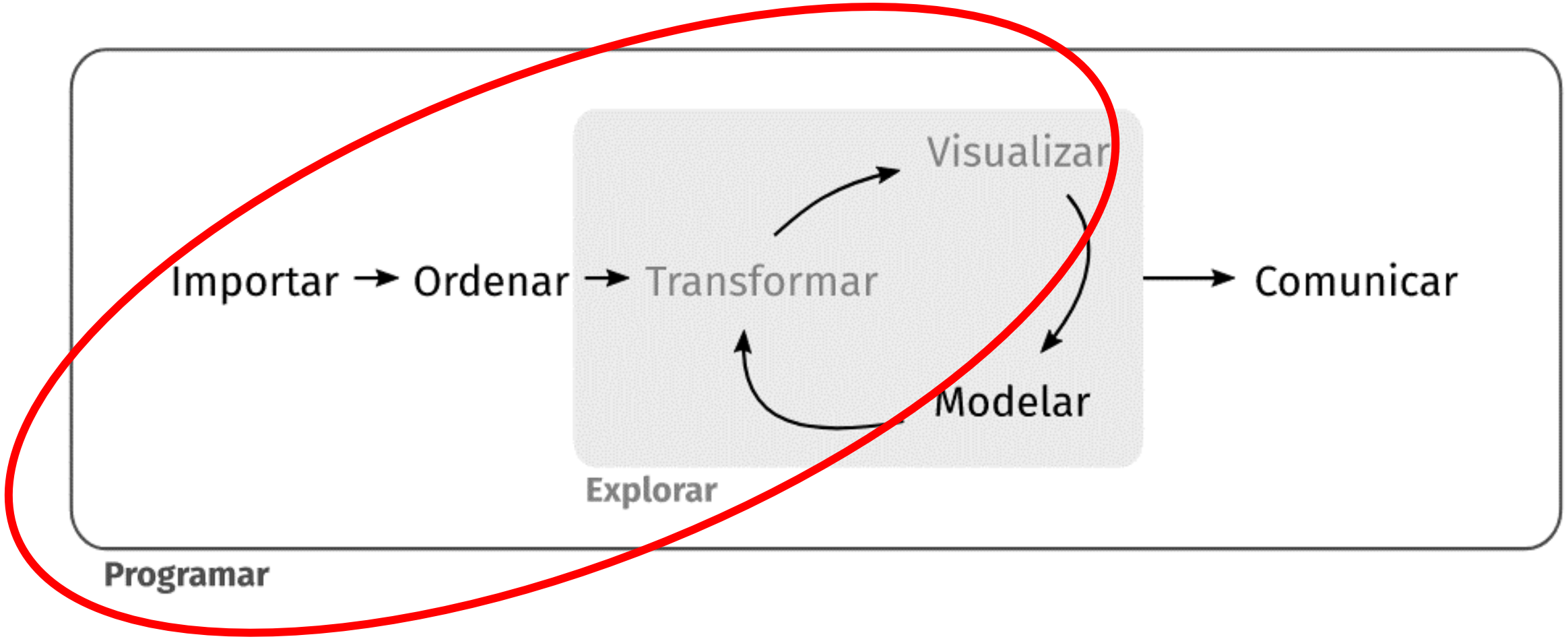
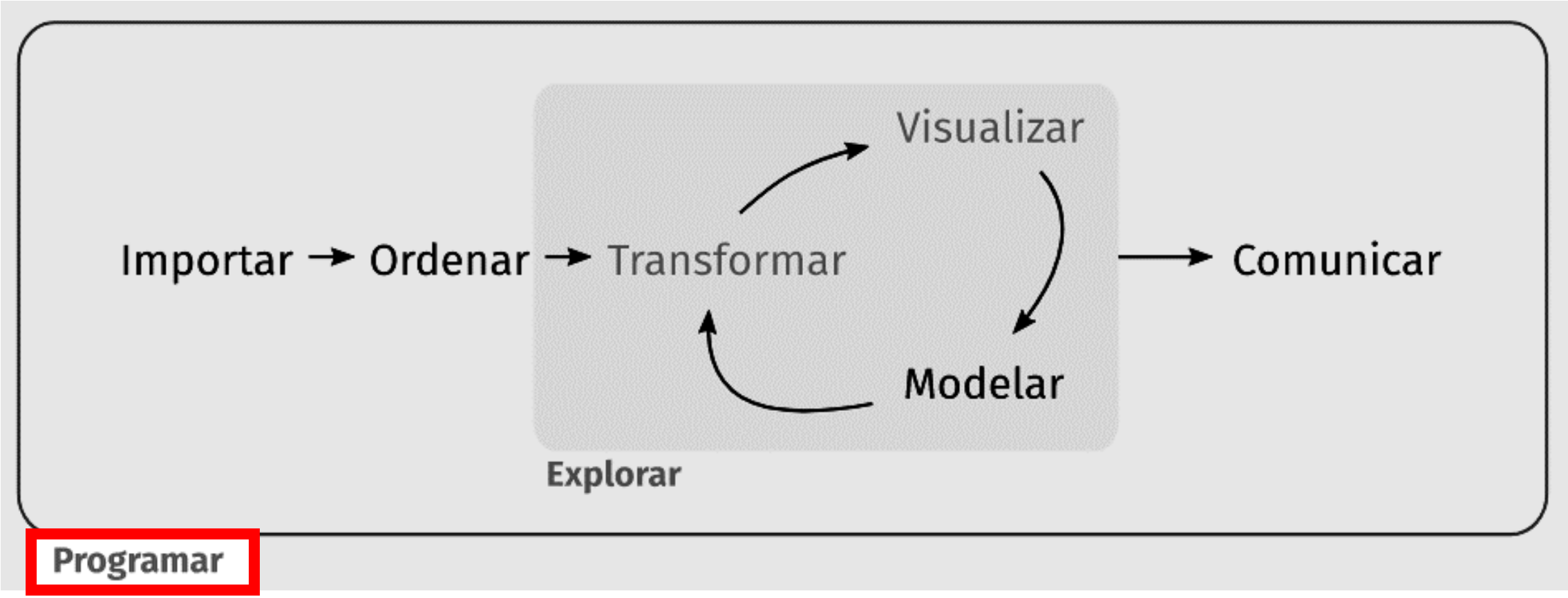
¿De qué se trata este curso?



https://es.r4ds.hadley.nz/explora-introduccion.html


Importar
Ordenar
Transformar
Visualizar
Modelar
Comunicar
Explorar
Programar

Qué es

Lenguaje y plataforma

- Lenguaje de programación estadística
- Herramienta de visualización de datos
- “Open source”

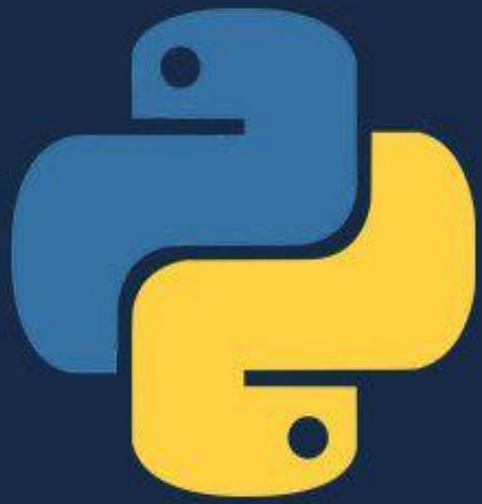
Ecosistema

- 12.000+ librerías gratuitas disponibles en CRAN
- Escalable (Big data)
- Muchas aplicaciones e integraciones con otras plataformas

Comunidad

- 2,5+ millones de usuarios
- Enseñando en la mayoría de las universidades
- Muchos y diversos grupos de usuarios a nivel mundial


VS



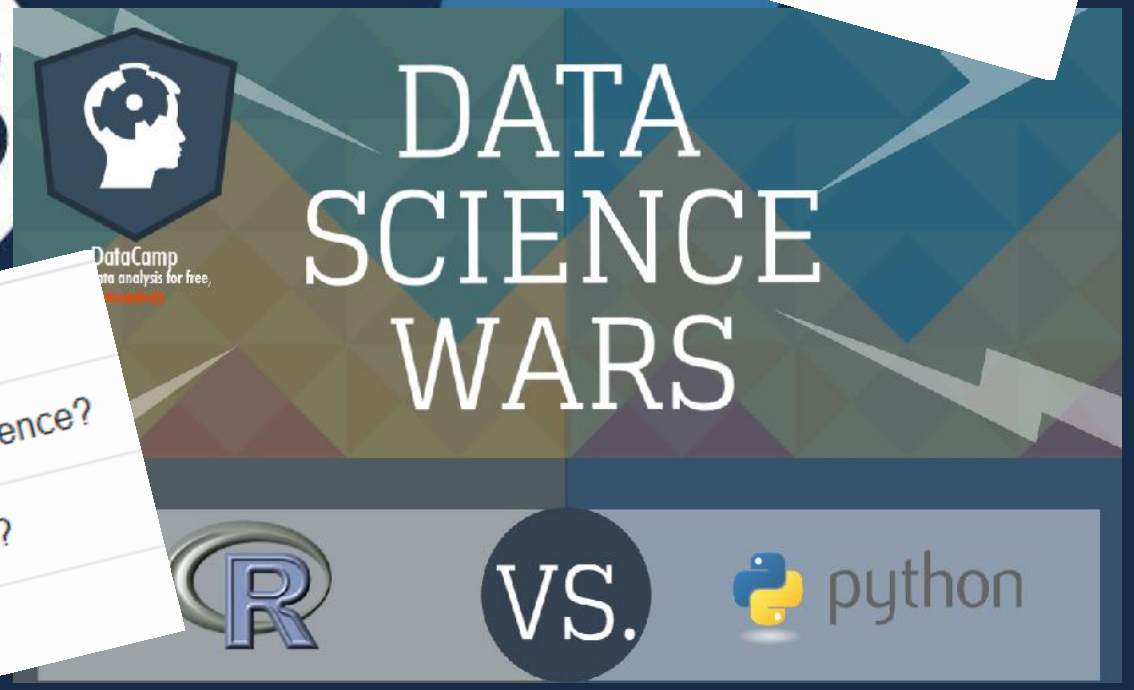
R Vs Python: What's the Difference?

R VS PYTHON 2018 ¿CUÁL ES EL MEJOR?

Publicado por: Rosana Ferrero Categoría: Data Science +R No hay comentarios

R y Python son dos de los lenguajes favoritos de los *Data Scientists*.

VS



Preguntas relacionadas

Is Python or R better for data science?

Should I learn Python or R first?

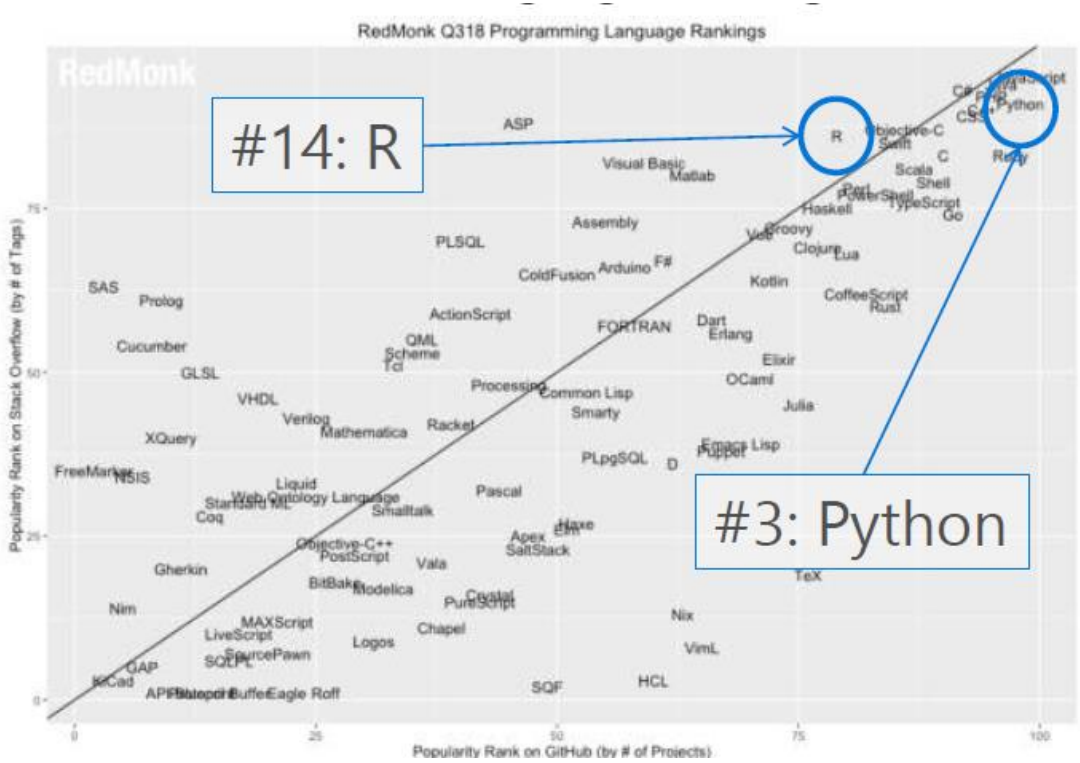
Is R similar to Python?

¿Qué dicen los usuarios?

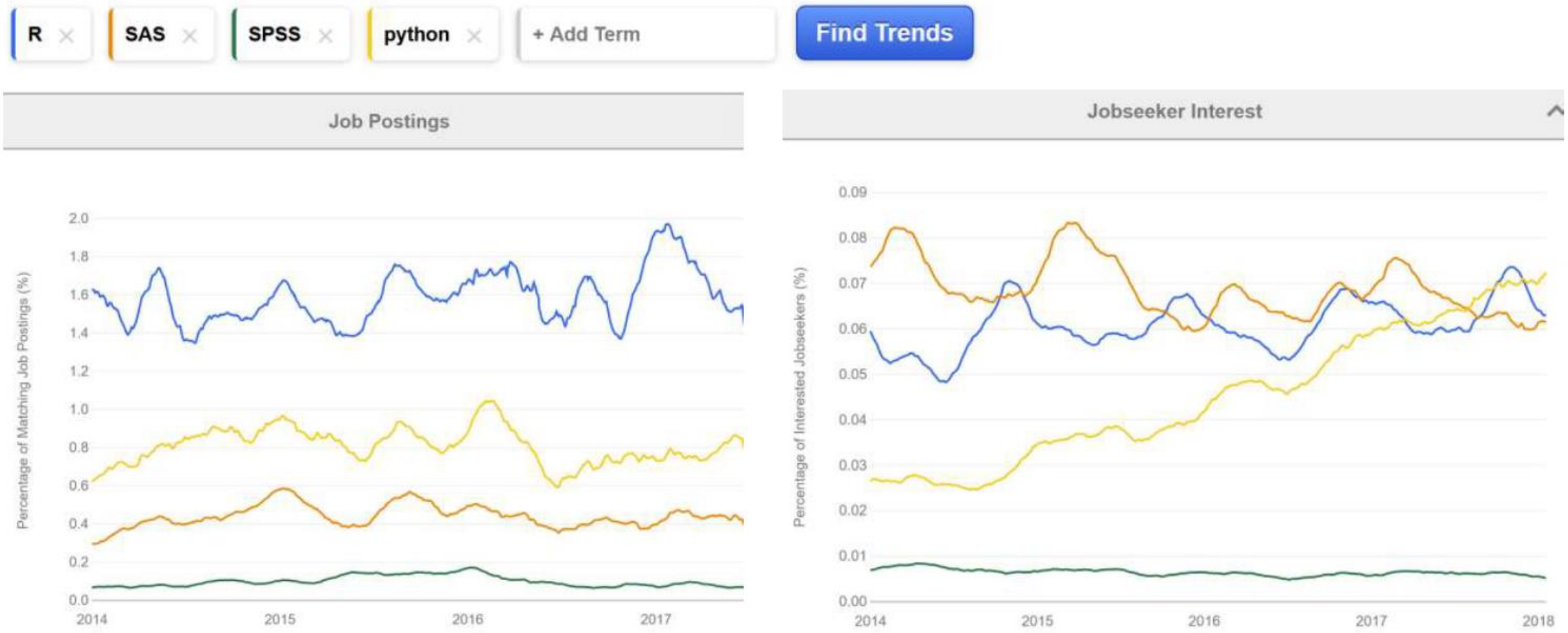
[IEEE Spectrum, Julio 2018]

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1

[Redmonk, Junio 2018]



¿Qué dicen los empleadores?



https://www.indeed.cl/?r=us

The reality is that learning both tools and using them for their respective strengths can only improve you as a data scientist. Versatility and flexibility are traits any data scientist at the top of their field. The Python vs R debate confines you to one programming language. You should look beyond it and embrace both tools for their respective strengths. Using more tools will only make you better as a data scientist.

Medium, Junio 2018

From 'R vs Python' to 'R and Python'

Leveraging the best of both 'Python and R' in a single project.

Medium, Marzo 2019

Is SQL Needed to be a Data Scientist?

Is SQL needed to be a Data Scientist? the answer is Yes, SQL (Structured Query Language) is Needed for Data Scientists to get the data and to work with that data. Everyone is busy to Learn R or Python for Data Science, but without Database Data Science is

Datacamp, 2018

“La comunidad de R”

rstudio::conf 2019

stackoverflow

R-Ladies

R-bloggers

R-statistics blog

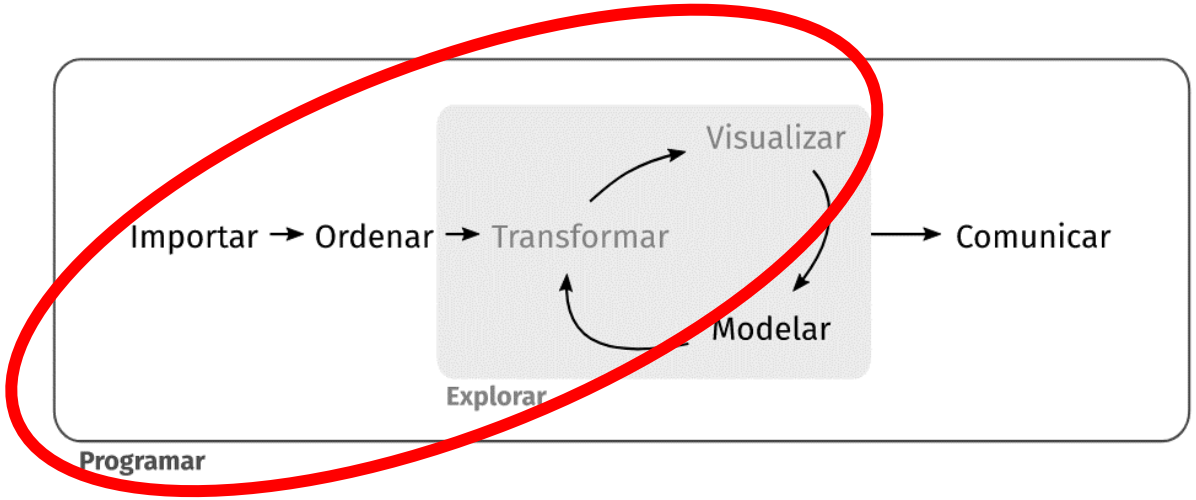
Statistics with R, and open source stuff (software, data, community)

useR! 2018

THE CONFERENCE FOR USERS OF R
JULY 10-13, 2018.
BRISBANE, AUSTRALIA.

Lo que veremos en las siguientes tres clases

- Introducción a R y R Studio
- Tipos y estructuras de datos
- Cómo importer y exportar datos
- Manipulación de datos
- Visualización de datos
- Comunicar resultados



Jessica Ward @JKRWard

```
# Yummy pasta recipe ----------
|
# get out the equipment we need (load the packages)
library(saucepan)
library(colander)

# get the ingredients out on to the counter (load data)
pasta<- read_csv("pasta.csv")
cheese<- read_csv("cheese.csv")
sauce<- read_csv("yummy_sauce.csv")
water<- read_csv("tap_water.csv")

#cook pasta then drain it and then add the cheese and the sauce
cooked_pasta<-Saucepan(pasta + water)
drained_pasta<-colander(cooked_pasta)
yummy_pasta <- c(drained_pasta, cheese, sauce)
```

Los scripts son recetas: registran como hacer las cosas. Escribe y guarda tus recetas para que R sepa que cocinar

El "environment" es como la mesa de la cocina: puedes poner ingredientes (datos) y platos terminados (resultados, modelos, etc) para usar mientras cocinas.

Los archivos (files) son como ingredientes en los muebles de la cocina: necesitas ponerlos en la mesa de la cocina (environment) para usarlos.
Los archivos que se necesitan pueden ser especificados en la receta para saber exactamente que se necesita sacar.

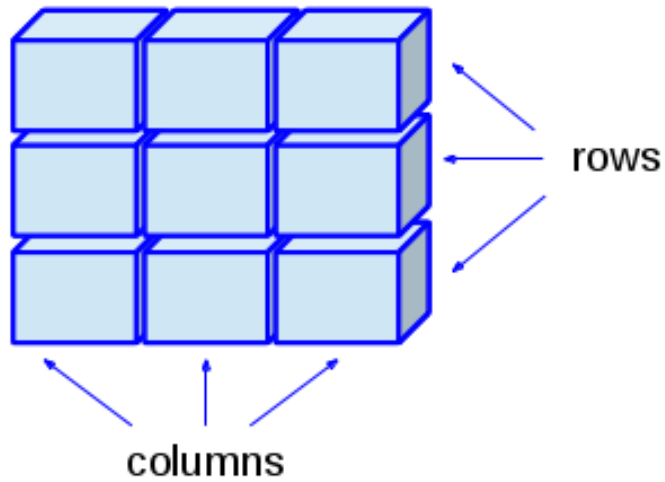
Las librerías (o paquetes) son como utensilios: cuando necesitas usar una olla hay que ir y comprar una que alguien ya haya diseñado y creado (install.packages()).
Cada vez que se quiera usar la olla hay que simplemente sacarla del mueble (library())

La consola es donde se cocina todo.
Acá se envían las recetas (se ejecuta el código) para cocinar.

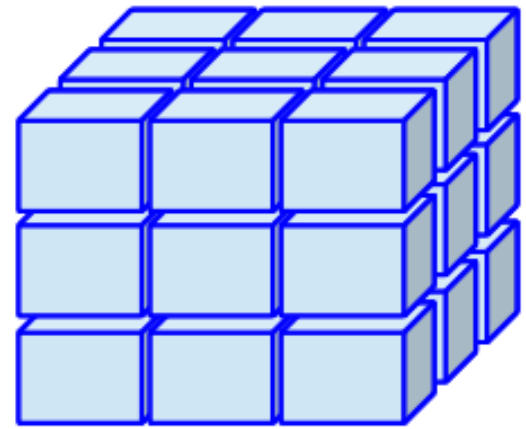
Puedes cocinar acá sin una receta pero será difícil recordar exactamente como recrear un plato en el futuro por lo que es recomendable siempre usar las recetas (scripts).

Vector

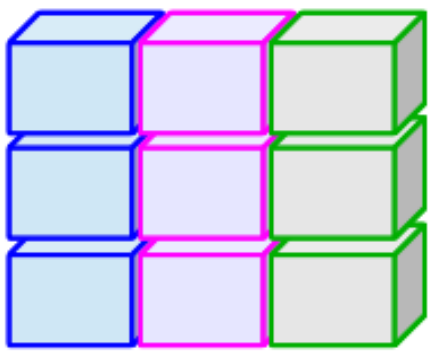
Matrix



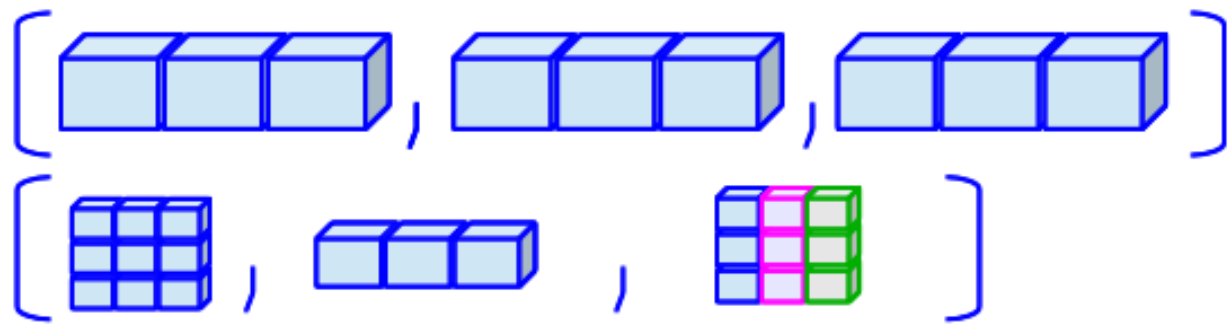
Array



Data Frame
(Table)

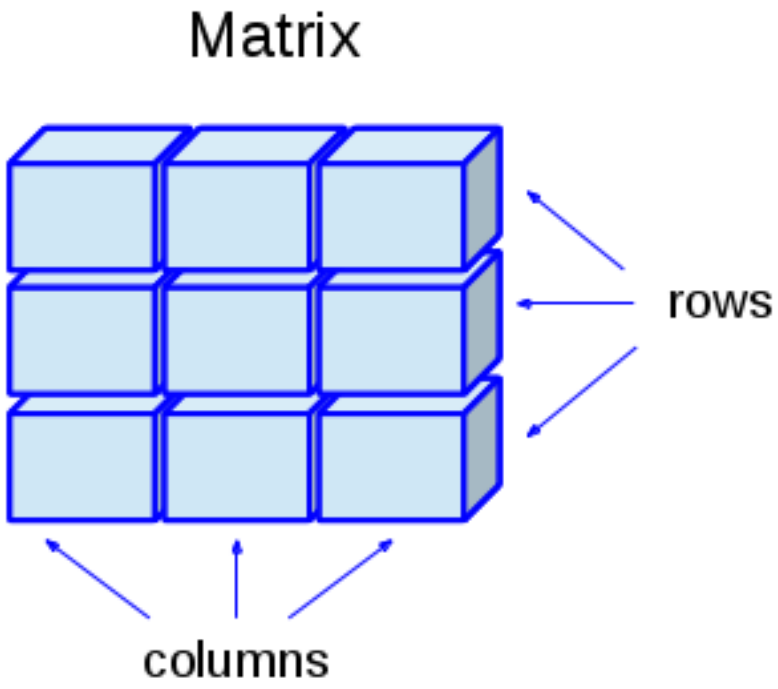


Lists

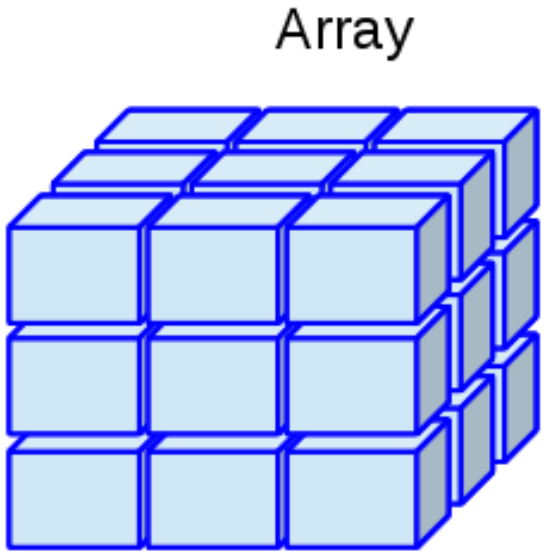


Vector

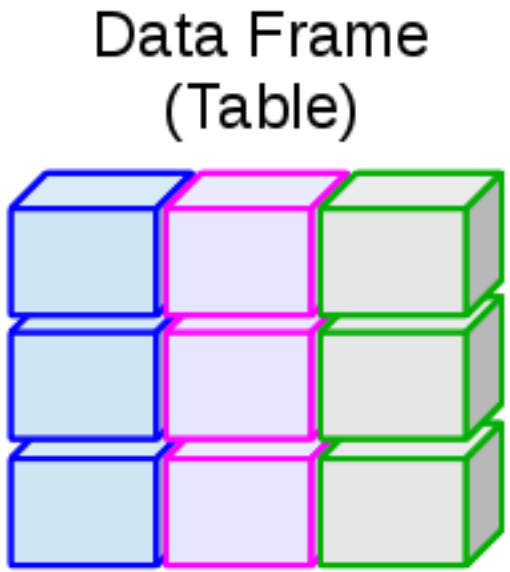
- 1 dimensión
- Un tipo de dato (por ej, solo números)



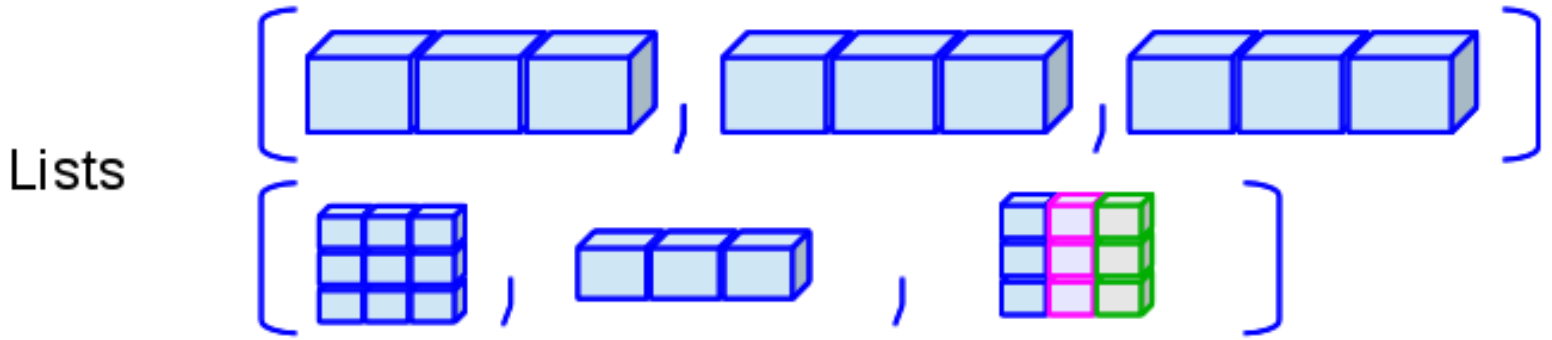
- 2 dimensiones
- Un tipo de dato (por ej, solo números)



- 3 dimensiones
- Un tipo de dato (por ej, solo números)



- 2 dimensiones
- Cada columna debe ser del mismo tipo de dato

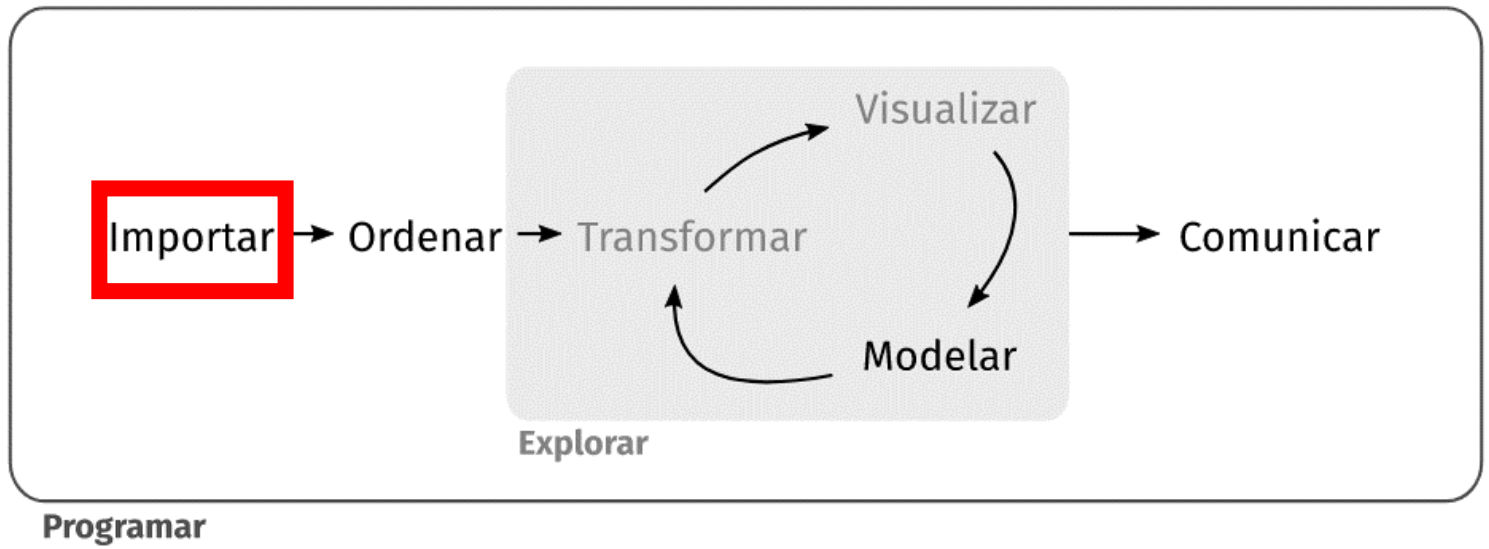


- Conjunto de objetos

- Cada elemento dentro de una lista puede corresponder a cualquiera de las estructuras que ya hemos visto

Tibble

- Version moderna de un data frame (recordar que R es un “lenguaje antiguo”)
  - `library(tibble)`
- El tidyverse trabaja integramente con tibbles
- Todo lo que aprendimos sobre data frames aplica para los tibbles
- Si alguna función no reconoce tibbles se puede usar `as.data.frame()`

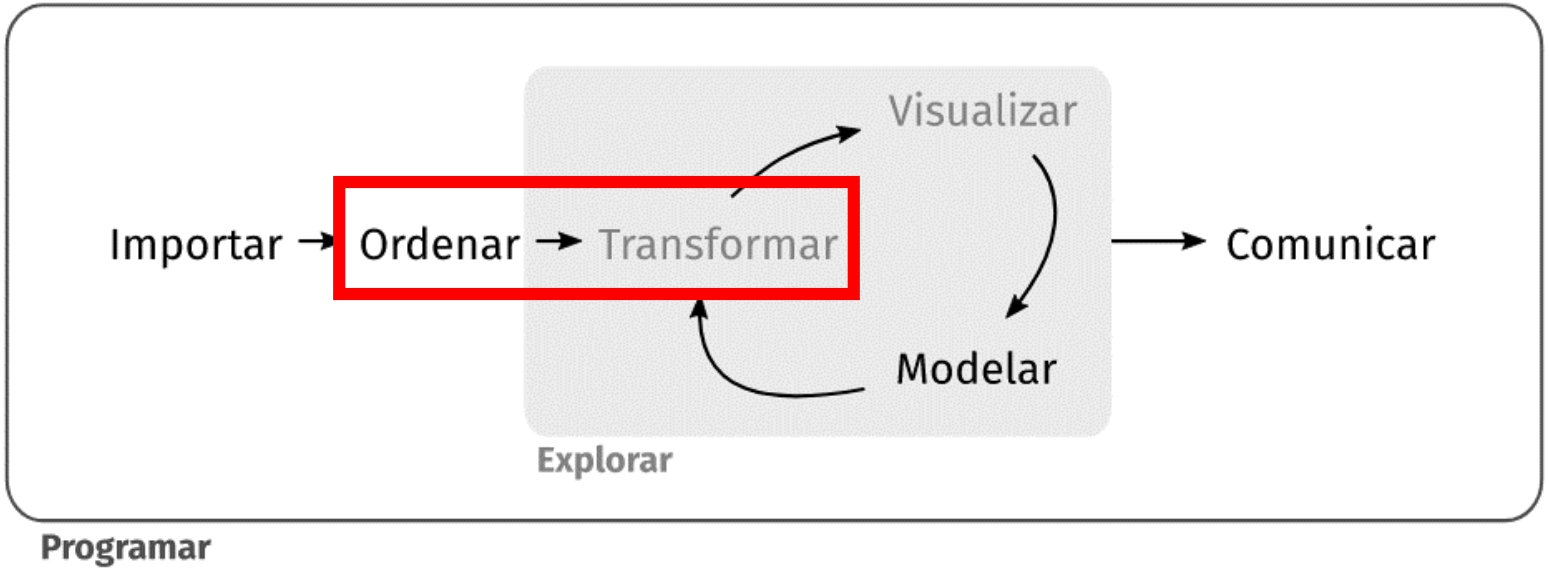

Importar
Ordenar
Visualizar
Transformar
Modelar
Comunicar
Explorar
Programar


```
read_tsv()
read_csv()
read_csv2()
```


```
read_excel()
```

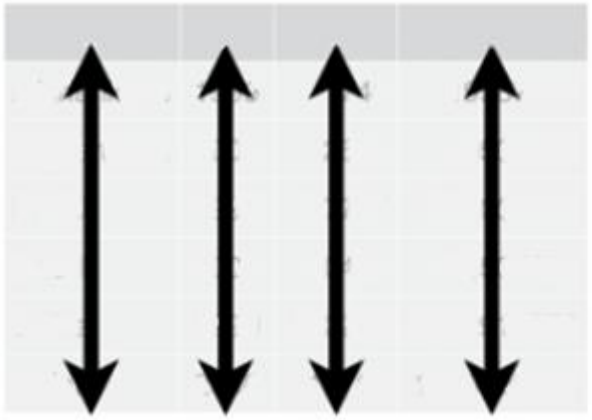

```
read_stata()
read_spss()
read_sas()
```

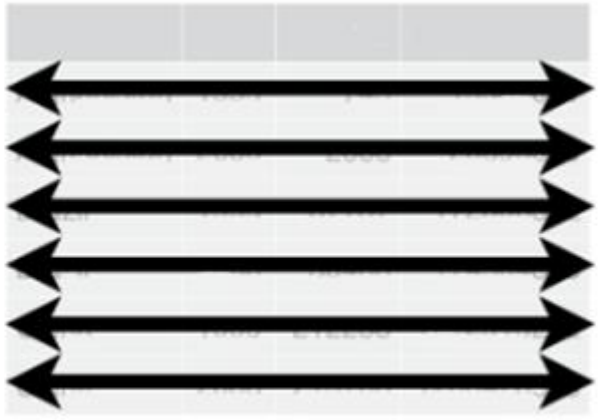
“Manipulación de datos” (Data Wrangling)

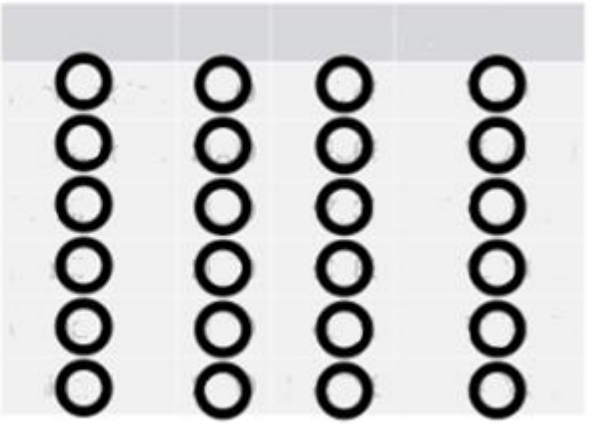


“Datos ordenados” (*Tidy Data*)

- Cada columna es una variable
- Cada fila es una observación
- Cada valor tiene su propia celda







¿Cuál corresponde a datos “tidy”?

1

	country	year	type	count
	<chr>	<int>	<chr>	<int>
1	Afghanistan	1999	cases	745
2	Afghanistan	1999	population	19987071
3	Afghanistan	2000	cases	2666
4	Afghanistan	2000	population	20595360
5	Brazil	1999	cases	37737
6	Brazil	1999	population	172006362
7	Brazil	2000	cases	80488
8	Brazil	2000	population	174504898
9	China	1999	cases	212258
10	China	1999	population	1272915272
11	China	2000	cases	213766
12	China	2000	population	1280428583

2

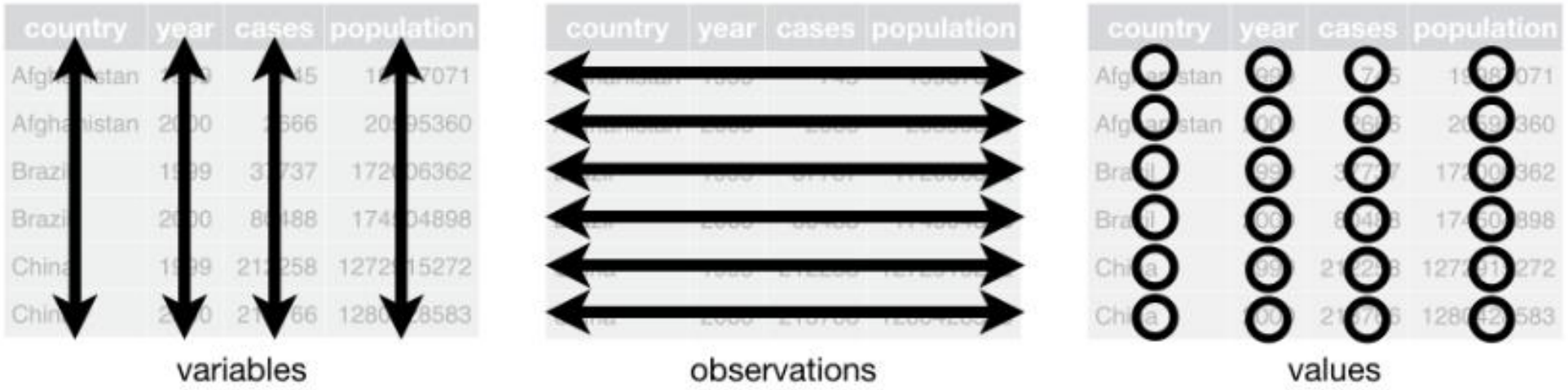
	country	year	cases	population
	<chr>	<int>	<int>	<int>
1	Afghanistan	1999	745	19987071
2	Afghanistan	2000	2666	20595360
3	Brazil	1999	37737	172006362
4	Brazil	2000	80488	174504898
5	China	1999	212258	1272915272
6	China	2000	213766	1280428583

3

	country	year	rate
*	<chr>	<int>	<chr>
1	Afghanistan	1999	745/19987071
2	Afghanistan	2000	2666/20595360
3	Brazil	1999	37737/172006362
4	Brazil	2000	80488/174504898
5	China	1999	212258/1272915272
6	China	2000	213766/1280428583

4

	country	`1999`	`2000`
*	<chr>	<int>	<int>
1	Afghanistan	19987071	20595360
2	Brazil	172006362	174504898
3	China	1272915272	1280428583



	country	year	cases	population
	<chr>	<int>	<int>	<int>
1	Afghanistan	1999	745	19987071
2	Afghanistan	2000	2666	20595360
3	Brazil	1999	37737	172006362
4	Brazil	2000	80488	174504898
5	China	1999	212258	1272915272
6	China	2000	213766	1280428583

¿Por qué ordenar los datos así?

- Bueno tener una sola forma consistente de almacenamiento de datos

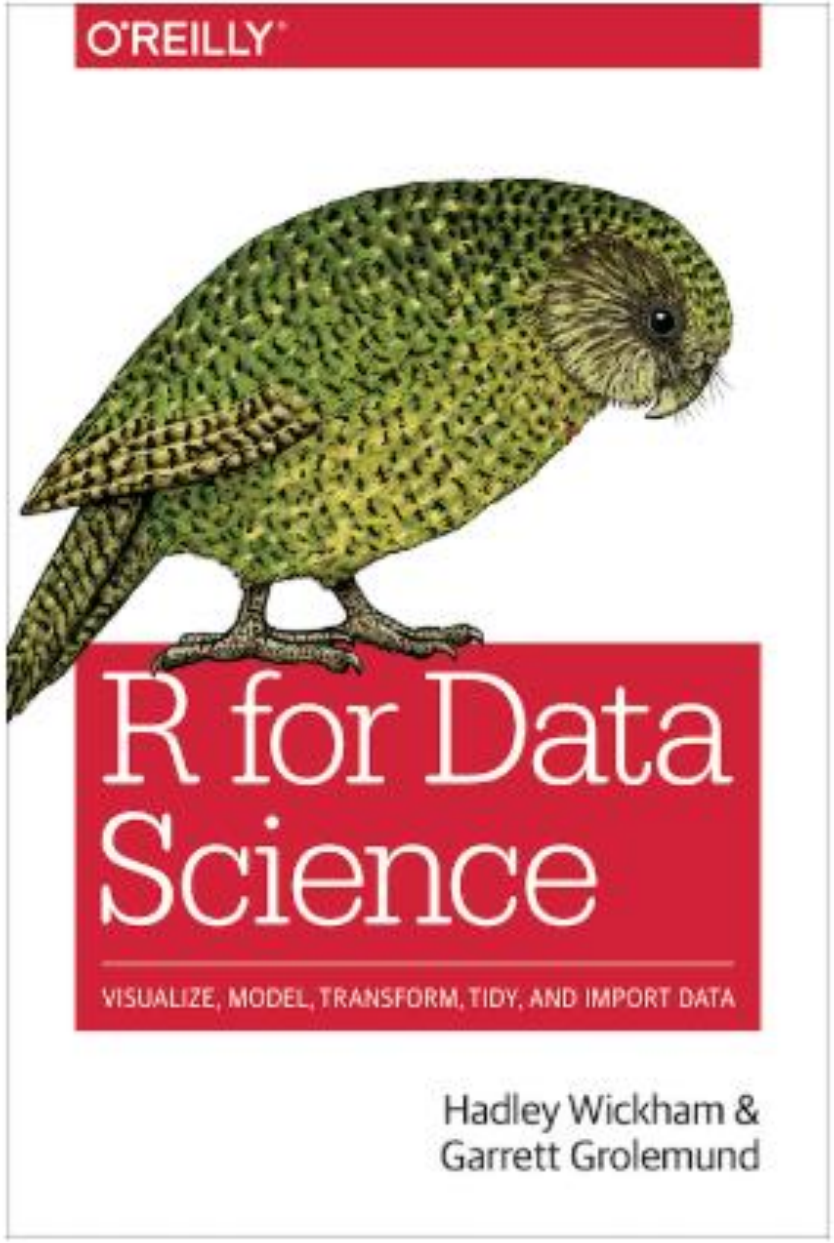
- Ventajas para explotar la forma de trabajo en R (vectores)

Tidyverse

https://www.tidyverse.org/

El ***tidyverse*** es una colección de paquetes R diseñados para la ciencia de datos. Todos los paquetes comparten un diseño, filosofía, gramática y estructuras de datos subyacente.

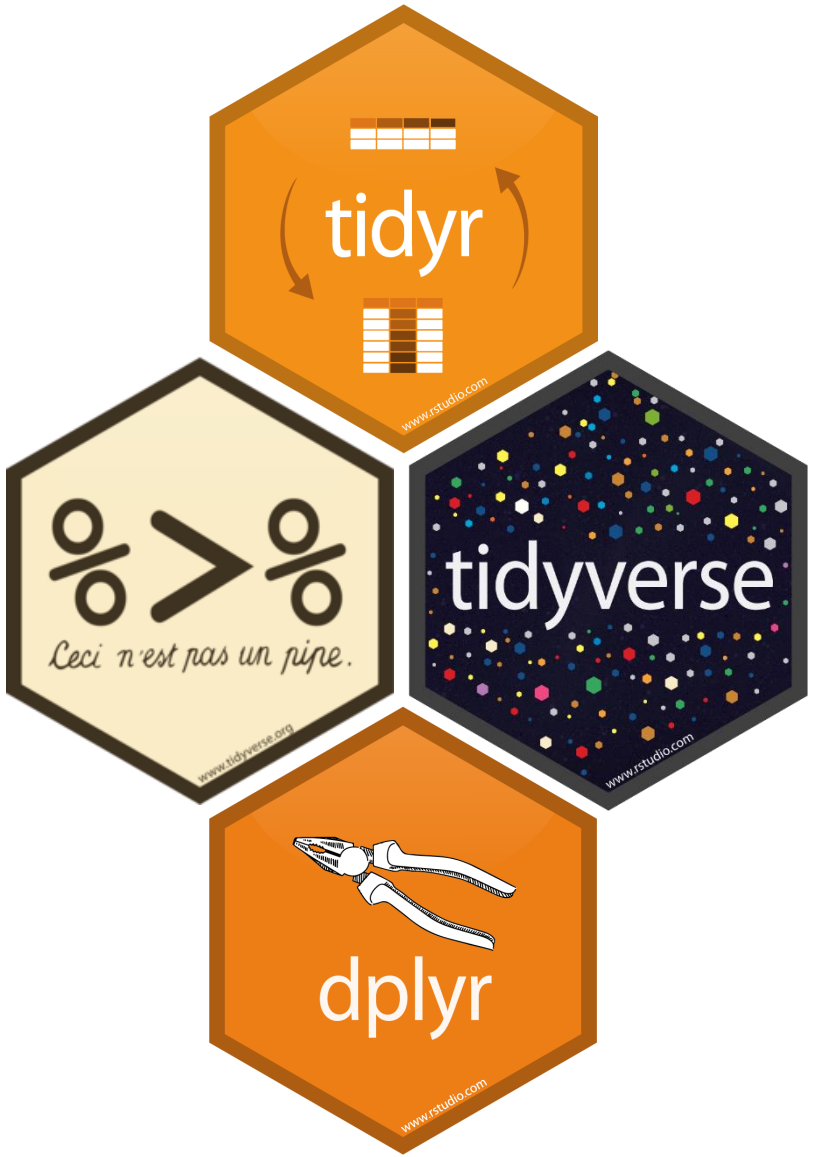
```
install.packages("tidyverse")
```



https://r4ds.had.co.nz/

Manipulación de datos con “tidyverse”

- **tidyr**
  - gather()
  - spread()
  - separate()
  - unite()
- **dplyr**
  - filter()
  - arrange()
  - select()
  - mutate()
  - summarise()
    - group_by()
- **magrittr**
  - %>% (pipe)



"Happy families are all alike; every unhappy family is unhappy in its own way."

— **Leo Tolstoy**

"Tidy datasets are all alike, but every messy dataset is messy in its own way."

— **Hadley Wickham**

- **tidyr**
  - gather()
  - spread()
  - separate()
  - unite()

gather()

- Usar cuando nombres de variables corresponden a valores

- Función utiliza tres argumentos además del tibble a modificar:
  - Nombres de columnas que representan valores y no variables
  - Nombre de la nueva variable formada a partir de las columnas(*key*)
  - Nombre de la nueva variable a partir de los valores repartidos entre columnas (*value*)

```
gather(table4, `1999`, `2000`, key = "year", value = “cases")
```

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

spread()

- Función opuesta a `gather()`
  - Usar cuando una observación está repartida entre varias filas
- Función utiliza dos argumentos además del tibble a modificar:
  - Nombre de columna que tiene el nombre de las variables (*key*)
  - Nombre de columna que tiene los valores de las variables (*value*)

```
spread(table2, key = key, value = value)
```

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

separate()/unite()

- `separate()` separa una columna en múltiples

country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

```
separate(table3, rate, into = c("cases", "population"), sep = "/", convert = TRUE
```

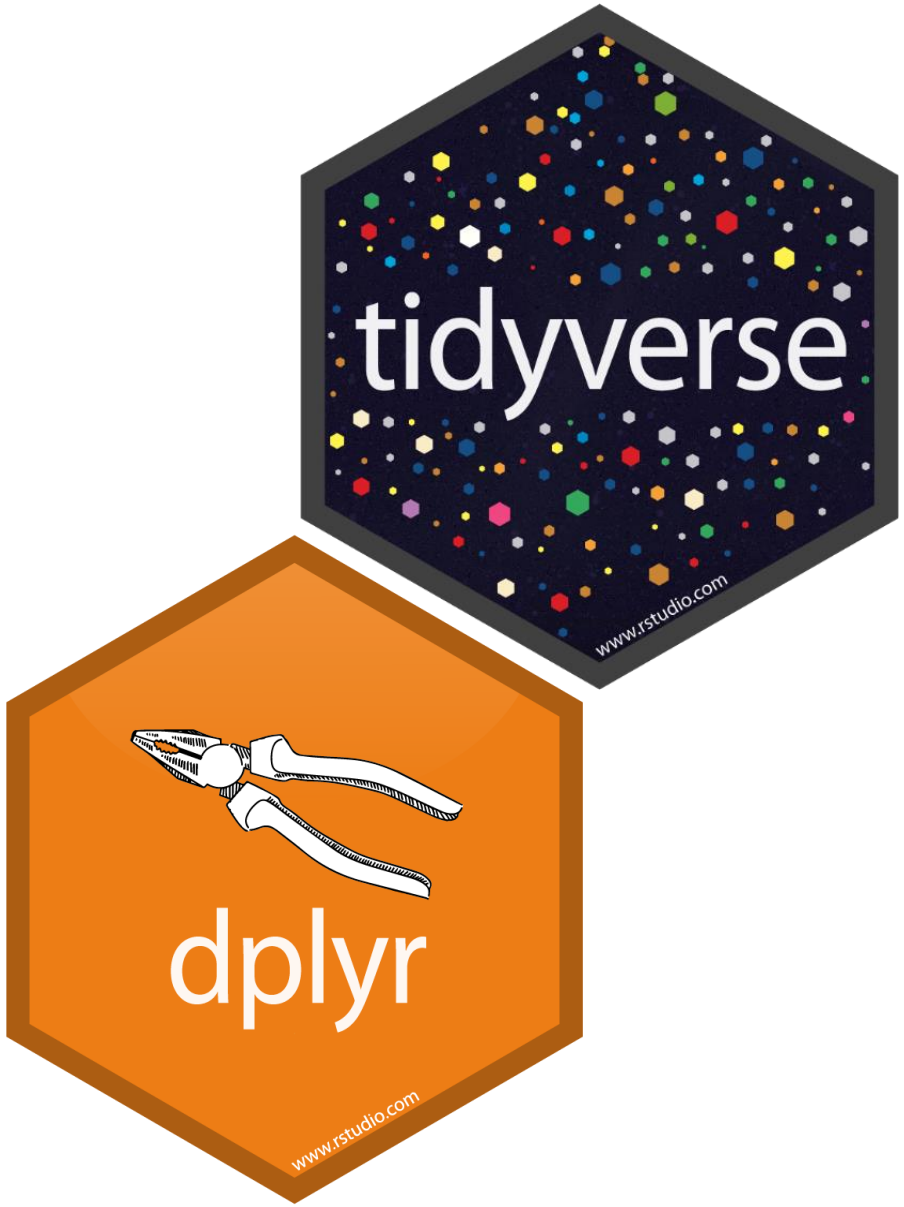
- `unite()` combina múltiples columnas en una

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	**99**	745 / 19987071
Afghanistan	20	**0**	2666 / 20595360
Brazil	19	**99**	37737 / 172006362
Brazil	20	**0**	80488 / 174504898
China	19	**99**	212258 / 1272915272
China	20	**0**	213766 / 1280428583

```
unite(table5, new, century, year, sep = "")
```

- **dplyr**
  - filter()
  - arrange()
  - select()
  - mutate()
  - summarise()



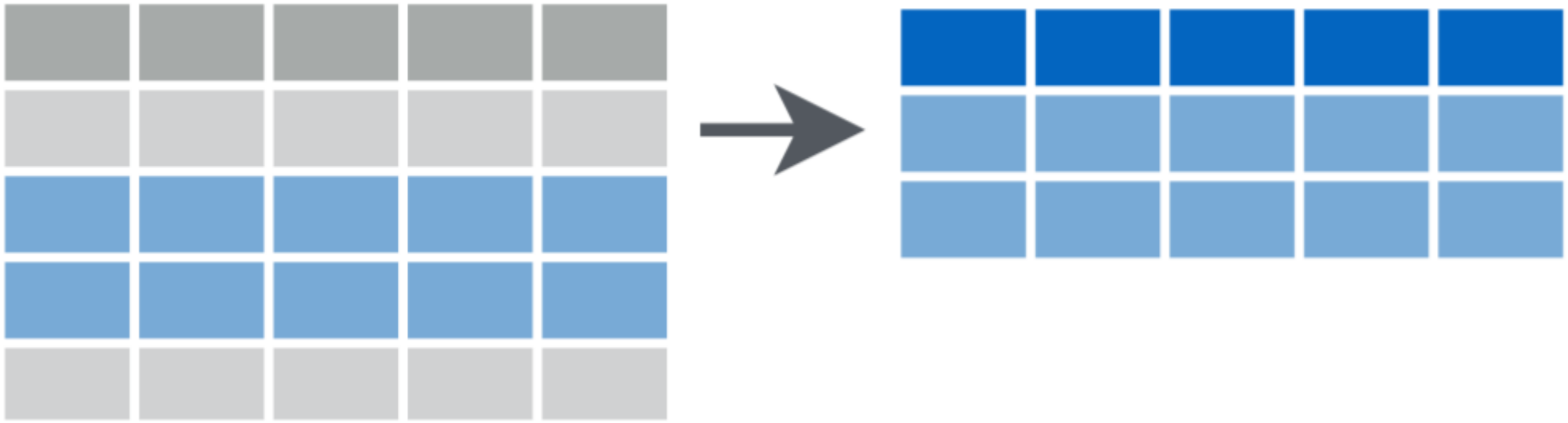
- **dplyr**
  - `filter()` – seleccion filas
  - `select()` – selecciona columnas
  - `mutate()` – crea nuevas variables
  - `arrange()` – reordena filas según columna/s
  - `summarise()` – "summary statistics"
    - `group_by()`

Similitudes entre los “verbos de dplyr”

- El primer argumento es un tibble (o data frame)
- Los siguientes argumentos describen que hacer con los datos ocupando los nombres de las columnas (variables) … sin usar “”!
- El resultado es un nuevo tibble

filter()

R Base	D[D$subject == 4 & D$trial == 10,]
dplyr	filter(D, subject == 4, trial == 10)



select()

R Base	D[,c("subject", "trial")]
dplyr	select(D, subject, trial)

mutate()

R Base	`D$freq <- D$count/D$time`
dplyr	`D <- mutate(D, freq = count/time)`

arrange()

R Base	D[order(D$subject, D$trial),]
dplyr	arrange(D, subject, trial)

Id	subject	trial	varX
10002	1	1	4
30005	1	2	6
10003	1	3	4
20002	2	1	3
10001	2	2	7
10008	2	3	3
30002	3	1	1
20005	3	2	4

group_by() / summary()

R Base	`aggregate(D$RT, list(subject = D$Subject), mean)`
dplyr	`a <- group_by(D, subject)` `summarize(a, totalcount = sum(count))`

magrittr - %>%


%>%
Ceci n'est pas un pipe.
www.tidyverse.org
tidyverse
www.rstudio.com

```
resultado_final <- sqrt(mean(abs(x)))
```

```
absoluto <- abs(x)
promedio <- mean(absoluto)
resultado_final <- sqrt(promedio)
```

```
resultado_final <- x %>% abs %>% mean %>% sqrt
```

¿Cómo funciona %>%?

```
f(x)                    =    x %>% f

f(x,y)                  =    x %>% f(y)

f(y,x)                  =    x %>% f(y,.)

h(g(f(x)))              =    x %>% f %>% g %>% h

filter(df, v1 == 2)     =    df %>% filter(v1 == 2)
```

```
elenco <- trimws(gsub("\n", "", html_text(html_nodes(machuca,
".primary_photo+ td a"))), "left")
```

```
elenco <- html_nodes(machuca, ".primary_photo+ td a")
elenco <- html_text(elenco)
elenco <- gsub("\n", "", elenco)
elenco <- trimws(elenco, "left")
```

```
elenco <- machuca %>% html_nodes(".primary_photo+ td a") %>%
                  html_text() %>%
                  gsub("\n", "", .) %>%
                  trimws("left")
```

```
elenco <- trimws(gsub("\n", "", html_text(html_nodes(machuca,
".primary_photo+ td a"))), "left")
```

```
elenco <- html_nodes(machuca, ".primary_photo+ td a")
elenco <- html_text(elenco)
elenco <- gsub("\n", "", elenco)
elenco <- trimws(elenco, "left")
```

```
elenco <- machuca %>% html_nodes(".primary_photo+ td a") %>%
                  html_text() %>%
                  gsub("\n", "", .) %>%
                  trimws("left")
```

```
elenco <- trimws(gsub("\n", "", html_text(html_nodes(machuca,
".primary_photo+ td a"))), "left")
```

```
elenco <- html_nodes(machuca, ".primary_photo+ td a")
elenco <- html_text(elenco)
elenco <- gsub("\n", "", elenco)
elenco <- trimws(elenco, "left")
```

```
elenco <- machuca %>% html_nodes(".primary_photo+ td a") %>%
                      html_text() %>%
                      gsub("\n", "", .) %>%
                      trimws("left")
```

```
elenco <- trimws(gsub("\n", "", html_text(html_nodes(machuca,
".primary_photo+ td a"))), "left")
```

```
elenco <- html_nodes(machuca, ".primary_photo+ td a")
elenco <- html_text(elenco)
elenco <- gsub("\n", "", elenco)
elenco <- trimws(elenco, "left")
```

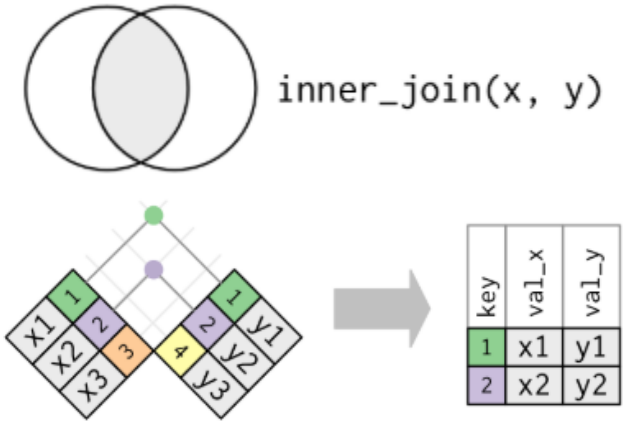
```
elenco <- machuca %>% html_nodes(".primary_photo+ td a") %>%
                      html_text() %>%
                      gsub("\n", "", .) %>%
                      trimws("left")
```

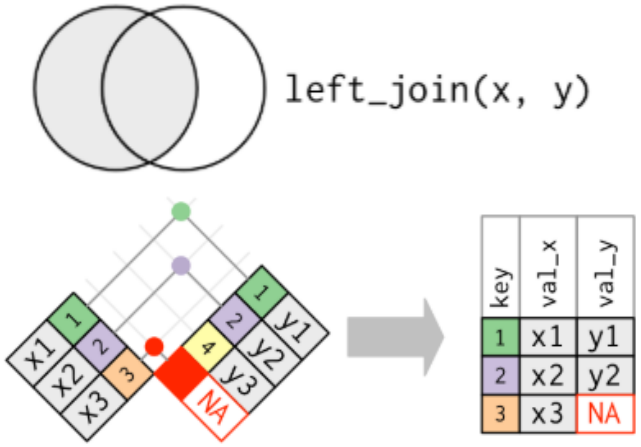
```r
elenco <- trimws(gsub("\n", "", html_text(html_nodes(machuca,
".primary_photo+ td a"))), "left")
```

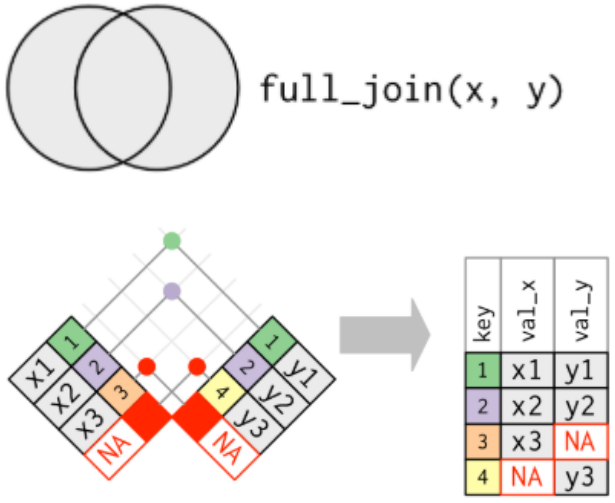
```r
elenco <- html_nodes(machuca, ".primary_photo+ td a")
elenco <- html_text(elenco)
elenco <- gsub("\n", "", elenco)
elenco <- trimws(elenco, "left")
```

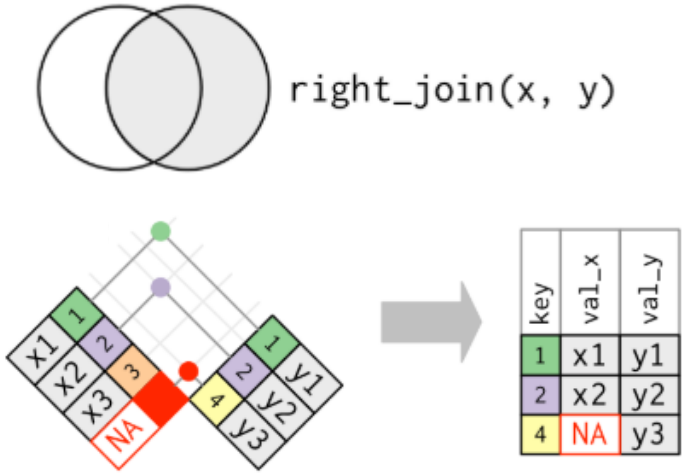
```r
elenco <- machuca %>% html_nodes(".primary_photo+ td a") %>%
                      html_text() %>%
                      gsub("\n", "", .) %>%
                      trimws("left")
```

Datos relacionales ("joins")



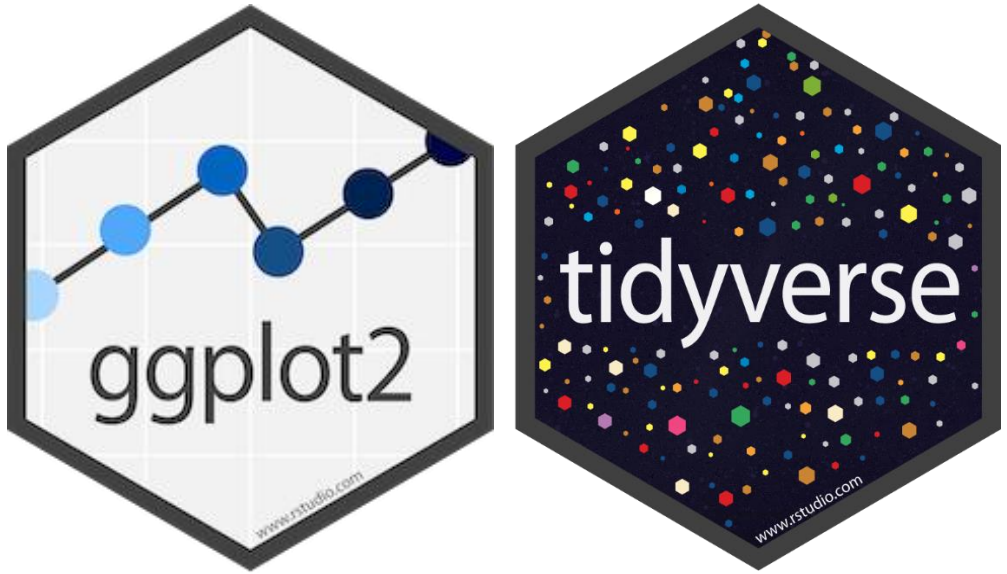


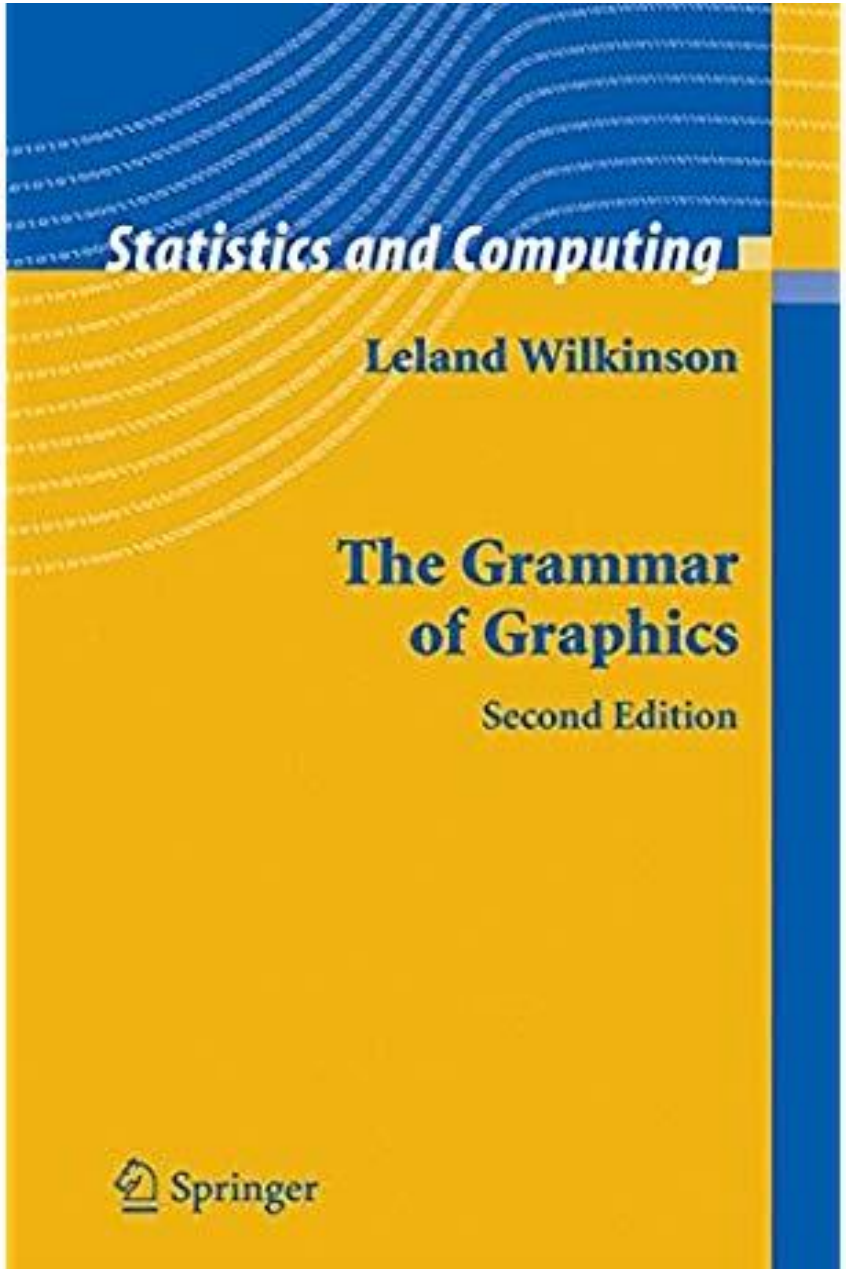


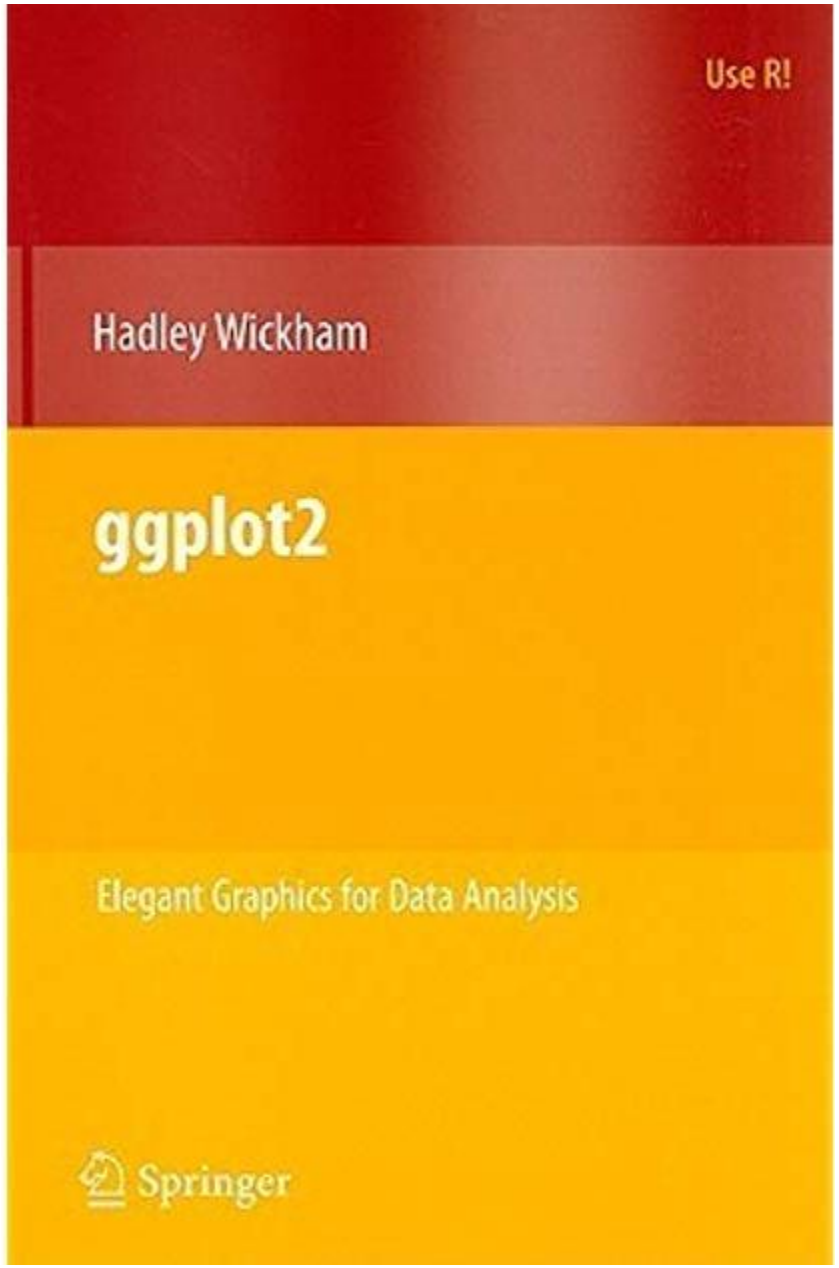


ggplot2

- Produce gráficos en forma de capas
- Utiliza una “gramática” subyacente para construir gráficos parte por parte en lugar de proporcionar gráficos prefabricados
- Suficientemente fácil para usarlo sin conocer la gramática subyacente
- Permite al usuario crear gráficos a partir de conceptos en lugar de recorder commandos y opciones

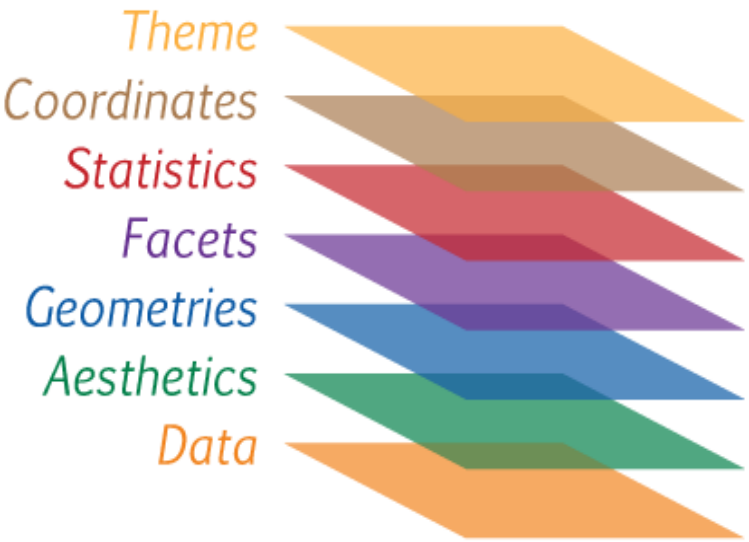



Statistics and Computing
Leland Wilkinson
The Grammar of Graphics
Second Edition
Springer


Use R!
Hadley Wickham
ggplot2
Elegant Graphics for Data Analysis
Springer

Componentes de `ggplot2` basados en "Grammar of graphics"

- Los **datos** (*data*) a graficar
- Conjunto de variables que definirán la **apariencia** (*aesthetics*) de los objetos geométricos
- **Objetos geométricos** (*geometric objects*) que aparecerán en el gráfico (circulos, lineas, etc.)
- Un **ajuste de posición** (*position adjustment*) para ubicar para objeto geométrico en el gráfico
- Una **escala** (*scale*), o rango de valores, para cada *aesthetics* que se use
- **Sistema de coordenadas** (*coordinate system*) para organizer los objetos geométricos
- **Facetas** (*facets*) o agrupación de datos a mostrar en un gráfico
- **Transformaciones estadísticas** (*statistical transformation*) necesarias para calculary valores a usar en el gráfico



Pasos básicos para crear gráficos con `ggplot2`

1. Llamar a la función ggplot()
2. Específicar las variables a mapear para la apariencia (aesthetics)
3. Agregar capas de objetos geométricos

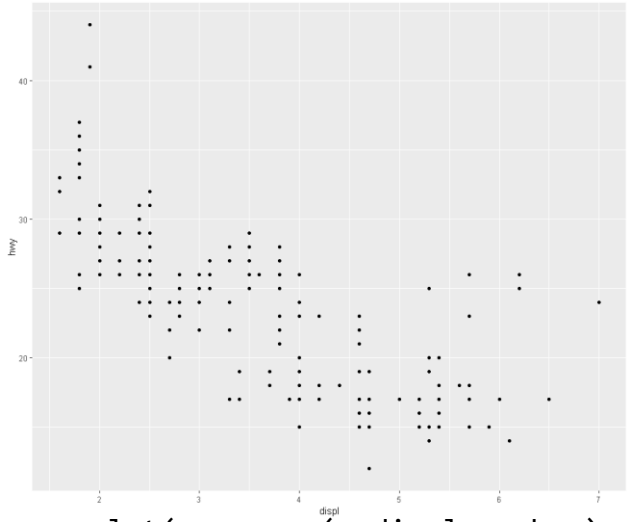
1. "Lienzo" donde se construirá el gráfico


```
ggplot(mpg)
```

2. "Lienzo" + variables mapeadas en los ejes "x" e "y"


```
ggplot(mpg, aes(x=displ, y=hwy)
```

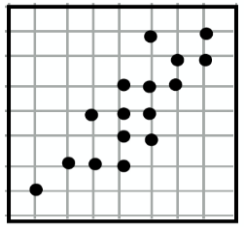
3. Agregar capa de objeto geométrico a graficar (en este caso puntos)



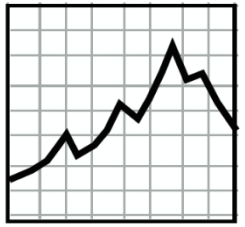

```
ggplot(mpg, aes(x=displ, y=hwy)
+ geom_point()
```

Diferentes capas `geom`

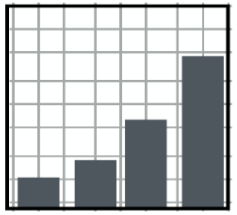
- `geom_point`



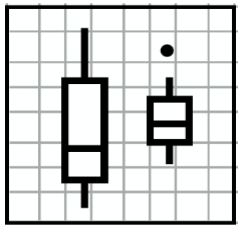
- `geom_line`



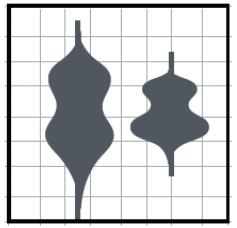
- `geom_bar`



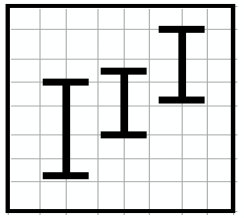
- `geom_boxplot`



- `geom_violin`



- `geom_errorbar`



- `geom_tile`

Y más...

```
ggplot(data = mpg) +
geom_point(mapping = aes(x=displ, y=hwy))
```

```
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

RStudio Community

https://community.rstudio.com/t/why-cant-ggplot2-use/4372

Why can't ggplot2 use %>%?

tidyverse ggplot2

Should ggplot2 use the pipe?

The first implicit question is should ggplot2 use the pipe? I think the answer is yes:

- I think the pipe is absolutely the right interface. It is a consistent principle that applies in many more situations, and because it's just syntactic sugar for function composition, you can still compose small pieces in other ways.
- Switching from `%>%` to `+` is a frequent source of errors (including for me!)
- The pipe avoids the poor match of the semantics of addition to ggplot2. You usually expect that `x + y` equals `y + x` and that `x + (y + z)` equals `(x + y) + z`. Neither of these are true (in general) for ggplot2.
- I think it's fine to have a pipe-y interface based around nouns instead of verbs. keras 50 is a good example - I don't think there would be any significant benefit to renaming (e.g.) `layer_dense()` to `add_layer_dense()`.

(`@rensa` points out one nice feature of `+` interface is that you can add multiple components by putting them in a list. But magrittr has an equivalent technique: `my_geoms <- . %>% geom_point() %>% geom_line() %>% geom_smooth()`. And I think that's an improvement because it uses ideas that can be applied in more contexts.)

As an interesting historical anecdote, ggplot (the precursor to ggplot2), was written in a function style that could have used the pipe (if the pipe had existed). To explore this idea little bit, I bought ggplot back to life as ggplot1 60:

```
library(ggplot1)

mtcars %>%
  ggplot(list(x = mpg, y = wt)) %>%
  ggpoint()
```

Could ggplot2 support both + and %>%?

So if ggplot2 *should* use the pipe, could it? Would it be possible to allow both `+` and `%>%`?

I'm pretty certain the answer is no:

- The first two arguments to all the geoms are currently `mapping` and `data`. For the pipe to work, the first argument would need to be `plot`.
- It would be possible to change the definition of the pipe specially to make it work with ggplot2, but that is unappealing because it would require changing a general tool to support a specific package.
- It's almost certainly possible to use some deep metaprogramming magic to tell when the pipe is being used and somehow offset every argument one place over. This is likely to be hard to implement, fragile, slow, and hard to document.

Would it be better to create ggplot3?

If we can't make the pipe work with ggplot2, maybe it's time for ggplot3? ggplot3 could behave identically to ggplot2 in every way, *except* that it would compose plots using `%>%` instead of `+`. This would solve the pipe problem but would come some major downsides:

- ggplot3 would need substantial (if fairly formulaic) changes to almost every function. This would be a lot of work.
- What would happen when someone reported a bug in ggplot2? Would I fix it only in ggplot3 and require users to upgrade? That seems unfair to ggplot2 users, so for every change, I'd need to make it simultaneously to ggplto2 and ggplot3, basically doubling all future development work.
- Similarly, ggplot3 would create a fork in all other documentation (e.g. stackoverflow and the ggplot2 book): you wouldn't be able to immediately apply ggplot2 anwers to ggplot3, and new answers created for ggplot3 wouldn't immediately apply to ggplot2.

Overall, I think making this change just to use the pipe is not worthwhile.

```
ggplot(mpg, aes(x=displ, y=hwy) + geom_point()
```

```
p <- ggplot(mpg, aes(x=displ, y=hwy)
p + geom_point()
```

```
mpg %>% ggplot(aes(x=displ, y=hwy) + geom_point()
```