Clubes de Ciencia
México

GDL 7

A tu edad no tenía tu edad: relojes epigenéticos y envejecimiento

At your age I wasn't your age: epigenetic clocks and aging

Instructores: Erick Navarro & Guadalupe Ayala

6-12 de Julio 2025
Guadalajara,México

Envejecimiento y sus características

Deterioro funcional de los organismos con el envejecimiento

Inestabilidad del ADN, desgaste de los telómeros , alteraciones epigenéticas, pérdida de proteostasis , detección de nutrientes desregulada, disfunción mitocondrial, senescencia celular, agotamiento de células madre y comunicación intercelular alterada.

Principalmente la metilación del DNA

Biological age is a concept that reflects the physiological state of an individual rather than the chronological time since birth.

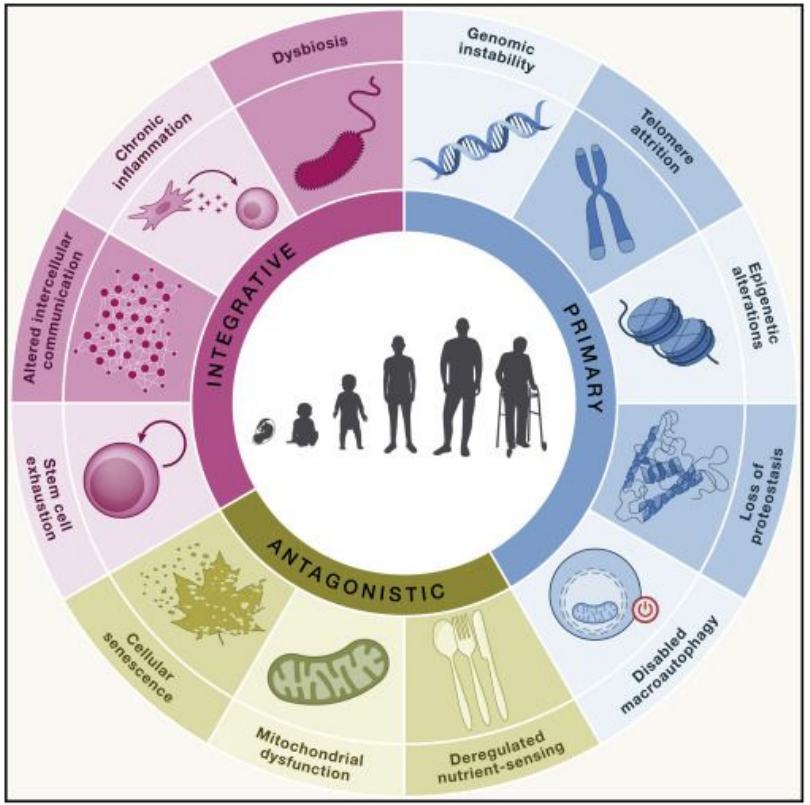


Figure from: López-Otín et al., 2023. *Cell*

Enfermedades neurodegenerativas

Pérdida de proteostasis

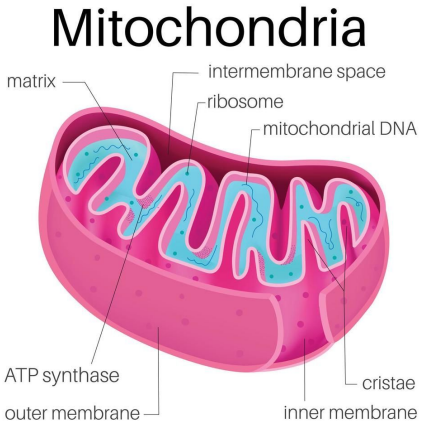
Se pierde las plegamiento de las proteínas y abundancia originales

El plegamiento incorrecto de proteínas ocurre en el cerebro y el músculo de los pacientes con Alzheimer

Disfunción mitocondrial

Especies reactivas de oxígeno

Detección de nutrientes desregularizada



Generación de energía ATP

Senescencia celular

Las células que antes se replicaban vigorosamente, pero que ahora han entrado en un estado permanente de no división, se denominan células senescentes.

Inestabilidad genómica

El genoma está en constante proceso de reparación, pero algunos daños se van acumulando con el tiempo.

Desgaste de los telómeros

Acortamiento de los telómeros

Telómeros estructuras que protegen el DNA

Comunicación intercelular alterada

Inflamación

Agotamiento de células madre

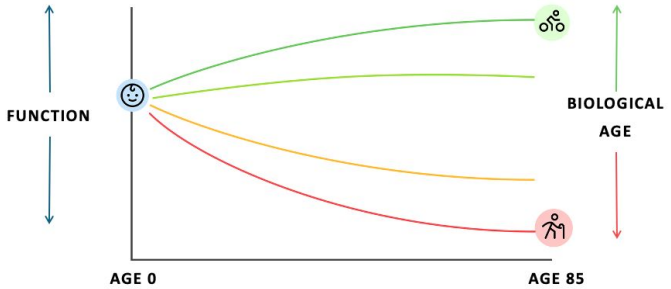
Pérdida de capacidad de autorrenovarse

Edad Biológica

Estado funcional de un individuo en referencia
La edad biológica puede ser mayor o menor que la edad cronológica, dependiendo de factores como la genética, el estilo de vida, el entorno y la exposición a factores de estrés.

Edad cronológica

Es la edad que determinamos habitualmente con base en el calendario, es decir, cuántos años han pasado desde que nacimos.
No tiene en cuenta necesariamente el estado de salud o el envejecimiento celular.

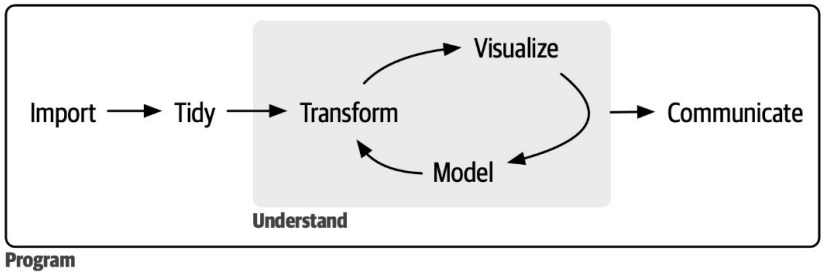


Tidyverse

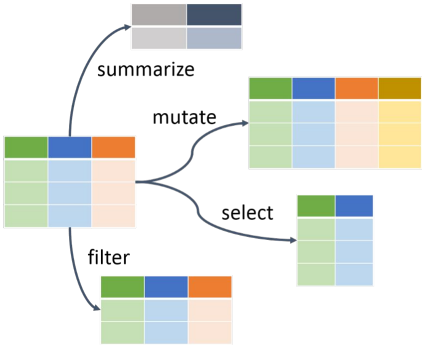
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```



Dplyr



Some of the main "verbs" of the dplyr package are:

- select: select variables (columns) based on their names.

- filter: filter, select observations (rows) based on their values

- mutate: add new variables/columns or transform existing variables

- arrange: sorts the values with respect to a column

1.The first argument is a data set (dataframe).

2. Subsequent arguments describe what to do with the data frame, using the variable names (without quotes).

3. The result is a new dataframe.

Data transformation with dplyr : : **CHEATSHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

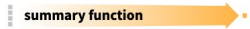
Each **observation**, or **case**, is in its own **row**

pipes

x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



summarize(.data, …**)** Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

ungroup(x, …**)** Returns ungrouped copy of table.

```
g_mtcars <- mtcars |> group_by(cyl)
ungroup(g_mtcars)
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
mtcars |> slice(10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
cars |> add_row(speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …**)** Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g., mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	**!**, e.g., !gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

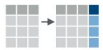
across(.cols, .funs, …, .names = NULL**)** Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.

```
df |>
  rowwise() |>
  mutate(x_total = sum(c_across(1:2)))
```

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).



mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column()**.
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(gpm = 1 / mpg, .keep = "none")

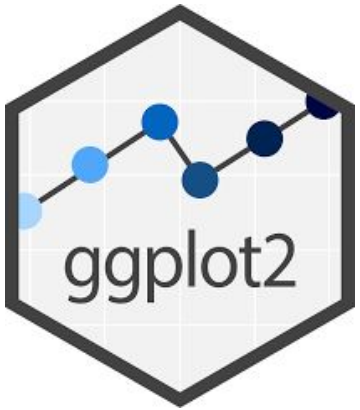
rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
mtcars |> rename(miles_per_gallon = mpg)

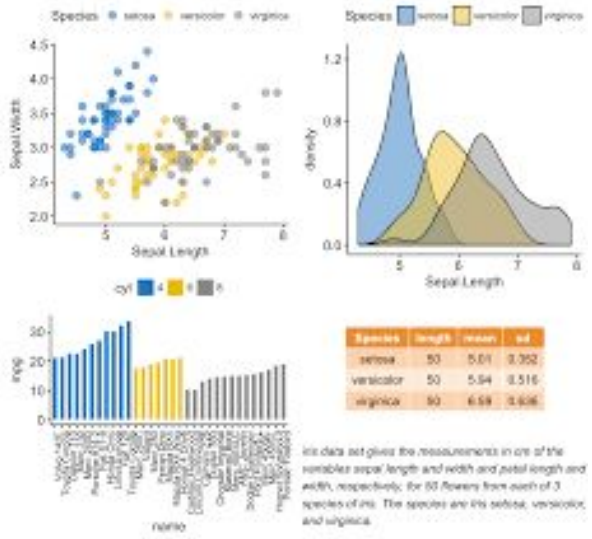
Pipe

One of the characteristics of tidyverse is the pipe operator, which is written as %>% and we can read it as “then” or “then”. The pipe allows to string together different code fragments and operations into a readable block of code. Since everything is executed sequentially, the operations are read as if were a written sentence.

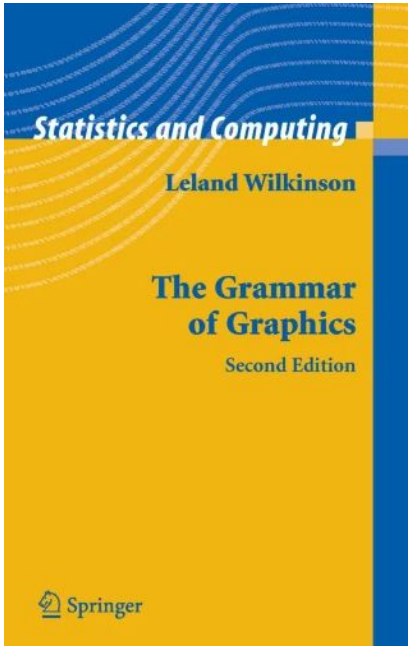
Ggplot2

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.





The Grammar of Graphics



In the same way that a grammar defines the regular structures and composition of a language, his book outlines a framework to structure statistical graphics.

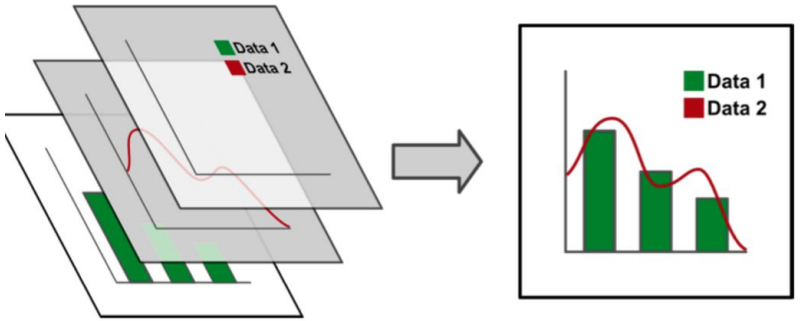
Its influence can be found in Tableau, Plotly, and the Python libraries bokeh, altair, seaborn, and plotnine. The most complete implementation of the grammar is found in an R package called ggplot2 by Hadley Wickham.

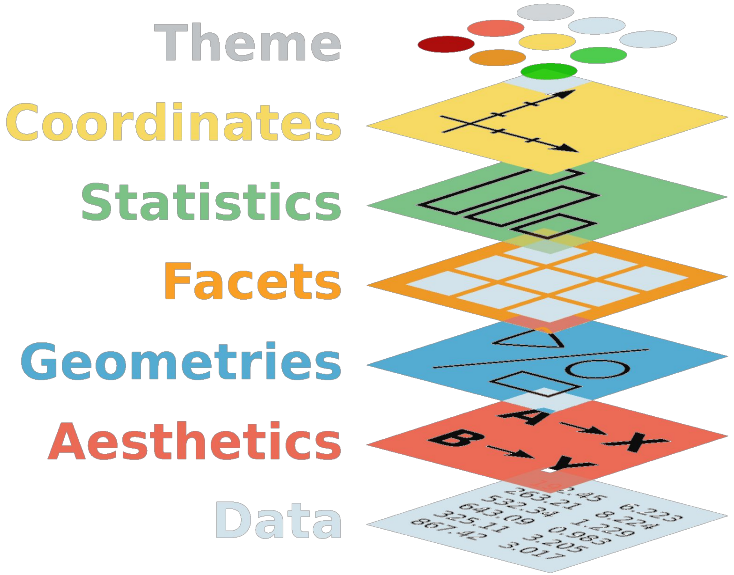
In Wickham's adaptation of the grammar of graphics, a plot can be decomposed into three primary elements:

1. the data,
2. the aesthetic mapping of the variables in the data to visual cues, and
3. the geometry used to encode the observations on the plot.

Painter’s model

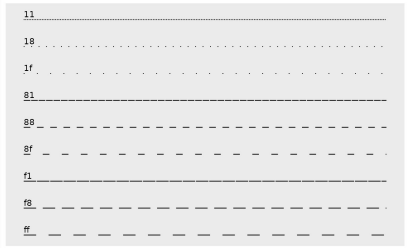
The default graphics system that comes with R, often called base R graphics is simple and fast. It is based on the painter’s model or canvas, where different output are directly overlaid on top of each other.

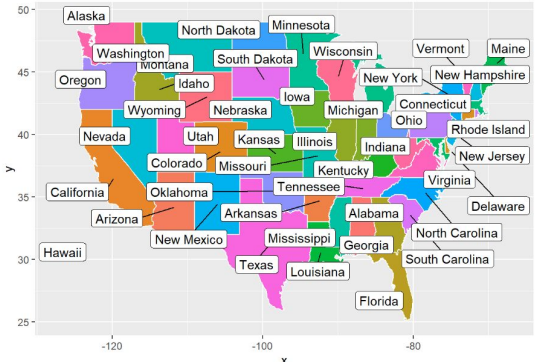




Aesthetic specifications

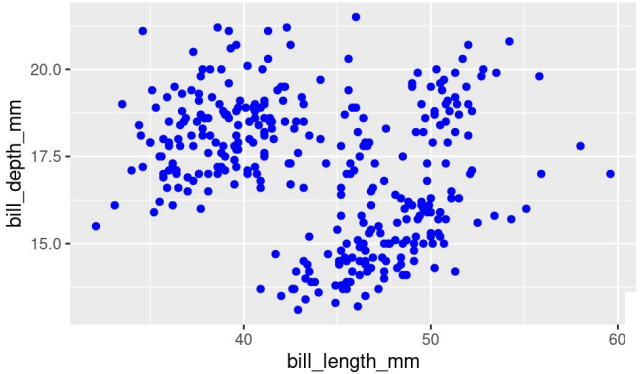
Visual property of objects in your graphics (aes)



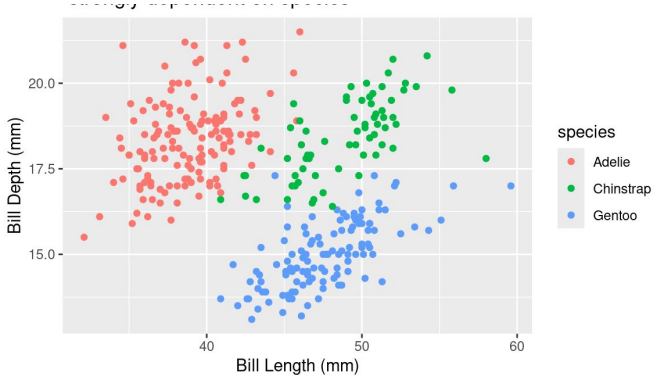


- color
- transparency
- shape
- size

Levels



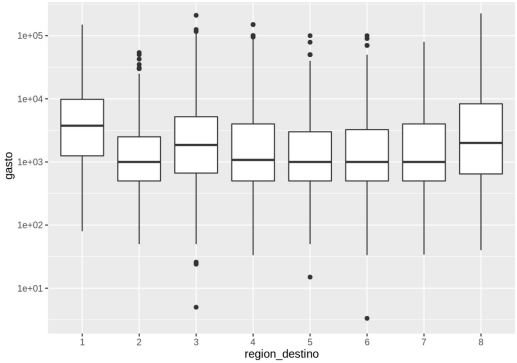
- constant
- associated with variable

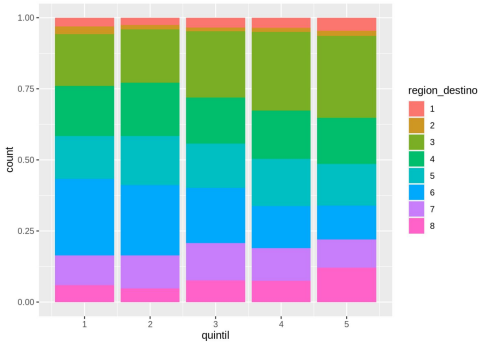


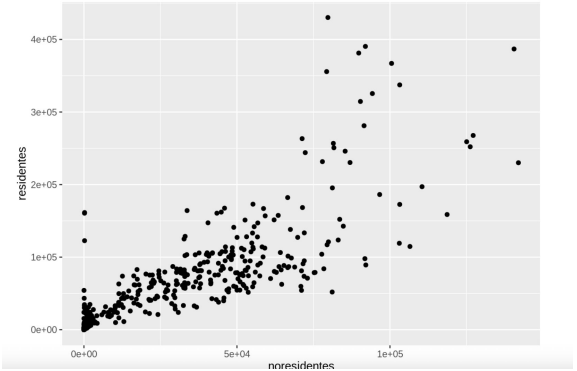
Geoms

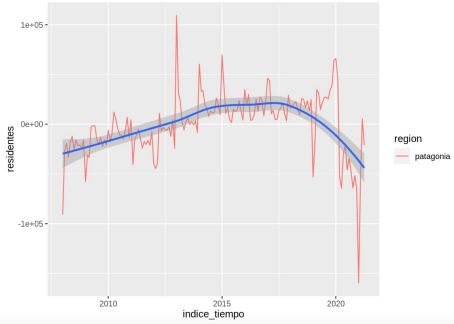
There are different ways to plot

With the data set in place and the aesthetic mappings selected, the final choice in making our plot is to decide how to graphically express the observations themselves.



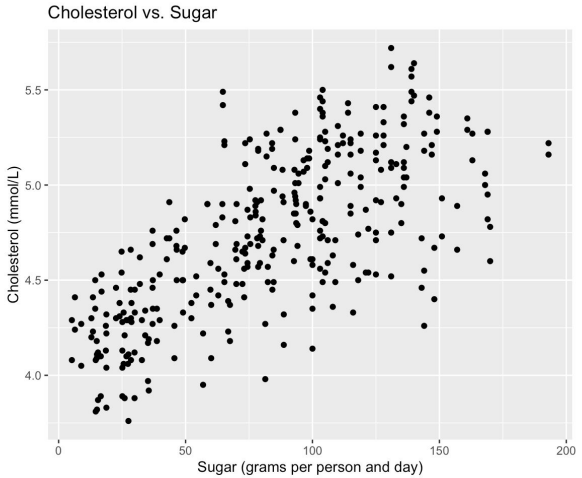


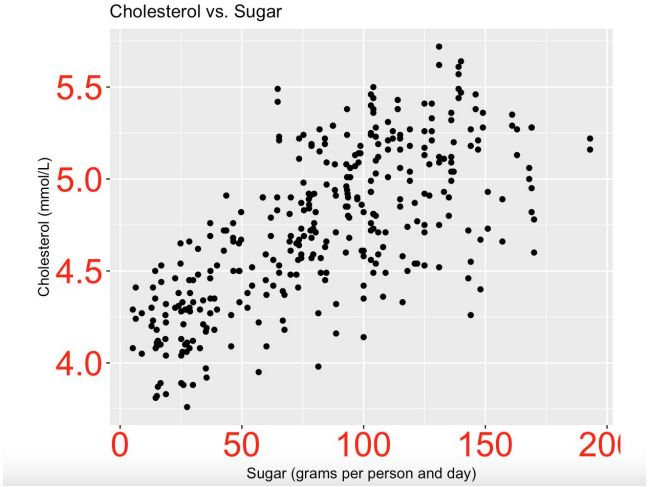


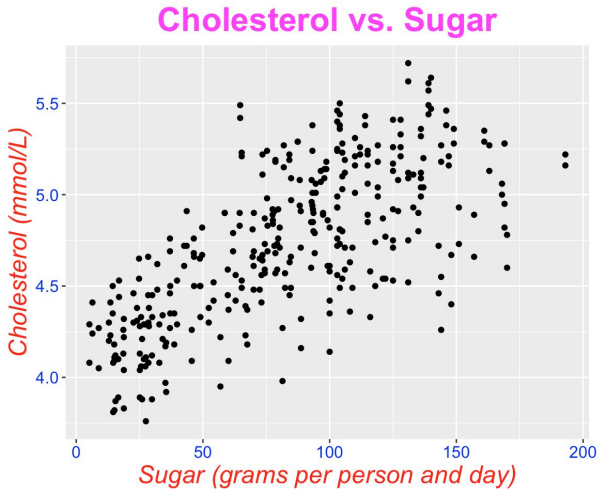


Customization in ggplot

Customizations on the axes, labels, and titles.







ggplot2 Theme Elements

theme(element_name = element_function())

- element_text()
- element_line()
- element_rect()
- element_blank()

Plot elements:

plot.background
element_rect()

plot.title
element_text()

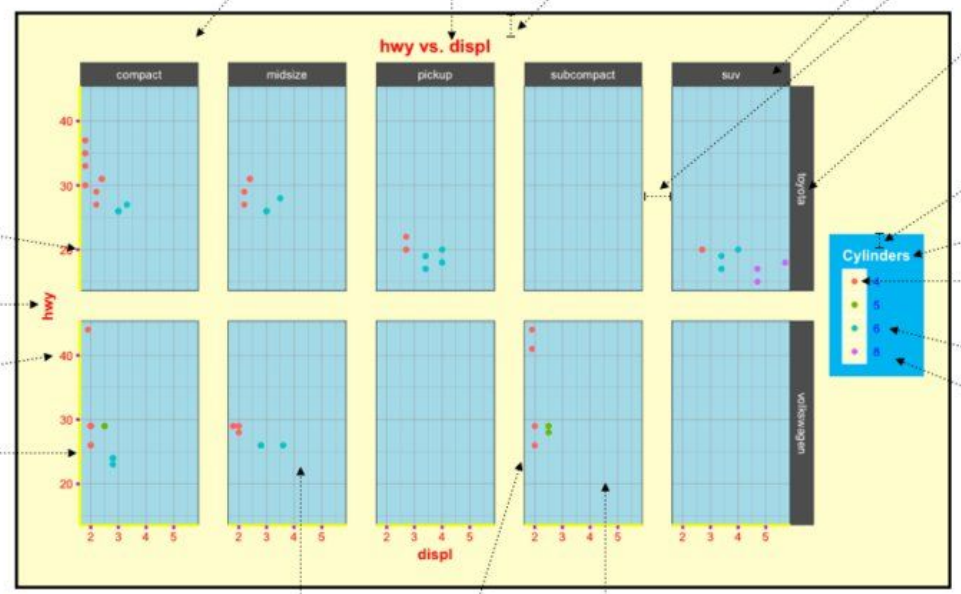
plot.margin
margin()

Facetting elements:

strip.background
element_rect()

panel.spacing
unit()

strip.text
element_text()



Axis elements:

axis.ticks
element_line()

axis.title
element_text()

axis.text
element_text()

axis.line
element_line()

Legend elements:

legend.margin
margin()

legend.title
element_text()

legend.key
element_rect()

legend.text
element_text()

legend.background
element_rect()

panel.background
element_rect()

panel.grid
element_line()

panel.border
element_rect(fill = NA)

Panel elements:

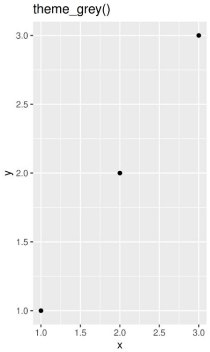
henrywang.nl

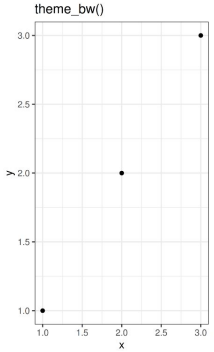
Derived from *"ggplot2: Elegant Graphics for Data Analysis"*

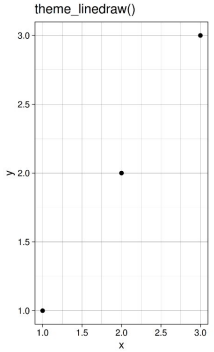
Theme

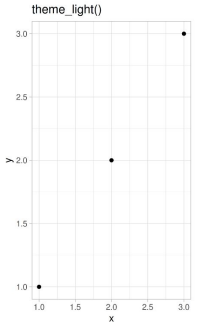
ggplot2 comes with a number of built-in themes.

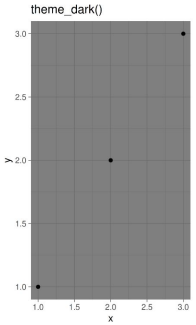
- `theme_gray()`
- `theme_bw()`
- `theme_minimal()`
- `theme_classic()`
- `theme_linedraw()`
- `theme_dark()`
- `theme_light()`
- `theme_void()`

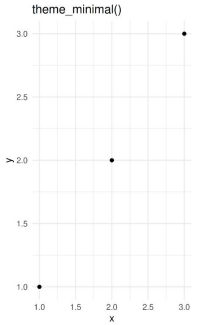


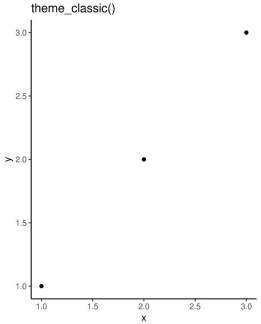












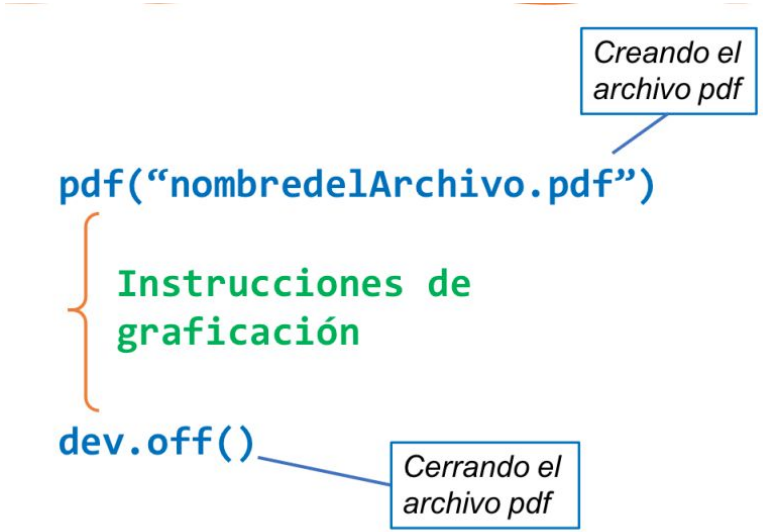
Ggsave

ggsave() is a convenient function for saving a plot. It defaults to saving the last plot that you displayed, using the size of the current graphics device. It also guesses the type of graphics device from the extension.

```
ggsave(
  filename,
  plot = last_plot(),
  device = NULL,
  path = NULL,
  scale = 1,
  width = NA,
  height = NA,
  units = c("in", "cm", "mm", "px"),
  dpi = 300,
  limitsize = TRUE,
  bg = NULL,
  create.dir = FALSE,
  ...
)
```

Dev.off

- bpm()
- jpeg()
- pdf()
- png()
- tiff()

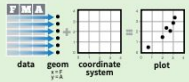


Visualización de Datos
usando ggplot2
Guía Rápida

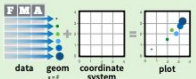
RStudio

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** ó **y**.



Para construir una gratica 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

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

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

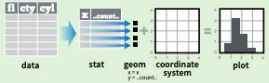
Argumentos

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.

función geom · función stat · geométricas

`f + stat_density2d(aes(fill = ..level..), geom = "polygon")` — variable que stat créa

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..

`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()`

Todo Uso

Escalas

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

`(n <- d + geom_bar(aes(fill = fl)))`

escala · estética afectada · escala del paquete · argumentos de la escala

`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"))`

valores permitidos · título del eje · etiqueta de cada valor visible · intervalo de cada valor visible

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))`

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

`p + scale_radius(range = c(1,6))`
`p + scale_size_area(max_size = 6)`

Usa el radio o el area

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

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 lineas
`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 indpendientes
- **"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)`

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

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

α^{c}	α^{d}	α^{e}	α^{p}	α^{r}

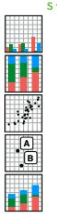
`t + facet_grid(. ~ fl, labeller = label_parsed)`

c	d	e	p	r

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))`

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>" `)`

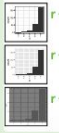
Use funciones escalas para controlar las etiquetas de las leyendas

`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

Tema



`r + theme_bw()` Fondo blanco con cuadrícula
`r + theme_gray()` Fondo gris (tema inicial)
`r + theme_dark()` Obscuro

`r + theme_classic()`
`r + theme_light()`
`r + theme_linedraw()`
`r + theme_minimal()` Temas minimalísticos
`r + theme_void()` Tema vacío

Agrandar una sección

Sin cortar (preferido)

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

Cortando (quita los puntos escondidos)

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