

Paquete caret

Guía Rápida

Especificando el Modelo

Possibles sintaxis para especificar las variables en el modelo:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
• rfe, sbf, gafs, y safs solo tienen el interfaz x/y.
• El método para la formula train creará siempre variables dummy
• El interfaz x/y de train no creará variables dummy (pero la función del modelo representado podría hacerlo).
```

Recuerda:

- Tener nombres en las columnas de tus datos.
- Usar factores para las variables objetivo en los modelos de clasificación (no usar 0/1 o enteros).
- Tener nombres de R válidos para los niveles de las clases (no "0"/"1")
- Fijar una semilla aleatoria antes de llamar a la función train repetidamente para obtener las mismas muestras en las diferentes llamadas.
- Usar la opción na.action = na.pass de train si vas a imputar valores faltantes. También, usa esta opción cuando vayas a predecir nuevos datos que contengan faltantes.

Para pasar opciones a la función del modelo, se puede hacer a través de train via las elipses:

```
train(y ~ ., data = dat, method = "rf",
      # options to `randomForest`:
      importance = TRUE)
```

Procesamiento en Paralelo

Se usa el paquete foreach para ejecutar modelos en paralelo. El código de train no cambia pero se tiene que llamar al "do" primero.

```
# en MacOS o Linux          # en Windows
library(doMC)                library(doParallel)
registerDoMC(cores=4)         cl <- makeCluster(2)
                             registerDoParallel(cl)
```

La función parallel::detectCores puede ayudar también.

Preprocesado

Se pueden aplicar transformaciones, filtros y otras operaciones a las variables predictoras con la opción preProc.

```
train(..., preProc = c("method1", "method2"), ...)
```

Incluye los siguientes métodos:

- "center", "scale", y "range" para normalizar.
- "BoxCox", "YeoJohnson", o "expoTrans" para transformar.
- "knnImpute", "bagImpute", o "medianImpute" para imputar.
- "corr", "nzv", "zv", y "conditionalX" para filtrar.
- "pca", "ica", o "spatialSign" para transformar grupos.

train determina el orden de las operaciones: el orden en el que se declaran los métodos no importa.

El paquete recipes contiene una lista más extensa de operaciones de procesamiento.

Añadiendo Opciones

Muchas opciones de train se pueden especificar usando la función trainControl:

```
train(y ~ ., data = dat, method = "cubist",
      trControl = trainControl(<opciones>))
```

Opciones de Remuestreo

trainControl se usa para elegir el método de remuestreo:

```
trainControl(method = <método>, <opciones>)
```

Los métodos y las opciones son:

- "cv" para K-iteraciones validación-cruzada (number define el # de iteraciones).
- "repeatedcv" para validación-cruzada repetida (repeats para el # de repeticiones).
- "boot" para bootstrap (number define las iteraciones).
- "LGOCV" para dejar-grupo-fuera (number y p son opciones).
- "L0O" para dejar-uno-fuera validación-cruzada.
- "oob" para el remuestreo fuera-de-la-bolsa (solo para algunos modelos).
- "timeslice" para series temporales (las opciones son initialWindow, horizon, fixedWindow, y skip).

Métricas de Rendimiento

Para elegir cómo resumir un modelo, se usa de nuevo la función trainControl

```
trainControl(summaryFunction = <función R>,
            classProbs = <lógico>)
```

Se pueden usar funciones de R a la medida, aunque caret incluye varias: defaultSummary (para Precisión, RMSE, etc), twoClassSummary (para curvas ROC), y prSummary (para recuperación de información). Para estas dos últimas funciones, se tiene que ajustar a TRUE la opción classProbs.

Búsqueda en Grid

Para conseguir que train determine los valores de el/los parámetro/s de ajuste, la opción tuneLength controla cuántos valores se evalúan por parámetro de ajuste.

De forma alternativa, se pueden declarar valores específicos de los parámetros de ajuste usando el argumento de tuneGrid:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
                      lambda = c(0.001, 0.01))
```

```
train(x = x, y = y, method = "glmnet",
      preProc = c("center", "scale"),
      tuneGrid = grid)
```

Búsqueda Aleatoria

Para ajustar train también se pueden generar combinaciones de parámetros de ajuste de forma aleatoria sobre un rango.

tuneLength controla el número total de combinaciones a evaluar. Para usar la búsqueda aleatoria:

```
trainControl(search = "random")
```

Submuestreos

Para un gran desbalanceo entre clases, train puede submuestrear los datos para balancear las clases antes de ajustar el modelo.

```
trainControl(sampling = "down")
```

Otros valores son "up", "smote", o "rose". Los dos últimos pueden requerir la instalación de paquetes adicionales.

Importar Datos

with `readr`, `tibble`, and `tidyR`

Guía Rápida



El **tidyverse** de R está construido alrededor de los **datos ordenados** almacenados en **tibbles**, una versión mejorada del data frame.

El anverso de esta guía muestra como leer ficheros de texto en R con `readr`. El reverso muestra como crear tibbles con `tibble` y como disponer datos ordenados con `tidyR`.

Otros tipos de datos

Prueba uno de los siguientes paquetes para importar otros tipos de ficheros

- **haven** - ficheros SPSS, Stata y SAS
- **readxl** - ficheros Excel (.xls y .xlsx)
- **DBI** - bases de datos
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Funciones Escritura

Salva `x`, un objeto R, en `path`, una ruta del fichero, con:

`write_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

Tibble/df a fichero delimitado con coma.

`write_delim(x, ruta, delim = " ", na = "NA", append = FALSE, col_names = !append)`

Tibble/df to file with any delimiter.

`write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

Tibble/df a CSV para Excel

`write_file(x, path, append = FALSE)`

Cadena a fichero.

`write_lines(x, path, na = "NA", append = FALSE)`

Vector cadena a fichero, un elemento por línea.

`write_rds(x, path, compress = c("none", "gz", "bz2", "xz", ...))`

Objeto a fichero RDS.

`write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)`

Tibble/df a ficheros delimitados por tab.

Funciones Lectura

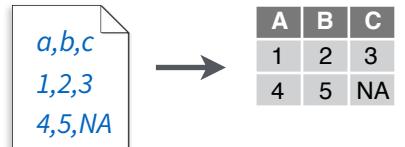
Lectura de datos tabulares a tibbles

Estas funciones comparten los siguientes argumentos comunes:

`read_*`(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())

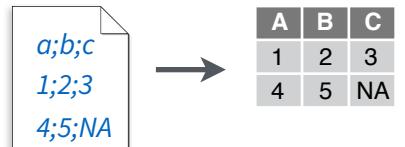
`read_csv()`

Lee ficheros delimitados por comas.
`read_csv("file.csv")`



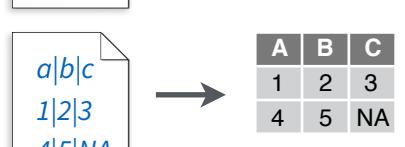
`read_csv2()`

Lee ficheros delimitados por punto y coma.
`read_csv2("file2.csv")`



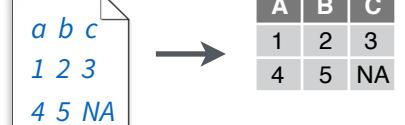
`read_delim(delim, quote = "\'", escape_backslash = FALSE, escape_double = TRUE)`

Lee ficheros con cualquier delimitador.
`read_delim("file.txt", delim = "|")`



`read_fwf(col_positions)`

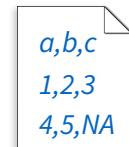
Lee ficheros con ancho fijo.
`read_fwf("file.fwf", col_positions = c(1, 3, 5))`



`read_tsv()`

Lee ficheros delimitados por tab. También `read_table()`.
`read_tsv("file.tsv")`

Argumentos útiles



Fichero ejemplo

`write_csv(path = "file.csv", x = read_csv("a,b,c|n1,2,3|n4,5,NA"))`

| | | |
|---|---|----|
| 1 | 2 | 3 |
| 4 | 5 | NA |



Sin cabecera

`read_csv("file.csv", col_names = FALSE)`

| | | |
|---|---|---|
| A | B | C |
| 1 | 2 | 3 |



Proporciona cabecera

`read_csv("file.csv", col_names = c("x", "y", "z"))`

| | | |
|----|----|----|
| A | B | C |
| 1 | 2 | 3 |
| NA | NA | NA |

Salta líneas

`read_csv("file.csv", skip = 1)`

| | | |
|---|---|---|
| A | B | C |
| 1 | 2 | 3 |

Lee un subconjunto

`read_csv("file.csv", n_max = 1)`

Valores Faltantes

`read_csv("file.csv", na = c("4", "5", "."))`

Lectura de datos no tabulares

`read_file(file, locale = default_locale())`

Lee un fichero en una sola cadena.

`read_file_raw(file)`

Lee un fichero en un vector raw.

`read_lines(file, skip = 0, n_max = -1L, locale = default_locale(), na = character(), progress = interactive())`

Lee cada línea en una cadena.

`read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())`

Lee cada línea en un vector raw.

`read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())`

Ficheros de log estilo Apache.

Filtrando tipos de datos

Las funciones de `readr` interpretan los tipos de cada columna y convierten los tipos de forma apropiada (pero NO convertirán cadenas a factores automáticamente).

Un mensaje muestra el tipo de columna en el resultado.

```
## Parsed with column specification:  
## cols(  
##   age = col_integer(),  
##   sex = col_character(),  
##   earn = col_double()  
## )
```

age es un entero
sex es un carácter
earn es un doble (numérico)

1. Usa `problems()` para diagnosticar problemas
`x <- read_csv("file.csv"); problems(x)`

2. Usa `col_function` para guiar el filtrado

- `col_guess()` - por defecto
 - `col_character()`
 - `col_double()`
 - `col_euro_double()`
 - `col_datetime(format = "")` También `col_date(format = "")` and `col_time(format = "")`
 - `col_factor(levels, ordered = FALSE)`
 - `col_integer()`
 - `col_logical()`
 - `col_number()`
 - `col_numeric()`
 - `col_skip()`
- ```
x <- read_csv("file.csv", col_types = cols(
 A = col_double(),
 B = col_logical(),
 C = col_factor()
)
```

3. Para el resto, los lee como vectores carácter y luego los filtra con la función `parse_`.

- `parse_guess(x, na = c("", "NA"), locale = default_locale())`
  - `parse_character(x, na = c("", "NA"), locale = default_locale())`
  - `parse_datetime(x, format = "", na = c("", "NA"), locale = default_locale())` Also `parse_date()` and `parse_time()`
  - `parse_double(x, na = c("", "NA"), locale = default_locale())`
  - `parse_factor(x, levels, ordered = FALSE, na = c("", "NA"), locale = default_locale())`
  - `parse_integer(x, na = c("", "NA"), locale = default_locale())`
  - `parse_logical(x, na = c("", "NA"), locale = default_locale())`
  - `parse_number(x, na = c("", "NA"), locale = default_locale())`
- ```
x$A <- parse_number(x$A)
```

Tibbles - un data frame mejorado

El paquete **tibble** proporciona una nueva clase S3 para almacenar datos tabulares, el tibble. Tibbles heredan la clase del data frame, pero mejora dos comportamientos:

- Visualiza** - Cuando se imprime un tibble, R proporciona una vista concisa de los datos que se ajustan en una pantalla.
- Subconjunto** - Siempre devuelve un nuevo tibble, [[y \$ siempre devuelve un vector.
- No emparejado parcial** - Se debe usar el nombre completo de las columnas cuando se selecciona un subconjunto.

| # A tibble: 234 x 6 | manufacturer | model | displ | year | cyl |
|---------------------|--------------|-------|-------|------|-----|
| 1 audi | a4 | 1.8 | | 1999 | 4 |
| 2 audi | a4 | 1.8 | | 1999 | 6 |
| 3 audi | a4 | 2.0 | | 1999 | 6 |
| 4 audi | a4 | 2.0 | | 1999 | 8 |
| 5 audi | a4 | 2.8 | | 1999 | 8 |
| 6 audi | a4 | 2.8 | | 1999 | 8 |
| 7 audi | a4 quattro | 3.1 | | 1999 | 8 |
| 8 audi a4 quattro | | 1.8 | | 1999 | 8 |
| 9 audi a4 quattro | | 1.8 | | 1999 | 8 |
| 10 audi a4 quattro | | 2.0 | | 1999 | 8 |

visualización tibble

| country | 1999 | 2000 |
|---------|------|------|
| A | 0.7K | 2K |
| B | 37K | 80K |
| C | 212K | 213K |

visualización data frame

- La apariencia por defecto se controla con las opciones:


```
options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf)
```
- La vista del conjunto completo de datos con **View(x, title)** o **glimpse(x, width = NULL, ...)**
- Revertir a data frame con **as.data.frame()** (requerido por algunos paquetes antiguos)

Construir un tibble en dos formas

| tibble(...) | Construye por columnas. |
|--|-------------------------|
| tibble(x = 1:3, y = c("a", "b", "c")) | Ambos crean este tibble |

| tribble(...) | Construye por filas. |
|--|--|
| tribble(~x, ~y, 1, "a", 2, "b", 3, "c") | A tibble: 3 x 2 x y <int> <dbl> 1 1 a 2 2 b 3 3 c |

as_tibble(x, ...) Convierte data frame a tibble.
enframe(x, name = "name", value = "value") Convierte un vector con nombre a un tibble con una columna con nombre y una columna valor.
is_tibble(x) Comprueba si x es un tibble.

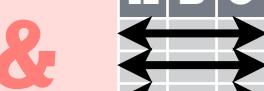
Datos Ordenados con tidyR

Datos Tidy es una forma de organizar datos tabulares. Proporciona una estructura consistente de datos entre paquetes.

Una tabla es tidy si:

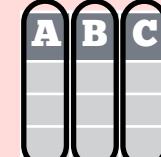


Cada **variable** está en su propia **columna**



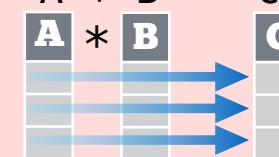
Cada **observación o caso**, está en su propia **fila**

Datos Tidy:



Permite el acceso a las variables como vectores

A * B → C



Preserva los casos en las operaciones vectorizadas

Remodelado de Datos - cambia la disposición de los valores en una tabla

Usa **gather()** y **spread()** para reorganizar los valores de una tabla en una nueva disposición. Usan la idea de una columna clave: par valor columna.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

Gather mueve los nombres de las columnas a una columna llave, reuniendo los valores de las columnas en una única columna.

table4a

| country | 1999 | 2000 |
|---------|------|------|
| A | 0.7K | 2K |
| B | 37K | 80K |
| C | 212K | 213K |



| country | year | cases |
|---------|------|-------|
| A | 1999 | 0.7K |
| B | 1999 | 37K |
| C | 1999 | 212K |
| A | 2000 | 2K |
| B | 2000 | 80K |
| C | 2000 | 213K |

llave valor

table2

| country | year | type | count |
|---------|------|-------|-------|
| A | 1999 | cases | 0.7K |
| A | 1999 | pop | 19M |
| A | 2000 | cases | 2K |
| A | 2000 | pop | 20M |
| B | 1999 | cases | 37K |
| B | 1999 | pop | 172M |
| B | 2000 | cases | 80K |
| B | 2000 | pop | 174M |
| C | 1999 | cases | 212K |
| C | 1999 | pop | 1T |
| C | 2000 | cases | 213K |
| C | 2000 | pop | 1T |

llave valor

spread(table2, type, count)

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

Gestión de Datos Faltantes

fill(data, ..., .direction = c("down", "up"))

Completa los NA's en ... columnas con los valores no-NA más cercanos.

x

| x1 | x2 |
|----|----|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

fill(x, x2)

replace_na(data, replace = list(), ...)

Reemplaza NA's por columna.

x

| x1 | x2 |
|----|----|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

replace_na(x, list(x2 = 2), x2)

drop_na(data, ...)

Elimina columnas con NAs.

x

| x1 | x2 |
|----|----|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

drop_na(x, x2)

Expansión de Tablas - crea tablas rápidamente con combinaciones de valores

complete(data, ..., fill = list())

Añade a los datos combinaciones faltantes de los valores listados en ...

complete(mtcars, cyl, gear, carb)

expand(data, ...)

Crea un nuevo tibble con todas las posibles combinaciones de valores de las variables listadas en ...

expand(mtcars, cyl, gear, carb)

Separar y Combinar Celdas

Usa estas funciones para separar o combinar celdas en valores individuales, aislados.

separate(data, col, into, sep = "[[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separar cada celda de una columna para crear varias columnas

| country | year | rate |
|---------|------|----------|
| A | 1999 | 0.7K/19M |
| A | 2000 | 2K/20M |
| B | 1999 | 37K/172M |
| B | 2000 | 80K/174M |
| C | 1999 | 212K/1T |
| C | 2000 | 213K/1T |

→

| country | year | cases | pop |
|---------|------|-------|-----|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172 |
| B | 2000 | 80K | 174 |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

separate_rows(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[[:alnum:]].+", convert = FALSE)

Separa cada celda en una columna para crear varias filas. También **separate_rows_()**.

| country | year | rate |
|---------|------|----------|
| A | 1999 | 0.7K/19M |
| A | 1999 | 19M |
| B | 1999 | 37K/172M |
| B | 1999 | 172M |
| C | 1999 | 212K/1T |
| C | 1999 | 1T |
| A | 2000 | 20M |
| B | 2000 | 80K |
| C | 2000 | 213K |

→

| country | year | rate |
|---------|------|------|
| A | 1999 | 0.7K |
| A | 1999 | 19M |
| A | 2000 | 20M |
| B | 1999 | 37K |
| B | 1999 | 172M |
| C | 1999 | 212K |
| C | 1999 | 1T |
| C | 2000 | 213K |
| C | 2000 | 1T |

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE)

Collapsa celdas de varias columnas para crear una única columna.

| country | century | year |
|---------|---------|------|
| Afghan | 19 | 99 |
| Afghan | 20 | 0 |
| Brazil | 19 | 99 |
| Brazil | 20 | 0 |
| China | 19 | 99 |
| China | 20 | 0 |

→

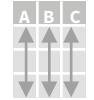
| country | year |
|---------|------|
| Afghan | 1999 |
| Afghan | 2000 |
| Brazil | 1999 |
| Brazil | 2000 |
| China | 1999 |
| China | 2000 |

unite(table5, century, year, col = "year", sep = "")

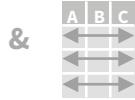
Transformación de Datos con dplyr :: HOJA DE REFERENCIA



dplyr funciona con conductos y requiere **datos ordenados**.
En datos ordenados:



Cada **variable** tiene su propia **columna**



Cada **observación** tiene su propia **fila**



Resumir Casos

Estos aplican **funciones de resumen** a columnas para crear un nuevo cuadro. Funciones de resumen toman vectores como entrada y devuelven un solo valor (ver reversa).

| | summary function |
|--|--|
| | summarise(.data, ...) Calcula cuadro de resúmenes. También summarise_() . <code>summarise(mtcars, avg = mean(mpg))</code> |
| | count(x, ..., wt = NULL, sort = FALSE) Conteo del número de filas en cada grupo, definido por las variables en ... También tally() . <code>count(iris, Species)</code> |

VARIACIONES

- summarise_all()** - Aplica funs a cada columna
- summarise_at()** - Aplica funs a columnas específicas.
- summarise_if()** - Aplica funs a todas las columnas de un tipo

Agrupar Casos

Usa **group_by()** para crear una copia “agrupada” de un cuadro. Funciones dplyr manipularán cada “grupo” por separado para luego combinar los resultados.

| | | |
|-----------------------------|--|--|
| | | |
| mtcars %>% | | |
| group_by(cyl) %>% | | |

group_by(.data, ..., add = FALSE)
Devuelve copia del cuadro agrupado por ...
`g_iris <- group_by(iris, Species)`

ungroup(x, ...)
Devuelve copia no-agrupada del cuadro
`ungroup(g_iris)`

Manipular Casos

EXTRAE CASOS

Funciones de Fila devuelven un sub-conjunto de filas como un nuevo cuadro. Usa la variante que termina en _ para código que funciona con evaluación no-estándar.

| | | |
|--|--|---|
| | | filter(.data, ...) Extrae filas que cumplen criterios lógicos. También filter_() . <code>filter(iris, Sepal.Length > 7)</code> |
| | | distinct(.data, ..., .keep_all = FALSE) Remueve filas duplicadas. También distinct_() . <code>distinct(iris, Species)</code> |
| | | sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Selecciona una fracción de filas al azar. <code>sample_frac(iris, 0.5, replace = TRUE)</code> |
| | | sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Selecciona n filas al azar. <code>sample_n(iris, 10, replace = TRUE)</code> |
| | | slice(.data, ...) Selecciona filas por posición. También slice_() . <code>slice(iris, 10:15)</code> |
| | | top_n(x, n, wt) Selecciona y ordena las n entradas mas altas (por grupo si los datos están agrupados). <code>.top_n(iris, 5, Sepal.Width)</code> |

Operadores Lógicos y Booleanos para usar con filter()

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

Busca **?base::logic** y **?Comparison** para la documentación.

ORDENA CASOS

| | | |
|--|--|--|
| | | arrange(.data, ...) Ordena filas por valores de una columna (bajo a alto), usa con desc() para ordenar de alto a bajo. <code>arrange(mtcars, mpg)</code> <code>arrange(mtcars, desc(mpg))</code> |
|--|--|--|

AÑADE CASOS

| | | |
|--|--|---|
| | | add_row(.data, ..., .before = NULL, .after = NULL) Añade una o mas filas a un cuadro. <code>add_row(faithful, eruptions = 1, waiting = 1)</code> |
|--|--|---|

Manipular Variables

EXTRAER VARIABLES

Funciones de Columnas devuelven un conjunto de columnas como un nuevo cuadro. Usa la variante que termina en _ para código que funciona con evaluación no-estándar.

| | | |
|--|--|--|
| | | select(.data, ...) Selecciona columnas por nombre o funciones de ayuda. También select_if() . <code>select(iris, Sepal.Length, Species)</code> |
|--|--|--|

Usa estos ayudantes con **select()**,
e.g. `select(iris, starts_with("Sepal"))`

| | | |
|-------------------------|---------------------------------|-------------------------------|
| contains(match) | num_range(prefix, range) | :, e.g. <code>mpg:cyl</code> |
| ends_with(match) | one_of(...) | -, e.g. <code>-Species</code> |
| matches(match) | starts_with(match) | |

CREA NUEVAS VARIABLES

Estos aplican **funciones vectorizadas** a columnas. Funs vectorizadas toman vectores como entrada y devuelven vectores de la misma longitud como salida (ver reverso).

| | función vectorizada |
|--|---|
| | mutate(.data, ...) Calcula columna(s) nueva(s). <code>mutate(mtcars, gpm = 1/mpg)</code> |
| | transmute(.data, ...) Calcula columna(s) nueva(s), elimina otros. <code>transmute(mtcars, gpm = 1/mpg)</code> |
| | mutate_all(.tbl, .funs, ...) Aplica funs a cada columna. Use con funs() . <code>mutate_all(faithful, funs(log(.), log2(.)))</code> |
| | mutate_at(.tbl, .cols, .funs, ...) Aplica funs a columnas específicas. Usa con funs() , vars() y la funciones de ayudar para select(). <code>mutate_at(iris, vars(-Species), funs(log(.)))</code> |
| | mutate_if(.tbl, .predicate, .funs, ...) Aplica funs a todas las columnas de un tipo. Usa con funs() . <code>mutate_if(iris, is.numeric, funs(log(.)))</code> |
| | add_column(.data, ..., .before = NULL, .after = NULL) Añade nueva(s) columna(s). <code>add_column(mtcars, new = 1:32)</code> |
| | rename(.data, ...) Renombra columnas. <code>rename(iris, Length = Sepal.Length)</code> |



Funciones de Vector

PARA USO CON MUTATE ()

mutate() y **transmute()** aplican funciones vectorizadas a columnas para crear nuevas columnas. Funciones vectorizadas toman vectores como entrada y devuelven vectores de la misma longitud.

función vectorizada →

CONTRARRESTAR

dplyr::lag() - Copia con valores atrasados por 1
dplyr::lead() - Copia con valores adelantados por 1

AGREGADOS AAcumulativoS

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

RANKINGS

dplyr::cume_dist() - Proporción de todos los valores <=
dplyr::dense_rank() - clasificación con empates = min, sin brechas
dplyr::min_rank() - clasificación con empates = min
dplyr::ntile() - asigna a n intervalos (bins)
dplyr::percent_rank() - min_rank escalado a [0,1]
dplyr::row_number() - clasificación con empates = "first" (el primero)

MATEMATICAS

+, -, *, /, ^, %/%, %% - ops aritméticas
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - comparaciones lógicas

MISCELÁNEAS

dplyr::between() - x >= izquierda & x <= derecha
dplyr::case_when() - casos-multiples if_else()
dplyr::coalesce() - primer elemento no-NA por elemento a lo largo de un conjunto de vectores
dplyr::if_else() - if() + else() por elemento
dplyr::na_if() - reemplaza valores específicos con NA
pmax() - max() por elemento
pmin() - min() por elemento
dplyr::recode() - switch() vectorizado
dplyr::recode_factor() - switch() vectorizado para factores

Funciones de Resumen

PARA USO CON SUMMARISE ()

summarise() aplica funciones de resumen a columnas para crear un nuevo cuadro. Funciones de resumen toman vectores como entrada y devuelven un solo valor.

función de resumen →

CONTEOS

dplyr::n() - número de valores / filas
dplyr::n_distinct() - # de únicos
sum(!is.na()) - # de no-NA's

POSICIÓN

mean() - promedio, también **mean(!is.na())**
median() - mediana

LÓGICOS

mean() - proporción de TRUE's
sum() - # de TRUE's

POSICIÓN/ORDEN

dplyr::first() - primer valor
dplyr::last() - último valor
dplyr::nth() - valor en posición n del vector

RANGO

quantile() - centil n
min() - valor mínimo
max() - valor máximo

PROPAGACIÓN

IQR() - rango inter-centil
mad() - desviación absoluta media
sd() - desviación estándar
var() - varianza

Nombres Filas

Datos ordenados no usan nombres de filas, que implica un valor fuera de las columnas. Para trabajar con nombres de filas primero muevelos a una columna.

→ **rownames_to_column()**
Mueve nombres de filas a una col.
a <- rownames_to_column(iris, var = "C")

→ **column_to_rownames()**
Mueve columna a nombre de filas.
column_to_rownames(a, var = "C")

También **has_rownames()**, **remove_rownames()**

Combina Cuadros

COMBINA VARIABLES

| X | y | = |
|--|--|--|
| A B C a t 1 b u 2 c v 3 | + A B D a t 3 b u 2 d w 1 | = A B C A B D a t 1 a t 3 b u 2 b u 2 c v 3 d w 1 |
| | | |

Usa **bind_cols()** para unir cuadros uno al lado del otro tal como son.

bind_cols(...) Devuelve cuadros posicionados lado a lado como un solo cuadro
ASEGÚRATE QUE LAS FILAS COINCIDEN.

Usa una "Unión Mutante" para unir un cuadro a columnas de otro cuadro, buscando valores correspondientes en las filas. Cada unión retiene una combinación diferente de los valores de los cuadros.

| A B C D | left_join(x, y, by = NULL, |
|----------------|--------------------------------------|
| a t 1 3 | copy=FALSE, suffix=c(".x",".y"),...) |
| b u 2 2 | Une filas coincidentes de y a x. |
| c v 3 NA | |

| A B C D | right_join(x, y, by = NULL, copy = |
|----------------|------------------------------------|
| a t 1 3 | FALSE, suffix=c(".x",".y"),...) |
| b u 2 2 | Une filas coincidentes de x a y. |
| d w NA 1 | |

| A B C D | inner_join(x, y, by = NULL, copy = |
|----------------|--|
| a t 1 3 | FALSE, suffix=c(".x",".y"),...) |
| b u 2 2 | Une datos. Mantener solo filas en ambos. |
| c v 3 NA | |

| A B C D | full_join(x, y, by = NULL, |
|----------------|---|
| a t 1 3 | copy=FALSE, suffix=c(".x",".y"),...) |
| b u 2 2 | Une datos. Mantener todos los valores, todas las filas. |
| c v 3 NA | |

Usa **by = c("col1", "col2")** para especificar cuales columnas usar para determinar coincidencias.
left_join(x, y, by = "A")

Usa un vector con nombres, **by = c("col1" = "col2")**, para determinar coincidencias en columnas con diferentes nombres en cada conjunto de datos.
left_join(x, y, by = c("C" = "D"))

Usa **suffix** para especificar el sufijo para dar a nombres de columnas duplicadas.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINA CASOS

| X | y | = |
|--|---|---|
| A B C a t 1 b u 2 c v 3 | + A B C C v 3 d w 4 | = A B C a t 1 b u 2 c v 3 d w 4 |
| | | |

Usa **bind_rows()** para unir cuadros uno debajo del otro tal como son.

bind_rows(..., .id = NULL)
Devuelve cuadros uno encima del otro como un solo cuadro. Fija .id a un nombre de columna para añadir una columna con los nombres del cuadro de proveniencia originales (como en la figura)

intersect(x, y, ...)
Filas que aparecen en ambos x y y.

setdiff(x, y, ...)
Filas que aparecen en x pero no en y.

union(x, y, ...)
Filas que aparecen en x o y (removiendo duplicados). union_all() retiene duplicados.

Usa **setequal()** para probar si dos conjuntos de datos contienen el número exactamente igual de filas (en cualquier orden).

EXTRAE FILAS

| X | y | = |
|--|--|--|
| A B C a t 1 b u 2 c v 3 | + A B D a t 3 b u 2 d w 1 | = A B D a t 3 b u 2 d w 1 |
| | | |

Usa una "Unión de Filtro" para filtrar un cuadro contra las filas de otro.

semi_join(x, y, by = NULL, ...)
Devuelve filas de x que coinciden en y.
ÚTIL PARA VER QUE SE VA A UNIR.

anti_join(x, y, by = NULL, ...)
Devuelve filas de x que no coinciden en y.
ÚTIL PARA VER QUE NO SE VA A UNIR.

Domar Datos con dplyr and tidyverse

Hoja de Referencia



Syntax - Convenciones útiles para la doma

dplyr::tbl_df(iris)

Convierte datos a una *tbl*. Objetos *tbl* son mas fáciles de inspeccionar que data frames. Ro solo muestra los datos que caben en la pantalla:

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length
1          5.1        3.5          1.4
2          4.9        3.0          1.4
3          4.7        3.2          1.3
4          4.6        3.1          1.5
5          5.0        3.6          1.4
...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::glimpse(iris)

Resumen con mucha información sobre los datos *tbl*.

utils::View(iris)

Observa el conjunto de datos en lo que parece una hoja de cálculo (nota la V capital).

| iris | | | | |
|------|--------------|-------------|--------------|-------------|
| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
| 1 | 5.1 | 3.5 | 1.4 | 0.2 |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 |
| 6 | 5.4 | 3.9 | 1.7 | 0.4 |
| 7 | 4.6 | 3.4 | 1.4 | 0.3 |
| 8 | 5.0 | 3.4 | 1.5 | 0.2 |

dplyr::%>%

Pasa el objeto a la izquierda al primer argumento (o argumento *.*) de la función a la derecha creando un tubo.

x %>% f(y) es lo mismo que *f(x, y)*

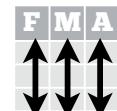
y %>% f(x, ., z) es lo mismo que *f(x, y, z)*

El uso de tubos y tuberías (en Ingles: "Piping") con *%>%* resulta en código mas legible. Por ejemplo:

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Datos Ordenados - La base para domar datos en R

En un conjunto de datos ordenado:



&



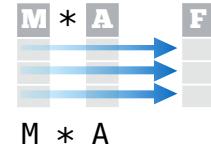
Cada variable se tiene su propia columna

Cada observación tiene su propia fila

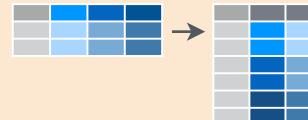
Datos ordenados complementan las

operaciones vectorizadas de R.

Automáticamente R preserva observaciones mientras manipula las variables. No hay otro formato que funciona tan intuitivamente en R.



Remodelar Datos - Cambia el esquema de los datos



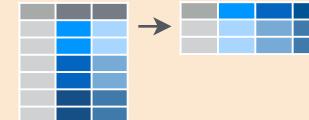
tidyverse::gather(cases, "year", "n", 2:4)

Reúne columnas en filas.



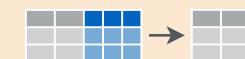
tidyverse::separate(storms, date, c("a", "m", "d"))

Separa una columna en varias.



tidyverse::spread(pollution, size, amount)

Extiende filas en columnas.



tidyverse::unite(data, col, ..., sep)

Une varias columnas en una.

dplyr::data_frame(a = 1:3, b = 4:6)

Combina vectores en un *data frame* (optimizado).

dplyr::arrange(mtcars, mpg)

Ordena filas por valores de una columna (bajo a alto).

dplyr::arrange(mtcars, desc(mpg))

Ordena filas por valores de una columna (alto a bajo).

dplyr::rename(tb, y = year)

Cambia el nombre de columnas de un *data frame*.

Subconjuntos de Observaciones



dplyr::filter(iris, Sepal.Length > 7)

Extrae filas que cumplen criterios lógicos.

dplyr::distinct(iris)

Remueve filas duplicadas.

dplyr::sample_frac(iris, 0.5, replace = TRUE)

Selecciona una fracción de filas al azar.

dplyr::sample_n(iris, 10, replace = TRUE)

Selecciona n filas al azar.

dplyr::slice(iris, 10:15)

Selecciona filas por posición.

dplyr::top_n(storms, 2, date)

Selecciona y ordena las n entradas mas altas (por grupo si los datos estan agrupados).

Subconjuntos de Variables



dplyr::select(iris, Sepal.Width, Petal.Length, Species)

Selecciona columnas por nombre o funciones de ayuda.

Funciones de ayuda para for select - ?select

select(iris, contains("."))

Selecciona columnas cuyos nombres contienen una cadena de caracteres.

select(iris, ends_with("Length"))

Selecciona columnas cuyos nombres terminan con una cadena de caracteres.

select(iris, everything())

Selecciona todas las columnas.

select(iris, matches("t.*"))

Selecciona columnas cuyo nombre cumple con una expresión regular.

select(iris, num_range("x", 1:5))

Selecciona columna con nombres x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))

Selecciona columnas cuyos nombres están en un grupo de nombres.

select(iris, starts_with("Sepa"))

Selecciona columnas cuyos nombres comienzan con una cadena de caracteres.

select(iris, Sepal.Length:Petal.Width)

Selecciona todas las columnas entre Sepal.Length y Petal.Width (incluyente).

select(iris, -Species)

Selecciona todas las columnas excepto Species.

| Lógica in R - ?Comparison, ?base::Logic | | | |
|---|-----------------------|------------------------|----------------------|
| < | Menor de | != | No equivale a |
| > | Mayor a | %in% | Membrecía de grupo |
| == | equivale a | is.na | Es NA |
| <= | Menos o equivalente a | !is.na | No es NA |
| >= | Mayor o equivalente a | &, , !, xor, any, all | Operadores Booleanos |

Resumir Datos



`dplyr::summarise(iris, avg = mean(Sepal.Length))`

Resume datos a una sola fila de valores.

`dplyr::summarise_each(iris, funs(mean))`

Aplica la función *summary* a cada columna.

`dplyr::count(iris, Species, wt = Sepal.Length)`

Cuenta el numero de valores únicos para cada variable (con o sin ponderación).



Summarise usa **funciones de resumen**, funciones que toman un vector de valores y devuelven un solo valor como:

`dplyr::first`

Primer valor de un vector.

`min`

Valor minimo en un vector.

`dplyr::last`

Ultimo valor de un vector.

`max`

Valor máximo en un vector.

`dplyr::nth`

N-avo valor de un vector.

`mean`

Valor promedio de un vector.

`dplyr::n`

de valores en u vector.

`median`

Valore mediano en un vector.

`dplyr::n_distinct`

valores distintos en un vector

`var`

Varianza de un vector.

`sd`

Desviación estándar de un vector.

`IQR`

IQR de un vector

Group Data

`dplyr::group_by(iris, Species)`

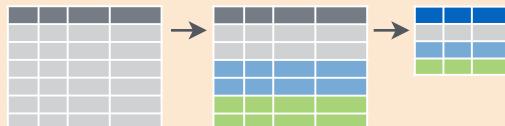
Agrope datos en filas por los valores en Species.

`dplyr::ungroup(iris)`

Remueve la agrupación del data frame.

`iris %>% group_by(Species) %>% summarise(...)`

Calcula una fila separada con el resumen para cada grupo.



Crea Nuevas Variables



`dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)`

Calcula y añade una o mas columnas nuevas.

`dplyr::mutate_each(iris, funs(min_rank))`

Aplica una función de ventana a cada columna.

`dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)`

Calcula una o mas columnas nuevas, borra columnas originales.



Mutate usa **funciones de ventana**, funciones que toman un vector de valores y devuelven otro vector de valores como:

`dplyr::lead`

Copia con valores adelantados por 1.

`dplyr::lag`

Copia con valores atrasados por 1.

`dplyr::dense_rank`

Rangos sin brechas.

`dplyr::min_rank`

Rangos. Empates reciben rango min.

`dplyr::percent_rank`

Rangos con escala del [0, 1].

`dplyr::row_number`

Rangos. Empates van al primer valor.

`dplyr::ntile`

Separa vector en n baldes.

`dplyr::between`

Los valores están entre a y b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

all cumulativo

`dplyr::cumany`

any cumulativo

`dplyr::cummean`

mean cumulativo

`cumsum`

sum cumulativo

`cummax`

max cumulativo

`cummin`

min cumulativo

`cumprod`

prod cumulativo

`pmax`

max por elementos

`pmin`

min por elementos

Combina Conjuntos de Datos

| a | b | | | |
|----|----|----|----|----|
| x1 | x2 | x3 | x1 | x3 |
| A | 1 | T | A | T |
| B | 2 | F | B | F |
| C | 3 | NA | D | T |

`dplyr::left_join(a, b, by = "x1")`

Une filas coincidentes de b a a.

`dplyr::right_join(a, b, by = "x1")`

Une filas coincidentes de a a b.

`dplyr::inner_join(a, b, by = "x1")`

Une datos. Reten solo filas en ambos.

`dplyr::full_join(a, b, by = "x1")`

Une datos. Reten todos los valores, todas las files.

Uniones con filtros

| x1 | x2 |
|----|----|
| A | 1 |
| B | 2 |

| x1 | x2 |
|----|----|
| C | 3 |

`dplyr::semi_join(a, b, by = "x1")`

Todas las filas con coincidencia en b.

`dplyr::anti_join(a, b, by = "x1")`

Todas las filas sin coincidencia en b.

| y | z | | |
|----|----|----|----|
| x1 | x2 | x1 | x2 |
| A | 1 | B | 2 |
| B | 2 | C | 3 |
| C | 3 | D | 4 |

Operaciones de conjuntos

| x1 | x2 |
|----|----|
| B | 2 |
| C | 3 |

| x1 | x2 |
|----|----|
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |

| x1 | x2 |
|----|----|
| A | 1 |

`dplyr::intersect(y, z)`

Filas que aparecen en y y z.

`dplyr::union(y, z)`

Filas que aparecen en una o ambas y y z.

`dplyr::setdiff(y, z)`

Filas que aparecen en y pero no en z.

| x1 | x2 |
|----|----|
| A | 1 |
| B | 2 |
| C | 3 |
| B | 2 |
| C | 3 |
| D | 4 |

| x1 | x2 | x1 | x2 |
|----|----|----|----|
| A | 1 | B | 2 |
| B | 2 | C | 3 |
| C | 3 | D | 4 |

`dplyr::bind_rows(y, z)`

Añade z a y como nuevas filas

`dplyr::bind_cols(y, z)`

Añade z a y como nuevas columnas.

Ojo: distribuye filas por posición.

Estadística descriptiva :: GUÍA RÁPIDA

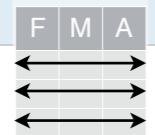


Resumen numérico

Conceptos básicos

VISUALIZA LOS DATOS

```
head(mtcars)  
View(mtcars)  
str(mtcars)
```



Tipos de variables:

- Continua (números reales)
- Discretas (números enteros)
- Ordinales (categorías con orden)
- Nominales (categorías sin orden)

funciones de resumen

Una variable categórica:

```
table(mtcars$cyl)  
prop.table(table(mtcars$cyl))
```

Más de una variable categórica:

```
table(mtcars$cyl, mtcars$vs)  
gmodels::CrossTable(mtcars$cyl,  
mtcars$vs)  
ftable(mtcars$cyl, mtcars$vs,  
mtcars$am)
```

Una variable numérica:

```
mean(mtcars$mpg)
```

Una numérica y una categórica

```
by(mtcars$mpg, mtcars$cyl, mean)  
plyr::ddply(mtcars,"cyl",summarise,  
N=length(mpg),  
mean=mean(mpg),  
sd=sd(mpg))
```

Variables numéricas y categóricas:

```
summary(mtcars)
```

ESTADÍSTICOS DESCRIPTIVOS

Posición central

opcional

```
mean(mtcars$mpg, tr=.2)  
median(mtcars$mpg)  
DescTools:::Mode(mtcars$mpg)
```

Posición no central

opcional

```
quantile(mtcars$mpg,c(.05,.95))
```

Dispersión

opcional

```
var(mtcars$mpg)  
sd(mtcars$mpg)  
IQR(mtcars$mpg)  
WRS2:::trimse(mtcars$mpg, tr=.2)  
msmedse(mtcars$mpg, sewarn=T)
```

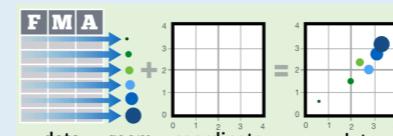
Forma

```
fBasics:::skewness(mtcars$mpg)  
fBasics:::kurtosis(mtcars$mpg)
```

Gráficos

Conceptos básicos

PAQUETE ggplot2



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

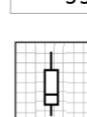
Requerido
No Requerido, se proveen valores iniciales

Una variable numérica:

```
c<-ggplot(mtcars,aes(mpg))
```



```
c+geom_histogram(binwidth=5)  
c+geom_density()
```



```
c<-ggplot(mtcars,aes(x="",y=mpg))  
c+geom_boxplot()
```



```
c+geom_bar()
```

Una variable categórica:

```
c<-ggplot(mtcars,aes(as.factor(am)))
```



```
c+geom_bar()
```

Una variable numérica y una categórica:

```
c<-ggplot(mtcars,aes(x=mpg,  
fill=as.factor(cyl)))
```



```
c+geom_histogram(binwidth=5)  
c+geom_density()
```

Una variable categórica:

```
c<-ggplot(mtcars,  
aes(x=as.factor(am)),  
fill=as.factor(cyl))
```

si no son factores



```
c+geom_bar()  
c+geom_bar(position="dodge")
```

FACETAS

Dividen el gráfico en subgráficos según una o más variables.

```
c<-ggplot(mtcars,aes(x=mpg,  
fill=as.factor(cyl)))
```



```
c+geom_histogram(binwidth=5)+face  
t_grid(~as.factor(am))  
c+geom_histogram(binwidth=5)+face  
t_grid(as.factor(am)~)  
c+geom_histogram(binwidth=5)+face  
t_grid(as.factor(am)~  
as.factor(vs))  
c+geom_histogram(binwidth=5)+face  
t_wrap(~as.factor(am))
```

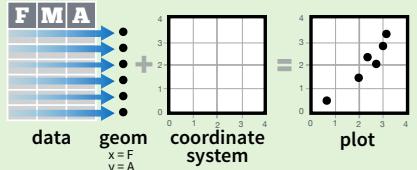
Visualización de Datos usando ggplot2

Guía Rápida

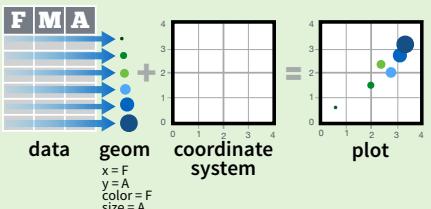


Conceptos Básicos

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 completa este patrón

```
ggplot(data = <DATOS>) +  
  <FUNCION_GEO> (  
    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, complétala 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, 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)) - xmin, xmax, ymin, ymax, 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

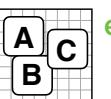
d <- ggplot(mpg, aes(fl))

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

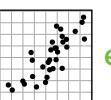
Dos Variables

X Continua, Y Continua

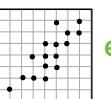
e <- ggplot(mpg, aes(cty, hwy))



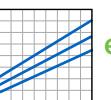
e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust



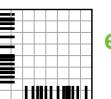
e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size



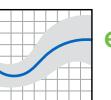
e + geom_point()
x, y, alpha, color, fill, shape, size, stroke



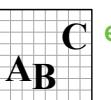
e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight



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



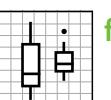
e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

X Discreta, Y Continua

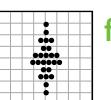
f <- ggplot(mpg, aes(class, hwy))



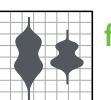
f + geom_col()
x, y, alpha, color, fill, group, linetype, size



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



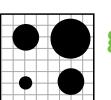
f + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group



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



seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

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



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



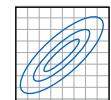
l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width

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



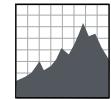
h + geom_density2d()
x, y, alpha, colour, group, linetype, size



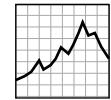
h + geom_hex()
x, y, alpha, colour, fill, size

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



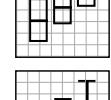
i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

Visualizando el Error

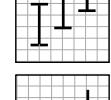
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)



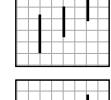
j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size



j + geom_errorbar()
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)



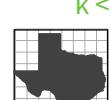
j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size



j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Mapas

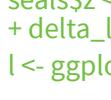
data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))



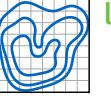
k + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat)
map_id, alpha, color, fill, linetype, size

Tres Variables

seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))



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



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



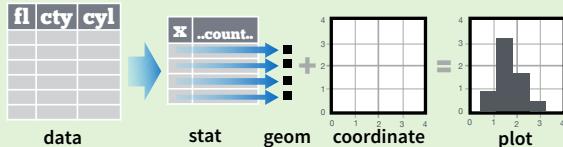
l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width

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**
`i + stat_density2d(aes(fill = ..level..),
geom = "polygon")` **variable que stat crea**

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")`

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

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.



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)

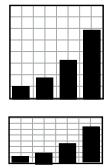
`o <- c + geom_dotplot(aes(fill = ..x..))`
`o + scale_fill_distiller(palette = "Blues")`
`o + scale_fill_gradient(low="red", high="yellow")`
`o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)`
`o + scale_fill_gradientn(colours=topo.colors(6))`
 También: `rainbow()`, `heat.colors()`, `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()`
`+ x`
`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`
`□○△+×◊▽★◆⊕⊗田⊗■●▲◆●○□◊△▽`
`p + scale_radius(range = c(1,6))`
`p + scale_size_area(max_size = 6)` **Usa el radio o el área**

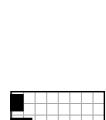
Sistema de Coordenadas

`r <- d + geom_bar()`
`r + coord_cartesian(xlim = c(0, 5))`



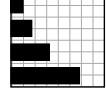
xlim, ylim
 Usa coordenadas cartesianas

`r + coord_fixed(ratio = 1/2)`



ratio, xlim, ylim
 Se fija la relación de aspecto

`r + coord_flip()`



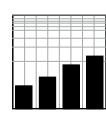
xlim, ylim
 Las coordenadas son volteadas

`r + coord_polar(theta = "x", direction=1)`

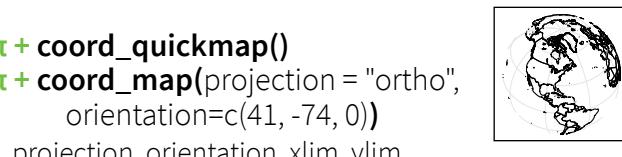


theta, start, direction
 Coordenadas polares

`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))`

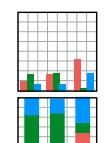
projection, orientation, xlim, ylim

Usa el paquete mapproj para proyectar mapas

Ajustes a las posiciones

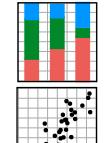
Determina que hacer con Geoms que ocuparían la misma posición en la gráfica.

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



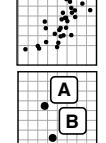
`s + geom_bar(position = "dodge")`

Pone los elementos a lado de cada uno



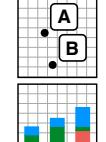
`s + geom_bar(position = "fill")`

Pone los elementos encima de cada uno y usa toda la altura de la gráfica



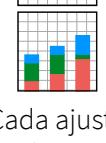
`e + geom_point(position = "jitter")`

Agrega ruido a los elementos



`e + geom_label(position = "nudge")`

Empuja las letras para ver los puntos



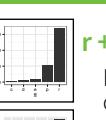
`s + geom_bar(position = "stack")`

Pone los elementos encima de 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



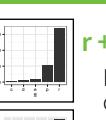
`r + theme_gray()`

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

Facetas

Las Facetas dividen una gráfica en múltiples subgráficas basadas en una o varias variables discretas

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



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

`t + facet_grid(drv ~ fl, scales = "free")`

Cada faceta tiene límites x e y independientes

- `"free_x"` - ajusta el límite del eje x
- `"free_y"` - ajusta el límite del eje y

Use `labeler` para cambiar las etiquetas de las facetas

`t + facet_grid(. ~ fl, labeler = label_both)`
`fl: c fl: d fl: e fl: p fl: r`
`t + facet_grid(fl ~ ., labeler = label_bquote(alpha ^ .(fl)))`
`αc αd αe αp αr`
`t + facet_grid(. ~ fl, labeler = label_parsed)`
`c d e p r`

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 en 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), o 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

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))`

Introducción al lenguaje R : : GUÍA RÁPIDA



Pedir ayuda

PÁGINAS DE AYUDA

```
?mean  
help.search('median')  
help(package='dplyr')
```

para una función

para una palabra o frase

SOBRE UN OBJETO

Resumen de la estructura del objeto

```
str(iris)
```

para un paquete

Utilizar paquetes

Extiende las funciones de R mediante paquetes

```
install.packages('dplyr')  
library(dplyr)
```

activa un paquete

instala un paquete

Accede a una base de datos de R

```
data(iris)
```

Utiliza una función de un paquete

```
dplyr::select
```

Directorio de trabajo

```
getwd()  
setwd(dplyr)
```

muestra tu directorio

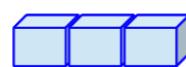
cambia tu directorio

Asignación

```
a<-'group' o a<-1
```

Tipos de datos

vector y factor

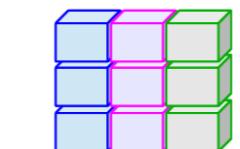


vector

y

factor

data frame o tabla



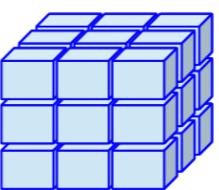
data

frame

o

tabla

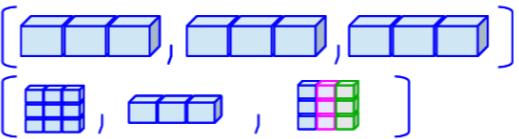
array



filas

columnas

Lista



Listas

CREAR

```
l<-list(x=1:5, y=c('a','b'))
```

SELECCIONAR

| opción | descripción |
|--------|-----------------------|
| I[[2]] | segundo elemento de l |
| I[1] | primer elemento de l |
| I\$x | elemento llamado 'x' |
| I['y'] | elemento llamado 'y' |

Vectores y factores

CREAR

```
x<-c(2,4,5)
```

| opción | descripción |
|-----------------------|--------------------------------|
| c(2,4,5) | une los elementos en un vector |
| 2:6 | crea una secuencia de enteros |
| seq(2,3,by=.5) | crea una secuencia más |
| rep(1:2, times, each) | repite elementos |
| factor(x) | convierte un vector en factor |

SELECCIONAR

Por posición

| opción | descripción |
|-----------|-----------------------------|
| x[4] | el cuarto elemento |
| x[-4] | todos menos el cuarto |
| x[2:4] | elementos desde el 2º al 4º |
| x[c(1,3)] | 1º y 3º elementos |

Por valor o nombre

| opción | descripción |
|------------------|------------------------------|
| x[x==10] | elementos con valor 10 |
| x[x<0] | elementos menores a 0 |
| x[x %in% c(1,3)] | elementos en el conjunto 1,3 |
| x['group'] | elementos con nombre 'group' |

CARACTERÍSTICAS

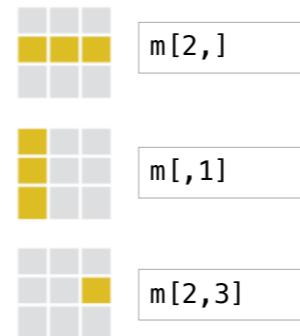
```
unique(x)  
sort(x)  
length(x)
```

Matrices

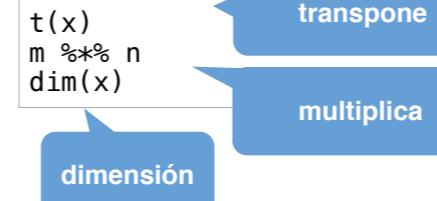
CREAR

```
m<-matrix(x=1:9,nrow=3,ncol=3)
```

SELECCIONAR



CARACTERÍSTICAS

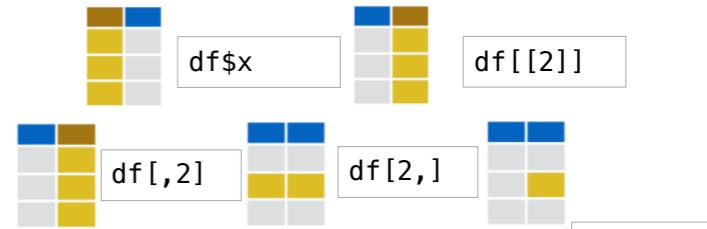


Data frames

CREAR

```
df<-data.frame(x=1:3,y=c('a','b','cs'))
```

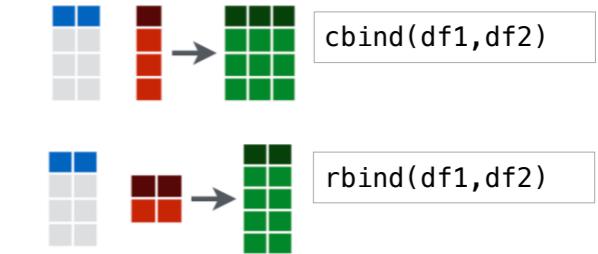
SELECCIONAR



CARACTERÍSTICAS

```
View(df)  
head(df)  
nrow(df) o ncol(df)
```

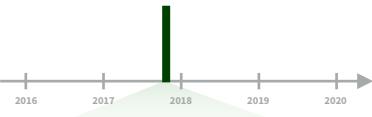
UNIR



Fechas and tiempos con lubridate :: GUÍA RÁPIDA



date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

Un **date-time** es un punto en el tiempo, almacenado como el número de segundos desde 1970-01-01 00:00:00 UTC

```
dt <- as_datetime(1511870400)  
## "2017-11-28 12:00:00 UTC"
```

FILTRA DATE-TIMES (Convierte cadenas o números a date-times)

- Identifica el orden del año (**y**), mes (**m**), día (**d**), hora (**h**), minuto (**m**) y segundo (**s**) en tus datos
- Usa la función inferior cuyo nombre reproduce el orden de tus datos. Cada una acepta una amplia variedad de formatos de entrada.

2017-11-28T14:02:00

ymd_hms(), **ymd_hm()**, **ymd_h()**.
ymd_hms("2017-11-28T14:02:00")

2017-22-12 10:00:00

ydm_hms(), **ydm_hm()**, **ydm_h()**.
ydm_hms("2017-22-12 10:00:00")

11/28/2017 1:02:03

mdy_hms(), **mdy_hm()**, **mdy_h()**.
mdy_hms("11/28/2017 1:02:03")

1 Jan 2017 23:59:59

dmy_hms(), **dmy_hm()**, **dmy_h()**.
dmy_hms("1 Jan 2017 23:59:59")

20170131

ymd(), **ydm()**. ymd(20170131)

July 4th, 2000

mdy(), **myd()**. mdy("July 4th, 2000")

4th of July '99

dmy(), **dym()**. dmy("4th of July '99")

2001: Q3

yq() Q para el trimestre. yq("2001: Q3")

2:01

hms::hms() También lubridate::hms(), hm() and ms(), que devuelve períodos.* hms::hms(sec = 0, min = 1, hours = 2)

2017.5

date_decimal(decimal, tz = "UTC") Q para trimestre. date_decimal(2017.5)



R Studio

now(tzone = "") Hora actual en tz (por defecto al valor del sistema tz). now()

today(tzone = "") Fecha actual en tz (por defecto al valor del sistema tz). today()

fast.strptime() strftime rápido.
fast.strptime('9/1/01', '%y/%m/%d')

parse_date_time() strftime fácil.
parse_date_time("9/1/01", "ymd")

2017-11-28

Un **date** es un día almacenado como el número de días desde 1970-01-01 00:00:00 UTC

```
d <- as_date(17498)  
## "2017-11-28"
```

12:00:00

Un hms es un **time** almacenado como el número de segundos desde 00:00:00

```
t <- hms::as.hms(85)  
## 00:01:25
```

CONSIGUE Y DEFINE COMPONENTES

Usa una función para conseguir un componente. **day(d) #28**
Asigna a una función para cambiar un componente. **day(d) <- 1**
d ## "2017-11-01"

2018-01-31 11:59:59 **date(x)** Componente fecha. **date(dt)**

2018-01-31 11:59:59 **year(x)** Year. **year(dt)**
isoyear(x) El año ISO 8601.
epiyear(x) Año epidemiológico.

2018-01-31 11:59:59 **month(x, label, abbr)** Mes. **month(dt)**

2018-01-31 11:59:59 **day(x)** Día del mes. **day(dt)**
wday(x, label, abbr) Día de la semana.
qday(x) Día del trimestre.

2018-01-31 11:59:59 **hour(x)** Hora. **hour(dt)**

2018-01-31 11:59:59 **minute(x)** Minutos. **minute(dt)**

2018-01-31 11:59:59 **second(x)** Segundos. **second(dt)**

2018-01-31 11:59:59 **week(x)** Semana del año. **week(dt)**
isoweek() SemanaISO 8601.
epiweek() Semana epidemiológica.

2018-01-31 11:59:59 **quarter(x, with_year = FALSE)** Trimestre.
quarter(dt)

2018-01-31 11:59:59 **semester(x, with_year = FALSE)** Semestre. **semester(dt)**

2018-01-31 11:59:59 **am(x)** ¿Es am? **am(dt)**
pm(x) ¿Es pm? **pm(dt)**

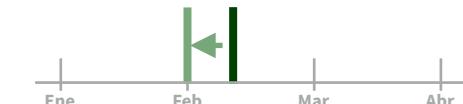
2018-01-31 11:59:59 **dst(x)** ¿Es horario de verano? **dst(dt)**

2018-01-31 11:59:59 **leap_year(x)** ¿Es año bisiesto?
leap_year(dt)

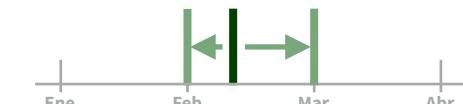
2018-01-31 11:59:59 **update(object, ...)**, simple = FALSE
update(dt, mday = 2, hour = 1)



Redondear Date-times



floor_date(x, unit = "second")
Redondear hacia la unidad más cercana inferior. **floor_date(dt, unit = "month")**



round_date(x, unit = "second")
Redondear a la unidad más cercana. **round_date(dt, unit = "month")**



ceiling_date(x, unit = "second", change_on_boundary = NULL)
Redondear hacia la unidad más cercana superior. **ceiling_date(dt, unit = "month")**

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Devuelve el último día del mes previo. **rollback(dt)**

Stamp Date-times

stamp() Genera una plantilla de una cadena de ejemplo y devuelve una nueva función que aplicará la plantilla a date-times. Además **stamp_date()** y **stamp_time()**.

- Genera una plantilla, crea una función **sf <- stamp("Created Sunday, Jan 17, 1999 3:34")**

- Aplica la plantilla a fechas

sf(ymd("2010-04-05"))
[1] "Created Monday, Apr 05, 2010 00:00"

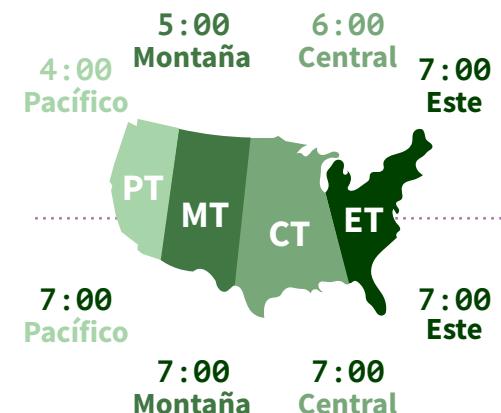
Tip: use a date with day > 12

Zonas horarias

R reconoce ~600 zonas horarias. Cada una incluye la zona horaria, Horario de verano, variaciones históricas del calendario para un área. R asigna **una zona horaria** por vector.

Usa la zona horaria **UTC** para evitar el Horario de Verano.

OlsonNames() Devuelve una lista de nombres válidos de zonas horarias.
OlsonNames()



with_tz(time, tzone = "")
Consigue la misma **misma date-time** en una nueva zona horaria (un nuevo huso horario). **with_tz(dt, "US/Pacific")**

force_tz(time, tzone = "")
Consigue el mismo et the **mismo huso horario** en una zona horaria (una nueva zona horaria). **force_tz(dt, "US/Pacific")**

Matemáticas con Date-times

– Lubridate proporciona tres clases de duraciones para facilitar las operaciones con fechas y date-times



Las operaciones con date-times se basa en un **intervalo temporal**, que se comporta inconsistentemente. Considera como un intervalo temporal se comporta durante:

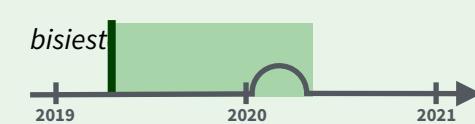
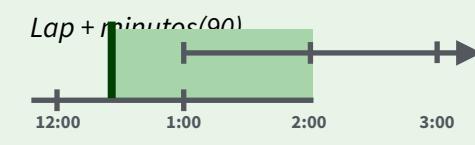
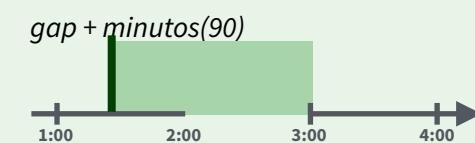
Un día normal
`nor <- ymd_hms("2018-01-01 01:30:00", tz="US/Eastern")`

El comienzo del horario de verano (primavera en adelante)
`gap <- ymd_hms("2018-03-11 01:30:00", tz="US/Eastern")`

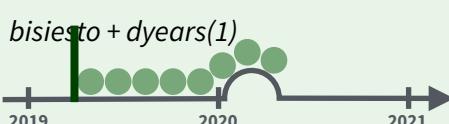
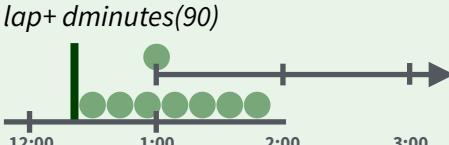
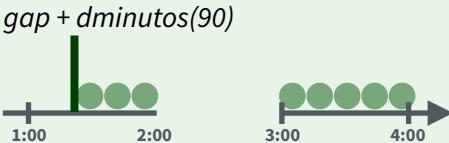
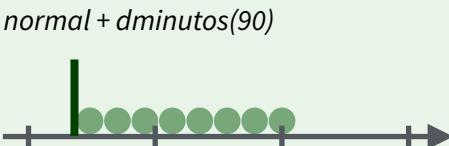
El final del horario de verano (vuelta atrás)
`lap <- ymd_hms("2018-11-04 00:30:00", tz="US/Eastern")`

Años bisiestos y segundos bisiestos
`leap <- ymd("2019-03-01")`

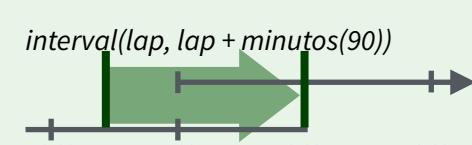
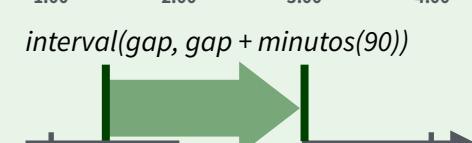
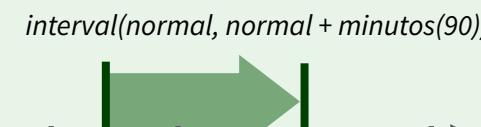
Periodos sigue los cambios en las horas, ignorando irregularidades en el intervalo temporal.



Duraciones sigue el paso del tiempo físico, que se desvía del tiempo de reloj cuando aparecen irregularidades.



Intervalos representan intervalos específicos de un intervalo temporal, delimitado por un comienzo y un final date-times.



No todos los años tienen 365 días por los **días bisiestos**.

No todos los minutos tienen 60 segundos debido a los **minutos bisiestos**.

Es posible crear fechas imaginarias añadiendo **meses**, e.g. Febrero 31

`jan31 <- ymd(20180131)`
`jan31 + months(1)`
`## NA`

%m+% y **%m-%** añadirán fechas imaginarias al último día del mes previo.

`jan31 %m+% months(1)`
`## "2018-02-28"`

add_with_rollback(e1, e2, roll_to_first = TRUE) añadirán fechas al primer día del nuevo mes.

`add_with_rollback(jan31, months(1), roll_to_first = TRUE)`
`## "2018-03-01"`

PERIODOS

Añade o resta periodos para modelizar eventos que ocurren en momentos específicos de reloj, como la apertura de la sesión del NYSE.

Crea un periodo con el nombre de la unidad de tiempo en **plural**, e.g.

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Número de meses Número de días etc.

years(x = 1) x años.
months(x) x meses.
weeks(x = 1) x semanas.
days(x = 1) x días.
hours(x = 1) x horas.
minutes(x = 1) x minutos.
seconds(x = 1) x segundos.
milliseconds(x = 1) x millisegundos.
microseconds(x = 1) x microsegundos
nanoseconds(x = 1) x millisegundos.
picoseconds(x = 1) x picosegundos.

period(num = NULL, units = "second", ...) Un constructor automático de periodos amigable.
`period(5, unit = "years")`

as.period(x, unit) Transforma un intervalo temporal a un periodo, opcionalmente en las unidades especificadas. También **is.period()**. **as.period(i)**

period_to_seconds(x) Convierte un periodo al número "estándar" de segundos del periodo. También **seconds_to_period()**.
`period_to_seconds(p)`

DURACIONES

Añade o resta duraciones para modelizar procesos físicos, como la duración de la batería. Las duraciones se guardan como segundos, la única unidad temporal con una longitud consistente. **Diftimes** son una clase de duraciones disponibles en R base.

Crea una duración con el nombre del periodo con el prefijo **d**, e.g.

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Longitud exacta en segundos Equivalente en unidades comunes

dyears(x = 1) 31536000x segundos.
dweeks(x = 1) 604800x segundos.
ddays(x = 1) 86400x segundos.
dhours(x = 1) 3600x segundos.
dminutes(x = 1) 60x segundos.
dseconds(x = 1) x segundos.
dmilliseconds(x = 1) $x \times 10^{-3}$ segundos.
dmicroseconds(x = 1) $x \times 10^{-6}$ segundos.
dnanoseconds(x = 1) $x \times 10^{-9}$ segundos.
dpicoseconds(x = 1) $x \times 10^{-12}$ segundos.

duration(num = NULL, units = "second", ...) Un constructor automático de periodos amigable. **duration(5, unit = "years")**

as.duration(x, ...) Transforma un intervalo temporal a una duración. También **is.duration()**, **is.difftime()**. **as.duration(i)**

make_difftime(x) Crea difftime con el número especificado de unidades.
`make_difftime(99999)`

INTERVALOS

Divide un intervalo por una duración para determinar su longitud física, divide un intervalo por un periodo para determinar su longitud en unidades de reloj.

Crea un intervalo con **interval()** or **%--%**, e.g.

```
i <- interval(ymd("2017-01-01"), d)
j <- d %--% ymd("2017-12-31")
## 2017-01-01 UTC--2017-11-28 UTC
## 2017-11-28 UTC--2017-12-31 UTC
```

Inicio Final

a **%within%** b Cae el intervalo o date-time en el intervalo a en el b? `now() %within% i`

int_start(int) Accede/define el inicio del date-time de un intervalo. Además **int_end()**.
`int_start(i) <- now(); int_start(i)`

int_aligns(int1, int2) ¿Comparten límites los dos intervalos? También **int_overlaps()**. `int_aligns(i, j)`

int_diff(times) Crea los intervalos que ocurren entre date-times en un vector.
`v <- c(dt, dt + 100, dt + 1000); int_diff(v)`

int_flip(int) Cambia la dirección de un intervalo. También **int_standardize()**. `int_flip(i)`

int_length(int) Longitud en segundos.
`int_length(i)`

int_shift(int, by) Mueve un intervalo adelante/atrás por un intervalo de tiempo. `int_shift(i, days(-1))`

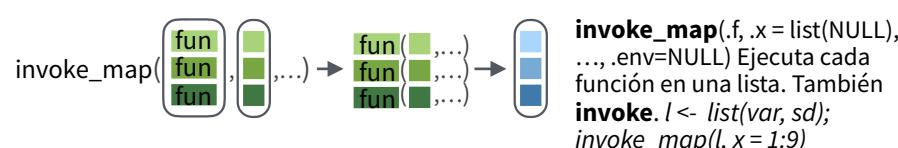
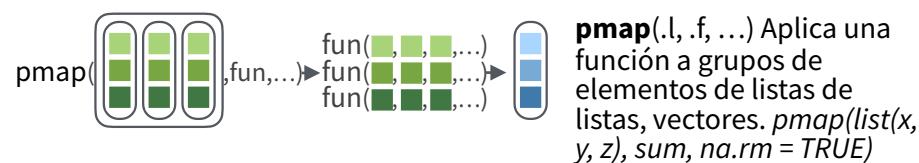
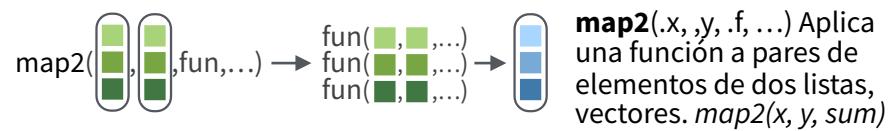
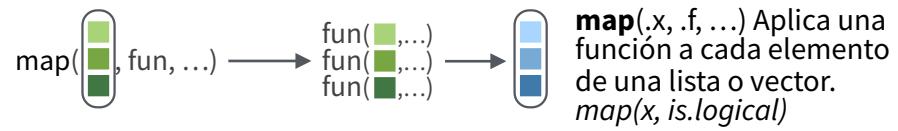
as.interval(x, start, ...) Transforma un intervalo de tiempo a un intervalo con un inicio. También **is.interval()**. `as.interval(days(1), start = now())`

Aplicar funciones con purrr :: GUÍA RÁPIDA



Aplicar Funciones

Las funciones map aplican una función iterativamente a cada elemento de una lista o un vector.



`lmap(x, f, ...)` Aplica una función a cada elemento de una lista o vector.
`imap(x, f, ...)` Aplica .f a cada elemento de una lista o vector y su índice.

SALIDA

`map()`, `map2()`, `pmap()`, `imap` y `invoke_map` devuelven una lista. Usar la versión con el sufijo para devolver el resultado de acuerdo a un tipo o un vector plano, e.g., `map2_chr`, `pmap_lgl`, etc.

Usar `walk`, `walk2`, y `pwalk` para producir efectos alternativos. Cada uno devuelve su entrada de forma invisible.

ATAJOS - en una función purrr:

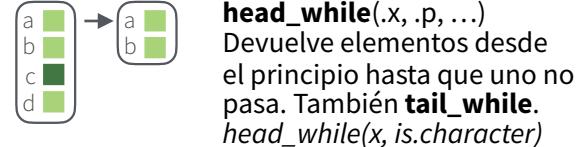
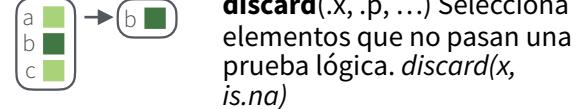
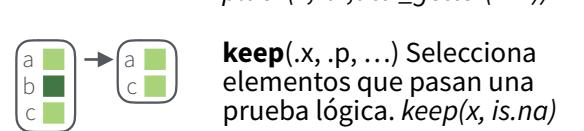
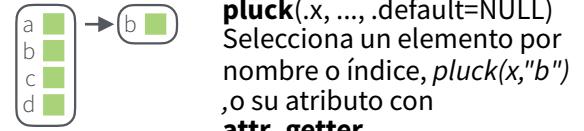
"`name`" pasa a ser `function(x)x$name`. e.g. `map(l, "a")` extrae \$a de cada elemento de l

`~ .x .y` pasa a ser `function(.x, .y).x .y`. e.g. `map2(l, p, ~ .x + .y)` pasa a ser `map2(l, p, function(l, p) l + p)`

`~ ..1 ..2` etc pasa a ser `function(..1, ..2, etc) ..1 ..2` etc e.g. `pmap(list(a, b, c), ~ ..3 + ..1 - ..2)` pasa a ser `pmap(list(a, b, c), function(a, b, c) c + a - b)`

Trabajar con Listas

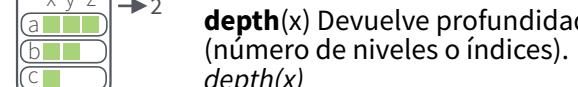
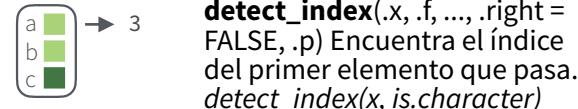
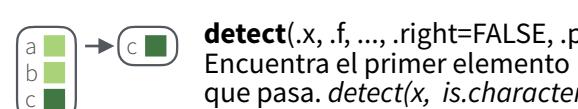
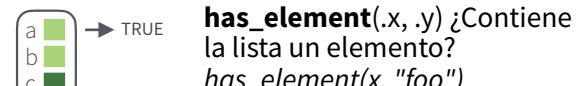
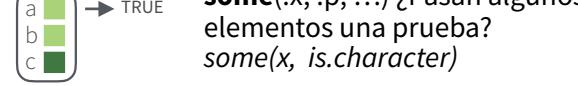
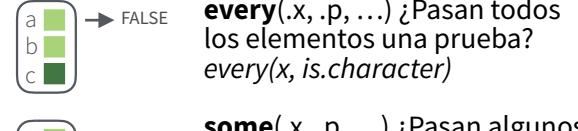
FILTRAR LISTAS



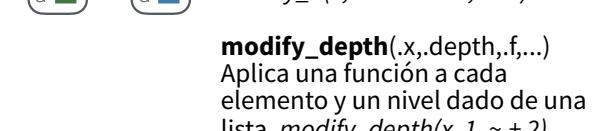
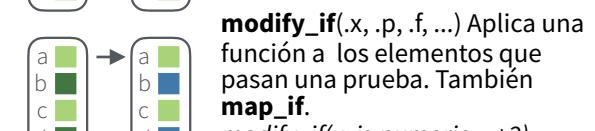
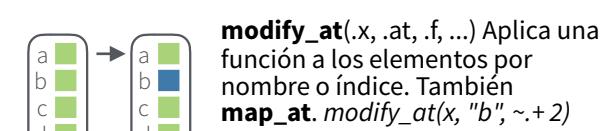
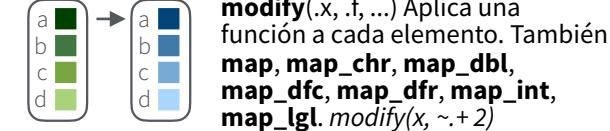
REMODELAR LISTAS



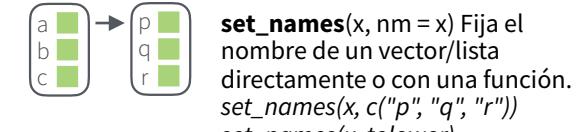
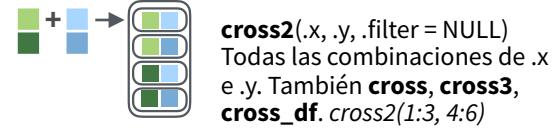
RESUMIR LISTAS



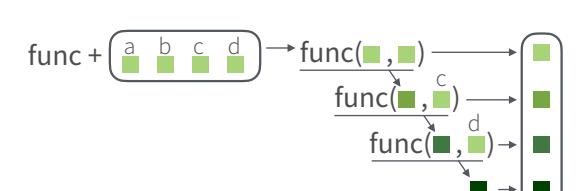
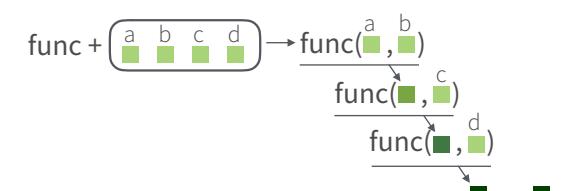
TRANSFORMAR LISTAS



TRABAJAR CON LISTAS



Reducir Listas



`reduce(x, f, ..., .init)` Aplica una función de forma recursiva a cada elemento de un a lista o vector. También `reduce_right`, `reduce2`, `reduce2_right`. `reduce(x, sum)`

`accumulate(x, f, ..., .init)` Reduce, pero también devuelve resultados intermedios. También `accumulate_right`. `accumulate(x, sum)`

`compose()` Compone múltiples funciones.

`lift()` Cambia el tipo de la entrada que una recibe una función. También `lift_dl`, `lift_lv`, `lift_vl`.

`safely()` Ejecuta una expresión n veces.

`negate()` Niega el predicado de una función (a pipe friendly !)

`partial()` Aplica parcialmente una función, completando algunos argumentos.

`possibly()` Modifica la función para devolver el valor por defecto para cualquier tipo de error que ocurra (en vez del error).

Modifica el comportamiento



Datos Anidados

Un **data frame anidado** almacena tablas individuales en las celdas de una tabla organizada más grande.

| "cell" contents | | | |
|-----------------|---------|---------|---------|
| Sepal.L | Sepal.W | Petal.L | Petal.W |
| 5.1 | 3.5 | 1.4 | 0.2 |
| 4.9 | 3.0 | 1.4 | 0.2 |
| 4.7 | 3.2 | 1.3 | 0.2 |
| 4.6 | 3.1 | 1.5 | 0.2 |
| 5.0 | 3.6 | 1.4 | 0.2 |

`n_iris$data[[1]]`

| nested data frame | | | | | |
|-------------------|-------------------|---------|---------|---------|---------|
| Species | data | Sepal.L | Sepal.W | Petal.L | Petal.W |
| setosa | <tibble [50 x 4]> | 7.0 | 3.2 | 4.7 | 1.4 |
| versicolor | <tibble [50 x 4]> | 6.4 | 3.2 | 4.5 | 1.5 |
| virginica | <tibble [50 x 4]> | 6.9 | 3.1 | 4.9 | 1.5 |

`n_iris`

| Sepal.L | Sepal.W | Petal.L | Petal.W |
|---------|---------|---------|---------|
| 6.3 | 3.3 | 6.0 | 2.5 |
| 5.8 | 2.7 | 5.1 | 1.9 |
| 7.1 | 3.0 | 5.9 | 2.1 |
| 6.3 | 2.9 | 5.6 | 1.8 |
| 6.5 | 3.0 | 5.8 | 2.2 |

`n_iris$data[[2]]`

| Sepal.L | Sepal.W | Petal.L | Petal.W |
|---------|---------|---------|---------|
| 6.3 | 3.3 | 6.0 | 2.5 |
| 5.8 | 2.7 | 5.1 | 1.9 |
| 7.1 | 3.0 | 5.9 | 2.1 |
| 6.3 | 2.9 | 5.6 | 1.8 |
| 6.5 | 3.0 | 5.8 | 2.2 |

`n_iris$data[[3]]`

Usa un data frame anidado para:

- Preservar las relaciones entre las observaciones y los subconjuntos de datos
- manipular varias sub-tablas a la vez con las funciones **purrr map()**, **map2()**, o **pmap()**.

Usa un proceso a dos pasos para crear un data frame anidado:

1. Agrupa los data frames en grupos con **dplyr::group_by()**
2. Usa **nest()** para crear un data frame anidado con una fila por grupo

| Species | S.L | S.W | P.L | P.W | Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|---------|-----|-----|-----|-----|
| setosa | 5.1 | 3.5 | 1.4 | 0.2 | setosa | 5.1 | 3.5 | 1.4 | 0.2 |
| setosa | 4.9 | 3.0 | 1.4 | 0.2 | setosa | 4.9 | 3.0 | 1.4 | 0.2 |
| setosa | 4.7 | 3.2 | 1.3 | 0.2 | setosa | 4.7 | 3.2 | 1.3 | 0.2 |
| setosa | 4.6 | 3.1 | 1.5 | 0.2 | setosa | 4.6 | 3.1 | 1.5 | 0.2 |
| setosa | 5.0 | 3.6 | 1.4 | 0.2 | setosa | 5.0 | 3.6 | 1.4 | 0.2 |
| versi | 7.0 | 3.2 | 4.7 | 1.4 | versi | 7.0 | 3.2 | 4.7 | 1.4 |
| versi | 6.4 | 3.2 | 4.5 | 1.5 | versi | 6.4 | 3.2 | 4.5 | 1.5 |
| versi | 6.9 | 3.1 | 4.9 | 1.5 | versi | 6.9 | 3.1 | 4.9 | 1.5 |
| versi | 5.5 | 2.3 | 4.0 | 1.3 | versi | 5.5 | 2.3 | 4.0 | 1.3 |
| virgini | 6.3 | 2.8 | 4.6 | 1.5 | virgini | 6.3 | 2.8 | 4.6 | 1.5 |
| virgini | 6.3 | 3.3 | 6.0 | 2.5 | virgini | 6.3 | 3.3 | 6.0 | 2.5 |
| virgini | 5.8 | 2.7 | 5.1 | 1.9 | virgini | 5.8 | 2.7 | 5.1 | 1.9 |
| virgini | 7.1 | 3.0 | 5.9 | 2.1 | virgini | 7.1 | 3.0 | 5.9 | 2.1 |
| virgini | 6.3 | 2.9 | 5.6 | 1.8 | virgini | 6.3 | 2.9 | 5.6 | 1.8 |
| virgini | 6.5 | 3.0 | 5.8 | 2.2 | virgini | 6.5 | 3.0 | 5.8 | 2.2 |

`n_iris <- iris %>% group_by(Species) %>% nest()`

tidy::nest(data, ..., .key = data)

Para datos agrupados, mueve grupos a celdas como data frames

Desanidar un data frame anidado con **unnest()**:

`n_iris %>% unnest()`

tidy::unnest(data, ..., .drop = NA, .id=NULL, .sep=NULL)

Desanida un data frame anidado.

Flujo Lista Columna

Los data frames anidados usan una **lista columna**, una lista que es almacenada como un vector columna de un data frame. El **flujo de trabajo** para listas de columnas::

1 Crear una lista columna

| Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|
| setosa | 5.1 | 3.5 | 1.4 | 0.2 |
| setosa | 4.9 | 3.0 | 1.4 | 0.2 |
| setosa | 4.7 | 3.2 | 1.3 | 0.2 |
| setosa | 4.6 | 3.1 | 1.5 | 0.2 |
| setosa | 5.0 | 3.6 | 1.4 | 0.2 |
| versi | 7.0 | 3.2 | 4.7 | 1.4 |
| versi | 6.4 | 3.2 | 4.5 | 1.5 |
| versi | 6.9 | 3.1 | 4.9 | 1.5 |
| versi | 5.5 | 2.3 | 4.0 | 1.3 |
| virgini | 6.3 | 2.8 | 4.6 | 1.5 |
| virgini | 6.3 | 3.3 | 6.0 | 2.5 |
| virgini | 5.8 | 2.7 | 5.1 | 1.9 |
| virgini | 7.1 | 3.0 | 5.9 | 2.1 |
| virgini | 6.3 | 2.9 | 5.6 | 1.8 |
| virgini | 6.5 | 3.0 | 5.8 | 2.2 |

```
n_iris <- iris %>%  
  group_by(Species) %>%  
  nest()
```

2 Trabajar con listas columnas

| Species | data | model |
|---------|-----------------|---|
| setosa | <tibble [50x4]> | Call: lm(S.L ~ ., df) Coefs: (Int) S.W P.L P.W 2.3 0.6 0.2 0.2 |
| versi | <tibble [50x4]> | Call: lm(S.L ~ ., df) Coefs: (Int) S.W P.L P.W 1.8 0.3 0.9 -0.6 |
| virgini | <tibble [50x4]> | Call: lm(S.L ~ ., df) Coefs: (Int) S.W P.L P.W 0.6 0.3 0.9 -0.1 |

```
mod_fun <- function(df)  
  lm(Sepal.Length ~ ., data = df)
```

```
m_iris <- n_iris %>%  
  mutate(model = map(data, mod_fun))
```

3 Simplificar la lista columna

| Species | beta |
|---------|------|
| setos | 2.35 |
| versi | 1.89 |
| virgini | 0.69 |

```
b_fun <- function(mod)  
  coefficients(mod)[[1]]
```

```
m_iris %>% transmute(Species,  
  beta = map_dbl(model, b_fun))
```

3. SIMPLIFICAR LA LISTA COLUMNAS (en una columna regular)

Usa las funciones de purrr **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, también con **unnest()** de tidy para reducir una lista columna a una columna regular.

purrr::map_lgl(x, f, ...)

Aplica .f elemento a elemento a .x como .f(x)

`n_iris %>% mutate(n = map(data, dim))`

purrr::map2(x, y, f, ...)

Aplica .f elemento a elemento a .x e .y como .f(x, y)

`m_iris %>% mutate(n = map2(data, model, list))`

purrr::pmap(.l, f, ...)

Aplica .f elemento a elemento a los vectores

guardados en .l

`m_iris %>%`

`mutate(n = pmap(list(data, model, data), list))`

purrr::map_lgl(x, f, ...)

Aplica .f elemento a elemento a .x, devuelve un vector lógico

`n_iris %>% transmute(n = map_lgl(data, is.matrix))`

purrr::map_int(x, f, ...)

Aplica .f elemento a elemento a .x, devuelve un vector entero

`n_iris %>% transmute(n = map_int(data, nrow))`

purrr::map_dbl(x, f, ...)

Aplica .f elemento a elemento a .x, devuelve un vector tipo doble



Trabajar con cadenas con stringr :: GUÍA RÁPIDA

El paquete **stringr** proporciona un conjunto consistente internamente de herramientas para trabajar con cadenas de caracteres i.e. secuencias de caracteres delimitados por comillas.

Detector Coincidencias



str_detect(cadena, patrón) Detecta la presencia de un patrón o la coincidencia en una cadena.
`str_detect(fruit, "a")`



str_which(cadena, patrón) Encuentra los índices de las cadenas que contienen un patrón coincidente.
`str_which(fruit, "a")`



str_count(cadena, patrón) Cuenta el número de coincidencias en una cadena.
`str_count(fruit, "a")`

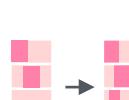


str_locate(cadena, patrón) Localiza las posiciones del patrón que coincide en la cadena. También **str_locate_all**.
`str_locate(fruit, "a")`

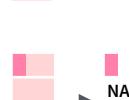
Subconjunto de cadenas



str_sub(cadena, start = 1L, end = -1L) Extrae subcadenas de un vector de caracteres.
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



str_subset(cadena, patrón) Devuelve solo las cadenas que contienen un patrón coincidente.
`str_subset(fruit, "b")`

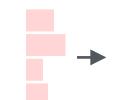


str_extract(cadena, patrón) Devuelve el primer patrón encontrado que coincide en cada cadena, como un vector. También **str_extract_all** para devolver cada patrón coincidente. `str_extract(fruit, "[aeiou]")`

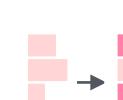


str_match(cadena, patrón) Devuelve el primer patrón encontrado que coincide en cada cadena, como una matriz, con una columna para cada una () agrupado por patrón. También **str_match_all**.
`str_match(sentences, "(a|the) ([^]+)")`

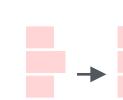
Gestionar Longitudes



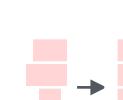
str_length(cadena) Los anchos de las cadenas (i.e. número de puntos de código, suele ser igual al número de caracteres). `str_length(fruit)`



str_pad(cadena, ancho, side = c("left", "right", "both"), pad = " ") Extiende cadenas a un ancho constante. `str_pad(fruit, 17)`



str_trunc(cadena, ancho, side = c("right", "left", "center"), ellipsis = "...") Trunca el ancho de una cadena, eliminando el contenido sobrante. `str_trunc(fruit, 3)`

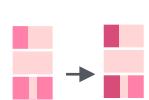


str_trim(cadena, side = c("both", "left", "right")) Elimina los espacios en blanco desde el comienzo y/o el final de una cadena.
`str_trim(fruit)`

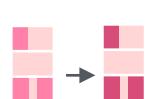
Transformar Cadena



str_sub() <- valor. Reemplaza subcadenas identificadas con `str_sub()` y se asignan al resultado.
`str_sub(fruit, 1, 3) <- "str"`



str_replace(cadena, patrón, reemplazo) Reemplaza el primer patrón coincidente en cada cadena. `str_replace(fruit, "a", "-")`



str_replace_all(cadena, patrón, replacement) Reemplaza todos los patrones coincidentes en cada cadena.
`str_replace_all(fruit, "a", "-")`

A STRING
↓
a string

a string
↓
A STRING

a string
↓
A String

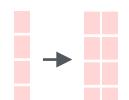
Juntar y Separar



str_c(..., sep = "", collapse = NULL) Une múltiples cadenas en una. `str_c(letters, LETTERS)`



str_c(..., sep = "", collapse = NULL) Colapsa un vector de cadenas en una sola cadena.
`str_c(letters, collapse = "")`



str_dup(cadena, veces) Repite cadenas varias veces. `str_dup(fruit, times = 2)`



str_split_fixed(cadena, patrón, n) Divide un vector de cadenas en una matriz de subcadenas (dividiendo en las ocurrencias del patrón de coincidencia). También **str_split** para devolver una lista de subcadenas.
`str_split_fixed(fruit, " ", n=2)`



glue::glue(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Crea una cadena a partir de cadenas y {expresión} para evaluar. `glue::glue("Pi is {pi}")`

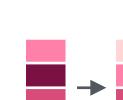


glue::glue_data(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Usa un data frame, lista, o entorno para crear una cadena a partir de cadenas y {expresión} para evaluar. `glue::glue_data(mtcars, "rownames(mtcars) has {hp} hp")`

Ordenar Cadena



str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Devuelve e vector de índices que ordena un vector de caracteres. `x[str_order(x)]`



str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)¹ Ordena un vector de caracteres.
`str_sort(x)`

Ayudas

apple
banana
pear

apple
banana
pear

str_conv(cadena, encoding) Sobre escribe el tipo de codificación de una cadena.
`str_conv(fruit, "ISO-8859-1")`

str_view(cadena, patrón, match = NA) Vista en HTML de la primera coincidencia de expresión regular en cada cadena. `str_view(fruit, "[aeiou]")`

str_view_all(cadena, patrón, match = NA) Vista en HTML de todas las coincidencias de la expresión regular. `str_view_all(fruit, "[aeiou])`

str_wrap(cadena, width = 80, indent = 0, exdent = 0) Envuelve cadenas en párrafos forrajeados de forma atractiva. `str_wrap(sentences, 20)`

¹ Ver bit.ly/ISO639-1 para una lista completa de locales.

Necesitas Saber

Los argumentos de los patrones en stringr son interpretados como expresiones regulares después de cada carácter que ha sido parseado.

En R, se escriben expresiones regulares como *cadenas*, secuencias de caracteres rodeados de comillas dobles ("") o simples ('').

Algunos caracteres no se pueden representar directamente como una cadena en R. Éstos son representados por **caracteres especiales**, secuencias de caracteres que tienen un significado específico., e.g.

Especial Character Representa

| | |
|-----|-------------|
| \\\ | \ |
| \\" | " |
| \n | nueva línea |

Escribe ?"" para ver una lista completa

Por esto, cuando \ aparece en una expresión regular, se tiene que escribir como \\ en la cadena que representa la expresión regular.

Usa **writeLines()** para ver como R ve tu cadena después de que todos los caracteres especiales se han parseado.

```
writeLines("\\")

# \.
```

```
writeLines("\\ is a backslash")
# \ is a backslash
```

INTERPRETACIÓN

Los patrones en stringr son interpretados como regexs. Para cambiar este comportamiento, envuelve el patrón con una de estas opciones:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)

Modifica una regex para ignorar casos, coincide fin de líneas como también fin de cadenas, permite que los comentarios de R dentro de las regex , y/o tienen . coincide cualquier cosa incluyendo \n. str_detect("I", regex("i", TRUE))

fixed() Empareja bytes pero ignorará algunos caracteres que se pueden representar de múltiples formas (rápido). str_detect("\u0130", fixed("I"))

coll() Empareja bytes y usará patrones específicos de los parámetros locales para reconocer caracteres que pueden ser representados en múltiples formas (lento). str_detect("\u0130", coll("I", TRUE, locale = "tr"))

boundary() Empareja límites entre caracteres, separadores de líneas, sentencias o palabras. str_split(sentencias, boundary("word"))



Expresiones Regulares

MATCH CHARACTERS

| Cadena (escribe esto) | regexp (para decir esto) | Coincidencias (que coincide con esto) | Ejemplos |
|--------------------------|-----------------------------|---|------------------|
| a (etc.) | a (etc.) | a (etc.) | see("a") |
| \. | \. | . | see("\.") |
| \! | \! | ! | see("\!") |
| \? | \? | ? | see("\?") |
| \\\ | \\\ | \ | see("\\\\") |
| \(| \(| (| see("\\\\") |
| \) | \) |) | see("\\\\") |
| \{ | \{ | { | see("\\\\") |
| \} | \} | } | see("\\\\") |
| \n | \n | nueva línea (retorno) | see("\n") |
| \t | \t | tab | see("\t") |
| \s | \s | espacios en blanco (\S para no-blancos) | see("\s") |
| \d | \d | dígitos (\D para no-dígitos) | see("\d") |
| \w | \w | cualquier carácter (\W para no-caracteres bordes de palabras) | see("\w") |
| \b | \b | dígitos | see("\b") |
| [:digit:] ¹ | [:digit:] ¹ | letras | see("[:digit:]") |
| [:alpha:] ¹ | [:alpha:] ¹ | letras minúsculas | see("[:alpha:]") |
| [:lower:] ¹ | [:lower:] ¹ | letras mayúsculas | see("[:lower:]") |
| [:upper:] ¹ | [:upper:] ¹ | letras y números | see("[:upper:]") |
| [:alnum:] ¹ | [:alnum:] ¹ | Puntuación | see("[:alnum:]") |
| [:punct:] ¹ | [:punct:] ¹ | letras, números, y puntuación | see("[:punct:]") |
| [:graph:] ¹ | [:graph:] ¹ | caracteres espacio (i.e. \s) | see("[:graph:]") |
| [:space:] ¹ | [:space:] ¹ | espacios y tab (pero no nueva línea) | see("[:space:]") |
| [:blank:] ¹ | [:blank:] ¹ | cada carácter excepto una nueva línea | see("[:blank:]") |
| . | . | | see("") |

Expresiones regulares, o *regexps*, es un lenguaje conciso para describir patrones en cadenas.

see <- function(rx) str_view_all("abc ABC 123 \t.?!\\(){}\n", rx)



[:space:]
nueva línea

[:blank:]
espacio
tab

[:graph:]

[:punct:]

. , : ; ? ! \ | / ` = * + - ^
_ ~ " ' [] { } () < > @ # \$

[:alnum:]

0 1 2 3 4 5 6 7 8 9

[:alpha:]

| [:lower:] | [:upper:] |
|-------------|-------------|
| a b c d e f | A B C D E F |
| g h i j k l | G H I J K L |
| m n o p q r | M N O P Q R |
| s t u v w x | S T U V W X |
| z | Z |

¹ Muchas funciones base de R requieren que las clases se envuelvan en un segundo juego de [], e.g. [[:digit:]]

ALTERNATIVAS

| regexp | coincidencias | ejemplo |
|--------|---------------|---------------|
| ab d | o | alt("ab d") |
| [abe] | una de | alt("[abe]") |
| [^abe] | Excepto | alt("[^abe]") |
| [a-c] | Rango | alt("[a-c]") |

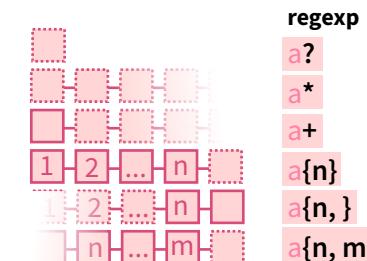
ANCLAS

| regexp | coincidencias | ejemplo |
|--------|-----------------|---------------|
| ^a | comienzo cadena | anchor("^a") |
| a\$ | fin de cadena | anchor("a\$") |

MIRAR ALREDEDOR

| regexp | coincidencias | ejemplo |
|---------|------------------|-----------------|
| a(=?c) | seguido por | look("a(=?c)") |
| a(?!c) | no seguido por | look("a(?!c)") |
| (?<=b)a | precedido por | look("(?<=b)a") |
| (?<!b)a | no precedido por | look("(?<!b)a") |

CUANTIFICADORES



| regexp |
|-----------|
| a? |
| a* |
| a+ |
| a{n} |
| a{n,} |
| a{n, m} |
| a{n,...m} |

| coincidencias |
|---------------|
| cero o uno |
| cero o más |
| uno o más |
| Exactamente n |
| n o más |
| entre n y m |

| Ejemplo |
|-----------------|
| quant("a?") |
| quant("a*") |
| quant("a+") |
| quant("a{2}") |
| quant("a{2,}") |
| quant("a{2,4}") |

GRUPOS

Usa paréntesis para fijar el precedente (orden de evaluación) y crea grupos

| regexp | coincidencia | ejemplo |
|---------|------------------|----------------------|
| (ab d)e | fija precedencia | alt("(ab d)e") abcde |

Usar un número escapado para referir un grupo en paréntesis que ocurre antes en un patrón. Referirse a cada grupo por su orden de aparición

| Cadena | regexp | coincidencia | ejemplo |
|----------------|-------------------|-------------------------|--|
| (escribe esto) | (para decir esto) | (que coincide con esto) | (el resultado es el mismo que ref("abba")) |

\1 (etc.) first () group, etc. ref("(a)(b)\1\2\1") abbaab