

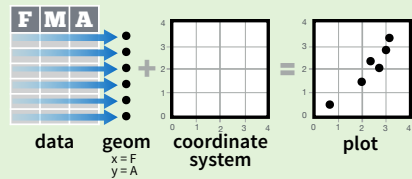
Visualización de Datos usando ggplot2

Guía Rápida

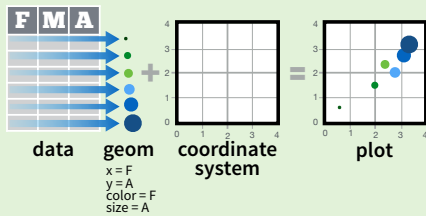


Conceptos Básicos

ggplot2 se basa en la idea que cualquier gráfica se puede construir usando estos tres componentes: **datos**, **coordenadas** y **objetos geométricos (geoms)**. Este concepto se llama: **gramática de las gráficas**.



Para visualizar resultados, asigne variables a las propiedades visuales, o **estéticas**, como **tamaño**, **color** y **posición x** o **y**.



Para construir una gráfica complete este patrón

```
ggplot(data = <DATOS>) +  
  <FUNCION_GEOM> (  
    mapping =aes( <ESTETICAS> ),  
    stat = <STAT> ,  
    position = <POSICION>  
  ) +  
  <FUNCION_COORDINADAS> +  
  <FUNCION_FACETA> +  
  <FUNCION_ESCALA> +  
  <FUNCION_TEMA>
```

Requerido

No
Requerido,
se proveen
valores
iniciales

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

Este comando comienza una gráfica, complétela mediante agregando capas, un **geom** por capa.

estéticas

datos

geom

qplot(x = cty, y = hwy, data = mpg, geom = "point")

Este comando es una gráfica completa, tiene datos, las estéticas están asignadas y por lo menos un geom.

last_plot()

Devuelve la última gráfica

ggsave("plot.png", width = 5, height = 5)

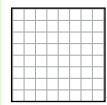
La última gráfica es grabada como una imagen de 5 por 5 pulgs., usa el mismo tipo de archivo que la extensión

Geoms - Funciones geom se utilizan para visualizar resultados. Asigne variables a las propiedades estéticas del geom. Cada geom forma una capa.

Geométricas Elementales

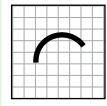
a <- ggplot(economics, aes(date, unemploy))

b <- ggplot(seals, aes(x = long, y = lat))

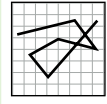


a + geom_blank()

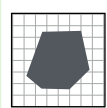
(Bueno para expandir límites)



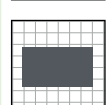
b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size



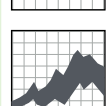
a + geom_path(lineend="butt", linejoin="round", linemitre=1) x, y, alpha, color, group, linetype, size



a + geom_polygon(aes(group = group)) x, y, alpha, color, fill, group, linetype, size



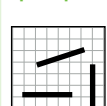
b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size



a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin alpha, color, fill, group, linetype, size

Segmentos Lineares

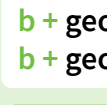
propiedades básicas: x, y, alpha, color, linetype, size



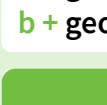
b + geom_abline(aes(intercept=0, slope=1))

b + geom_hline(aes(yintercept = lat))

b + geom_vline(aes(xintercept = long))



b + geom_segment(aes(yend=lat+1, xend=long+1))

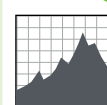


b + geom_spoke(aes(angle = 1:1155, radius = 1))

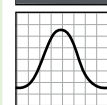
Una Variable

Continua

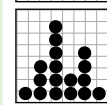
c <- ggplot(mpg, aes(hwy)); **c2** <- ggplot(mpg)



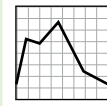
c + geom_area(stat = "bin") x, y, alpha, color, fill, linetype, size



c + geom_density(kernel = "gaussian") x, y, alpha, color, fill, group, linetype, size, weight



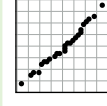
c + geom_dotplot() x, y, alpha, color, fill



c + geom_freqpoly() x, y, alpha, color, group, linetype, size



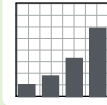
c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight



c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

Discreta

d <- ggplot(mpg, aes(fl))

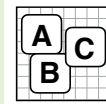


d + geom_bar() x, alpha, color, fill, linetype, size, weight

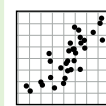
Dos Variables

X Continua, Y Continua

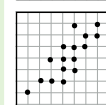
e <- ggplot(mpg, aes(cty, hwy))



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



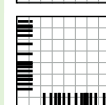
e + geom_jitter(height = 2, width = 2) x, y, alpha, color, fill, shape, size



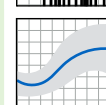
e + geom_point() x, y, alpha, color, fill, shape, size, stroke



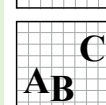
e + geom_quantile() x, y, alpha, color, group, linetype, size, weight



e + geom_rug(sides = "bl") x, y, alpha, color, linetype, size



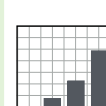
e + geom_smooth(method = lm) x, y, alpha, color, fill, group, linetype, size, weight



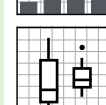
e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

X Discreta, Y Continua

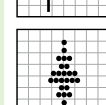
f <- ggplot(mpg, aes(class, hwy))



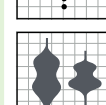
f + geom_col() x, y, alpha, color, fill, group, linetype, size



f + geom_boxplot() x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight



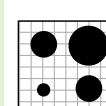
f + geom_dotplot(binaxis = "y", stackdir = "center") x, y, alpha, color, fill, group



f + geom_violin(scale = "area") x, y, alpha, color, fill, group, linetype, size, weight

X Discreta, Y Discreta

g <- ggplot(diamonds, aes(cut, color))

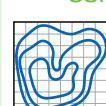


g + geom_count() x, y, alpha, color, fill, shape, size, stroke

Trés Variables

seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))

l <- ggplot(seals, aes(long, lat))



l + geom_contour(aes(z = z)) x, y, z, alpha, colour, group, linetype, size, weight



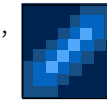
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE) x, y, alpha, fill



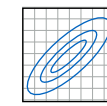
l + geom_tile(aes(fill = z)) x, y, alpha, color, fill, linetype, size, width

Distribución Bivariada Continua

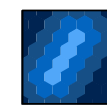
h <- ggplot(diamonds, aes(carat, price))



h + geom_bin2d(binwidth = c(0.25, 500)) x, y, alpha, color, fill, linetype, size, weight



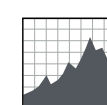
h + geom_density2d() x, y, alpha, colour, group, linetype, size



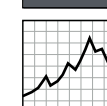
h + geom_hex() x, y, alpha, colour, fill, size

Función Continua

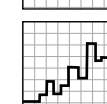
i <- ggplot(economics, aes(date, unemploy))



i + geom_area() x, y, alpha, color, fill, linetype, size



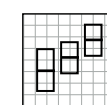
i + geom_line() x, y, alpha, color, group, linetype, size



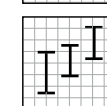
i + geom_step(direction = "hv") x, y, alpha, color, group, linetype, size

Visualizando el Error

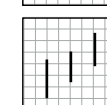
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))



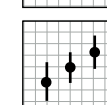
j + geom_crossbar(fatten = 2) x, y, ymax, ymin, alpha, color, fill, group, linetype, size



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

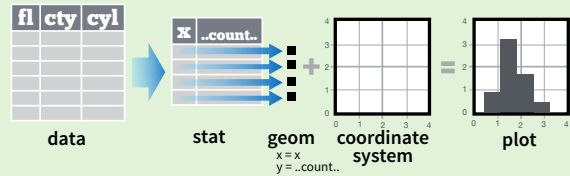
Argumentos

Traducción de argumentos comunes

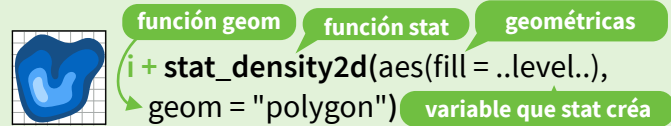
label = etiqueta, angle=ángulo
size=tamaño, weight=peso
alpha=transparencia
fontface=tipo de letra
hjust=ajuste horizontal
lineheight=grosor de línea

Stats - Otra manera de construir una capa

Stat crea nuevas variables para la gráfica, como `count`



Cambie el Stat que la función Geom usa para visualizarla, así: `geom_bar(stat="count")`. También puede usar la función Stat, así: `stat_count(geom="bar")` que igual como una función Geom, esta función también crea una capa.



Distribución Unidimensional

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

Distribución Bidimensional

- `e + stat_bin_2d(bins = 30, drop = T)` 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")`

3 Variables

- `l + stat_contour(aes(z = z))` x, y, z, order | ..level..
- `l + stat_summary_hex(aes(z = z), bins = 30, fun = max)` x, y, z, fill | ..value..
- `l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)` x, y, z, fill | ..value..

Comparativas

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

Funciones

- `e + stat_ecdf(n = 40)` x, y | ..x.., ..y..
- `e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq")` x, y | ..quantile..
- `e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95)` x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

Todo Uso

- `ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5))` x | ..x.., ..y..
- `e + stat_identity(na.rm = TRUE)`
- `ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=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

Las **escalas** asignan los valores que hay en los datos a los valores visuales de una estética.



Escalas para todo uso
Uselas con la mayoría de las estéticas

- `scale_*_continuous()` - asigna valores continuos a visuales
- `scale_*_discrete()` - asigna valores discretos a visuales
- `scale_*_identity()` - crea una estética visual por cada valor
- `scale_*_manual(values = c())` - asigna valores específicos a valores visuales escogidos manualmente.
- `scale_*_date(date_labels = "%m/%d")`, `date_breaks = "2 weeks"` - Usa los valores como fechas
- `scale_*_datetime()` - Usa los valores como fecha-horas Igual que `scale_*_date` pero usando `strptime`

Escalas de localización para X e Y
Use con las estéticas x e y (aquí se muestra x)

- `scale_x_log10()` - Usa escala logarítmica base 10
- `scale_x_reverse()` - Posiciona x al revés
- `scale_x_sqrt()` - Usa escala raíz cuadrada

Escalas para Color y Relleno (Discretas)
`n <- d + geom_bar(aes(fill = fl))`

- `n + scale_fill_brewer(palette = "Blues")`
Ver opciones de colores: `RColorBrewer::display.brewer.all()`
- `n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`

Escalas para Color y Relleno (Continuas)
`o <- c + geom_dotplot(aes(fill = ..x..))`

- `o + scale_fill_distiller(palette = "Blues")`
- `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=topo.colors(6))`
También: `rainbow()`, `heat.colors()`, `terrain.colors()`, `cm.colors()`, `RColorBrewer::brewer.pal()`

Escalas que usan tamaño y figuras
`p <- e + geom_point(aes(shape = fl, size = cyl))`

- `p + scale_shape() + scale_size()`
- `p + scale_shape_manual(values = c(3:7))`
- `p + scale_radius(range = c(1,6))`
Usa el radio o el area
- `p + scale_size_area(max_size = 6)`

Sistema de Coordenadas

- `r <- d + geom_bar()`
- `r + coord_cartesian(xlim = c(0, 5))`
xlim, ylim
Usa coordenadas cartesianas
- `r + coord_fixed(ratio = 1/2)`
ratio, xlim, ylim
Se fija la relación de aspecto
- `r + coord_flip()`
xlim, ylim
Las coordenadas son volteadas
- `r + coord_polar(theta = "x", direction=1)`
theta, start, direction
Coordenadas polares
- `r + coord_trans(ytrans = "sqrt")`
xtrans, ytrans, limx, limy
xtrans e ytrans se asignan a funciones ventanas para transformar las coordenadas cartesianas

- `π + coord_quickmap()`
- `π + coord_map(projection = "ortho", orientation=c(41, -74, 0))`
projection, orientation, xlim, ylim
Usa el paquete mapproj para proyectar mapas

Ajustes a las posiciones

Determina que hacer con Geoms que ocuparían la misma posición en la gráfica.

- `s <- ggplot(mpg, aes(fl, fill = drv))`
- `s + geom_bar(position = "dodge")`
Pone los elementos a lado de cada uno
- `s + geom_bar(position = "fill")`
Pone los elementos encima the cada uno y usa toda la altura de la gráfica
- `e + geom_point(position = "jitter")`
Agrega ruido a los elementos
- `e + geom_label(position = "nudge")`
Empuja las letras para ver los puntos
- `s + geom_bar(position = "stack")`
Pone los elementos encima the cada uno

Cada ajuste se puede usar como función para fijar el ancho and alto

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

Tema

- `r + theme_bw()`
Fondo blanco con cuadrícula
- `r + theme_classic()`
Fondo gris (tema inicial)
- `r + theme_light()`
Temas minimalísticos
- `r + theme_linedraw()`
Tema vacío
- `r + theme_minimal()`
- `r + theme_void()`
- `r + theme_dark()`
Obscuro

Facetas

Las Facetas dividen una gráfica en multiple sub-gráficas basada en una ó varias variables discretas

- `t <- ggplot(mpg, aes(cty, hwy)) + geom_point()`
- `t + facet_grid(. ~ fl)`
usa fl para dividir en columnas
- `t + facet_grid(year ~ .)`
usa year para dividir en líneas
- `t + facet_grid(year ~ fl)`
usa los dos para dividir
- `t + facet_wrap(~ fl)`
divide en una manera rectangular

Use **scales** para que dejar que el límite cambie por cada faceta

- `t + facet_grid(drv ~ fl, scales = "free")`
Cada faceta tiene limites x e y independientes
- **"free_x"** - ajusta el límite del eje x
- **"free_y"** - ajusta el límite del eje y

Use **labeller** para cambiar las etiquetas de las facetas

- `t + facet_grid(. ~ fl, labeller = label_both)`
- `t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)))`
- `t + facet_grid(. ~ fl, labeller = label_parsed)`

Etiquetas

- `t + labs(x = "Etiqueta X", y = "Etiqueta Y", title = "Título de la gráfica", subtitle = "Subtítulo de la gráfica", caption = "Nota de la gráfica", <AES> = "Texto in la <AES>")`
- `t + annotate(geom = "text", x = 8, y = 9, label = "A")`
Anotaciones
- geom a usar
- valores manuales del geom

Leyendas

- `n + theme(legend.position = "bottom")`
Pone la leyenda debajo (bottom), arriba(top), izquierda (left), ó derecha (right)
- `n + guides(fill = "none")`
Tipo de leyenda por cada estética : colorbar, legend, or none (no legend)
- `n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))`
Fija el título y etiquetas de la leyenda

Agrandar una sección

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