

— Parte 4. —

Tidyverse

Paolo Bosetti (paolo.bosetti@unitn.it)

Indice

1	Introduzione al Tidyverse	1
1.1	Strutture dati	2
1.2	Lettura e importazione dati	3
1.3	Manipolazione dati	4
1.4	Grafici base	8
1.4.1	Aesthetics	9
1.4.2	Facets	11
1.4.3	Scale	11
1.4.4	Palette	12
1.4.5	Limiti degli assi	13
2	Esempi	15
2.1	Distribuzioni	15
2.1.1	Distribuzioni discrete	15
2.1.2	Distribuzioni continue	16
2.1.3	Iistogrammi e QQ-plot	17
2.1.4	Boxplot	18
2.1.5	istogrammi marginali	19
2.2	Serie temporali	19

1 Introduzione al Tidyverse

Tidyverse rappresenta una collezione di librerie nate attorno a RStudio (spesso dagli stessi sviluppatori), allo scopo di modernizzare R sia come linguaggio sia come potenzialità soprattutto nella creazione di grafici complessi e nella gestione di *big data*.

Un elemento comune alle librerie Tidyverse è quello di sostituire funzioni nidificate, tipiche di R, con sequenze di operazioni: da `f(g(x))` a `g(x) %>% f()`. Lo scopo è generalmente rendere il codice più leggibile e flessibile.

Questo nuovo approccio è reso possibile dal due caratteristiche di R:

1. R è un linguaggio funzionale, in cui le funzioni sono *first class objects*;
2. Le funzioni accettano parametri nominati in qualsiasi ordine, anonimi nell'ordine in cui sono definiti;
3. Gli stessi operatori come `+`, `-`, ..., sono in realtà *funzioni* con due argomenti;
4. Gli operatori possono essere *ridefiniti* per nuovi oggetti, ed è possibile definire *nuovi operatori* del tipo `%op%`, dove `op` può essere un qualsiasi insieme di caratteri.

Grazie a queste caratteristiche, Tidyverse introduce la possibilità di accodare operazioni, avendo ridefinito l'operatore `+`, e di passare strutture dati mediante l'operatore `%>%` (*pipe*). In quest'ultimo caso un esempio è particolarmente efficace:

```
set.seed(123)
(x <- rnorm(10)) %>% mean %>% round(digits=2)

## [1] 0.07
```

che è equivalente alla “vecchia” sintassi:

```
set.seed(123)
round(mean(x <- rnorm(10)), digits=2)

## [1] 0.07
```

ma molto più leggibile.

Nell’ultimo esempio, si noti che il vettore `x` **rimane invariato**, mentre se avessimo scritto `x <- rnorm(10) %>% mean %>% round(digits=2)`, allora `x` conterrebbe la media del vettore generato (che a sua volta andrebbe perso).

In generale, tuttavia, la “nuova moda” non sostituisce completamente la vecchia, per alcuni motivi:

- Tidyverse è comunque abbastanza pesante
- se certe funzionalità (ad esempio il *pipe*) sostanzialmente non hanno svantaggi, altre vanno poco d'accordo con librerie e funzioni “vecchie”, ed è soprattutto il caso dei grafici
- in certi casi, ed è di nuovo il caso dei grafici, le vecchie funzioni sono semplicemente più concise e rapide, quindi preferibili se si generano grafici per analisi dati piuttosto che per la pubblicazione finale
- spesso le funzioni grafiche sono molto potenti, ma tendono a nascondere all'utente i dettagli degli algoritmi utilizzati

Le librerie che fanno parte di Tidyverse possono essere caricate individualmente oppure con un'unica istruzione `library(tidyverse)`. Tuttavia, se se ne usa solo un limitato sottoinsieme spesso conviene caricarle individualmente.

1.1 Strutture dati

Le librerie Tidyverse si aspettano i dati in formato *tidy*, cioè un’osservazione per riga, un osservando per colonna. I dati sono tipicamente contenuti in data frame o, preferibilmente, nella nuova classe `tibble`, che è per lo più interscambiabile con i data frame, dato che ne eredita la classe. Una `tibble` può essere creata convertendo un data frame, oppure passando i dati per colonne (come per la funzione `data.frame()`), oppure ancora passando i dati **per righe**, mediante la funzione `tribble()` (notare la `r`, per `rows`):

```
n <- 10
a <- data.frame(
  In=1:n,
  Out=(1:n)^2 + rnorm(n, sd=1),
  Size=rnorm(n, 10, 0.1)) %>%
  tribble %>%
  print

## # A tibble: 10 x 3
##       In     Out   Size
##   <int> <dbl> <dbl>
## 1     1    2.22  9.89
## 2     2    4.36  9.98
## 3     3    9.40  9.90
## 4     4   16.1   9.93
## 5     5   24.4   9.94
```

```

## 6      6 37.8  9.83
## 7      7 49.5 10.1
## 8      8 62.0 10.0
## 9      9 81.7  9.89
## 10     10 99.5 10.1

(b <- tribble(
  ~name, ~age,
  "Paolo", 50,
  "Luca", 45,
  "Lucia", 38,
  "Anna", 52
)) %>% knitr::kable()



|       | name | age |
|-------|------|-----|
| Paolo | 50   |     |
| Luca  | 45   |     |
| Lucia | 38   |     |
| Anna  | 52   |     |


```

Gli argomenti della funzione `tibble()` sono valutati in maniera *lazy*, cioè solo quando sono effettivamente utilizzati, a differenza che in `data.frame()`:

```

rm(x, y)
try(
  df <- data.frame(x=1:10, y=x^2)
)

## Error in data.frame(x = 1:10, y = x^2) : object 'x' not found

tb <- tibble(x=1:5, y=x^2)
tb

## # A tibble: 5 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     2     4
## 3     3     9
## 4     4    16
## 5     5    25

```

1.2 Lettura e importazione dati

La libreria `readr` mette a disposizione:

- `read_csv()`/`read_csv2()`: comma separated (CSV) files
- `read_tsv()`: tab separated files
- `read_delim()`: general delimited files
- `read_fwf()`: fixed width files
- `read_table()`: tabular files where columns are separated by white-space.
- `read_log()`: web log files

Tutte queste funzioni restituiscono una `tibble`.

```
stat <- read_csv2(mydata("censimento_TAA_2011.csv"))
```

```

## i Using ',',," as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
## Rows: 2055 Columns: 154
## -- Column specification -----
## Delimiter: ";"
## chr (5): REGIONE, PROVINCIA, COMUNE, LOCALITA, ALTITUDINE
## dbl (149): CODREG, CODPRO, CODCOM, PROCOM, LOC2011, CODLOC, TIPOLOC, AMPLOC, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
trace <- read_csv2(mydata("tracciato_2011_localita.csv"))

## i Using ',',," as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.
## Rows: 154 Columns: 2
## -- Column specification -----
## Delimiter: ";"
## chr (2): Nome Campo, Definizione
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

Ovviamente esistono anche le controparti `write_*`(), che accettano come primo argomento un data rame oppure una `tibble`. Generalmente queste funzioni sono molto più veloci delle controparti della libreria `base` (`write.*()`).

1.3 Manipolazione dati

La libreria `dplyr` mette a disposizione un'ampia scelta di funzioni per la manipolazione di tabelle e `tibble`. In generale, queste funzioni semplificano e rendono più leggibili operazioni che sarebbero comunque realizzabili mediante approcci più standard, anche se con più passaggi e in maniera meno efficiente.

Nella nomenclatura tidy, una `tibble` contiene una o più *variabili* (colonne), ciascuna con valori per uno o più *casi* (righe). Secondo questa convenzione, le principali funzioni di `dplyr` sono:

- `mutate()` aggiunge nuove variabili, funzione di variabili esistenti
- `select()` seleziona variabili in base al loro nome
- `filter()` seleziona casi in base ai loro valori
- `summarise()` riporta più valori ad un unico indicatore
- `arrange()` riordina i casi (righe)

Vediamo esempi di filtraggio:

```

starwars %>%
  filter(homeworld == "Naboo" & species == "Human")

## # A tibble: 5 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>     <int> <dbl> <chr>       <chr>       <dbl> <chr> <chr>
## 1 Palpati~    170     75 grey        pale        yellow       82 male   mascul-
## 2 Gregar ~    185     85 black       dark        brown        NA male   mascul-
## 3 Cordé       157     NA brown      light       brown        NA fema~ feminis-
## 4 Dormé       165     NA brown      light       brown        NA fema~ feminis-
## 5 Padmé A~    165     45 brown      light       brown       46 fema~ feminis-
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>

```

Selezione:

```
starwars %>%
  select(name, ends_with("color")) %>%
  names

## [1] "name"      "hair_color" "skin_color" "eye_color"
```

Modifica:

```
starwars %>%
  filter(species == "Human") %>%
  mutate(name, bmi = round(mass / ((height / 100) ^ 2), 1)) %>%
  select(name:mass, bmi) %>%
  arrange(desc(bmi), )

## # A tibble: 35 x 4
##   name           height  mass   bmi
##   <chr>        <int> <dbl> <dbl>
## 1 Owen Lars       178   120  37.9
## 2 Jek Tono Porkins    180   110  34
## 3 Darth Vader      202   136  33.3
## 4 Beru Whitesun lars  165    75  27.5
## 5 Wedge Antilles     170    77  26.6
## 6 Luke Skywalker     172    77  26
## 7 Palpatine          170    75  26
## 8 Lobot              175    79  25.8
## 9 Lando Calrissian     177    79  25.2
## 10 Biggs Darklighter    183    84  25.1
## # ... with 25 more rows
```

Si noti che la variante %<>% dell'operatore pipe consente di fare una *modifica sul posto* di una tibble:

```
library(magrittr)

##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##   set_names

## The following object is masked from 'package:tidyverse':
##
##   extract

humans <- starwars %>% filter(species == "Human")
humans %<>% mutate(name, bmi = round(mass / ((height / 100) ^ 2), 1))
humans

## # A tibble: 35 x 15
##   name    height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>        <dbl> <chr> <chr>
## 1 Luke S~     172    77 blond     fair       blue         19  male  masculin
## 2 Darth ~     202   136 none      white      yellow       41.9 male  masculin
## 3 Leia O~     150    49 brown     light      brown        19  femal feminin
## 4 Owen L~     178   120 brown, grey light      blue        52  male  masculin
## 5 Beru W~     165    75 brown     light      blue        47  femal feminin
## 6 Biggs ~     183    84 black     light      brown        24  male  masculin
## 7 Obi-Wa~     182    77 auburn, wh~ fair      blue-gray     57  male  masculin
```

```

##  8 Anakin~   188    84 blond      fair      blue       41.9 male  masculin-
##  9 Wilhuf~   180    NA auburn, gr~ fair      blue        64 male  masculin-
## 10 Han So~   180    80 brown      fair      brown       29 male  masculin-
## # ... with 25 more rows, and 6 more variables: homeworld <chr>, species <chr>,
## #   films <list>, vehicles <list>, starships <list>, bmi <dbl>

```

Talvolta i dati sono inclusi come *nomi di riga*. Ciò può avvenire solo se i dati sono originariamente in un data frame, dato che le **tibble** non supportano i nomi di riga:

La funzione `summarise()` è particolarmente utile assieme a `group_by()`:

```
starwars %>%
  group_by(species) %>%
  summarise(
    n = n(),
    mass = round(mean(mass, na.rm=T), 1),
    "max height" = max(height, na.rm=T),
    "min height" = min(height, na.rm=T)
  ) %>%
  filter(
    n > 1,
```

```

    mass > 50
  )

## # A tibble: 8 x 5
##   species     n   mass `max height` `min height`
##   <chr>     <int> <dbl>        <int>        <int>
## 1 Droid       6   69.8        200         96
## 2 Gungan      3    74        224        196
## 3 Human      35   82.8        202        150
## 4 Kaminoan    2    88        229        213
## 5 Mirialan    2   53.1        170        166
## 6 Twi'lek      2    55        180        178
## 7 Wookiee     2   124        234        228
## 8 Zabrak      2    80        175        171

```

Sono infine particolarmente utili le funzioni per **combinare tabelle di dati** secondo il paradigma relazionale:

```

persons <- tribble(
  ~name, ~surname, ~role,
  "Paolo", "Bosetti", 1,
  "John", "Smith", 2,
  "Phil", "Cameron", 1,
  "Eddy", "Hunt", 3,
  "Sebastian", "Hauer", 3
)

roles <- tribble(
  ~id, ~role,
  1, "attack",
  2, "play",
  3, "defense"
)

team <- left_join(persons, roles, by=c("role"="id"))
team %>% filter(role.y=="attack")

## # A tibble: 2 x 4
##   name surname role role.y
##   <chr> <chr>   <dbl> <chr>
## 1 Paolo Bosetti     1 attack
## 2 Phil Cameron      1 attack

# Equivalento a:
team[team$role.y=="attack",]

## # A tibble: 2 x 4
##   name surname role role.y
##   <chr> <chr>   <dbl> <chr>
## 1 Paolo Bosetti     1 attack
## 2 Phil Cameron      1 attack

mtcars %>% mutate(
  kpl=round(mpg*0.621371/3.78541, 1),
  lp100k=round(100/kpl, 1)) %>%
  rownames_to_column(var="Model") %>%
  head

##           Model mpg cyl disp  hp drat    wt  qsec vs am gear carb kpl

```

```

## 1      Mazda RX4 21.0   6 160 110 3.90 2.620 16.46 0 1 4 4 3.4
## 2      Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02 0 1 4 4 3.4
## 3      Datsun 710 22.8    4 108 93 3.85 2.320 18.61 1 1 4 1 3.7
## 4      Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44 1 0 3 1 3.5
## 5      Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02 0 0 3 2 3.1
## 6      Valiant 18.1     6 225 105 2.76 3.460 20.22 1 0 3 1 3.0
##   lp100k
## 1 29.4
## 2 29.4
## 3 27.0
## 4 28.6
## 5 32.3
## 6 33.3

```

1.4 Grafici base

Forse una delle innovazioni più importanti di Tidyverse è la nuova interfaccia di plotting, che è fornita dalla libreria `ggplot2`, dove la doppia `g` sta per *Grammar of Graphics*, secondo l'idea di comporre un grafico accostando funzioni secondo una grammatica base che consente di tenere separati i dati dagli algoritmi e dall'estetica.

Una interessante caratteristica di `ggplot2` è che consente di adottare dei temi (personalizzabili) in modo da controllare l'aspetto estetico dei grafici realizzati. I temi vengono caricati così:

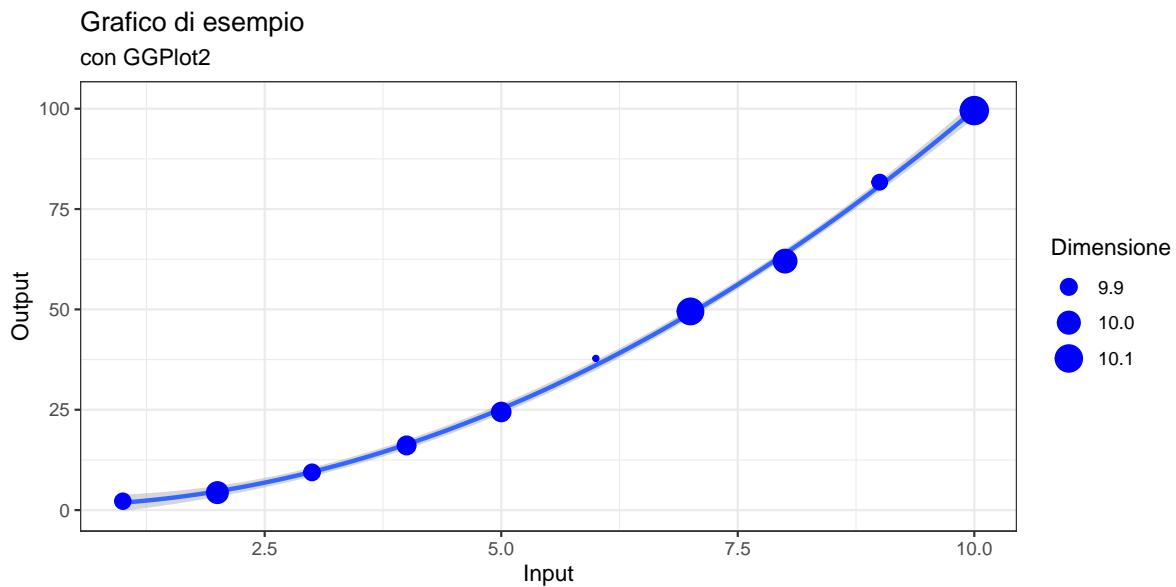
```
theme_set(theme_bw())
```

Un GGPlot viene creato inizialmente con la funzione `ggplot()` che richiede la struttura dati (`tibble` o `dataframe`) e gli *aesthetics*, cioè un comando che specifica *cosa* deve essere visualizzato nel grafico (funzione `aes()`). Quest'unico comando tuttavia produce esclusivamente un grafico vuoto: per riempirlo è necessario aggiungere dei *layer*, **commando** al `ggplot` opportuni comandi:

```

ggplot(a, aes(In, Out)) +
  geom_smooth(formula = y~poly(x,2), method="lm") +
  geom_point(aes(size=Size), color="blue") +
  labs(title="Grafico di esempio", subtitle = "con GGPlot2") +
  xlab("Input") +
  ylab("Output") +
  labs(size="Dimensione")

```

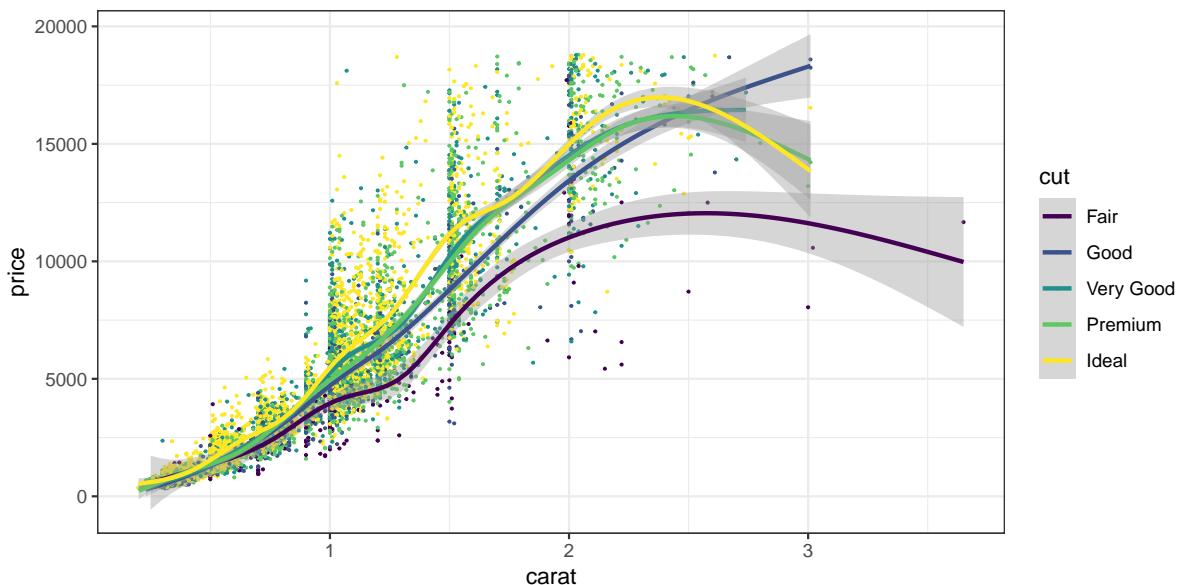


1.4.1 Aesthetics

È importante comprendere il ruolo degli *aesthetics*: essi definiscono quali *variabili* nei dati originali vengono mappati sui due assi e sui vari indicatori del grafico (colori, dimensioni, forma, riempimento, ecc.). A seconda dei casi `aes()` può essere indicata solo in `ggplot()` oppure anche nei comandi `geom_*`() successivi:

```
gp <- diamonds %>%
  slice_sample(prop=0.2) %>%
  ggplot(aes(x=carat, y=price, color=cut)) +
  geom_point(size=1/4) +
  geom_smooth()
print(gp)

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

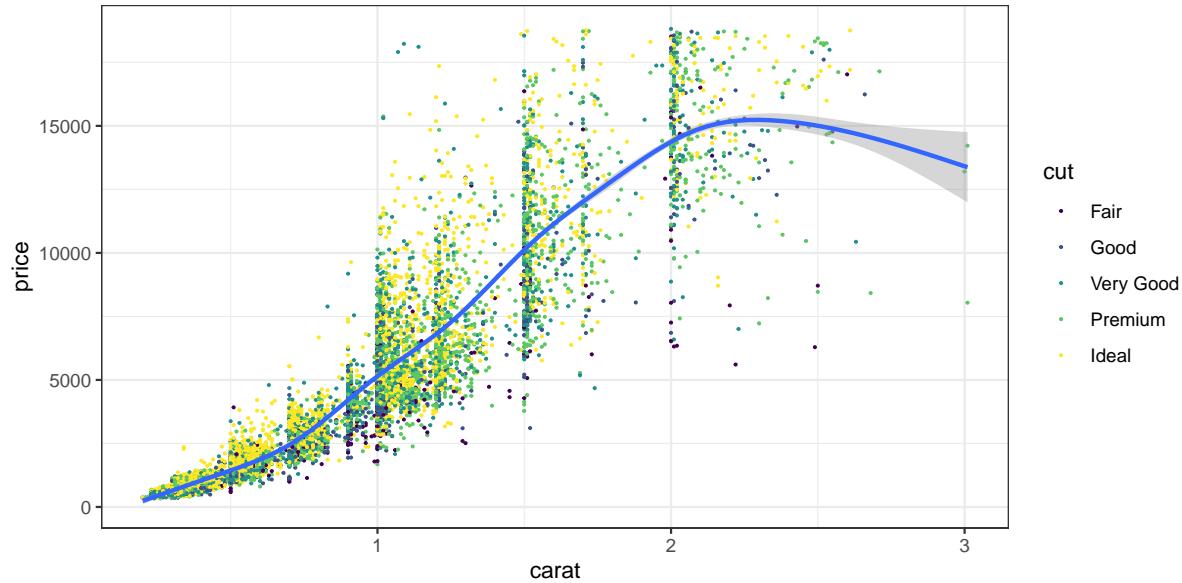


Come si vede, siccome l'estetica include i tre parametri, se non si specifica nessuna estetica per `geom_smooth()` essa viene applicata per ciascun raggruppamento previsto dall'estetica `color`. Se invece volessimo un'u-

nica linea di tendenza dovremmo specificare un'estetica apposita e **separata** per `geom_point()` e per `geom_smooth()`:

```
diamonds %>%
  slice_sample(prop=0.2) %>%
  ggplot() +
  geom_point(aes(x=carat, y=price, color=cut), size=1/4) +
  geom_smooth(aes(x=carat, y=price))

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

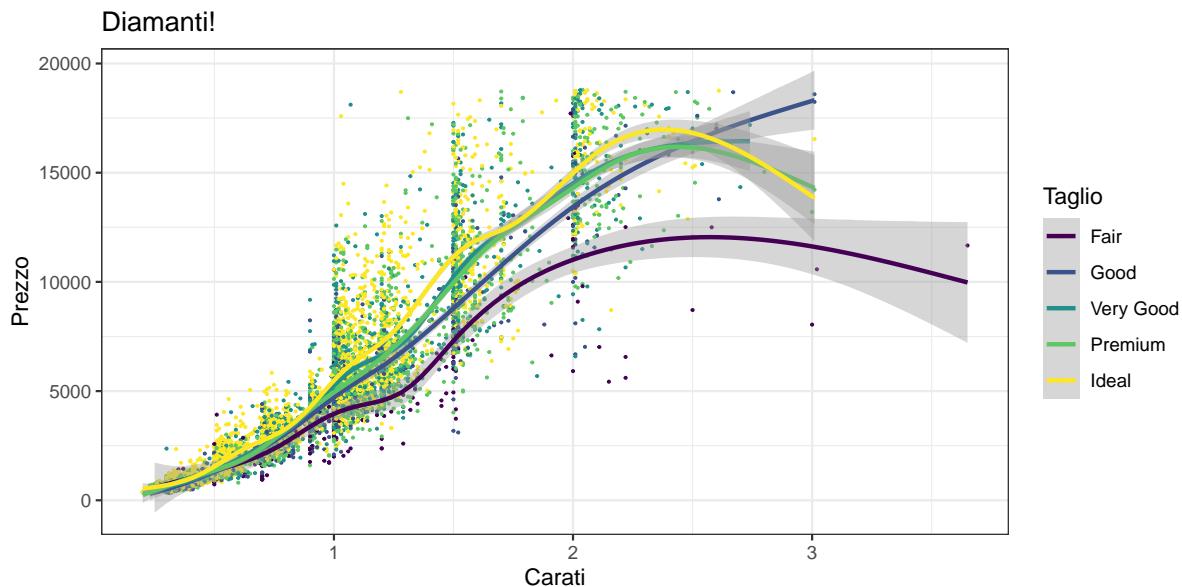


Si noti l'uso della funzione `slice_sample()` (in `dplyr`) per estrarre a caso solo il 20% dei dati, in modo da velocizzare la creazione del grafico.

Le etichette di testo possono essere modificate con il verbo `labs()`:

```
gp + labs(title="Diamanti!", x="Carati", y="Prezzo", color="Taglio")

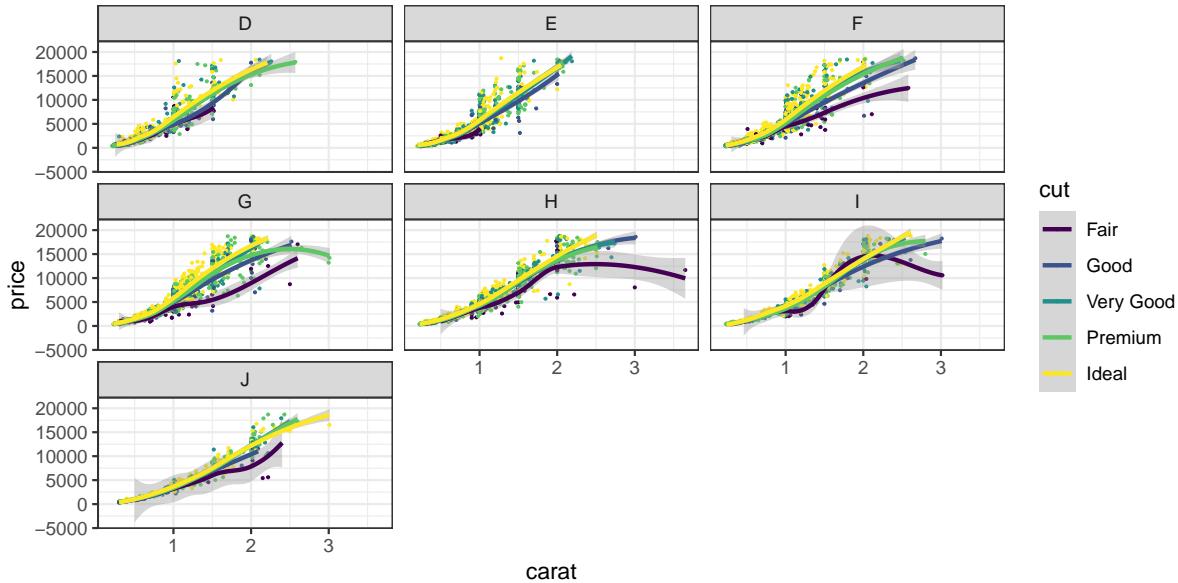
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



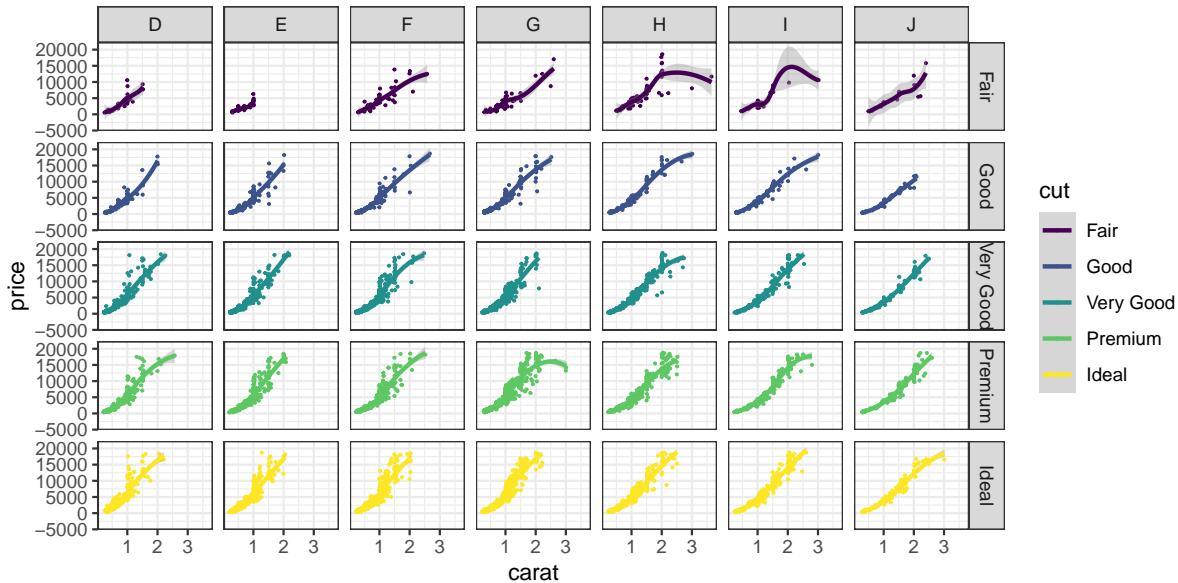
1.4.2 Facets

Altri termini di raggruppamento possono essere inclusi variando ad esempio `size`, `shape`, `fill` dei punti del grafico, oppure creando matrici di grafici in cui le variabili da utilizzare sulle righe e sulle colonne della matrice vanno indicate mediante una `formula` di tipo `row ~ column`, omettendo eventualmente la prima o la seconda:

```
gp + facet_wrap(~color)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
gp + facet_grid(cut~color)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

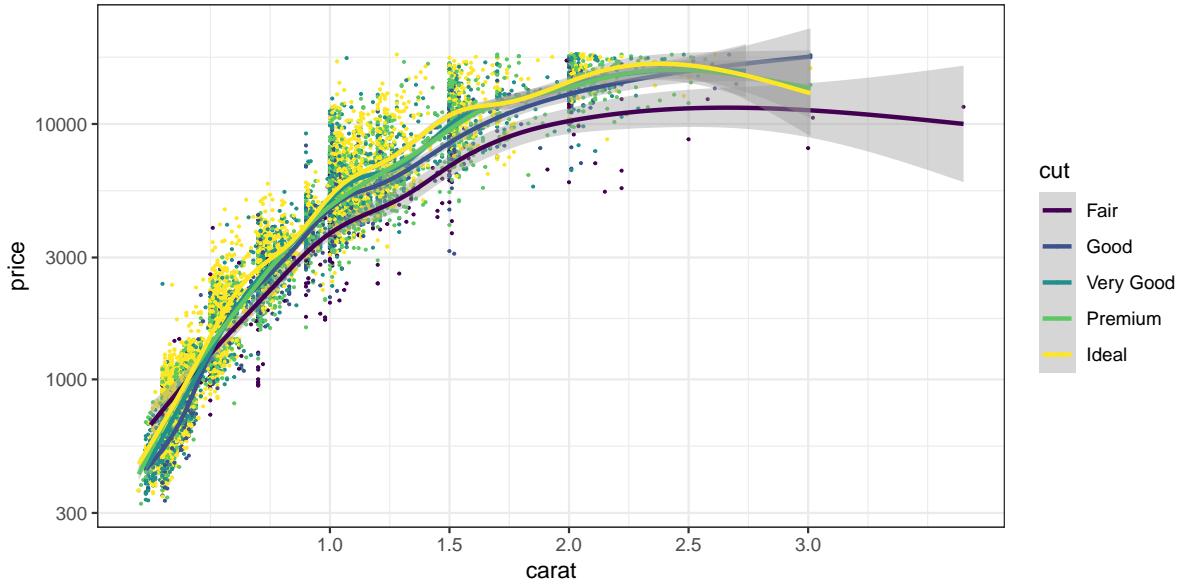


1.4.3 Scale

I verbi ggplot che cominciano con `scale_` possono essere utilizzati per manipolare le scale degli assi (sia gli assi `x` e `y` che gli assi virtuali). Ad esempio è possibile cambiare la spaziatura delle etichette

degli assi (`scale_*_continuous(breaks=c(...))`) che il tipo di scala (`scale_*_log10()` piuttosto che `scale_*_reverse()`):

```
gp + scale_x_continuous(breaks = seq(1, 3, 0.5)) +
  scale_y_log10()
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



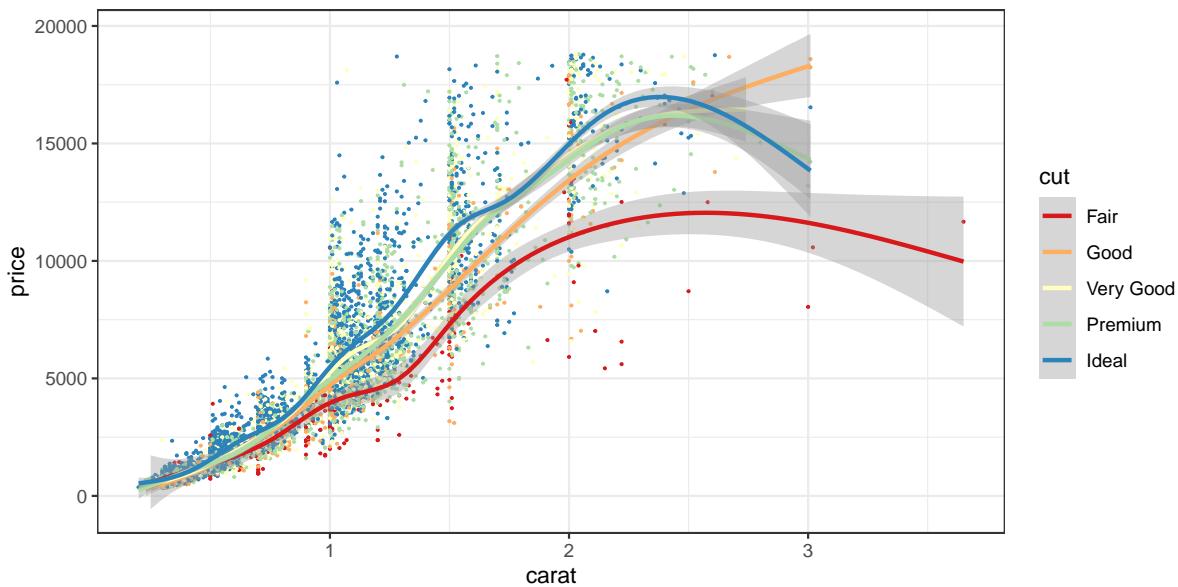
1.4.4 Palette

Un particolare tipo di scala è quella dei colori. Le palette di colori disponibili sono visualizzabili con il comando (libreria `RColorBrewer`):

```
display.brewer.all()
```

Ad esempio:

```
gp +
  scale_color_brewer(palette = "Spectral")
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



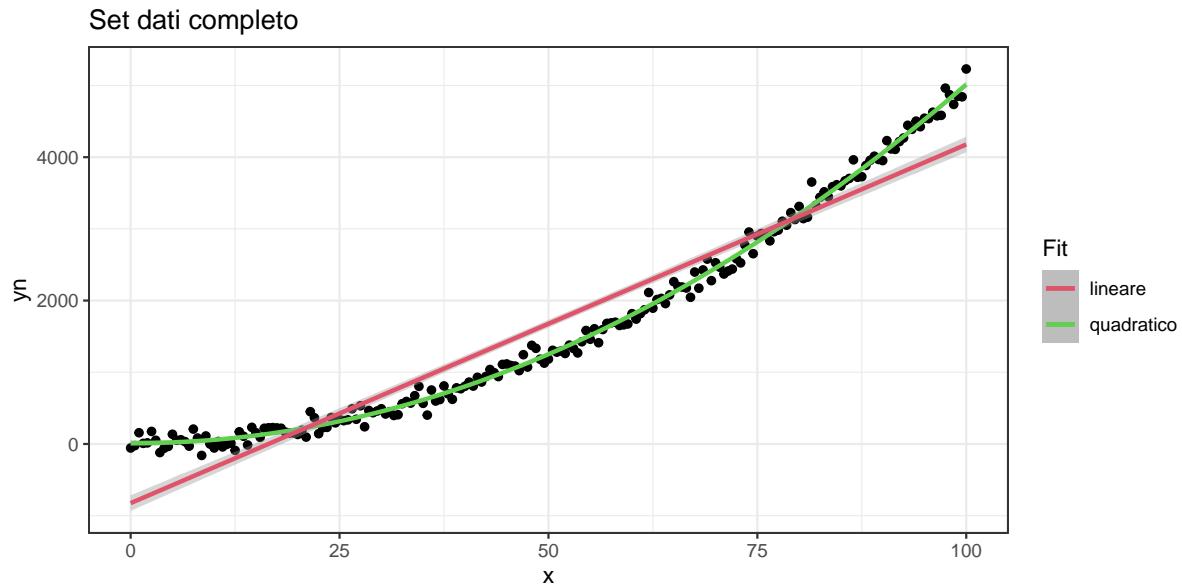
1.4.5 Limiti degli assi

In GGplot2 ci sono due modi per modificare i limiti degli assi:

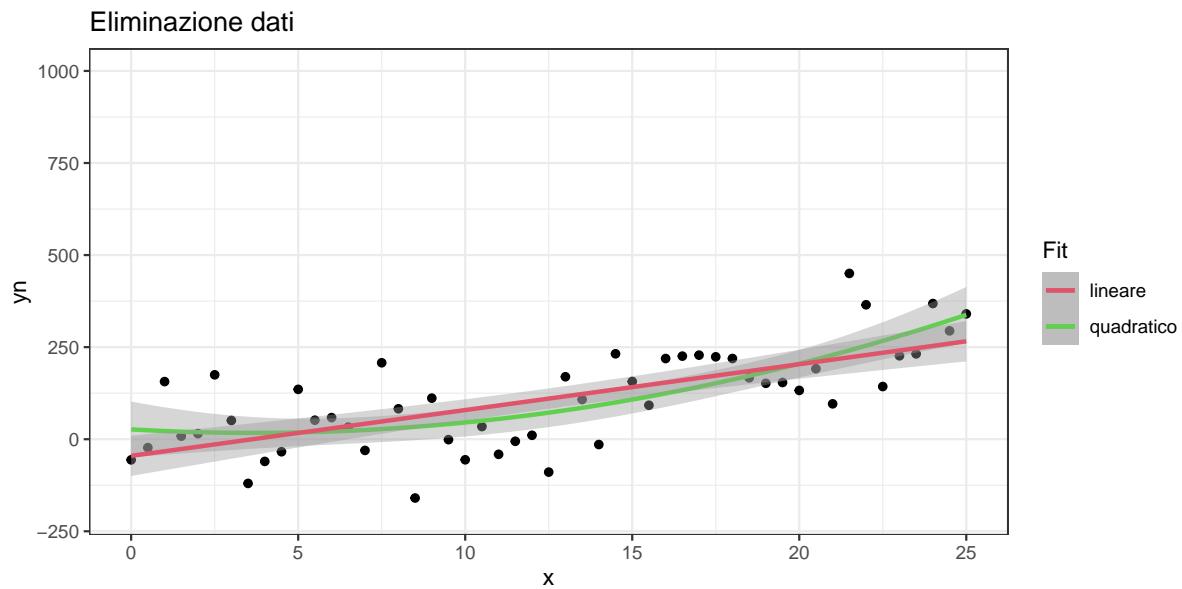
1. *rimuovendo* i dati esterni ad un intervallo: questo modifica il set di dati iniziali, e quindi cambia il comportamento di funzioni come `geom_smooth()`
2. *zoomando* sul grafico, mantenendo intatto il set di dati.

```
set.seed(123)
data <- tibble(
  x=seq(0,100,0.5),
  y=0.5*x^2+0.1*x,
  yn=y+rnorm(length(x), 0, 100)
)

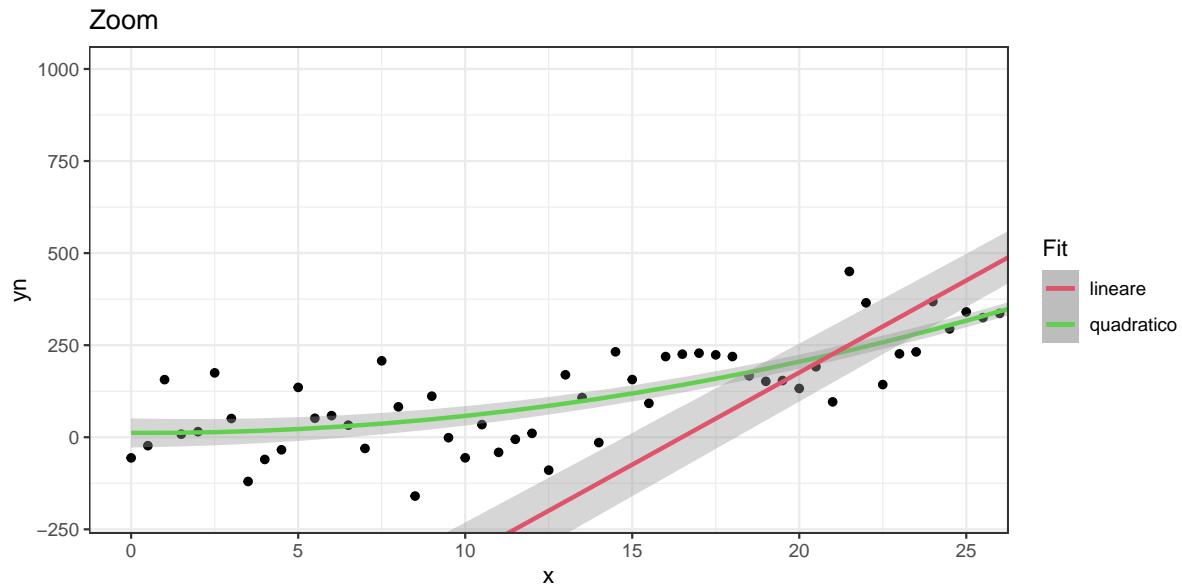
(gp <- data %>% ggplot(aes(x=x, y=yn)) +
  geom_point() +
  geom_smooth(method="lm", formula=y~poly(x, 2)+x, aes(color="quadratico")) +
  geom_smooth(method="lm", formula=y~x, aes(color="lineare")) +
  scale_color_manual(name="Fit", values=c(2, 3)) +
  ggtitle("Set dati completo"))
```



```
gp +
  xlim(c(0, 25)) +
  ylim(c(-200, 1000)) +
  ggtitle("Eliminazione dati")
```



```
gp +
  coord_cartesian(xlim=c(0, 25), ylim=c(-200, 1000)) +
  ggtitle("Zoom")
```



2 Esempi

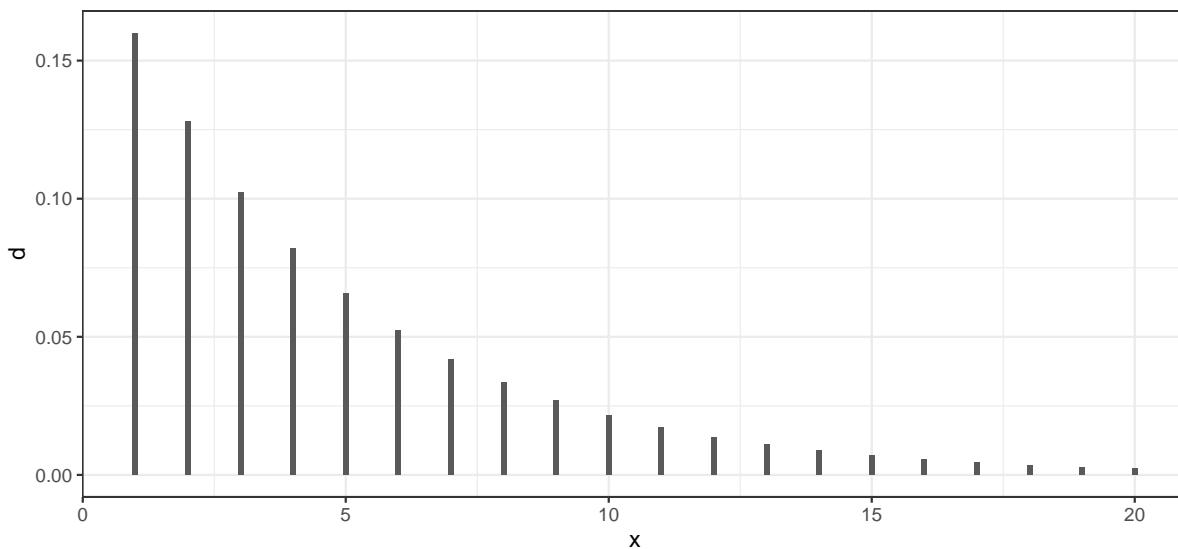
In questo capitolo riprendiamo alcuni concetti espressi nelle parti precedenti e ricreiamo gli stessi grafici in GGplot2.

2.1 Distribuzioni

2.1.1 Distribuzioni discrete

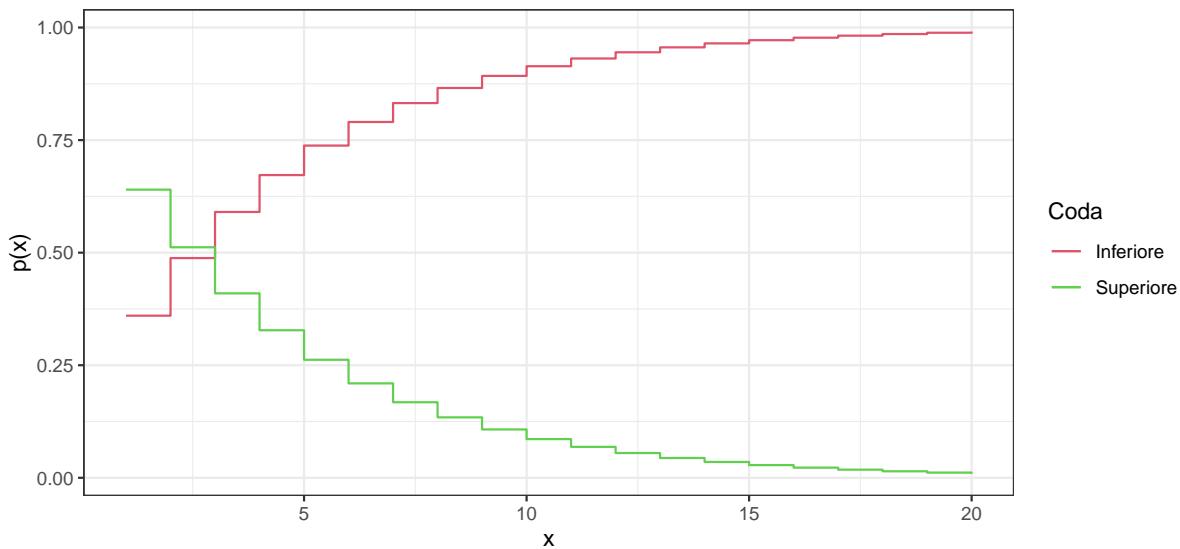
```
df <- tibble(
  x=1:20,
  d=dgeom(x, prob=0.2),
  pl=pgeom(x, prob=0.2, lower.tail = T),
  pu=pgeom(x, prob=0.2, lower.tail = F)
)
ggplot(df, aes(x=x, y=d)) +
  geom_col(width=0.1) +
  labs(title="Densità di distribuzione geometrica", ylab="p(x)")
```

Densità di distribuzione geometrica



```
ggplot(df) +
  geom_step(aes(x=x, y=pl, col="Inferiore")) +
  geom_step(aes(x=x, y=pu, col="Superiore")) +
  scale_color_manual(name="Coda", values=c(2,3)) +
  ggtitle("Probabilità cumulativa geometrica") +
  ylab("p(x)")
```

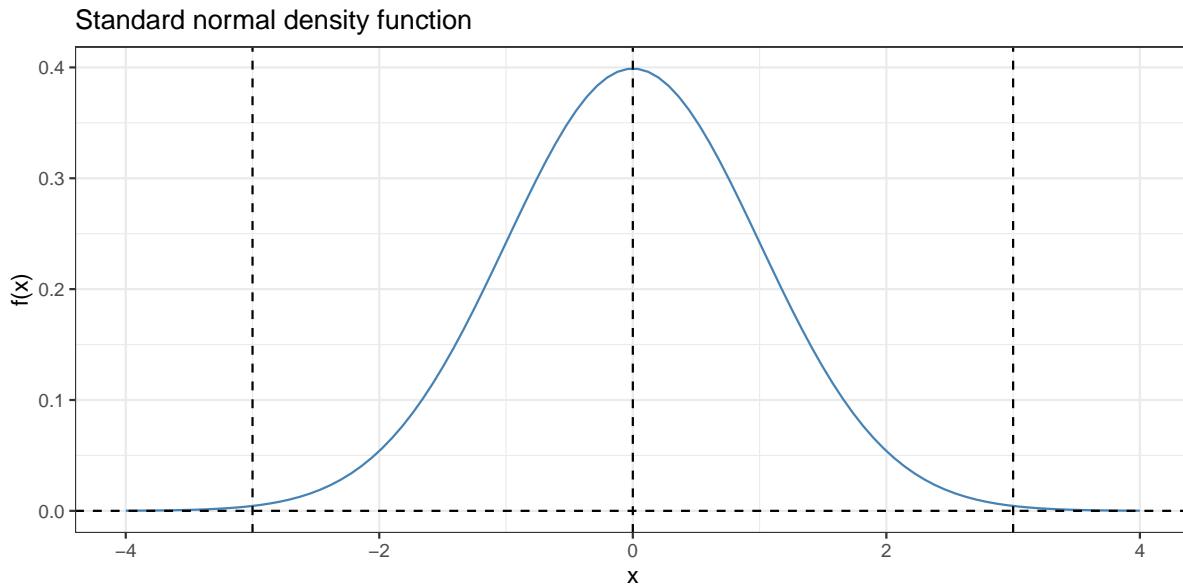
Probabilità cumulativa geometrica



2.1.2 Distribuzioni continue

```
df <- tibble(
  x=seq(-4,4,length.out=100),
  f=dnorm(x, 0, 1)
)
df %>% ggplot(aes(x=x, y=f)) +
  geom_line(color="steelblue") +
  geom_hline(yintercept=0, linetype=2) +
  geom_vline(xintercept=c(-3, 0, 3), linetype=2) +
```

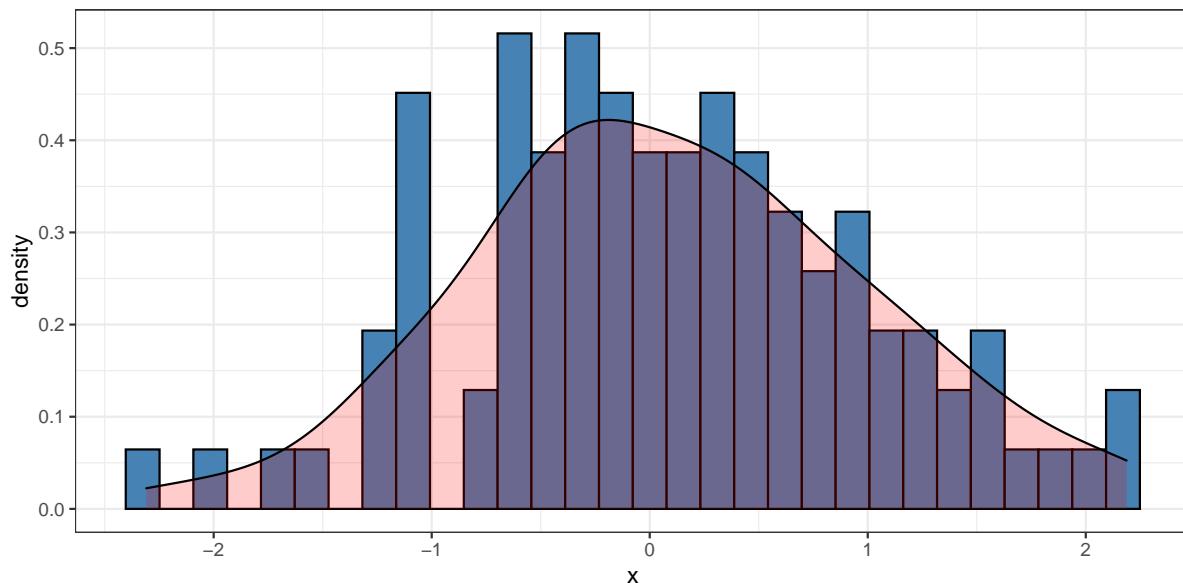
```
ggtitle("Standard normal density function") +
  ylab("f(x)")
```



2.1.3 Istogrammi e QQ-plot

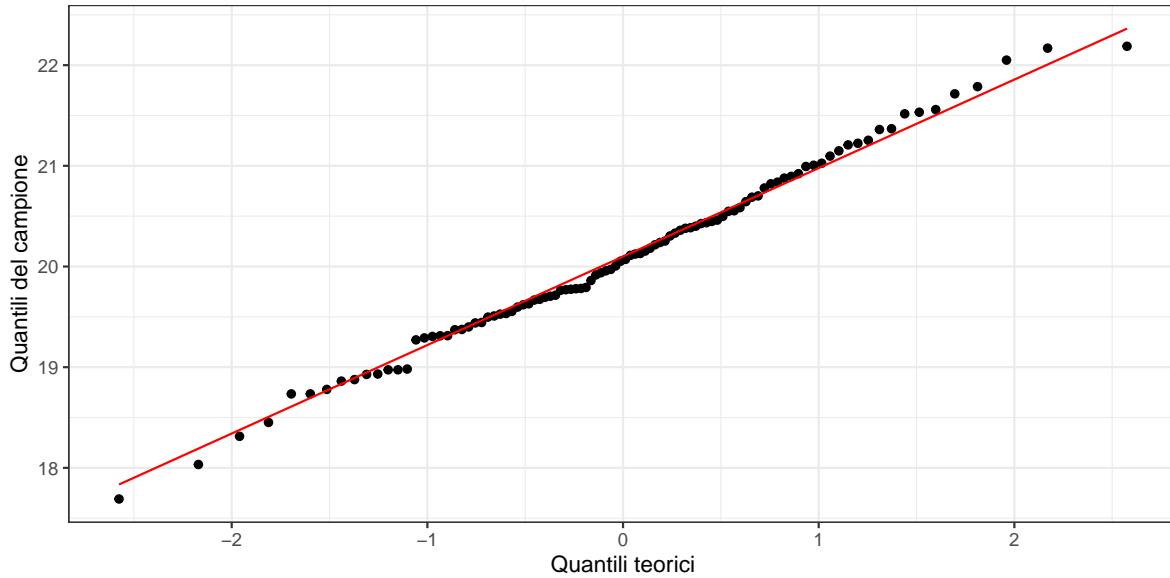
```
set.seed(123)
tibble(
  x=rnorm(100)
) %>%
  ggplot(aes(x=x)) +
  geom_histogram(aes(y=..density..), fill="steelblue", color="black") +
  geom_density(fill="red", alpha=0.2)

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



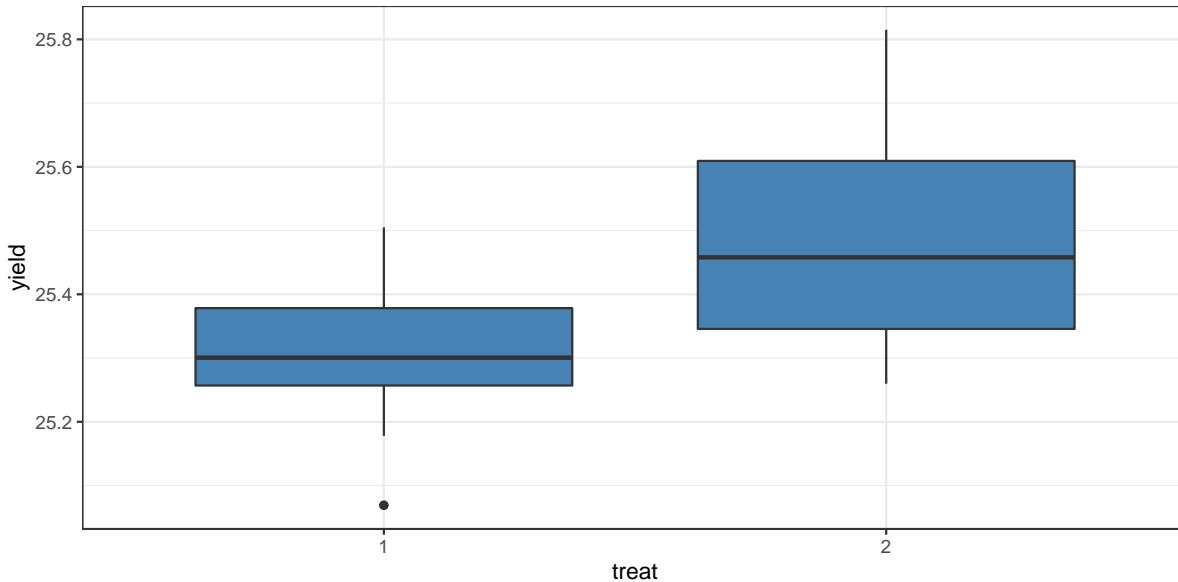
Nota: in `aes()` la notazione `..density..` sta a indicare “applica la funzione `density` ai dati in ingresso”.

```
set.seed(123)
tibble(
  x=rnorm(100, mean=20)
) %>%
ggplot(aes(sample=x)) +
  geom_qq() +
  geom_qq_line(color="red") +
  xlab("Quantili teorici") +
  ylab("Quantili del campione")
```



2.1.4 Boxplot

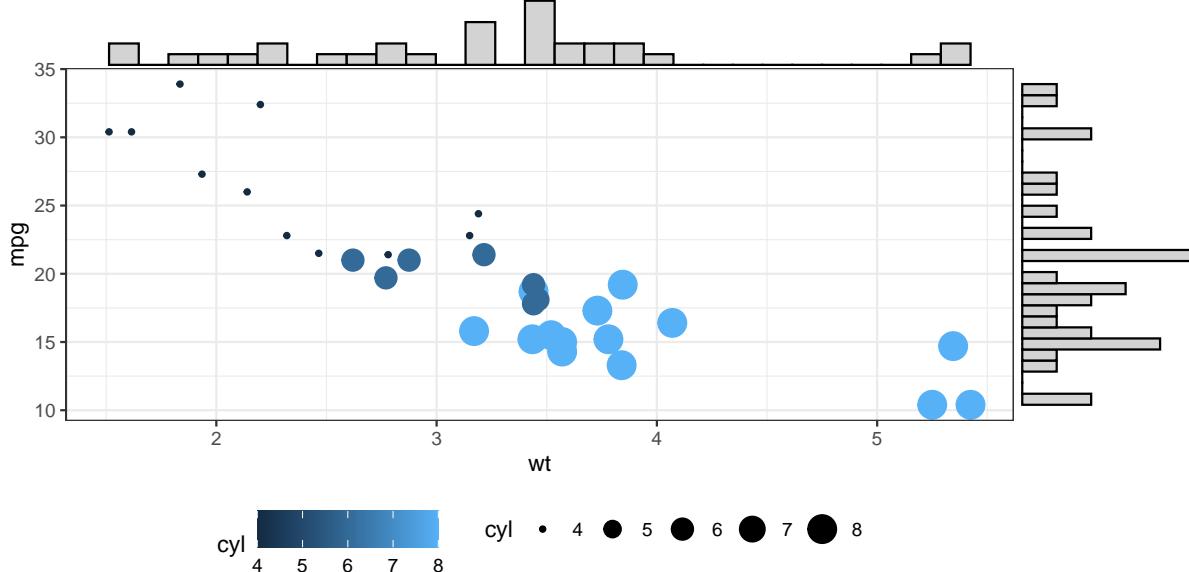
```
df <- read_table(mydata("twosample.dat"), col_types=cols(
  treat=col_factor(),
  yield=col_double()))
df %>% ggplot(aes(x=treat, y=yield)) +
  geom_boxplot(fill="steelblue")
```



2.1.5 histogrammi marginali

```
df <- tibble(mtcars)
gp <- df %>% ggplot(aes(x=wt, y=mpg, color=cyl, size=cyl)) +
  geom_point() +
  theme(legend.position="bottom")

gp %>% ggMarginal(type="histogram", fill="lightgray")
```



2.2 Serie temporali

```
datafile <- "http://repos.dii.unitn.it:8080/data/temperature-anomaly.csv"
data <- read.csv(datafile)
data <- data[data$Entity=="Global",]
t.global <- xts(data$Median.temp,
                  order.by=as.Date(as.character(data$Year), format="%Y"),
                  frequency = 1)

x1 <- t.global["/1999-12-31"]
p1 <- autoplot(x1) +
  geom_hline(yintercept = 0) +
  geom_area(aes(x=index(x1), y=ifelse(x1<0, x1, 0)), fill="blue") +
  geom_area(aes(x=index(x1), y=ifelse(x1>0, x1, 0)), fill="orange") +
  labs(title="Anomalia termica", subtitle = "Terra/mare, globale") +
  xlab("Anno") +
  ylab("Anomalia (°C)")

x2 <- t.global["2000-1-1/"]
p1 + geom_line(data=x2, aes(Index, x2), color="red") +
  #scale_x_continuous(breaks=seq(start(x1), end(x2), by="20 years")) +
  scale_x_date(breaks="10 years", date_labels="%Y") +
  scale_y_continuous(breaks=seq(round(min(t.global), 0.1), round(max(t.global), 0.1), by=0.1))
```

