

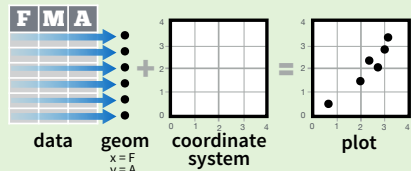
Visualização de Dados com ggplot2

Folha de Referência

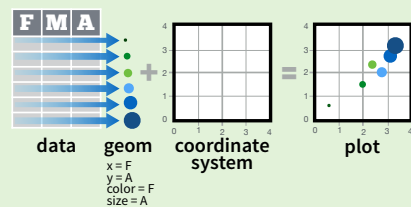


Basics

ggplot2 é baseado na gramática de gráficos, a ideia é que você possa construir qualquer gráfico com alguns poucos componentes básicos: um conjunto de **dados**, um conjunto de **geoms** — marcações visuais que representam pontos de dados, e um **sistema de coordenadas**.



Para mostrar os valores dos dados, as variáveis são mapeadas para propriedades estéticas do geom como **tamanho**, **cor**, e locais **x** e **y**.



Construa um gráfico com **ggplot()** ou **qplot()**.

```
ggplot(data = mpg, aes(x = cty, y = hwy))
```

Começa um gráfico que você termina adicionando camadas a ele. Não tem valores padrões, mas permite maior controle do que qplot().

dados

```
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method = "lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

adiciona camadas com +

camada = geom + stat padrão + mapeamentos específicos da camada

elementos adicionais

Adiciona uma nova camada a um gráfico com a funções **geom_***() ou **stat_***(). Cada um disponibiliza um geom, um conjunto de mapeamentos estéticos, um stat padrão e um ajuste de posição.

mapeamentos estéticos **dados** **geom**

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
Cria um gráfico completo com os dados, geom, e mapeamentos fornecidos. Possui vários valores padrões úteis.









last_plot()
Retorna o último gráfico.

ggsave("plot.png", width = 5, height = 5)
Salva o último gráfico em arquivo 5' x 5' nomeado de "plot.png" no diretório de trabalho. Define o tipo do arquivo pela extensão.

Geoms - Use um geom para representar pontos de dados e suas propriedades estéticas para representar variáveis. Cada função retorna uma camada.






Graphical Primitives

```
a <- ggplot(seals, aes(x = long, y = lat))
b <- ggplot(economics, aes(date, unemploy))
```


-  **a + geom_blank()**
(Útil para expandir os limites)
-  **a + geom_curve(aes(yend = lat + delta_lat, xend = long + delta_long, curvature = z))**
x, xend, y, yend, alpha, angle, color, curvature, linetype, size
-  **b + geom_path(lineend="butt", linejoin="round", linemitre=1)**
x, y, alpha, color, group, linetype, size
-  **b + geom_polygon(aes(group = group))**
x, y, alpha, color, fill, group, linetype, size
-  **a + geom_rect(aes(xmin = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat))**
x, xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
-  **b + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900))**
x, y, alpha, color, fill, group, linetype, size
-  **a + geom_segment(aes(yend=lat + delta_lat, xend = long + delta_long))**
x, xend, y, yend, alpha, color, linetype, size
-  **a + geom_spoke(aes(yend = lat + delta_lat, xend = long + delta_long))**
x, y, angle, radius, alpha, color, linetype, size

Uma variável

Contínua








- ```
c <- ggplot(mpg, aes(hwy))
```
-  **c + geom\_area(stat = "bin")**  
x, y, alpha, color, fill, linetype, size  
a + geom\_area(aes(y = ..density..), stat = "bin")
  -  **c + geom\_density(kernel = "gaussian")**  
x, y, alpha, color, fill, group, linetype, size, weight
  -  **c + geom\_dotplot()**  
x, y, alpha, color, fill
  -  **c + geom\_freqpoly()**  
x, y, alpha, color, group, linetype, size  
a + geom\_freqpoly(aes(y = ..density..))
  -  **c + geom\_histogram(binwidth = 5)**  
x, y, alpha, color, fill, linetype, size, weight  
a + geom\_histogram(aes(y = ..density..))

#### Discreta





- ```
d <- ggplot(mpg, aes(fl))
```
-  **d + geom_bar()**
x, alpha, color, fill, linetype, size, weight

Duas Variáveis


Contínua X, Contínua Y

- ```
e <- ggplot(mpg, aes(cty, hwy))
```
-  **e + geom\_label(aes(label = cty), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE)**  
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
  -  **e + geom\_jitter(height = 2, width = 2)**  
x, y, alpha, color, fill, shape, size
  -  **e + geom\_point()**  
x, y, alpha, color, fill, shape, size, stroke
  -  **e + geom\_quantile()**  
x, y, alpha, color, group, linetype, size, weight
  -  **e + geom\_rug(sides = "bl")**  
x, y, alpha, color, linetype, size
  -  **e + geom\_smooth(method = lm)**  
x, y, alpha, color, fill, group, linetype, size, weight
  -  **e + geom\_text(aes(label = cty), nudge\_x = 1, nudge\_y = 1, check\_overlap = TRUE)**  
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust




#### Discreta X, Contínua Y

- ```
f <- ggplot(mpg, aes(class, hwy))
```
-  **f + geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight
 -  **f + geom_boxplot()**
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight
 -  **f + geom_dotplot(binaxis = "y", stackdir = "center")**
x, y, alpha, color, fill, group
 -  **f + geom_violin(scale = "area")**
x, y, alpha, color, fill, group, linetype, size, weight




Discreta X, Discreta Y

- ```
g <- ggplot(diamonds, aes(cut, color))
```
-  **g + geom\_count()**  
x, y, alpha, color, fill, shape, size, stroke

#### Distribuição Contínua Bivariada





- ```
h <- ggplot(diamonds, aes(carat, price))
```
-  **h + geom_bin2d(binwidth = c(0.25, 500))**
x, y, alpha, color, fill, linetype, size, weight
 -  **h + geom_density2d()**
x, y, alpha, colour, group, linetype, size
 -  **h + geom_hex()**
x, y, alpha, colour, fill, size

Função Contínua

- ```
i <- ggplot(economics, aes(date, unemploy))
```
-  **i + geom\_area()**  
x, y, alpha, color, fill, linetype, size
  -  **i + geom\_line()**  
x, y, alpha, color, group, linetype, size
  -  **i + geom\_step(direction = "hv")**  
x, y, alpha, color, group, linetype, size


#### Visualizando Erros

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```



-  **j + geom\_crossbar(fatten = 2)**  
x, y, ymax, ymin, alpha, color, fill, group, linetype, size
-  **j + geom\_errorbar()**  
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom\_errorbarh()**)
-  **j + geom\_linerange()**  
x, ymin, ymax, alpha, color, group, linetype, size
-  **j + geom\_pointrange()**  
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

#### Mapas

```
data <- data.frame(murder = USArrests$Murder,
 state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

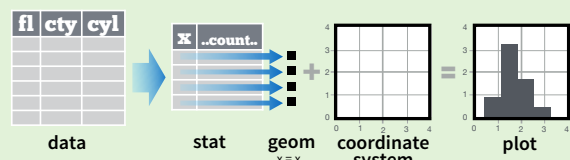
-  **k + geom\_map(aes(map\_id = state), map = map) + expand\_limits(x = map\$long, y = map\$lat)**  
map\_id, alpha, color, fill, linetype, size

### Três Variáveis

- ```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))
```
-  **l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)**
x, y, alpha, fill
 -  **l + geom_tile(aes(fill = z))**
x, y, alpha, color, fill, linetype, size, width, weight

Stats - An alternative way to build a layer

Alguns gráficos realizam uma **transformação** nos dados originais. Use um **stat** para escolher uma transformação comum para visualizar, e.g. **a + geom_bar(stat = "count")**



Cada stat cria variáveis adicionais que são mapeadas para a estética. Essas variáveis usam uma sintaxe comum **..name..**

Ambas as funções stat e geom combinam um stat com um geom para criar uma camada, i.e. **stat_count(geom="bar")** faz o mesmo que **geom_bar(stat="count")**

função stat **mapeamento de camadas**

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)

geom da camada **parâmetros do stat** **variável criada pela transformação**

c + stat_bin(binwidth = 1, origin = 10) **Distribuições 1D**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
c + stat_count(width = 1)
x, y, l | ..count.., ..prop..
c + stat_density(adjust = 1, kernel = "gaussian")
x, y, l | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = TRUE) **Distribuições 2D**
x, y, fill | ..count.., ..density..
e + stat_bin_hex(bins = 30)
x, y, fill | ..count.., ..density..
e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..
e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **3 Variáveis**
x, y, z, order | ..level..
l + stat_summary_hex(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **Comparações**
x, y | ..lower.., ..middle.., ..upper.., ..width.., ..ymin.., ..ymax..
f + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **Funções**
x, y | ..x.., ..y..
e + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile..
e + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)) **Propósito Geral**
x | ..x.., ..y..
e + stat_identity(na.rm = TRUE)
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..sample.., ..theoretical..
e + stat_sum()
x, y, size | ..n.., ..prop..
e + stat_summary(fun.data = "mean_cl_boot")
h + stat_summary_bin(fun.y = "mean", geom = "bar")
e + stat_unique()

Escalas

As **Escalas** controlam como um gráfico mapeia os valores dos dados para os valores visuais de uma estética. Para mudar o mapeamento, adicione uma escala personalizada.

n <- b + geom_bar(aes(fill = fl))
n

scale_ **estética para ajustar** **escala pré-definida** **argumentos específicos**

n + scale_fill_manual(values = c("skyblue", "royalblue", "blue", "navy"), limits = c("d", "e", "p", "r"), breaks = c("d", "e", "p", "r"), name = "fuel", labels = c("D", "E", "P", "R"))

intervalo de valores para incluir no mapeamento **título para usar na legenda/eixo** **rótulo para usar na legenda/eixo** **breaks para usar na legenda/eixo**

Escalas de Propósito Geral
Use com qualquer estética: **alpha, color, fill, linetype, shape, size**

scale_*_continuous() - mapeia valores contínuos para visuais
scale_*_discrete() - mapeia valores discretos para visuais
scale_*_identity() - usa valores dos dados **como** visuais
scale_*_manual(values = c()) - mapeia valores discretos para valores visuais manualmente escolhidos

Escala de local de X e Y
Use com a estética **x** ou **y** (**x** exposto aqui)

scale_x_date(date_labels = "%m/%d", date_breaks = "2 weeks") - trata os valores de x como datas. Ver ?strptime para o formatos.
scale_x_datetime() - trata os valores de x como data e hora. Usa os mesmos argumentos que **scale_x_date()**.
scale_x_log10() - Transforma x para a escala log10.
scale_x_reverse() - Inverte a direção do eixo x.
scale_x_sqrt() - Transforma x para a escala da raiz quadrada de x.

Escalas de Cor e Preenchimento

Discreta **Contínua**

n <- d + geom_bar(aes(fill = fl))
n + scale_fill_brewer(palette = "Blues")
Para opções de paleta: **library(RColorBrewer)** **display.brewer.all()**

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

o <- c + geom_dotplot(aes(fill = ..x..))
o + scale_fill_gradient(low = "red", high = "yellow")
o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)
o + scale_fill_gradientn(colours = terrain.colors(6))

Também: **rainbow()**, **heat.colors()**, **topo.colors()**, **cm.colors()**, **RColorBrewer::brewer.pal()**

Escalas de Formato

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape(solid = FALSE)
p + scale_shape_manual(values = c(3:7))
Valores de formato expostos no gráfico à direita

Manual shape values

0	6	12	18	24
1	7	13	19	25
2	8	14	20	*
3	9	15	21	
4	10	16	22	o
5	11	17	23	o

Escalas de Tamanho

p + scale_radius(range=c(1,6))
p + scale_size(max_scale=6)
Mapeia para área do círculo (não raios)

Sistema de Coordenadas

r <- d + geom_bar()
r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
Sistema de coordenadas padrão

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Sistema de coordenadas com proporção fixa entre as unidades de x e y.

r + coord_flip()
xlim, ylim
Coordenadas cartesianas invertidas

r + coord_polar(theta="x", direction=1)
theta, start, direction
Coordenadas polares

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Coordenadas cartesianas transformadas. Define xtrans e ytrans para o nome da função de janelamento.

π + coord_map(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Mapeia projeções do pacote mapproj (mercator (padrão), azequialarea, lagrange, etc.)

Ajustes de Posição

Ajustes de posição definem como os geoms se localizam, evitando que ocupem o mesmo espaço.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Coloca os elementos lado a lado.

s + geom_bar(position = "fill")
Empilha os elementos um sobre o outro, normaliza a altura.

e + geom_point(position = "jitter")
Adiciona um ruído aleatório para as posições X e Y de cada elemento evitando a sobreposição.

e + geom_label(position = "nudge")
Afasta os rótulos dos pontos.

s + geom_bar(position = "stack")
Empilha os elementos um sobre o outro.

Cada ajuste de posição pode ser redefinido como um ajuste de posição manual dos argumentos **width** e **height**.

s + geom_bar(position = position_dodge(width = 1))

Temas

r + theme_bw()
Fundo branco com linhas em grande

r + theme_classic()
r + theme_light()
r + theme_linedraw()
Temas mínimos

r + theme_gray()
fundo cinza (tema padrão)

r + theme_minimal()
r + theme_void()
Temas vazios

r + theme_dark()
escuro para contraste

Facetas

Facetas dividem um gráfico em sub-gráficos baseando-se em uma ou mais variáveis discretas.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
divide em colunas baseado em fl

t + facet_grid(year ~ .)
divide em linhas baseado em year

t + facet_grid(year ~ fl)
divide em linhas e colunas

t + facet_wrap(~ fl)
ajusta as facetas em um formato retangular

Defina **scales** para que os limites variem entre os eixos

t + facet_grid(drv ~ fl, scales = "free")
Ajusta os limites dos eixos x e y para facetas individuais

- **"free_x"** - ajusta os limites do eixo x
- **"free_y"** - ajusta os limites do eixo y

Defina **labeller** para ajustar os rótulos das facetas

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r
-------	-------	-------	-------	-------

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))

			α^p	α^r
--	--	--	------------	------------

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r
---	---	---	---	---

Rótulos

t + ggtitle("New Plot Title")
Adiciona um título principal sobre o gráfico

t + xlab("New X label")
Muda o rótulo do eixo X

t + ylab("New Y label")
Muda o rótulo do eixo Y

t + labs(title = "New title", x = "New x", y = "New y")
Todos acima

Use funções **scale** para atualizar o rótulo das legendas

Legendas

n + theme(legend.position = "bottom")
Coloca a legenda no "bottom", "top", "left", ou "right"

n + guides(fill = "none")
Define o tipo da legenda para cada estética: colorbar, legend, ou none (sem legenda)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Define o título da legenda e os rótulos com uma função **scale**.

Zoom

Sem recorte (preferido)

t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))

Com recorte (remove dados não visualizados)

t + xlim(0, 100) + ylim(10, 20)
t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))