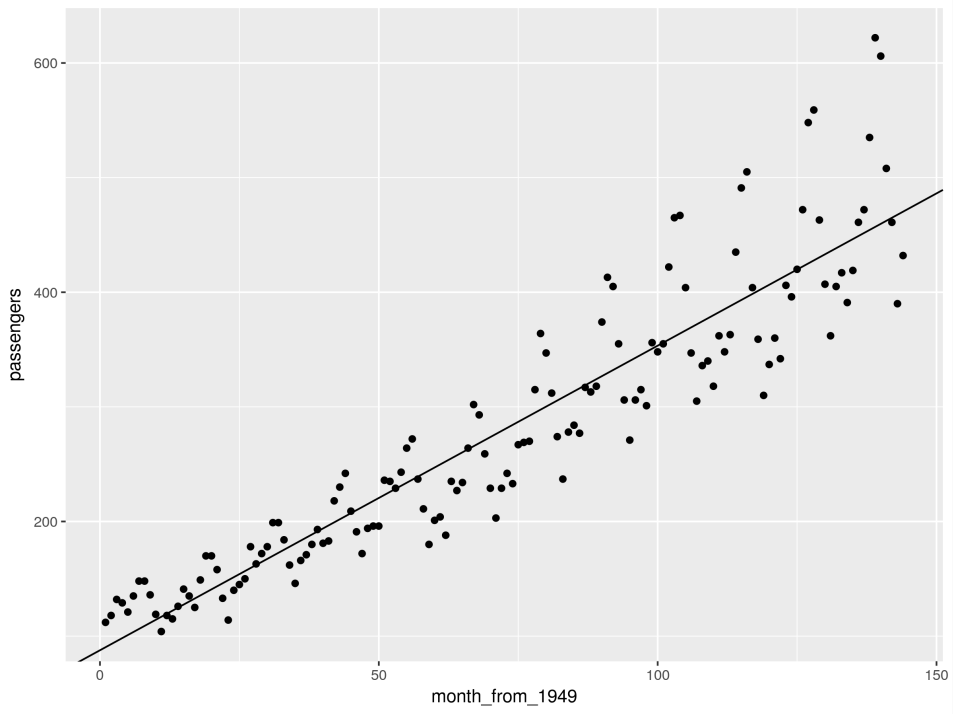
Uno dei problemi piú comuni da affrontare in analisi dati é quello di quantificare l'andamento di una serie di dati.

Vedremo di fare un'interpolazione lineare operando sul classico dataset di Box&Jenkins, che riporta il totale mensile di passeggeri delle linee aeree internazionali dal 1949 al 1960.

```r
library(ggplot2)
library(dplyr)
data("AirPassengers")
passengers <- data.frame(year = rep(1949:1960, each=12),
                         month = c(1:12),
                         passengers = AirPassengers)
#This could have been done using cbind()... try at home!
passengers <- mutate(passengers, month_from_1949 =
                       month + 12*(year-year[1]))
#We're fitting with a linear model
fitted <- lm(passengers$passengers ~
               passengers$month_from_1949)
#Get the summary for extracting statistics such as
#R-squared
summarised <- summary(fitted)
#Predict the new values
predicted <- predict(fitted)
```

Notiamo che, oltre alle chiamate alle librerie, nelle prime righe abbiamo riorganizzato il dataset: ogni colonna una variabile, ogni riga un'osservazione. Fatto questo il fit è estremamente rapido da fare!

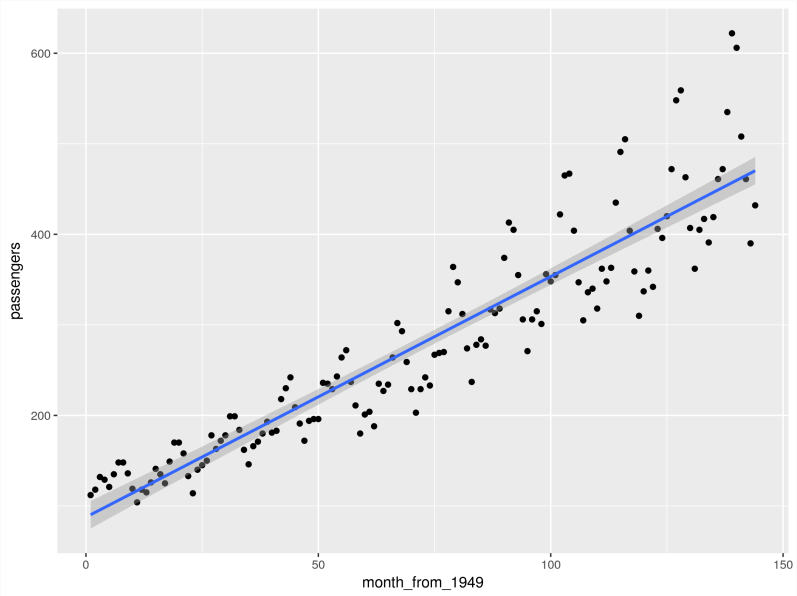
Vediamo di graficare il risultato per vedere se i nostri risultati sono ragionevoli.



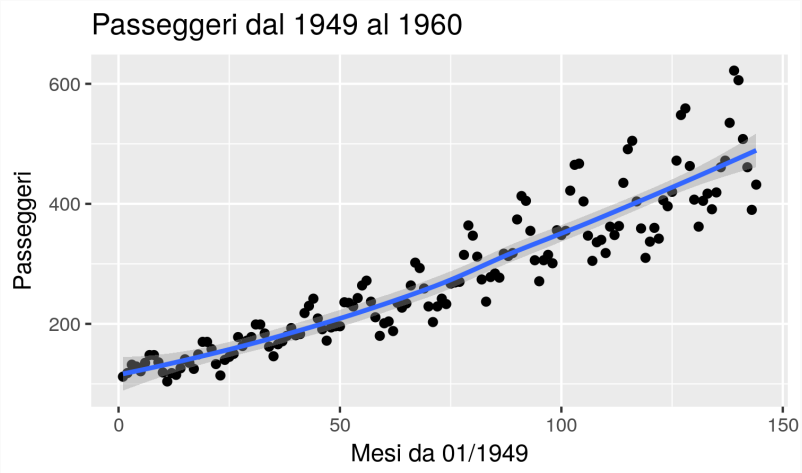

```
g1 <- ggplot(data=passengers, aes(x=month_from_1949,
                                  y=passengers)) +
      geom_point() +
      geom_abline(intercept = fitted$coefficients[1],
                  slope=fitted$coefficients[2])
g1
```

Ma ggplot2 ha la funzione di smoothing giá implementata, avremmo potuto (ed é consigliabile farlo) usarla.

```
g2 <- ggplot(data=passengers, aes(x=month_from_1949,
                                  y=passengers)) +
      geom_point() + geom_smooth(method = 'lm')
g2
```



E se volessimo mettere un titolo? E se il fit non fosse lineare?




```
g1 <- ggplot(data=passengers, aes(x=month_from_1949,
                                  y=passengers)) +
      geom_point() + geom_smooth() +
      labs(title ='Passeggeri dal 1949 al 1960',
           x = 'Mesi da 01/1949',
           y = 'Passeggeri')
g1
```

Ggplot2

Ggplot2

Ggplot2 é un pacchetto per la visualizzazione dati scritta da Hadley Wickham nel 2005, basato sulla *Grammar of Graphics* (Wilkinson).
In breve tempo é diventato uno dei piú popolari pacchetti di R, diventando lo standard de facto per la rappresentazione grafica.

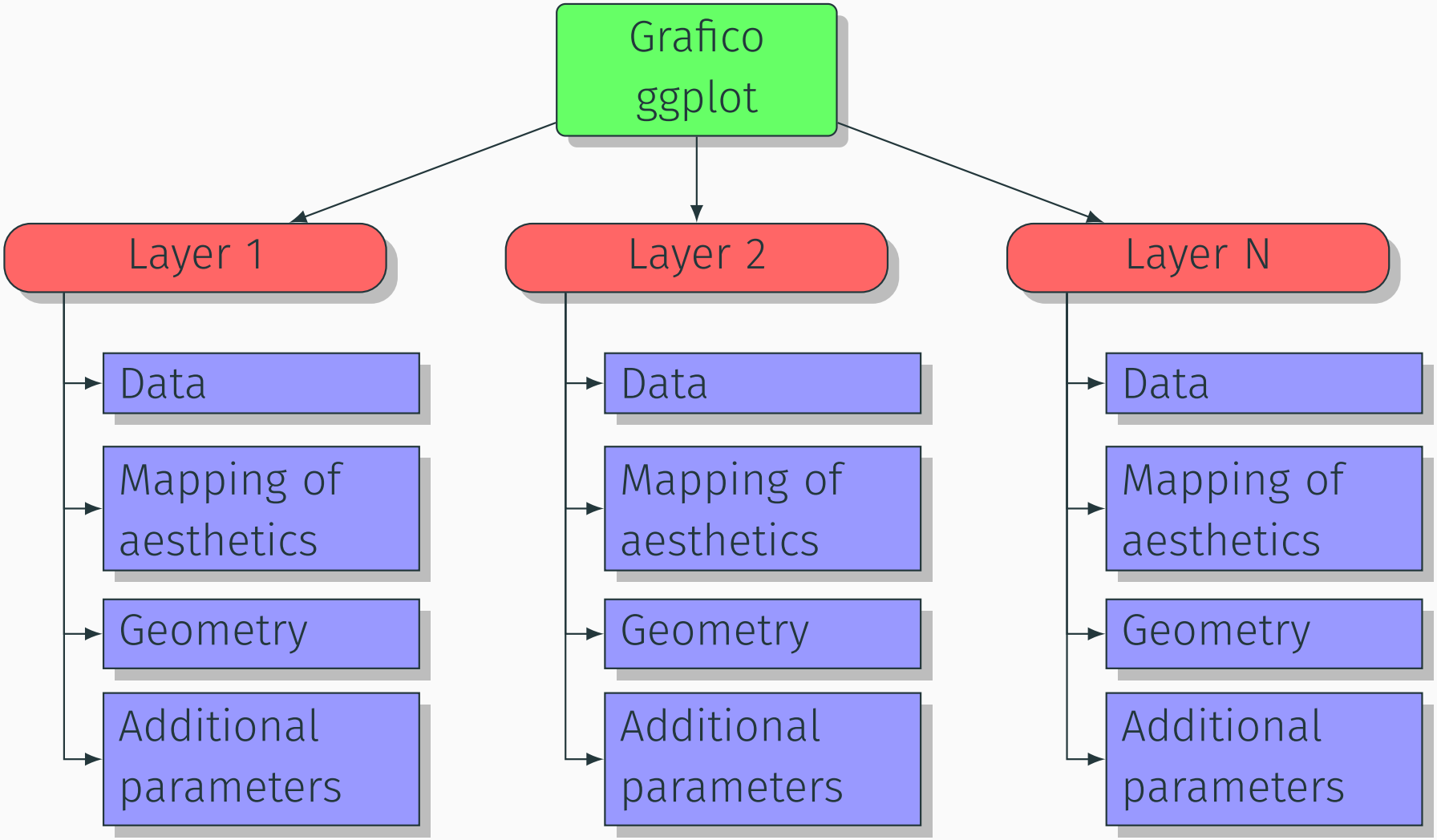
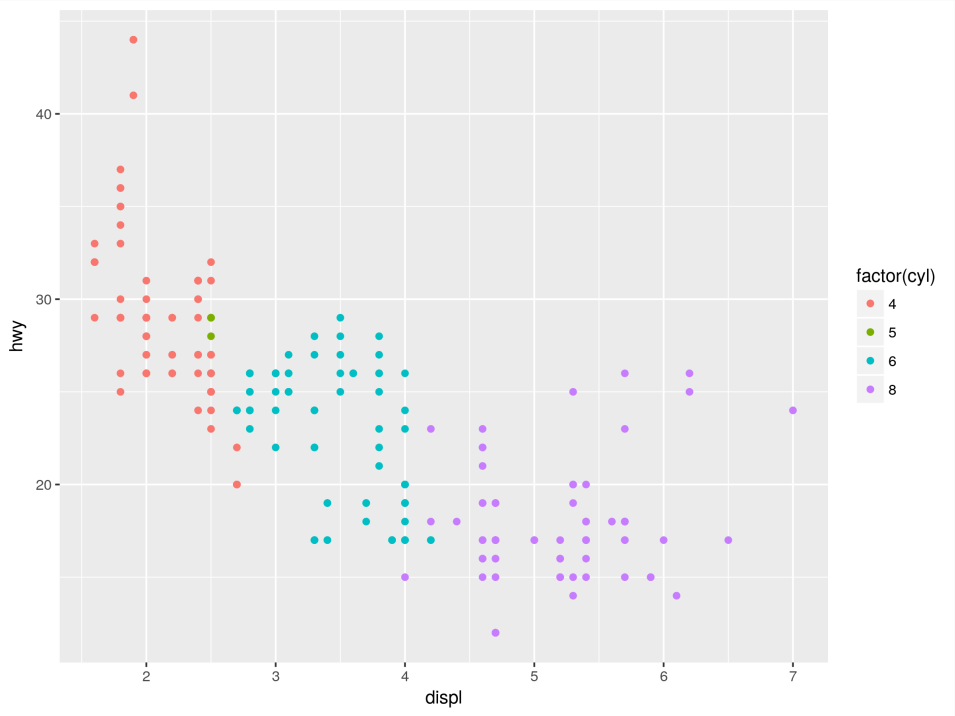


Grafico
ggplot
Layer 1
Layer 2
Layer N
Data
Mapping of
aesthetics
Geometry
Additional
parameters
Data
Mapping of
aesthetics
Geometry
Additional
parameters
Data
Mapping of
aesthetics
Geometry
Additional
parameters

Hands on: fuel economy data

Usiamo ggplot2 sul dataset *mpg*, contenente i dati relativi ai consumi di 38 modelli di auto dal 1999 al 2008, rilasciati dall'EPA.

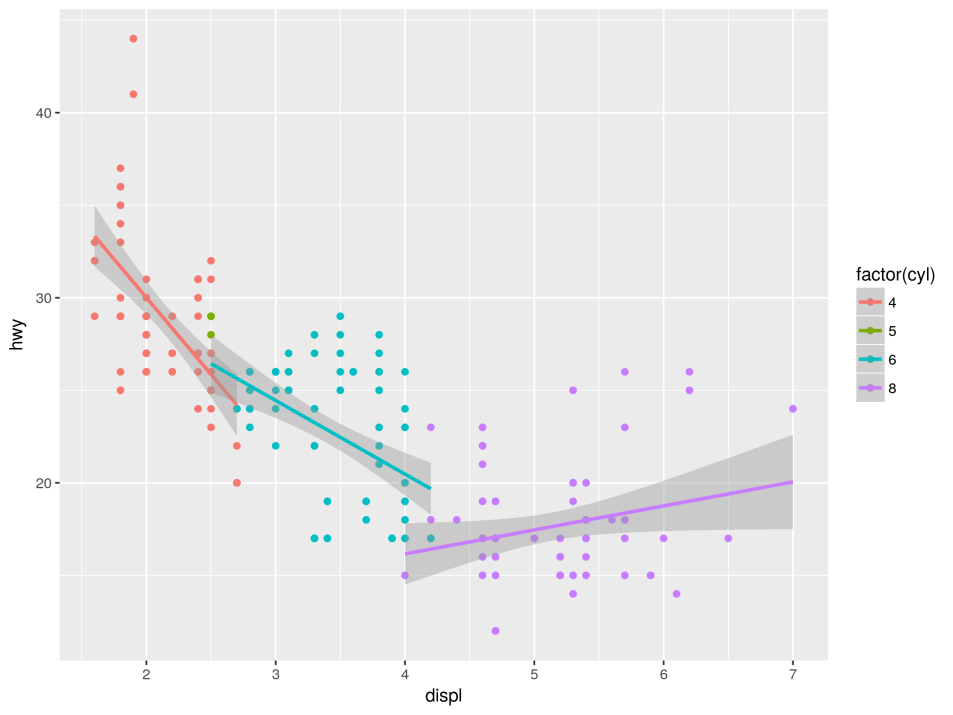
```
> str(mpg)
Classes 'tbl_'df', ''tbl and 'data.frame':^^I234 obs. of
11 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1  ...
 $ year        : int  1999 1999 2008 2008 1999 1999 ...
```




```
g <- ggplot(data=mpg, aes(x=displ, y=hwy)) +
geom_point(aes(color=factor(cyl)))
```

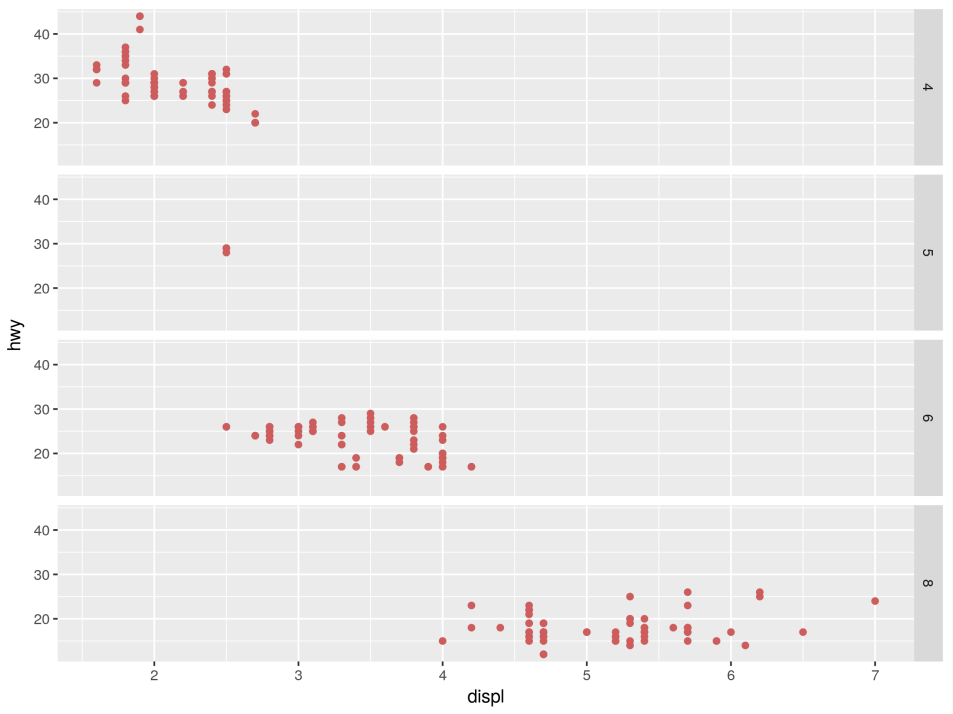
To do: provare a mappare i colori sul produttore. Chi produce i modelli piú economi?

Complichiamo un po' il grafico:




```
g <- ggplot(data=mpg, aes(x=displ, y=hwy)) +
     geom_point(aes(color=factor(cyl))) +
     geom_smooth(aes(color=factor(cyl)), method='lm')
```

Faceting




```
g <- ggplot(data=mpg, aes(x=displ, y=hwy)) +
     geom_point(color='indianred') +
     facet_grid(cyl ~ .)
```

Altre geometrie comuni in ggplot:

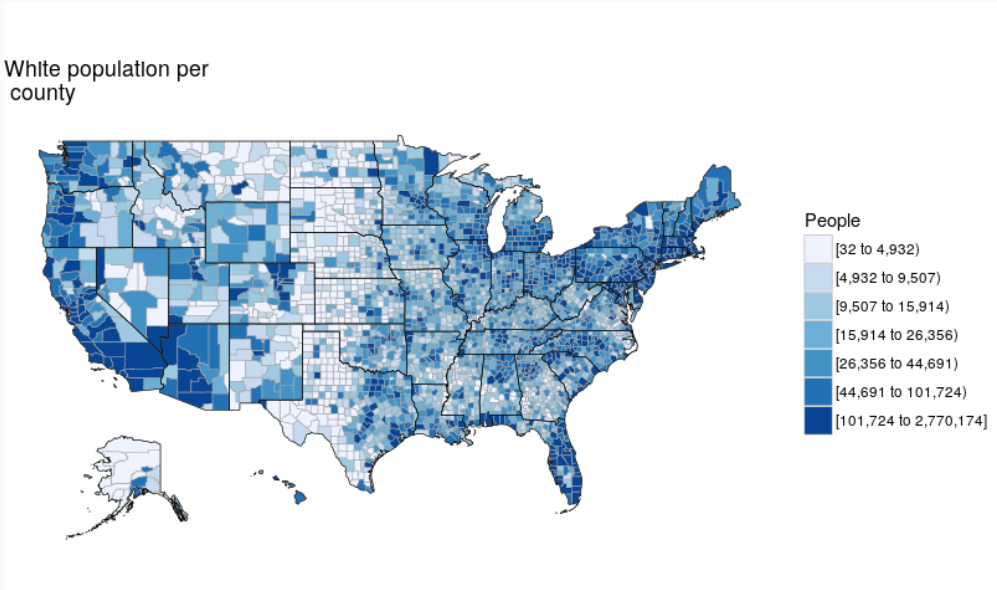
- geom_histogram
- geom_line
- geom_path
- geom_polygon
- geom_ribbon
- ...

Per via dell'elevato livello di astrazione di ggplot2 molti pacchetti si basano su di lui.

Choroplethr e Ggmap

Choroplethr é un pacchetto per fare mappe a coropleto

Ggmap é un pacchetto che permette di fare una query a GoogleMaps e OpenStreetMaps, scaricare una mappa e plottarci sopra usando ggplot2.



Multicore

doParallel

É possibile eseguire codice in parallelo facilmente attraverso l'utilizzo del pacchetto **doParallel** e la funzione **foreach**.

Alcune note:

- é possibile parallelizzare se il risultato non dipende dal passaggio precedente, ovvero se tutti i passaggi sono indipendenti l'uno dall'altro.
- la parallelizzazione ha un costo: nel caso di cicli computazionalmente poco onerosi, potrebbe essere piú veloce la versione sequenziale.

```r
library(doParallel)
registerDoParallel()
#Sequential version
foreach(i=1:100000) %do% sum(tanh(1:i))
#Parallel version
foreach(i=1:100000) %dopar% sum(tanh(1:i))
```

foreach

Description

%do% and %dopar% are binary operators that operate on a foreach object and an R expression. The expression, ex, is evaluated multiple times in an environment that is created by the foreach object, and that environment is modified for each evaluation as specified by the foreach object. %do% evaluates the expression sequentially, while %dopar% evalutes it in parallel. The results of evaluating ex are returned as a list by default, but this can be modified by means of the .combine argument.

```
> system.time(
    foreach(i=1:100000) %do% sum(tanh(1:i)))

   user  system elapsed
 64.005   0.034  64.307
> system.time(
    foreach(i=1:100000) %dopar% sum(tanh(1:i)))

   user  system elapsed
 31.253   1.166  32.562
```

Un esempio un po' meno banale:

```
data(iris)
x <- iris[which(iris[,5] != "setosa"), c(1,5)]
trials <- 10000
ptime <- system.time({
         r <- foreach(i=1:trials, .combine=cbind)
         %dopar% {
         ind <- sample(100, 100, replace=TRUE)
         result1 <- glm(x[ind,2]~x[ind,1],
         family=binomial(logit))
         coefficients(result1)
       }
   })[3]
ptime
```

Risorse

Risorse

Risorse distribuite sotto licenza libera:

- *R Inferno*, Patrick Burns (http://www.burns-stat.com/pages/Tutor/R_inferno.pdf)
- *Advanced R*, Hadley Wickham (http://adv-r.had.co.nz/)
- *R Packages*, Hadley Wickham (http://r-pkgs.had.co.nz/)
- *R Bloggers* (https://www.r-bloggers.com/)

E un libro tradizionale:

- *ggplot2: Elegant Graphics for Data Analysis*, Hadley Wickham

In piú, YouTube é pieno di video esplicativi e interventi di varie conferenze, anche di relatori importanti, estremamente interessanti.