

```
library(dplyr)  
library(magrittr)  
  
rladies_global %>%  
filter(city == "Santa Fe" & country = "Argentina")
```



# TALLER DE R PARA CIENCIA DE DATOS

Miércoles 4/Julio/2018 – El Entrevero



# NOTICIAS



<http://rladies-santafe.slack.com>  
Invitación: [santafe@rladies.org](mailto:santafe@rladies.org)

¡Tenemos el soporte de  
RStudio y RConsortium!



¡Gracias a El Entrevero por el lugar,  
los mates, y la buena disposición!



# PARTE 1: PRIMEROS PASOS VARIABLES, OPERACIONES BÁSICAS



# ¿Qué es una variable?

- ✓ En programación, una variable está formada por un espacio de memoria y un nombre simbólico (**identificador**).
- ✓ Ese espacio contiene una cantidad de información conocida o desconocida, es decir un **valor**.
- ✓ El **nombre de la variable** es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa.



# ¡Probemos RStudio!



R script

R console

Graphical output

The screenshot shows the RStudio interface with several components highlighted:

- File bar:** Shows tabs for "01-Introduction.Rnw", "02-SRS.Rnw", and "Data analysis Kalimantan.R". A yellow circle highlights the "File" button.
- Code editor:** Displays R code for biomass calculation per tree, summing values per plot, and creating a nested plot. A yellow box highlights the word "script".
- Console:** Shows R commands for merging datasets and calculating biomass. A yellow box highlights the word "console".
- Environment pane:** Lists global variables like "hil.trees", "kal.plot", "kalimantan", etc., with their respective values. A yellow box highlights the "Environment" tab.
- Plots pane:** Contains a box labeled "R environment" and a box labeled "Biomass estimation per plot with different models". The plot shows biomass (Mg·ha<sup>-1</sup>) for various plots across different models. A red box highlights the "Plots" tab in the pane header.

# ¡Probemos RStudio!



The screenshot shows the RStudio interface with the following components:

- Console pane:** Displays the R startup message, license information, and a user command: `> anio <- 2018`, `> anio`, `[1] 2018`.
- Environment pane:** Shows the variable `anio` with the value `2018`.
- Files pane:** Shows the directory structure: D:/RWorkspace/TallerR, containing a file named `TallerR.Rproj`.

```
anio <- 2018 #Para guardar la variable  
anio #Para imprimir la variable
```

# TIPOS ATÓMICOS DE DATOS

- Carácter: `letra <- "b"`
- Numérico: `miVuelto <- 28,53`
- Entero: `otroAnio <- 1975`
- Lógico: `esCierto <- TRUE`

Los valores lógicos son binarios y sólo pueden representar verdadero o falso.

Se pueden realizar operaciones básicas matemáticas entre los datos!

```
# Sumar dos valores  
5 + 8,3
```

```
# Restar dos valores  
37 - 29
```

```
# Multiplicar valores  
3 * 5 * 2,4
```

```
# Una división  
(10 + 20) / 2
```

```
# Potencia  
# Esto es: dos a la quinta  
2 ^ 5
```

```
# Módulo  
28 %% 6
```

# VECTORES

Listas (secuencias, conjuntos) de datos, donde todos los datos *son del mismo tipo*.

Se crea con la función **c()**

# Creamos un vector de tipo lógico

```
vectorLogico <- c(TRUE, TRUE, FALSE, FALSE)
```

# Creamos un vector de nombres

```
nombres <- c("Grace", "Ada", "Margaret")
```

# Creamos un vector de un tipo de datos, pero vacío

```
vectorVacio <- character(26)
```

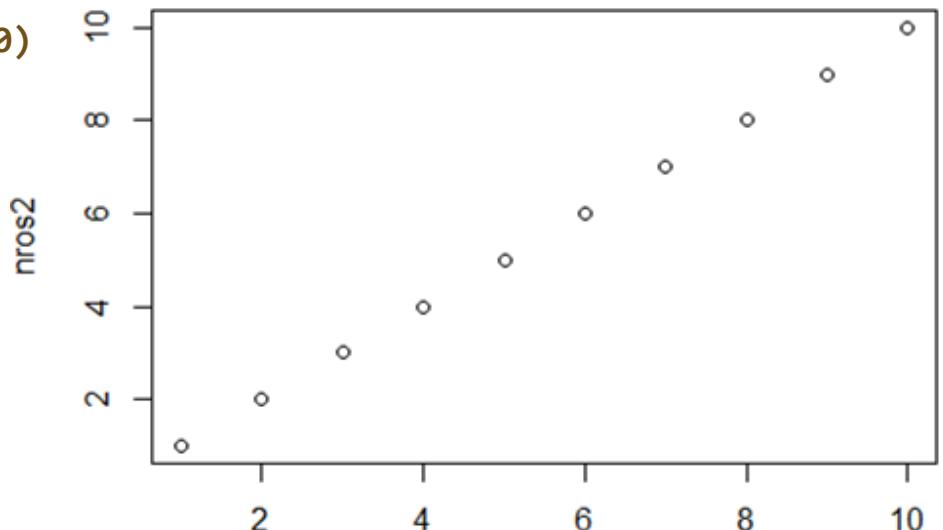
# Creemos una secuencia de números

```
nros1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
nros2 <- c(1:10)
```

```
plot(nros2)
```

Una función es un grupo de instrucciones que toma un "input" o datos de entrada, usa estos datos para computar otros valores y retorna un resultado/producto.



# Grupos de Datos: Matrices

- ✓ Una matriz en programación es justamente lo mismo que una matriz en matemáticas: es un conjunto de vectores con un atributo de dimensión.
- ✓ Es algo parecido a una tabla, pero todos los elementos son del mismo tipo de dato.

```
# Creamos una matriz vacía
matrix1 <- matrix(nrow = 2, ncol = 3)
matrix1
```

```
[,1] [,2] [,3]
[1,] NA   NA   NA
[2,] NA   NA   NA
```

**NA = NOT AVAILABLE**  
 Es un dato que no existe y que no está disponible. Es un operador lógico que puede compararse.

```
# Podemos cargar números automáticamente,
# pero carga los valores POR COLUMNA
matrix(1:6, nrow = 2, ncol = 3)
```

```
[,1] [,2] [,3]
[1,] 1   3   5
[2,] 2   4   6
```

Como no estamos ASIGNANDO el valor de la matriz, R imprime ese valor y luego no podemos accederlo

```
# Podemos crear un vector
mat <- c(1:10)
```

```
# Y darle dimensiones,
# ...para que sea una matriz
dim(mat) <- c(2, 5)
```

```
[,1] [,2] [,3] [,4] [,5]
[1,] 1   3   5   7   9
[2,] 2   4   6   8   10
```

# Grupos de Datos: Matrices

Podemos crear matrices desde vectores de forma más compleja:

- **cbind()** crea una matriz por **columnas**: recibe como parámetro (información) los valores de cada columna.
- **rbind()** crea una matriz por **filas**: recibe como parámetro (información) los valores de cada fila.

```
# Creamos los vectores
x <- c(1:3)
y <- c(1:10)
```

```
# Generamos una matriz por columnas
```

```
cbind(x, y)
      x  y
[1,] 1  1
[2,] 2  2
[3,] 3  3
[4,] 1  4
[5,] 2  5
[6,] 3  6
[7,] 1  7
[8,] 2  8
[9,] 3  9
[10,] 1 10
```

```
# Generamos una matriz por filas
```

```
> x <- 1:3
> y <- 1:10
> rmat <- rbind(x, y)
Warning message:
In rbind(x, y) :
  number of columns of result is not a multiple of vector length (arg 1)
> rmat
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
x     1     2     3     1     2     3     1     2     3     1
y     1     2     3     4     5     6     7     8     9     10
```



```
# Accedemos a una posición específica
rmat[1, 5]
```

x

2

# Grupos de Datos: Listas

- ✓ Tipo especial de vectores que puede tener elementos de distintos tipos.
- ✓ Se puede crear explícitamente con la función `list()`

```
# Creamos una lista sin asignar  
list(4, "julio", TRUE, 2018)
```

```
[[1]]  
[1]  
4
```

```
[[2]]  
[1]  
"julio"
```

```
[[3]]  
[1]  
TRUE
```

```
[[4]]  
[1]  
2018
```

```
# Creamos una lista vacía como vector  
# pero no le damos valores  
vector(mode = "list", length = 4)
```

```
[[1]]  
[1]  
NULL
```

```
[[2]]  
[1]  
NULL
```

```
[[3]]  
[1]  
NULL
```

```
[[4]]  
[1]  
NULL
```

NULL representa un espacio que no tiene nada adentro. Se usa para indicar retornos de funciones cuyos valores son indefinidos.

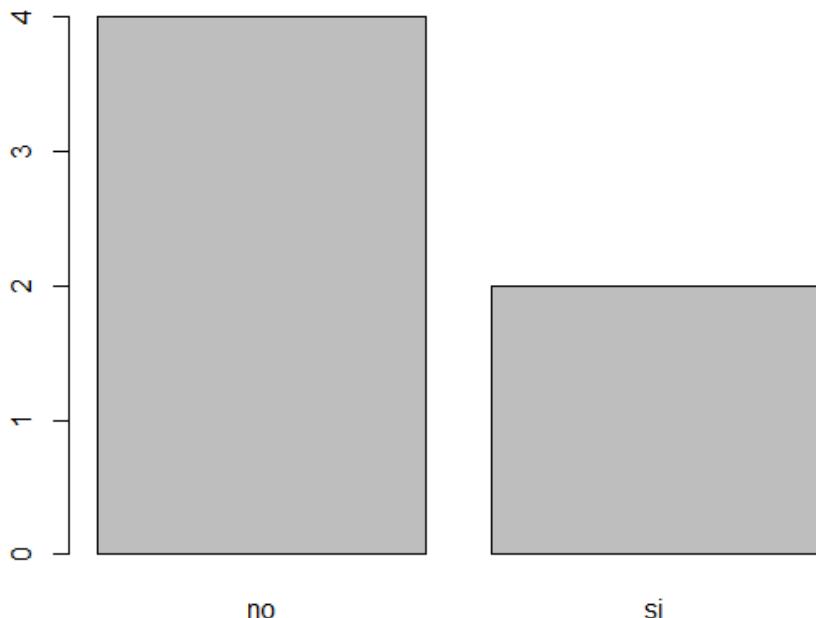
# Grupos de Datos: Factores

- ✓ Representan datos categóricos que pueden estar ordenados o no.
- ✓ Es parecido a un vector de enteros, donde cada entero tiene una etiqueta.
- ✓ Cuando se leen datos automáticamente (por ejemplo, desde un archivo Excel), R crea factores de forma automática cuando encuentra datos que son cadenas de texto.

```
# Creamos un factor con valores "sí" y "no"
f <- factor(c("si", "si", "no", "no", "no", "no"))
f
```

```
[1] si si no no no no
Levels: no si
```

```
# Podemos visualizar los factores
plot(f)
```



# Grupos de Datos: Data Frames

- ✓ Almacenan datos tabulares para R
- ✓ Cada fila representa un registro, y cada columna es un “atributo” o información adicional para ese registro.
- ✓ Similares a los datos en Excel o en bases de datos relacionales.
- ✓ Cada columna **puede ser de un tipo de datos diferente.**

R tiene cargados datos de ejemplos, que se pueden usar para realizar diferentes pruebas. Vamos a intentar usar uno de ellos.

```
# Guardamos en una variable los datos del dataset de automóviles  
df <- mtcars
```

```
# Mostramos los datos  
df
```

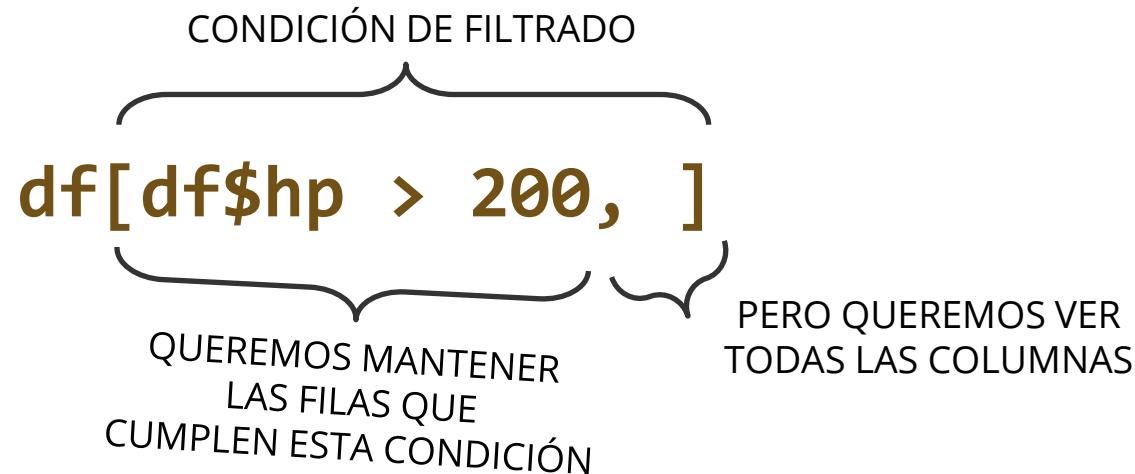
The screenshot shows the RStudio interface. In the center, the Data View pane displays the 'df' dataset with columns: mpg, cyl, disp, hp, drat, wt, qsec, vs. Below it, the Console pane shows the command 'df'. To the left, a green arrow points to the Data View pane. To the right, a red arrow points to the Environment pane, which shows the 'df' object with the description '32 obs. of 11 variables'. A blue arrow points down from the top right towards the Environment pane.

	mpg	cyl	disp	hp	drat	wt	qsec	vs
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	

# Grupos de Datos: Data Frames

¡Las columnas tienen nombres!

Vamos a intentar **filtrar los datos**: quedarnos con aquellos que sólo cumplen ciertas condiciones.



`<mi_data_frame>$<columna_específica>`

Nuestra condición son todas las filas que tienen autos con más de 200 caballos de fuerza.

---

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Duster 360	14.3	8	360	245	3.21	3.570	15.84	0	0	3	4
Cadillac Fleetwood	10.4	8	472	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440	230	3.23	5.345	17.42	0	0	3	4
Camaro Z28	13.3	8	350	245	3.73	3.840	15.41	0	0	3	4
Ford Pantera L	15.8	8	351	264	4.22	3.170	14.50	0	1	5	4
Maserati Bora	15.0	8	301	335	3.54	3.570	14.60	0	1	5	8

# Grupos de Datos: Data Frames

¿Qué sucede si sólo escribimos la condición?

`df$hp > 200`

```
> df$hp > 200
[1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[14] FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
[27] FALSE FALSE TRUE FALSE TRUE FALSE
```

¡Obtenemos un vector de valores lógicos!

- ✓ Hay un valor por cada fila del *data frame*.
- ✓ Si es **true**, indica que la fila cumple la condición.
- ✓ Si es **false**, indica que la fila no cumple la condición.

```
# Podemos guardar la condición como un vector
condicion <- df$hp > 200
```

```
# Y luego usarla para filtrar el data frame
df[condicion, ]
```

¡Obtenemos el mismo resultado que antes!

# Grupos de Datos: Data Frames

¡Las columnas tienen nombres!

## names(df)

La función names() devuelve el nombre de las columnas del data frame, matriz o dato compuesto.

# Nombres de columnas

colnames(df)

```
[1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec"
[8] "vs"    "am"    "gear"  "carb"
```

# Nombres de las filas

rownames(df)

```
[1] "Mazda RX4"           "Mazda RX4 Wag"
[3] "Datsun 710"          "Hornet 4 Drive"
[5] "Hornet Sportabout"   "Valiant"
[7] "Duster 360"          "Merc 240D"
[9] "Merc 230"             "Merc 280"
[11] "Merc 280C"            "Merc 450SE"
[13] "Merc 450SL"           "Merc 450SLC"
[15] "Cadillac Fleetwood"  "Lincoln Continental"
[17] "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"          "Toyota Corolla"
[21] "Toyota Corona"        "Dodge Challenger"
[23] "AMC Javelin"          "Camaro Z28"
[25] "Pontiac Firebird"     "Fiat X1-9"
[27] "Porsche 914-2"         "Lotus Europa"
[29] "Ford Pantera L"        "Ferrari Dino"
[31] "Maserati Bora"         "Volvo 142E"
```

# Podemos cambiar el nombre

colnames(df) <- c("mpg", "cyl", "disp",
 "hpow", "drat", "wt",
 "qsec", "vs", "am",
 "gear", "carb")

colnames(df)

```
[1] "mpg"   "cyl"   "disp"  "hpow"  "drat"  "wt"    "qsec"
[8] "vs"    "am"    "gear"  "carb"
```

# ¡Limpiemos el espacio de trabajo!

Para limpiar las variables cargadas, usamos la función `rm()`.

```
# Así borramos todas las variables  
# ¡A tener cuidado!  
rm(list = ls())  
  
# Sino, tenés que escribir el nombre exacto a borrar  
rm(x, y, df)
```

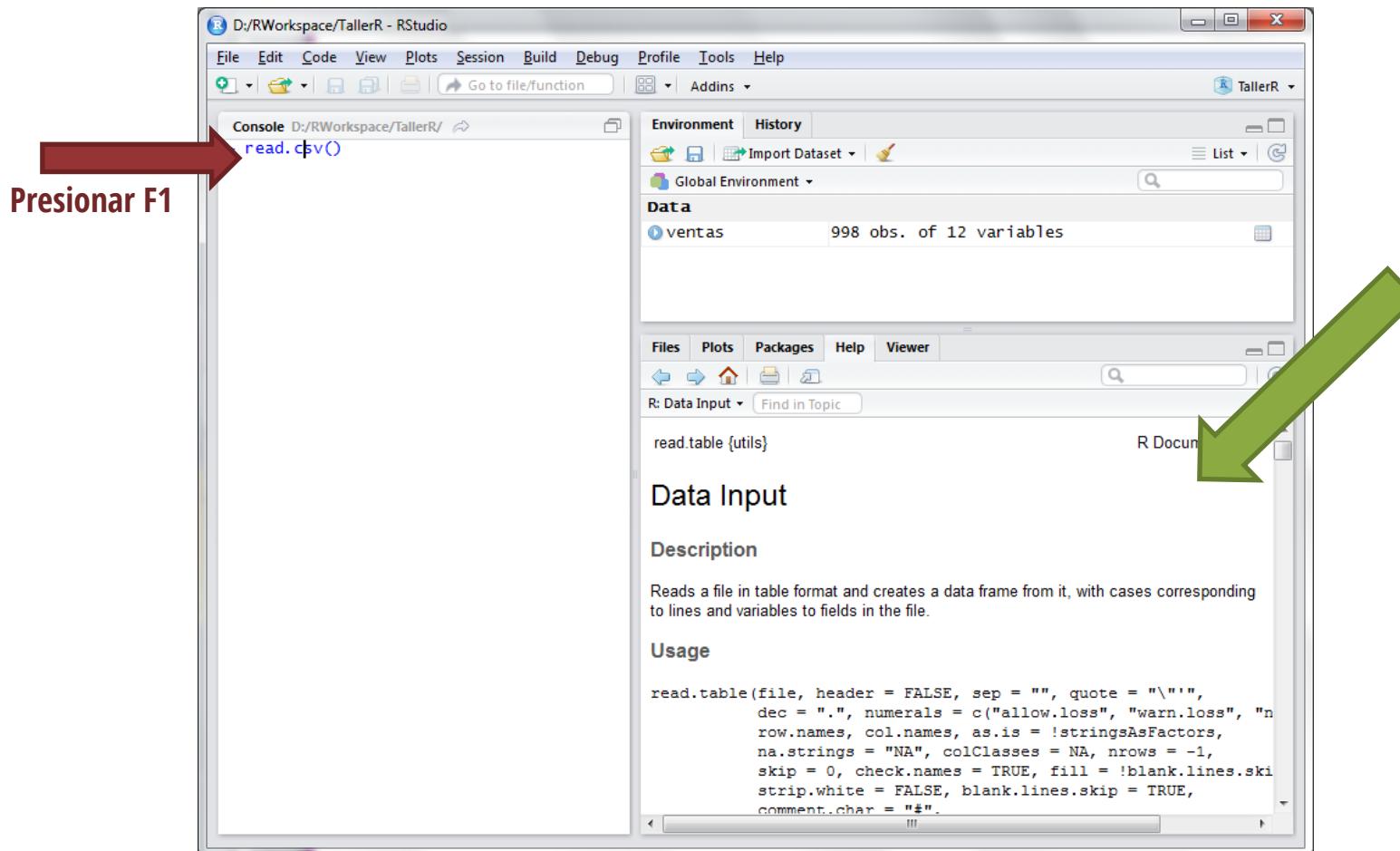


Para limpiar el texto escrito en la consola presionás **CTRL + L**

# Lectura de Datos: CSV

- ✓ Trabajamos con datos de ejemplo, guardados como CSV.
- ✓ Ubicación: <https://tinyurl.com/Csv1TallerR>

Leemos los archivos CSV con una función específica: **read.csv()**



# Lectura de Datos: CSV

- ✓ Trabajamos con datos de ejemplo, guardados como CSV.
- ✓ Ubicación: <https://tinyurl.com/Csv1TallerR>

Leemos los archivos CSV con una función específica: `read.csv()`

```
# Importamos el CSV como un data frame
ventas <- read.csv(file = "https://tinyurl.com/Csv1TallerR",
```

`header = TRUE,`  
`sep = ",")`

`header = TRUE`

Esto significa que la primera fila de cada columna del archivo es realidad el nombre de la columna.

Por ser un archivo CSV, cada valor está separado por una coma.

`sep = ","`

Indicamos desde donde leemos el archivo. Puede ser una ubicación local.

The screenshot shows the RStudio interface with the following details:

- Title Bar:** D:/RWorkspace/TallerR - RStudio
- Menu Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help
- Toolbar:** Includes icons for file operations like Open, Save, Print, and Go to file/function.
- Data View:** A grid showing the first 5 rows of the 'ventas' data frame. The columns are: fecha, producto, precio, tipo\_pago, nombre, and ciudad.
- Environment Panel:** Shows the variable 'ventas' defined in the Global Environment.
- History Panel:** Shows the history of R commands run in the session.
- Imports Panel:** Shows the imported datasets.
- Global Environment:** Shows the global environment with the variable 'ventas'.
- Data View:** Shows the data frame 'ventas' with 998 observations and 12 variables.

# Lectura de Datos: Archivo de Texto

- ✓ Trabajamos con datos de un poema, guardado como texto.
- ✓ Cada renglón del archivo es una línea del poema.
- ✓ Ubicación: <https://tinyurl.com/PoemaTallerR>

**Leemos los archivos TXT línea a línea, con una función específica: `readLines()`**

```
poema <- readLines(con <- file("https://tinyurl.com/PoemaTallerR",
                                encoding = "UTF-8"))
```

```
# Imprimimos el poema
# Vemos que es un vector!
poema
[1] "No te des por vencido, ni aun vencido,"
[2] "no te sientas esclavo, ni aun esclavo;"
[3] "trémulo de pavor, piénsate bravo,"
[4] "y arremete feroz, ya mal herido."
[5] "Ten el tesón del clavo enmohecido"
[6] "que ya viejo y ruin, vuelve a ser clavo;"
[7] "no la cobarde intrepidez del pavo"
[8] "que amaina su plumaje al menor ruido."
[9] "Procede como Dios que nunca llora;"
[10] "o como Lucifer, que nunca reza;"
[11] "o como el robledal, cuya grandeza"
[12] "necesita del agua y no la implora..."
[13] "Que muerda y vocifere vengadora,"
[14] "ya rodando en el polvo, tu cabeza!"
```

```
# Podemos ver una línea particular
poema[3]
[1] "trémulo de pavor, piénsate bravo,"
```

Los textos luego se pueden trabajar  
a través de técnicas de minería de  
texto (o text mining)!



# Lectura de Datos: Bases de Datos

- ✓ Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
- ✓ Hay varios tipos: relacionales, NoSQL o NewSQL.

Cada base de datos usa una **librería o paquete** diferente.

Un **paquete** es una colección de funciones, datos y código R que se almacenan en conforme a una estructura bien definida, fácilmente accesible para R.

Tenemos diferentes paquetes para cada base de datos:

- Oracle: **ROracle**
- MySQL: **RMySQL**
- PostgreSQL: **RPostgreSQL**
- MongoDB: **RMongo** o **mongolite**
- Cassandra: **RCassandra**

Trabajar con una base de datos no es tan directo. Implica:

- » Preparar la conexión.
- » Preparar la consulta u operación.
- » Ejecutar la operación.
- » Si es una lectura, recibir los datos en una variable.



# ¡HAGAMOS UN BREAK!

Volvemos en 30 minutos...



# PARTE 2A: MANIPULACIÓN ESTRUCTURAS DE CONTROL Y BUCLES



# Trabajamos en un Script



The screenshot shows the RStudio interface. A large red arrow points downwards from the title 'Trabajamos en un Script' towards the interface. The interface is divided into several panes:

- R script:** The leftmost pane contains R code for biomass calculation and estimation. A yellow circle highlights the 'File' menu at the top of this pane.
- R console:** The bottom-left pane shows the R command-line interface with some commands entered.
- Environment:** The rightmost pane displays the global environment variables, including `hil.trees`, `kal.plot`, `kalimantan`, `lsi`, `pub`, and `wei`.
- Graphical output:** The bottom-right pane displays a box plot titled "Biomass estimation per plot with different models". The y-axis is labeled "Biomass (Mg·ha<sup>-1</sup>)".

Three boxes with labels are overlaid on the interface:

- R script** (highlighting the script editor)
- R environment** (highlighting the Environment pane)
- Graphical output** (highlighting the box plot)

Pueden usar **CTRL+S** para ir guardando

# Estructuras de Control

- ✓ Las estructuras de control permiten controlar el flujo (u orden) de ejecución de un conjunto de sentencias de R.
- ✓ Permiten evaluar las entradas, y ejecutar código diferente según ciertos casos (determinados por condiciones).

Se ejecutan sobre todo al escribir funciones propias, o en expresiones largas.

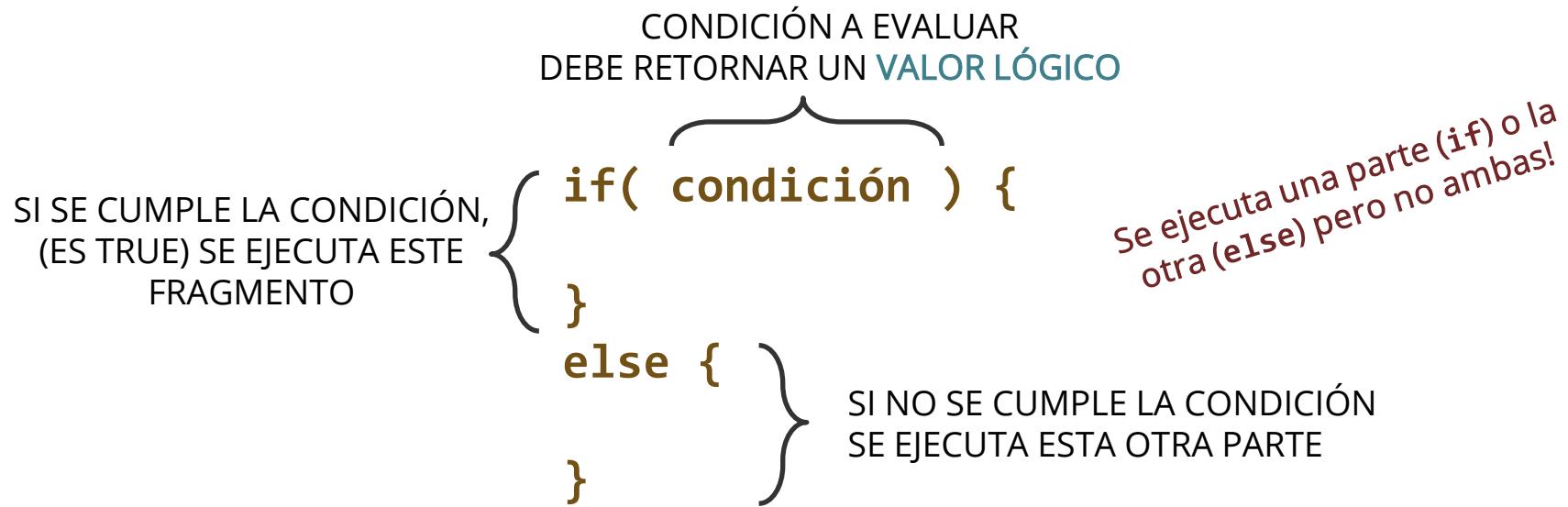


Hay diferentes situaciones:

- ⇒ Una opción o la otra: **if-else**
- ⇒ Repetir veces definidas: **for**
- ⇒ Repetir mientras se cumple una condición: **while**
- ⇒ Dejar de repetir: **break**
- ⇒ O saltar una repetición: **skip**

# Estructuras de Control: if-else

- ✓ Evalúa una condición, y ejecuta código dependiendo de si esa condición se cumple o no se cumple.



```

# --- Probemos un ejemplo ---
# Si hay al menos 100 ventas, decimos algo!
if( nrow(ventas) >= 100 ) {
  "¡Hemos vendido mucho!"
}

# Sino, decimos una cosa diferente
else { "Tenemos muy pocas ventas" }
  
```

```

> if( nrow(ventas) >= 100 ) {
+   "¡Hemos vendido mucho!"
+ } else { "Tenemos muy pocas ventas" }
[1] "¡Hemos vendido mucho!"
  
```

# Estructuras de Control: for

- ✓ Es un ciclo o bucle: repite ciertas líneas de código durante una cantidad conocida de veces.
- ✓ Es el bucle más usado en R.
- ✓ Requiere un **iterador**: una variable que toma valores sucesivos desde un origen hasta un destino. El iterador puede ser un vector.

```
for(<variable> in <rango>) { <instrucciones a repetir> }
```

```
# Para los números impar del 1 al 9
for(i in c(1, 3, 5, 7, 9)) {
  # Imprimo esa fila,
  # pero sólo latitud y longitud
  print(ventas[i, 11:12])

  # Y para la fila anterior
  # muestro el precio
  print(ventas[i-1, 3])
}
```

```
Latitud Longitud
3 46.18806 -123.83
[1] 1200
Levels: 1200 1250 13,000 1800 2100 250 3600 7500 800
Latitud Longitud
5 33.52056 -86.8025
[1] 1200
Levels: 1200 1250 13,000 1800 2100 250 3600 7500 800
Latitud Longitud
7 40.69361 -89.58889
[1] 1200
Levels: 1200 1250 13,000 1800 2100 250 3600 7500 800
Latitud Longitud
9 32.06667 34.76667
[1] 1200
Levels: 1200 1250 13,000 1800 2100 250 3600 7500 800
Latitud Longitud
11 40.71417 -74.00639
[1] 1200
Levels: 1200 1250 13,000 1800 2100 250 3600 7500 800
```



# Estructuras de Control: while

- ✓ Es otro tipo de bucle.
- ✓ Este se repite (**itera**) *mientras* se cumpla una condición lógica (es decir, que la condición sea TRUE).
- ✓ En cada ciclo, la condición se vuelve a evaluar, para decidir si continuar ejecutando el código interno del bucle, dejar de repetirlo.

```
while(<condición>) { <instrucciones a repetir> }
```

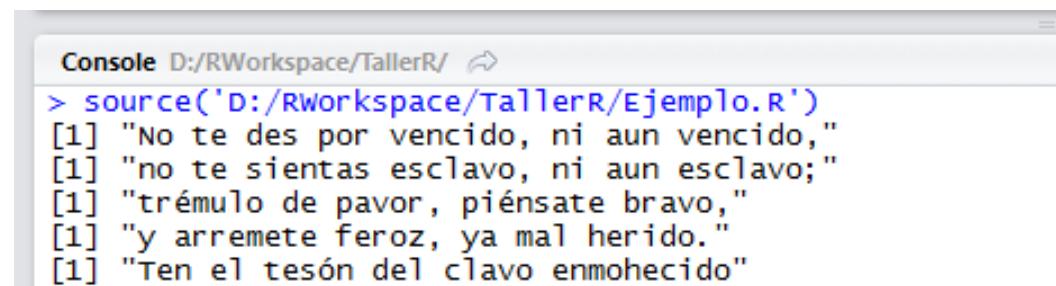
```
# Creamos un contador para poder movernos
contador <- 1
```

El **for** crea el contador de forma “automática” en la declaración entre los paréntesis. Aquí se hace de modo “manual”.

```
# Mientras que la frase contenga “te”
while(grepl("te", poema[contador])) {
  # Imprimimos esa frase
  print(poema[contador])
```

La función **grepl** revisa que el primer texto, esté dentro del segundo texto.

```
# Y avanzamos el contador al sumar 1
contador <- contador + 1
}
```



```
Console D:/RWorkspace/TallerR/ 
> source('D:/RWorkspace/TallerR/Ejemplo.R')
[1] "No te des por vencido, ni aun vencido,"
[1] "no te sientas esclavo, ni aun esclavo;"
[1] "trémulo de pavor, piénsate bravo,"
[1] "y arremete feroz, ya mal herido."
[1] "Ten el tesón del clavo enmohecido"
```

# Funciones de Loop



R siempre apunta a  
**simplificar** el código.

Estas funciones permiten escribir bucles  
como **one-liners**: en un solo renglón y de  
forma muy sencilla.



Trabajan sobre vectores, listas,  
matrices o data-frames.

# Funciones de Loop: lapply()

- ✓ Itera sobre una lista o vector, y evalúa una función en cada elemento.

Sigue los siguientes pasos:

1. Recorre una lista/vector/data-frame, usando (**iterando**) un elemento por vez.
2. Aplica una **función** a cada elemento de la lista.
3. Retorna/devuelve una lista.

El programador define qué función es la que se aplica. Puede ser una existente o una creada por ustedes.

```
# Devolvemos una lista con la cantidad
# de caracteres por renglón del poema
lapply(poema, FUN = nchar)

[[1]]
[1] 38
[[2]] # Si queremos que el retorno sea un vector, lo convertimos
[1] 38 as.integer( lapply(poema, FUN = nchar) )
[[3]]
[1] 33 > as.integer( lapply(poema, FUN = nchar) )
[1] 38 38 33 32 33 40 33 37 34 31 33 34 32 34

[[4]]
[1] 32 # Siempre se va a retornar una lista
[[5]] lapply(list(a = 1:5), mean)
[1] 33
[[6]] $a
[1] 3
[1] 33
```

# Funciones de Loop: sapply()

- ✓ Funciona igual que `lapply()`
- ✓ Intenta simplificar los resultados obtenidos.

Sigue los siguientes pasos:

1. Invoca a `lapply()`
2. Si el resultado es una lista con elementos de largo 1, lo convierte a vector.
3. Si el resultado es una lista donde cada elemento tiene el mismo largo, lo convierte en una matriz.
4. Si lo demás no funciona, devuelve una lista.

```
# Aplicamos lo mismo de antes, y obtenemos un vector!
sapply(poema, FUN = nchar)
```

No te des por vencido, ni aun vencido,	38	no te sientas esclavo, ni aun esclavo;	38
trémulo de pavor, piénsate bravo,	33	y arremete feroz, ya mal herido.	32
Ten el tesón del clavo enmohecido	33	que ya viejo y ruin, vuelve a ser clavo;	40
no la cobarde intrepidez del pavo	33	que amaina su plumaje al menor ruido.	37
Procede como Dios que nunca llora;	34	o como Lucifer, que nunca reza;	31
o como el robledal, cuya grandeza	33	necesita del agua y no la implora...	34
Que muerda y vocifere vengadora,	32	ya rodando en el polvo, tu cabeza!	34

# Funciones de Loop: split()

- ✓ Toma un vector u otros objetos, lo divide en grupos, de acuerdo a un factor o a una lista de factores.
- ✓ Es común combinar **split** con **lapply** u otras funciones.

```
# Aplicamos el split, y guardamos el resultado
splitVentas <- split(ventas, ventas$precio)
# Queremos ver sólo la posición 1
splitVentas[1]
```

Se puede realizar el split con múltiples factores a la vez, pero puede generar nulos!

	fecha	producto	precio	tipo_pago	nombre		ciudad	estado
1	01/02/2009	06:17	Product1	1200	Mastercard	carolina	Basildon	England
2	01/02/2009	04:53	Product1	1200	Visa	Betina		MO
3	01/02/2009	13:08	Product1	1200	Mastercard	Federica	Astoria	OR
4	01/03/2009	14:44	Product1	1200	Visa	e Andrea		Victoria
6	01/04/2009	13:19	Product1	1200	Visa	Gouya	Mickleton	NJ
7	01/04/2009	20:11	Product1	1200	Mastercard	LAURENCE	Peoria	IL
8	01/02/2009	20:09	Product1	1200	Mastercard	Fleur	Martin	TN
9	01/04/2009	13:17	Product1	1200	Mastercard	adam		Tel Aviv
10	01/04/2009	14:11	Product1	1200	Visa	Renee	Tel Aviv	Ille-de-France
11	01/05/2009	02:42	Product1	1200	Diners	Aidan	Chatou	NY
12	01/05/2009	05:39	Product1	1200	Amex	Stacy	New York	Noord-Brabant
13	01/02/2009	09:16	Product1	1200	Mastercard	Heidi	Eindhoven	TX
14	01/05/2009	10:08	Product1	1200	Visa	Sean	Shavano Park	ID
15	01/02/2009	14:18	Product1	1200	Visa	Georgia	Eagle	NJ
16	01/04/2009	01:05	Product1	1200	Diners	Richard	Riverside	Meath
17	01/05/2009	11:37	Product1	1200	Visa	Leanne		Ontario
18	01/06/2009	05:02	Product1	1200	Diners	Janet	Julianstown	Andhra Pradesh
20	01/02/2009	07:35	Product1	1200	Diners	barbara	Ottawa	UT
21	01/06/2009	12:56	Product1	1200	Visa	Hani	Hyderabad	England
22	01/01/2009	11:05	Product1	1200	Diners	Jeremy	Salt Lake City	Cork
23	01/05/2009	04:10	Product1	1200	Mastercard	Janis	Manchester	Gauteng
24	01/06/2009	07:18	Product1	1200	Visa	Nicola	Ballynora	CA
25	01/02/2009	01:11	Product1	1200	Mastercard	asuman	Roodepoort	Ita-Suomen Laani
26	01/01/2009	02:24	Product1	1200	Visa	Lena	Chula Vista	TX
					Lisa	Kuopio	Sugar Land	

# Funciones de Loop: apply()

- ✓ Evalúa una función sobre los márgenes de un array.
- ✓ Suele usarse para aplicar funciones a las filas o columnas de una matriz.
- ✓ No es más rápido que usar un `for()` tradicional,

`apply(x = <mis_datos>, MARGIN = <col_o_filas>, FUN = <mi_funcion>)`

1 indica filas  
2 indica columnas  
c(1,2) es ambos

Puede ser una función existente, una propia o una anónima.



```
# Creamos una matriz con dist. normal
matrix <- matrix(rnorm(200), 20, 10)
# Obtenemos la media de cada columna
apply(matrix, 2, mean)
[1] -0.04466722 -0.10567843  0.15864597 -0.11833802
[5] -0.08971721 -0.02423741  0.20777232 -0.02030479
[9]  0.07720558  0.21040571
```

```
# Función propia llamada 'miFuncion'
miFuncion <- function(entrada, ...) {
  # ¿Qué hace la función?
}
```

```
# Llamamos a apply con función anónima
apply(matrix, 1, function(x) { #operaciones })
```

# PARTE 2B: VISUALIZACIÓN GRÁFICOS BÁSICOS





# Instalación de Paquete: ggplot2

Vamos a trabajar con este paquete para obtener visualizaciones lindas.

```
# Primero, instalamos la librería
# Esto usualmente puede demorar mucho
install.package("ggplot2")
Installing package into 'D:/Mis Documentos/R/win-library/3.4'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/ggplot2_2.2.1.zip'
Content type 'application/zip' length 2784902 bytes (2.7 MB)
downloaded 2.7 MB

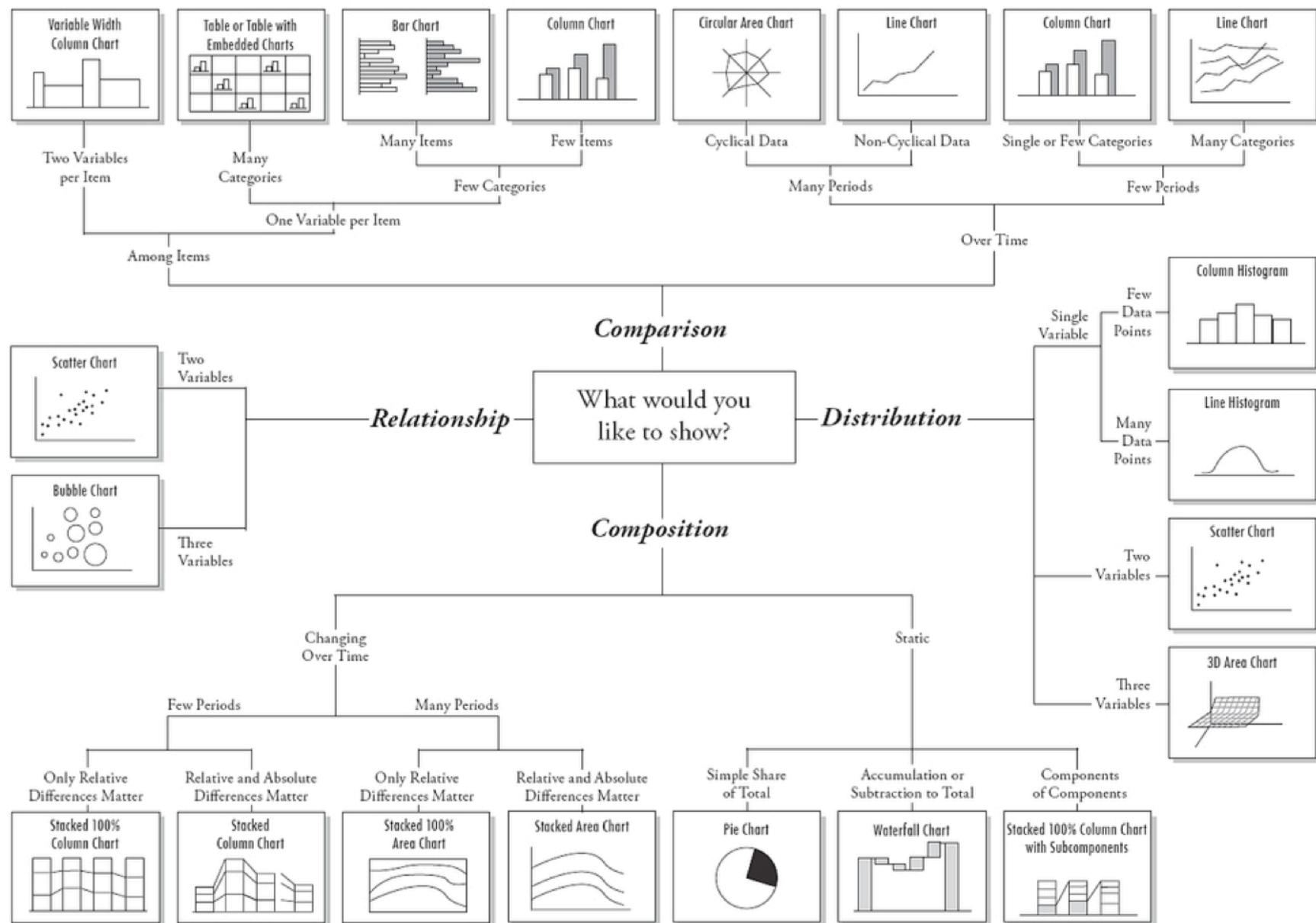
package 'ggplot2' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\ALDO\AppData\Local\Temp\RtmpUpY277\downloaded_packages

# Ahora "importamos" la librería/paquete
# Esto le dice a R que queremos la librería activa
# Es decir, que vamos a trabajar con dicha librería
library(ggplot2)
```

*iAtento a las comillas dobles!  
Instalar REQUIERE comillas,  
pero importar NO LAS USA.*

# ¿Qué gráficos nos conviene usar?



# Gramática de Gráficos

- ✓ En la gramática de los gráficos tienen cinco componentes básicos a partir de los que podemos controlar prácticamente todos los aspectos de un gráfico.
- ✓ Varias de los elementos se crean por defecto.

1. **Especificación de mapeo de variables:** ¿Cómo se quiere dibujar? Incluye colores, tamaños, ejes, líneas, etc. Se especifica con `aes()`.
2. **Un sistema de coordenadas:** proyecta los datos al espacio. Por defecto son coordenadas cartesianas, pero se puede cambiar.
3. **Escala:** la relación entre los números y el espacio. Por defecto, los ejes caen en el punto (0,0).
4. **Transformación estadística:** ¿cómo se transforman los datos para visualizarlos? Por ejemplo, contar frecuencias, calcular medias, etc.
5. **Elemento geométrico:** representa a los datos en el gráfico. Se pueden usar varios elementos o combinarlos en el mismo gráfico.

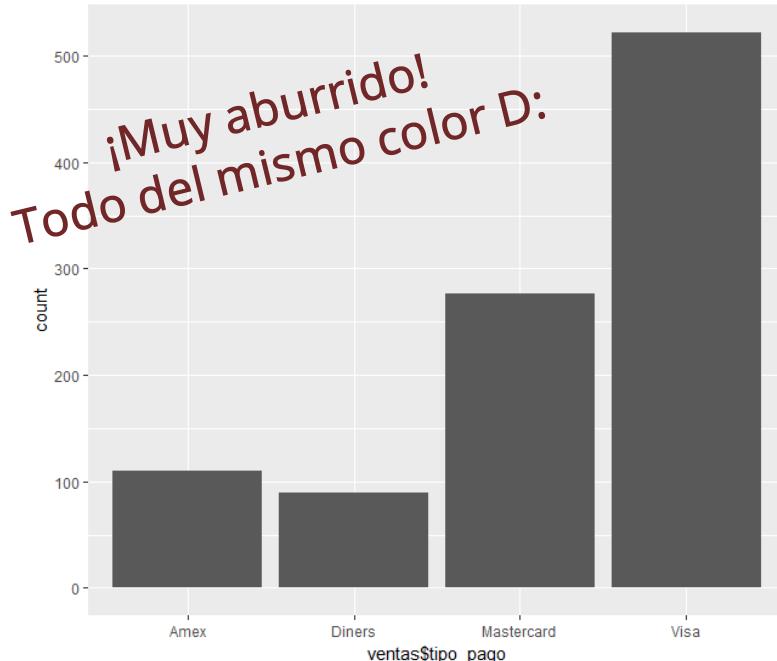


# Ejemplo: Gráfico de Barras

- Queremos mostrar cuántas ventas hay para cada tipo de pago.
- Elegimos un gráfico de barras

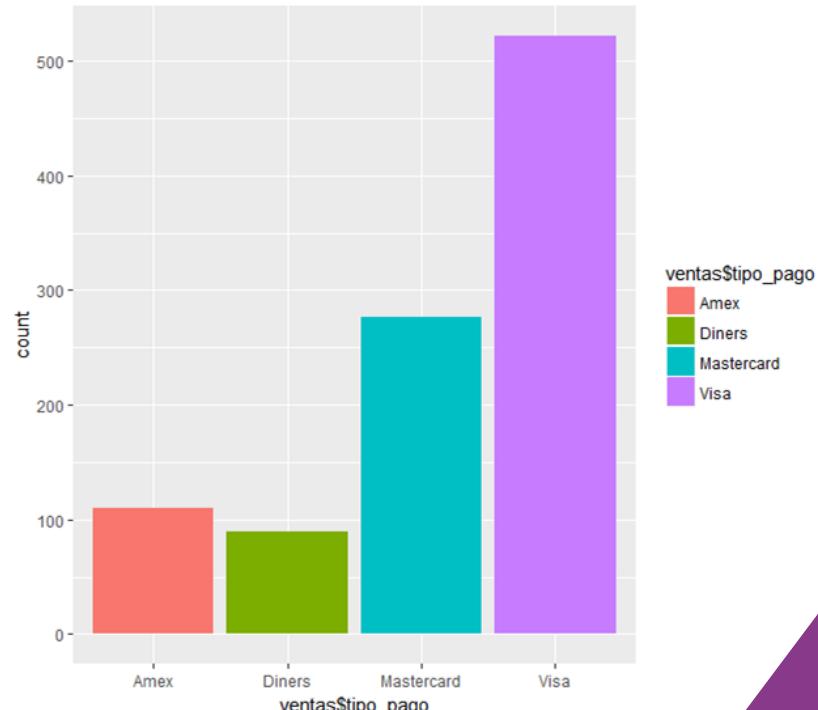
```
# Dibujamos un gráfico de barras
ggplot(ventas, aes(x=ventas$tipo_pago))
  + geom_bar(stat = "count")
```

Esto indica que las barras se van a generar contando la cantidad de filas que hay para cada tipo de pago.



```
# Dibujamos un gráfico de barras
ggplot(ventas, aes(x=ventas$tipo_pago,
                    fill = ventas$tipo_pago))
  + geom_bar(stat = "count")
```

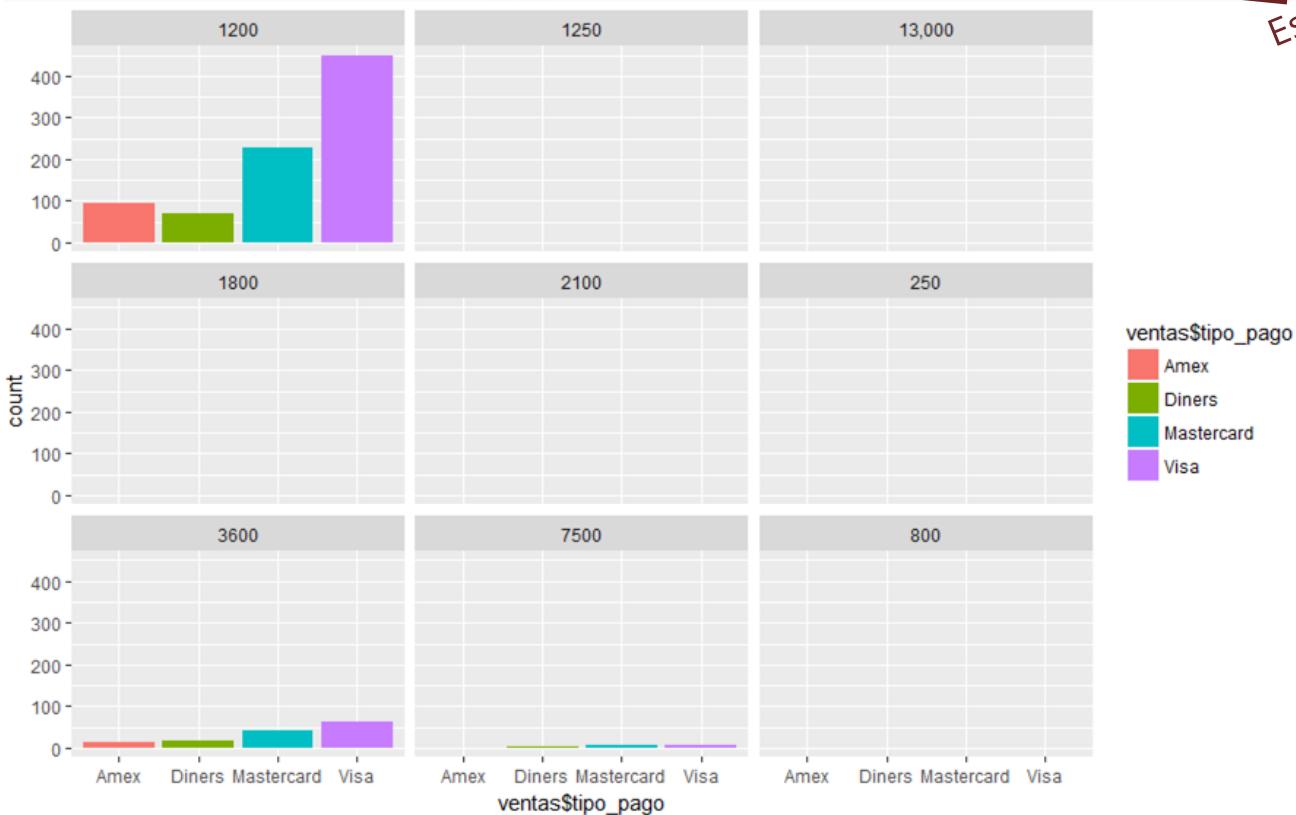
Le indica al gráfico que ponga tantos colores como tipos de ventas encuentre.



# Ejemplo: Agrupando Datos

- Queremos modificar el gráfico anterior!
- Buscamos hacer un gráfico por cada monto pagado.

```
# Agreguemos algo de información al gráfico previo
ggplot(ventas, aes(x=ventas$tipo_pago, fill = ventas$tipo_pago))
  + geom_bar(stat = "count")
  + facet_wrap(~ ventas$precio)
```



Esto indica que queremos un gráfico separado por cada precio, y que los queremos acomodados como una grilla.



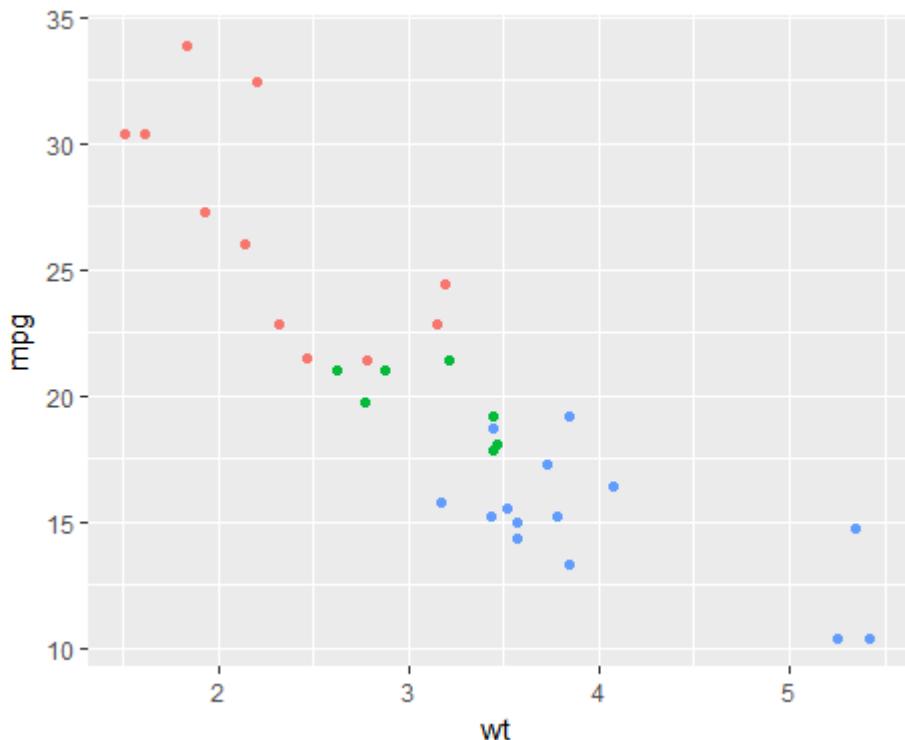
El gráfico anterior no se modificó: R creó un gráfico nuevo con las especificaciones que le dimos ahora!

# Ejemplo: Combinando Gráficos

- Volvemos a trabajar con los datos de los autos.
- ¿Hay alguna relación entre el peso del auto y el consumo, según cilindrada?

```
# Obtenemos los autos  
autos <- mtcars
```

```
# Hacemos un gráfico de puntos entre WT (peso del auto) y mpg (el consumo)  
# Pero, convertimos la cilindrada a factor, sin afectar el dataset  
ggplot(autos, aes(x = wt, y = mpg, col = factor(cyl))) + geom_point()
```



Con esto, indicamos que trate a la cilindrada como factores, y no como una columna de enteros.

factor(cyl)

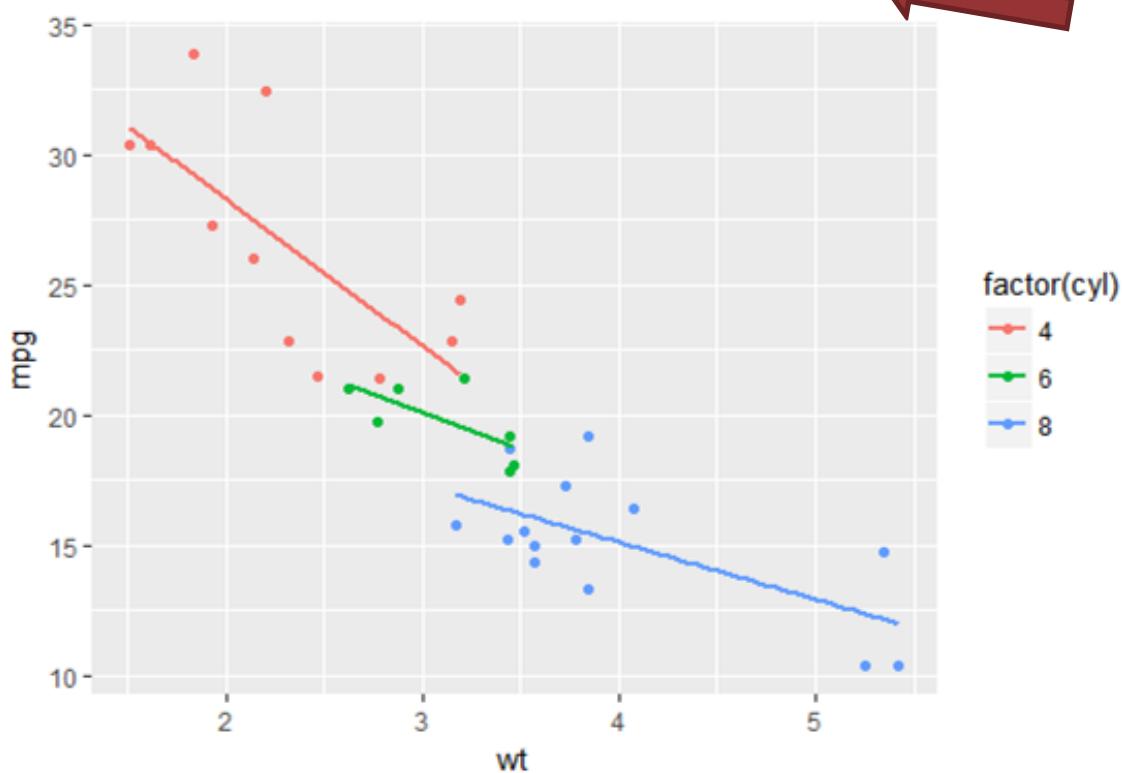
•	4
•	6
•	8

Al agrupar de esta forma, ya se genera un color de forma automática.

# Ejemplo: Combinando Gráficos

- Volvemos a trabajar con los datos de los autos.
- ¿Hay alguna relación entre el peso del auto y el consumo, según cilindrada?

```
# Hacemos un gráfico de puntos entre WT (peso del auto) y mpg (el consumo)
# Pero, convertimos la cilindrada a factor, sin afectar el dataset
ggplot(autos, aes(x = wt, y = mpg, col = factor(cyl))) + geom_point() +
  # Pero agregamos líneas de tendencia con mínimos cuadrados
  stat_smooth(method = "lm", se = F)
```



Esta línea hace que se calcule una  
línea de tendencia de forma  
automática.

# ¿CÓMO CONTINUAMOS?



# Ejercicios de Práctica

- Pueden encontrar ejercicios básicos en: <https://goo.gl/eL6J1z>
- Las diapositivas y los datasets usados se subirán a Drive y GitHub.
- La grabación del taller estará en YouTube (tengan paciencia!).
- Todos los links serán comunicados por Meetup.

¿Qué pasa si surgen dudas o preguntas?



<http://rladies-santafe.slack.com>  
Invitación: [santafe@rladies.org](mailto:santafe@rladies.org)



# ¡Tenemos Planes a Futuro!



**AGOSTO: Charla de Economía + Sociales**  
Casos de Estudio y Aplicaciones Reales



**SEPTIEMBRE: Taller de Algoritmos ML**  
Ejemplos básicos con práctica y ejercicios. Comparación a Python.  
¡Necesitamos voluntarios para mesa de difusión!



**OCTUBRE: Taller Colaborativo**  
¡Analicemos juntos un dataset!  
Colaboremos aplicando lo aprendido.



**NOVIEMBRE: ¡Sorpresa!**  
Pero estamos buscando voluntarios ;)

# ¡GRACIAS POR VENIR!®

