

# [Programación] R



- [Introducción y primeros pasos con el lenguaje R](#)

R es un lenguaje de programación y un entorno de software libre y de código abierto diseñado específicamente para el análisis estadístico y la visualización de datos. Es ampliamente utilizado en diversas disciplinas, incluyendo la ciencia de datos, la bioinformática, la economía, la epidemiología y muchas otras áreas donde se requiere análisis cuantitativo.



## Vectorized Functions

TO USE WITH MUTATE ()

**mutate()** and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

### OFFSET

dplyr: **lag()** - offset elements by 1  
 dplyr: **lead()** - offset elements by -1

### CUMULATIVE AGGREGATE

dplyr: **cumall()** - cumulative all()  
 dplyr: **cumany()** - cumulative any()  
 dplyr: **cummax()** - cumulative max()  
 dplyr: **cummean()** - cumulative mean()  
 dplyr: **cummin()** - cumulative min()  
 dplyr: **cumprod()** - cumulative prod()  
 dplyr: **cumsum()** - cumulative sum()

### RANKING

dplyr: **cume\_dist()** - proportion of all values <=  
 dplyr: **dense\_rank()** - rank w ties = min, no gaps  
 dplyr: **min\_rank()** - rank with ties = min  
 dplyr: **ntile()** - bins into n bins  
 dplyr: **percent\_rank()** - min\_rank scaled to [0,1]  
 dplyr: **row\_number()** - rank with ties = "first"

### MATH

**+**, **-**, **\***, **/**, **^**, **%/%**, **%%** - arithmetic ops  
**log()**, **log2()**, **log10()** - logs  
**<**, **<=**, **>**, **>=**, **!=**, **==** - logical comparisons  
 dplyr: **between()** - x >= left & x <= right  
 dplyr: **near()** - safe == for floating point numbers

### MISCELLANEOUS

dplyr: **case\_when()** - multi-case if\_else()  
 starwars %>%  
 mutate(type = case\_when(  
 height > 200 | mass > 200 ~ "large",  
 species == "Droid" ~ "robot",  
 TRUE ~ "other")

dplyr: **coalesce()** - first non-NA values by element across a set of vectors  
 dplyr: **if\_else()** - element-wise if() + else()  
 dplyr: **na\_if()** - replace specific values with NA  
 dplyr: **pmax()** - element-wise max()  
 dplyr: **pmin()** - element-wise min()

## Summary Functions

TO USE WITH SUMMARISE ()

**summarise()** applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

### COUNT

dplyr: **n()** - number of values/rows  
 dplyr: **n\_distinct()** - # of uniques  
 dplyr: **sum(is.na())** - # of non-NA's

### POSITION

**mean()** - mean, also **mean(is.na())**  
**median()** - median

### LOGICAL

**mean()** - proportion of TRUE's  
**sum()** - # of TRUE's

### ORDER

dplyr: **first()** - first value  
 dplyr: **last()** - last value  
 dplyr: **nth()** - value in nth location of vector

### RANK

**quantile()** - nth quantile  
**min()** - minimum value  
**max()** - maximum value

### SPREAD

**IQR()** - Inter-Quartile Range  
**mad()** - median absolute deviation  
**sd()** - standard deviation  
**var()** - variance

## Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

**rownames\_to\_column()**  
 Move row names into col.  
 a <- rownames\_to\_column(mtcars,  
 var = "C")

**column\_to\_rownames()**  
 Move col into row names.  
 column\_to\_rownames(a, var = "C")

Also tibble: **has\_rownames()** and **remove\_rownames()**.

## Combine Tables

COMBINE VARIABLES

**bind\_cols(..., .name\_repair)** Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

**bind\_rows(..., id = NULL)** Returns tables one on top of the other as a single table. Set id to a column name to add a column of the original table names (as pictured).

COMBINE CASES

**bind\_rows(..., id = NULL)** Returns tables one on top of the other as a single table. Set id to a column name to add a column of the original table names (as pictured).

### RELATIONAL DATA

Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"), ..., keep = FALSE, na\_matches = "na")** Join matching values from y to x.

**right\_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"), ..., keep = FALSE, na\_matches = "na")** Join matching values from x to y.

**inner\_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"), ..., keep = FALSE, na\_matches = "na")** Join data. Retain only rows with matches.

**full\_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"), ..., keep = FALSE, na\_matches = "na")** Join data. Retain all values, all rows.

Use a "Filtering Join" to filter one table against the rows of another.

**semi\_join(x, y, by = NULL, copy = FALSE, ..., na\_matches = "na")** Return rows of x that have a match in y. Use to see what will be included in a join.

**anti\_join(x, y, by = NULL, copy = FALSE, ..., na\_matches = "na")** Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a "Nest Join" to inner join one table to another into a nested data frame.

**nest\_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, ...)** Join data, nesting matches from y in a single new data frame column.

### COLUMN MATCHING FOR JOINS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.  
**left\_join(x, y, by = "x")**

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.  
**left\_join(x, y, by = c("C" = "D"))**

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.  
**left\_join(x, y, by = c("C" = "D"), suffix = c("I", "Z"))**

### SET OPERATIONS

**intersect(x, y, ...)**  
 Rows that appear in both x and y.

**setdiff(x, y, ...)**  
 Rows that appear in x but not y.

**union(x, y, ...)**  
 Rows that appear in x or y. (Duplicates removed). **union\_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).





# Data visualization with ggplot2 : : CHEAT SHEET

## Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

required  
not required, sensible defaults supplied

**ggplot**(data = mpg, aes(x = cty, y = hwy)) Begins a plot that you finish by adding layers. Add one geom function per layer.

**last\_plot()** Returns the last plot.

**ggsave**("plot.png", width = 5, height = 5) Saves last plot as 5" x 5" file named "plot.png" in working directory. Matches file type to file extension.

## Aes

**Common aesthetic values.**  
**color and fill** - string ("red", "#RRGGBB")  
**linetype** - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")  
**lineend** - string ("round", "butt", or "square")  
**linejoin** - string ("round", "mitre", or "bevel")  
**size** - integer (line width in mm)  
**shape** - integer/shape name or a single character ("a")



## Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

### GRAPHICAL PRIMITIVES

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

a + geom_blank() and a + expand_limits()
Ensure limits include values across all plots.

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

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

a + geom_polygon(aes(alpha = 50)) - x, y, alpha,
color, fill, group, subgroup, linetype, size

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

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

### LINE SEGMENTS

```
common aesthetics: x, y, alpha, color, linetype, size

b + geom_spline(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 = 111.155, radius = 1))
```

### ONE VARIABLE continuous

```
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(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group

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
```

### discrete

```
d <- ggplot(mpg, aes(f1))
d + geom_bar()
x, y, alpha, color, fill, linetype, size, weight
```

### TWO VARIABLES

```
both continuous
e <- ggplot(mpg, aes(cty, hwy))

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

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) - x, y, label, alpha, angle, color,
family, fontface, hjust, lineheight, size, vjust
```

### one discrete, one continuous

```
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
```

### both discrete

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

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

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

### THREE 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, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z))
x, y, alpha, color, fill, group, linetype, size, subgroup
```

### continuous bivariate distribution

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

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

h + geom_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size
```

### continuous function

```
i <- ggplot(economics, aes(date, unemployment))

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
```

### visualizing 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, y, 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
```

### maps

```
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
```

From: <https://euloxio.myds.me/dokuwiki/> - Euloxio wiki

Permanent link: <https://euloxio.myds.me/dokuwiki/doku.php/doc:tec:prg:r:inicio>

Last update: 2024/04/12 11:26

