Introducción a R

2 parte

COMPASS

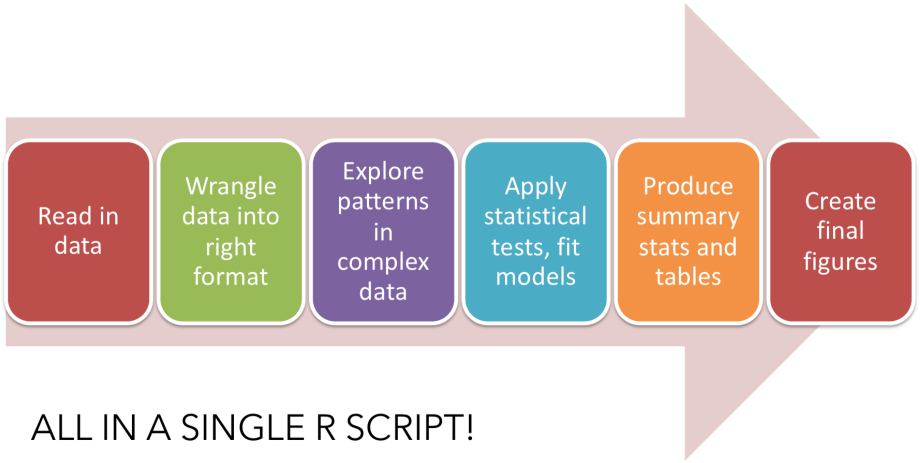
Community Platform for Agricultural Sciences

Paulo Izquierdo
Viviana Ortiz

Revisión

Intro a R

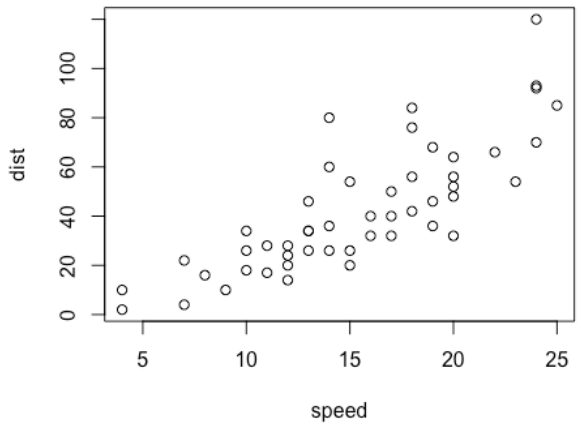
- Qué es R?
- Asignar variables
- Vectorizar operaciones
- Funciones básicas
- Bases de datos integrados y otras




```
mi nombre <- "paulo"
Error: unexpected symbol in "mi
nombre”
```

```
> mean(x)
```

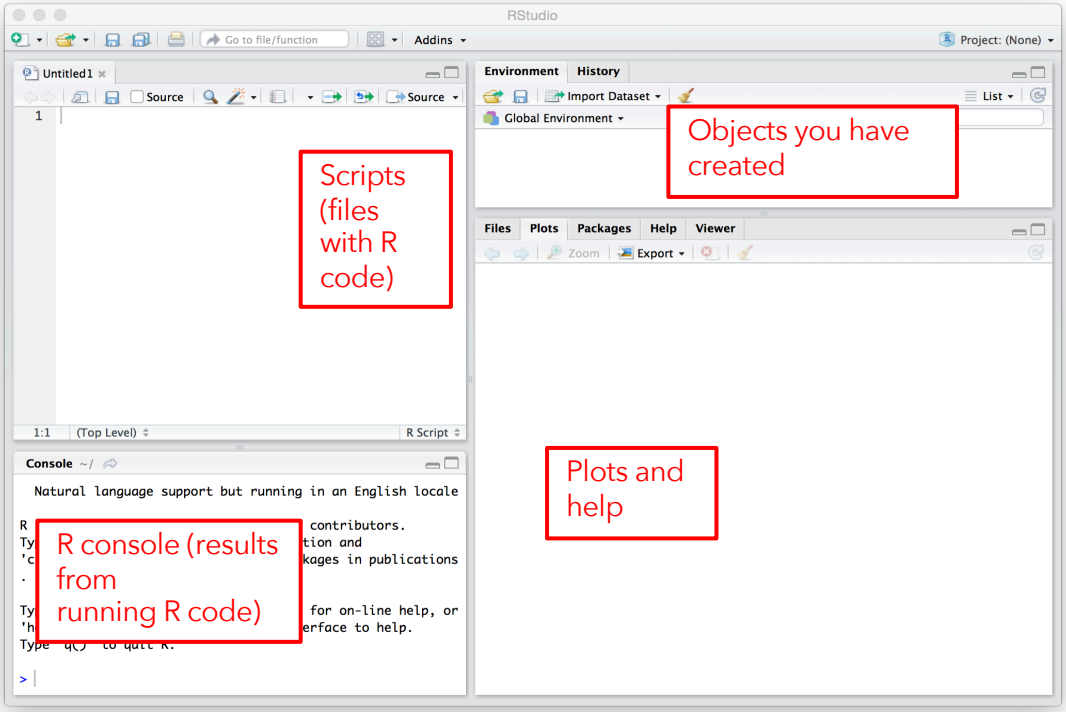
```
> plot(cars)
```



Revisión

Objetos y estructuras

- RStudio!
- Cómo cargar datos en R
- Cuáles son las estructuras de datos más comunes en R
- Cómo crear vectores
- Cómo verificar y cambiar el tipo de datos
- Cómo unir vectores en matrices, dataframes, lists



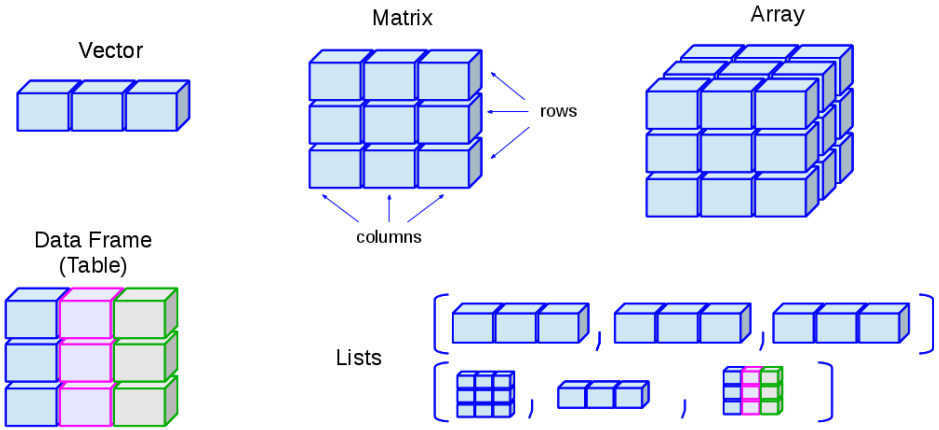
Revisión

Objetos y estructuras

- RStudio!
- Cómo cargar datos en R
- Cuáles son las estructuras de datos más comunes en R
- Cómo crear vectores
- Cómo verificar y cambiar el tipo de datos
- Cómo unir vectores en matrices, dataframes, lists


```
getwd ()
setwd ("~ / Desktop")
x <- read.csv(file="FileName.csv")
```

```
test <- c("a","b","c")
```



Revisión

Manipular datos

- Como indexar diferentes tipos de objetos
- Usar lógica para indexar
- Remover NA para los análisis.
- Cómo ordenar, transponer, eliminar duplicados, etc.


```
> x[1] #primer elemento de un vector
> x[ ,5] #quinto elemento de una matriz
> x[[2]] #segundo objeto de una lista
> x[[2]][1] # primer elemento del segundo objeto de una lista
```

Revisión

Manipular datos

- Como indexar diferentes tipos de objetos
- Usar lógica para indexar
- Remover NA para los análisis.
- Cómo ordenar, transponer, eliminar duplicados, etc.


```
a <-c(2,3,4,)
[1] 2 3 4

b <- a>3
[1] FALSE FALSE TRUE
```

```
x <- c(63.33, NA, 64.6, 68.38,
NA, 79.1, 77.46)

mean(x, na.rm=T)
[1] 70.58
```

Revisión

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
x <- iris[iris$Species=="setosa" &
            iris$Petal.Length > 1.5, ]

mean(x$Petal.Width)
[1] 0.3076923

mean(iris[iris$Species=="setosa" &
            iris$Petal.Length > 1.5,
            Petal.Width"])
[1] 0.3076923
```

Revisión

```
> myDataFrame <-
    data.frame(a=c(11,13,12,15,17,20),
               b=c(8,NA, 6, 4,NA,15))
```

```
   a  b
1 11  8
2 13 NA
3 12  6
4 15  4
5 17 NA
6 20 15
```

```
> subset(x=myDataFrame, subset=b>5) #filas
   a  b
1 11  8
3 12  6
6 20 15
```

```
> subset(x=myDataFrame, subset=b>7,
          select=a)  #columnas
   a
1 11
6 20
```

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.
rev(x)
Return x reversed.
table(x)
See counts of values.
unique(x)
See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which are equal to 10.
x[x < 0]	All elements less than zero.
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.

Named Vectors

x['apple']	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/ write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.
`as.factor`	`'1', '0', '1', levels: '1', '0'`	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrixes

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)` Transpose

`m %*% n` Matrix Multiplication

`solve(m, n)` Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** library.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)` Number of rows.

`ncol(df)` Number of columns.

`dim(df)` Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Strings

Also see the **stringr** library.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)` Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)` Turn a numeric vector into a factor but 'cutting' into sections.

Statistics

`lm(x ~ y, data=df)` Linear model.

`glm(x ~ y, data=df)` Generalised linear model.

`summary` Get more detailed information out a model.

`t.test(x, y)` Preform a t-test for difference between means.

`pairwise.t.test` Preform a t-test for paired data.

`prop.test` Test for a difference between proportions.

`aov` Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** library.

`plot(x)` Values of x in order.

`plot(x, y)` Values of x against y.

`hist(x)` Histogram of x.

Dates

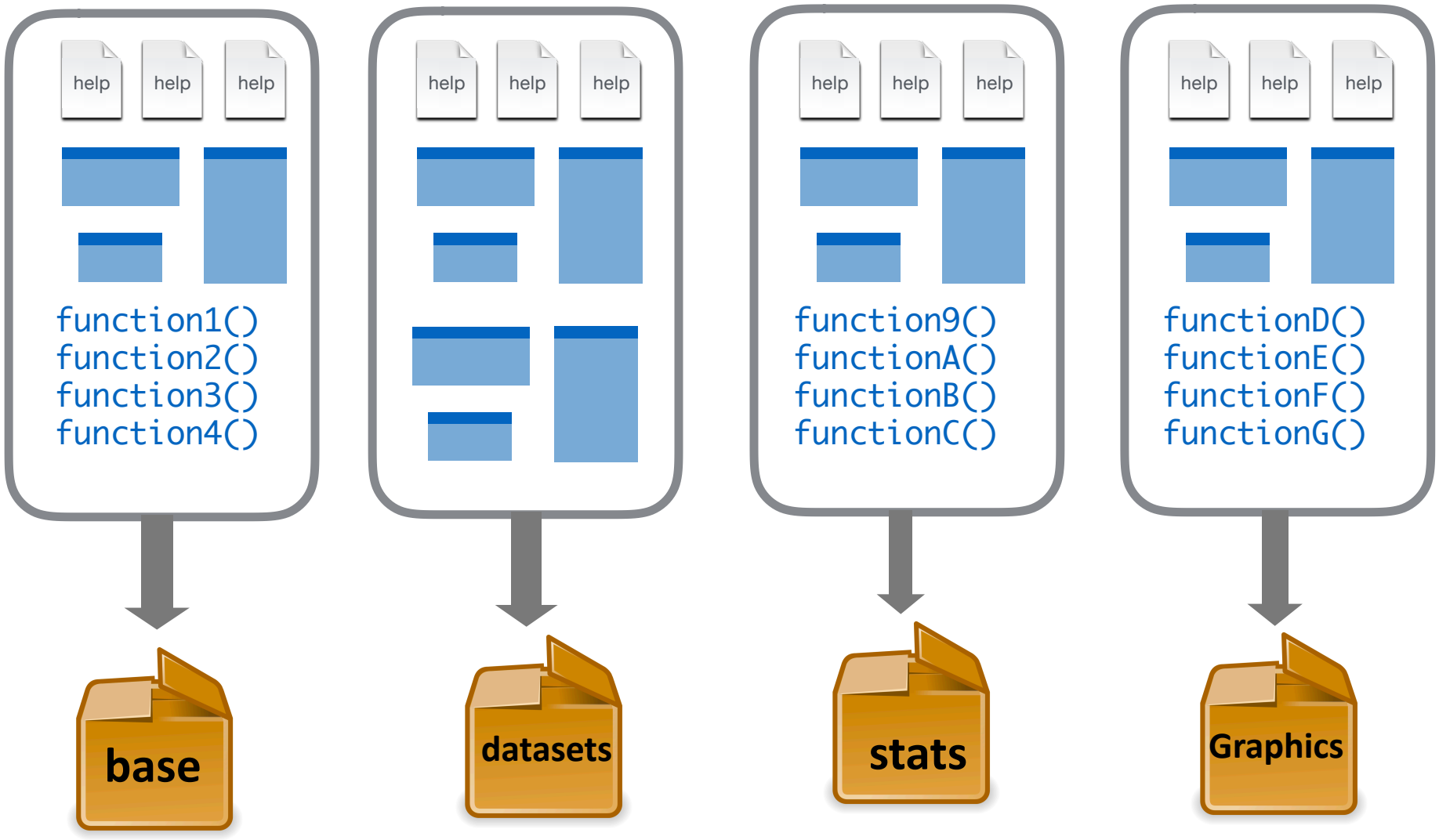
See the **lubridate** library.

Ahora vamos a usar : dplyr

- Instalación de paquetes externos
- Organizar datos con dplyr
- Otros paquetes..

Hasta ahora solo usamos funciones integradas en R

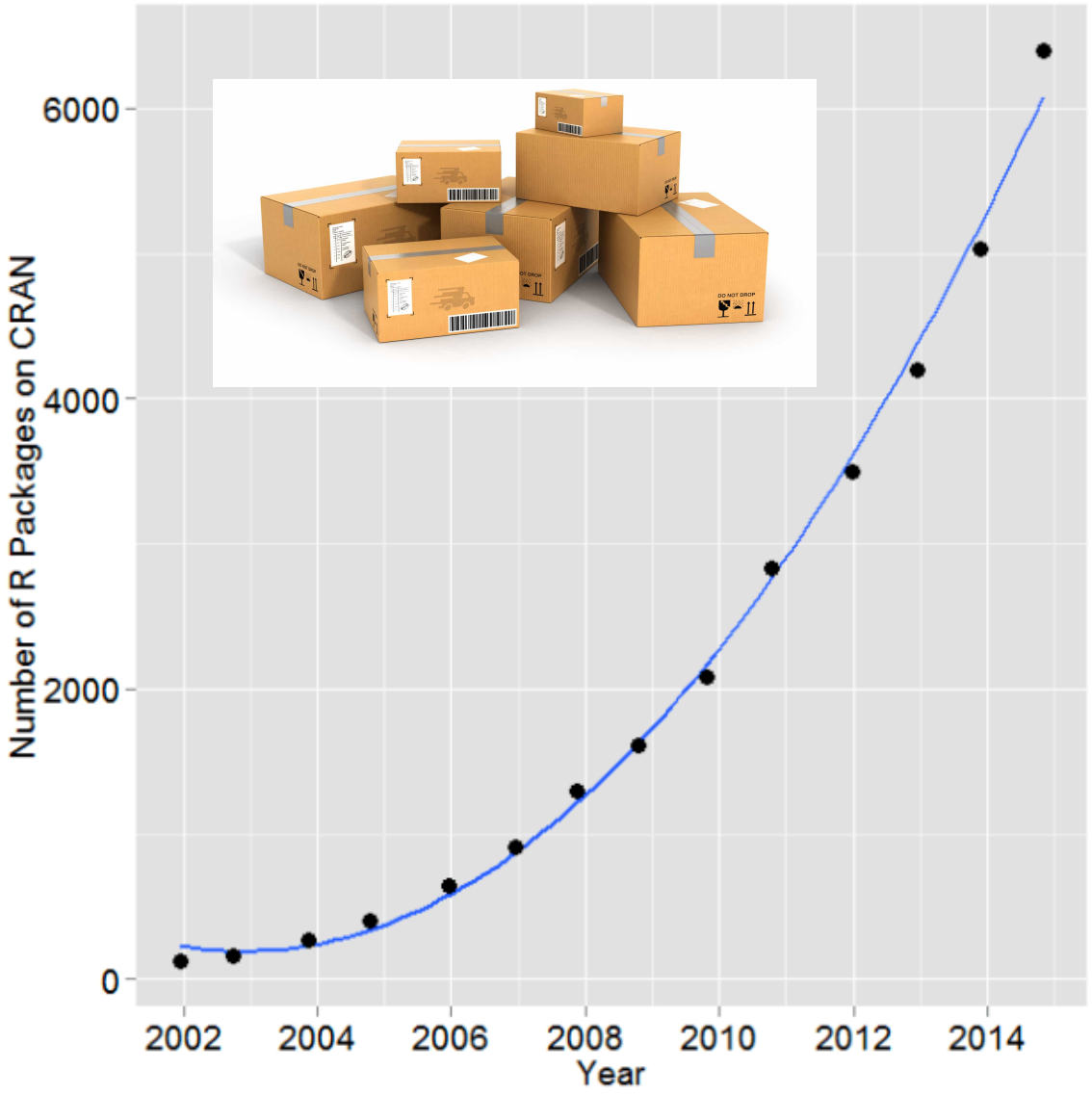
Ya tienes algunos paquetes R cargados automáticamente



Miles de paquetes para explorar

```
dim(available.packages())[1]
```

para ver el numero de paquetes disponibles



dplyr

- Paquete con herramientas para fácil manipulación de datos.
- Creado para data frames
- Rápido (usa C++)

The tidyverse

Components

The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

Usar paquetes en R

1

```
install.packages("pkname")
```

Descargar el paquete

1 x por computador

2

```
library("pkname")
```

Cargar el paquete

1 x por sesión en R

```
install.packages("dplyr")
```

Ahora debemos cargarlo

- Cree un nuevo script en R studio

```
# nombre y fecha en la
parte superior
```

- Limpia tu directorio de trabajo

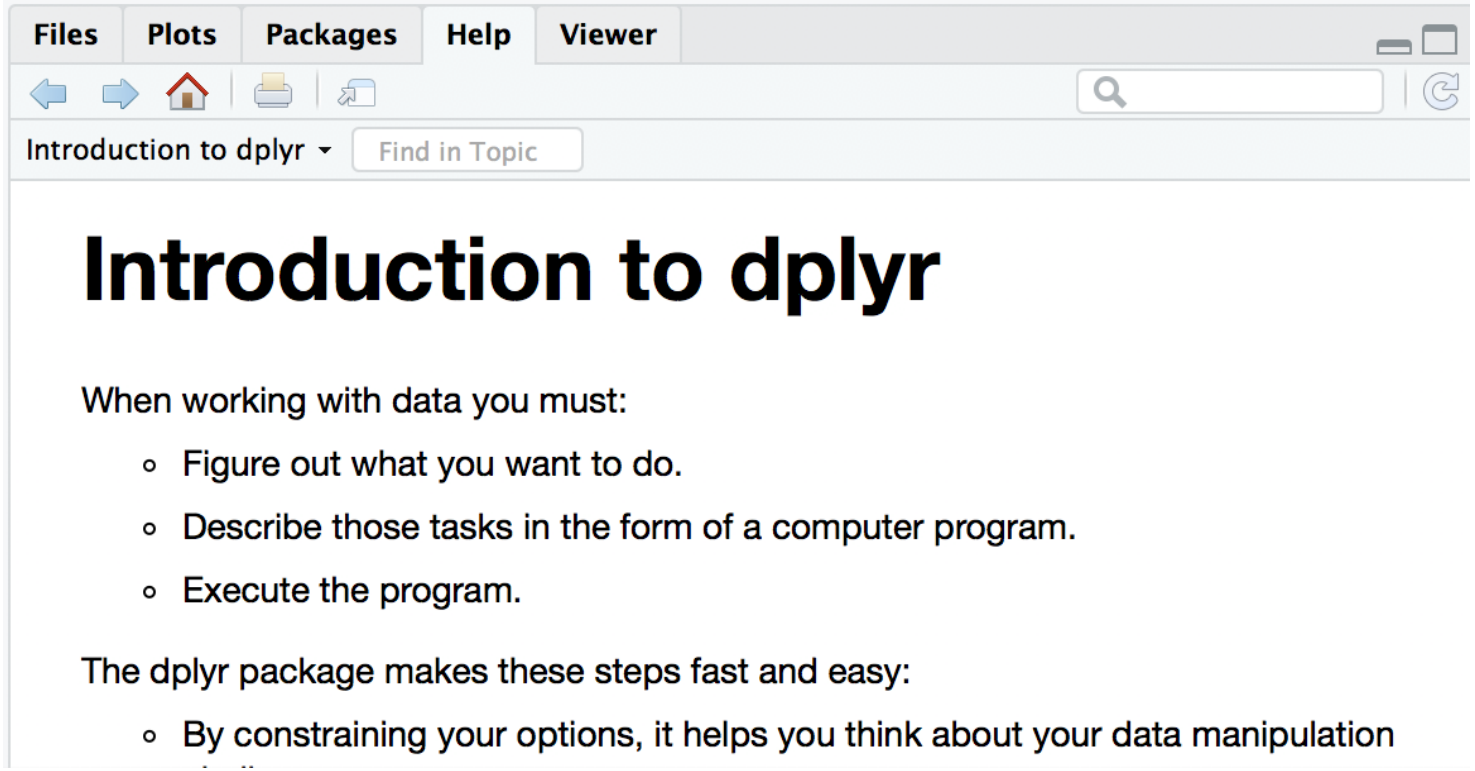
```
rm (list = ls ())
```

- Cargue su nuevo paquete:

```
library (dplyr)
```

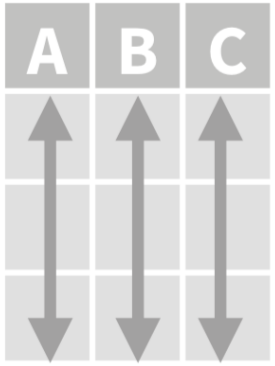
Usen "**vignette**" para leer la guía del paquete

```
vignette("dplyr")
```

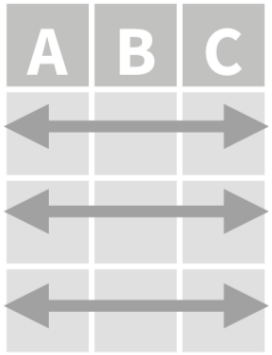


Datos ordenados

dplyr asume que los datos están ordenados



&

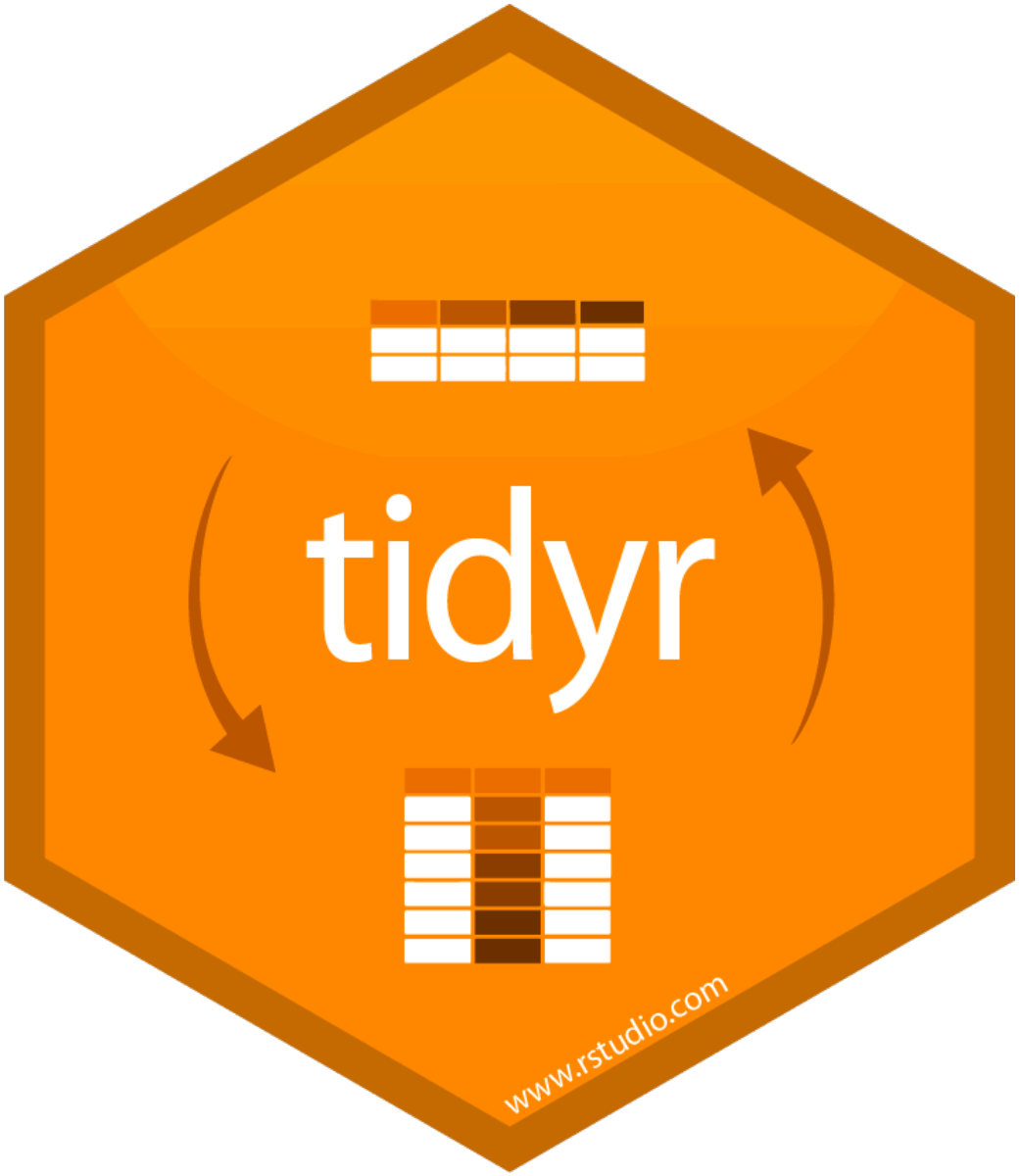


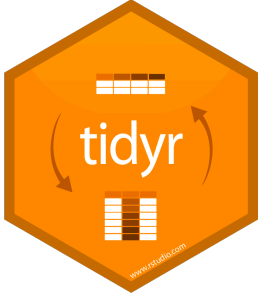
Each **variable** is in its own **column**

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

Hay un paquete adicional para ordenar sus datos

```
install.packages("tidyr")
library("tidyr")
```





Tiene funciones como:

```
install.packages("tidyr")
library("tidyr")
```

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

Separar columnas

```
df <- data.frame(x = c(11, 21, 13, 41))
```

```
   x
1 11
2 21
3 13
4 41
```

(separate)

```
separate(df, x, c("A", "B"),sep=1)
```

```
   A B
1 1 1
2 2 1
3 1 3
4 4 1
```

Tiene funciones como:

Cambiar los datos de formato "ancho" a "largo"

```
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
```

```
> stocks
# A tibble: 10 x 4
   time            X      Y      Z
   <date>      <dbl>  <dbl>  <dbl>
 1 2009-01-01 -1.54  -1.60   0.991
 2 2009-01-02 -1.57   0.737 -4.55
 3 2009-01-03  1.60  -0.326 -1.47
 4 2009-01-04  0.122 -4.91  -4.48
 5 2009-01-05 -1.17   2.31   1.84
 6 2009-01-06  1.19   0.244 -6.20
 7 2009-01-07 -2.28  -0.250 -3.56
 8 2009-01-08  1.12   0.194 -3.25
 9 2009-01-09  0.378 -3.03   7.09
10 2009-01-10  0.515 -0.334 -7.17
```

(gather)

Tiene funciones como:

Cambiar los datos de formato "ancho" a "largo"

```
gather(stocks, "stock", "price", -time)
```

```
> stocks
# A tibble: 10 x 4
   time            X      Y      Z
   <date>      <dbl>  <dbl>  <dbl>
 1 2009-01-01 -1.54  -1.60   0.991
 2 2009-01-02 -1.57   0.737 -4.55
 3 2009-01-03  1.60  -0.326 -1.47
 4 2009-01-04  0.122 -4.91  -4.48
 5 2009-01-05 -1.17   2.31   1.84
 6 2009-01-06  1.19   0.244 -6.20
 7 2009-01-07 -2.28  -0.250 -3.56
 8 2009-01-08  1.12   0.194 -3.25
 9 2009-01-09  0.378 -3.03   7.09
10 2009-01-10  0.515 -0.334 -7.17
```

```
> gather(stocks, "stock", "price", -time)
# A tibble: 30 x 3
   time       stock  price
   <date>     <chr>  <dbl>
 1 2009-01-01 X     -1.54
 2 2009-01-02 X     -1.57
 3 2009-01-03 X      1.60
 4 2009-01-04 X      0.122
 5 2009-01-05 X     -1.17
 6 2009-01-06 X      1.19
 7 2009-01-07 X     -2.28
 8 2009-01-08 X      1.12
 9 2009-01-09 X      0.378
10 2009-01-10 X      0.515
# … with 20 more rows
```

Tiene funciones como:

Cambiar los datos de formato "ancho" a "largo"

```
gather(stocks, "stock", "price", -time)
```

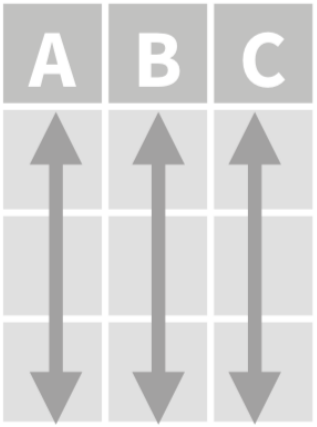
```
> stocks
# A tibble: 10 x 4
   time            X      Y      Z
   <date>      <dbl>  <dbl>  <dbl>
 1 2009-01-01 -1.54  -1.60   0.991
 2 2009-01-02 -1.57   0.737 -4.55
 3 2009-01-03  1.60  -0.326 -1.47
 4 2009-01-04  0.122 -4.91  -4.48
 5 2009-01-05 -1.17   2.31   1.84
 6 2009-01-06  1.19   0.244 -6.20
 7 2009-01-07 -2.28  -0.250 -3.56
 8 2009-01-08  1.12   0.194 -3.25
 9 2009-01-09  0.378 -3.03   7.09
10 2009-01-10  0.515 -0.334 -7.17
```

```
> gather(stocks, "stock", "price", -time)
# A tibble: 30 x 3
   time       stock  price
   <date>     <chr>  <dbl>
 1 2009-01-01 X     -1.54
 2 2009-01-02 X     -1.57
 3 2009-01-03 X      1.60
 4 2009-01-04 X      0.122
 5 2009-01-05 X     -1.17
 6 2009-01-06 X      1.19
 7 2009-01-07 X     -2.28
 8 2009-01-08 X      1.12
 9 2009-01-09 X      0.378
10 2009-01-10 X      0.515
# … with 20 more rows
```

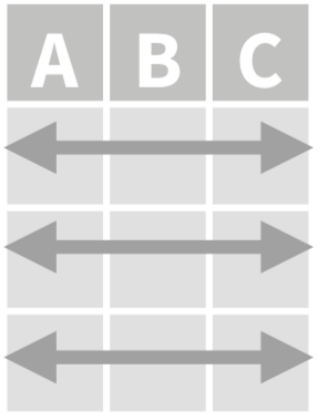
```
pivot_longer(stocks, -time, "stock", "price") #nueva función
```

Supongamos que nuestros datos estan ordenados..

iris es un buen ejemplo



&



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

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

Tibbles

dplyr usa data frames, pero los convierte a “tibbles”

“Es parecido, pero se ve mejor en la consola”

```
Source: local data frame [53,940 x 10]

   carat       cut color clarity depth table
1   0.23     Ideal     E     SI2  61.5    55
2   0.21   Premium     E     SI1  59.8    61
3   0.23      Good     E     VS1  56.9    65
4   0.29   Premium     I     VS2  62.4    58
5   0.31      Good     J     SI2  63.3    58
6   0.24 Very Good     J    VVS2  62.8    57
7   0.24 Very Good     I    VVS1  62.3    57
8   0.26 Very Good     H     SI1  61.9    55
9   0.22      Fair     E     VS2  65.1    61
10  0.23 Very Good     H     VS1  59.4    61
..   ...       ...   ...     ...   ...   ...
Variables not shown: price (int), x (dbl), y
  (dbl), z (dbl)
```

tbl

```
967    55.0  2892  6.12  6.14  3.79
968    59.0  2892  6.62  6.55  4.29
969    55.0  2893  5.66  5.71  3.56
970    56.0  2893  5.99  6.04  3.76
971    57.0  2893  5.73  5.75  3.53
972    57.0  2893  5.78  5.83  3.63
973    60.0  2893  5.87  5.78  3.45
974    55.0  2893  5.89  5.92  3.69
975    62.0  2893  6.02  6.04  3.61
976    55.0  2893  6.00  5.93  3.78
977    59.0  2893  6.09  6.06  3.64
978    57.0  2894  5.91  5.99  3.71
979    57.0  2894  5.96  6.00  3.72
980    56.0  2894  5.88  5.92  3.62
981    56.0  2895  5.75  5.78  3.51
982    59.0  2895  5.66  5.76  3.53
983    53.0  2895  5.71  5.75  3.56
984    58.0  2896  5.85  5.89  3.51
985    60.0  2896  5.81  5.91  3.59
986    63.0  2896  6.00  6.05  3.51
987    56.0  2896  5.18  5.24  3.21
988    56.0  2896  5.91  5.96  3.65
989    55.0  2896  5.82  5.86  3.59
990    56.0  2896  5.83  5.89  3.64
991    58.0  2896  5.94  5.88  3.60
992    57.0  2896  6.39  6.35  4.02
993    57.0  2896  6.46  6.45  3.97
994    57.0  2897  5.48  5.51  3.33
995    58.0  2897  5.91  5.85  3.59
996    52.0  2897  5.30  5.34  3.26
997    55.0  2897  5.69  5.74  3.57
998    61.0  2897  5.82  5.89  3.48
999    58.0  2897  5.81  5.77  3.58
1000   59.0  2898  6.68  6.61  4.03
 [ reached getOption("max.print") --
omitted 52940 rows ]
```

data.frame

Convertir un data frame a tibble:

as_tibble()

```
iris_tib <- as_tibble(iris)
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

Tiene herramientas para:

- Reordenar
- Subconjuntos
- Resumir
- Crear nuevas variables
- Combinar

Reordenar

arrange ordena las filas por valores de una columna de menor a mayor.

```
arrange(iris, Sepal.Width)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.0	2.0	3.5	1.0	versicolor
2	6.0	2.2	4.0	1.0	versicolor
3	6.2	2.2	4.5	1.5	versicolor
4	6.0	2.2	5.0	1.5	virginica
5	4.5	2.3	1.3	0.3	setosa
6	5.5	2.3	4.0	1.3	versicolor

...

Subconjuntos

`filter()` por filas

`select()` por columnas

Subconjuntos

Intenten esto:

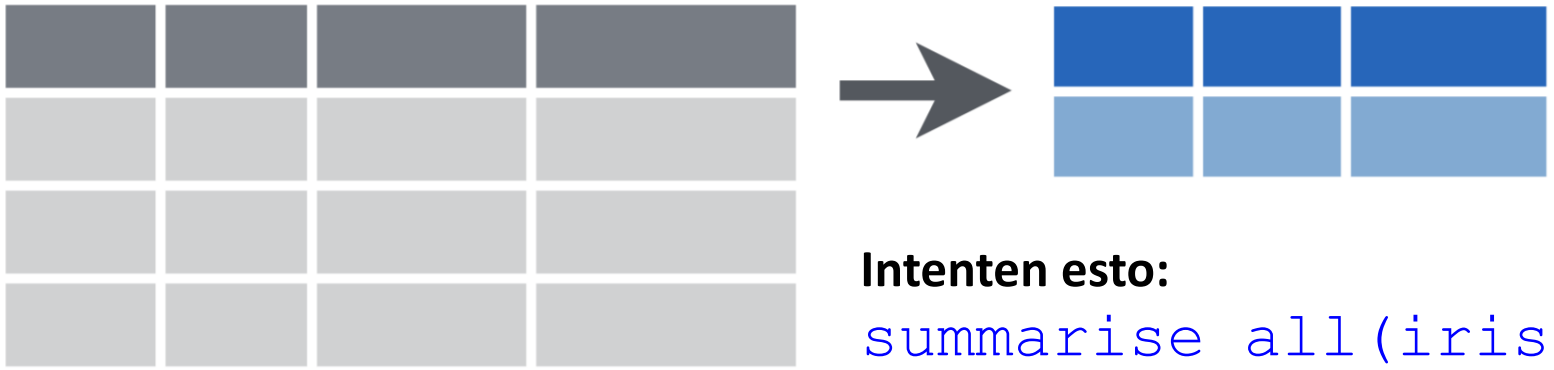
```
filter(iris, Sepal.Length > 7)
```

```
select(iris, Sepal.Width,
       Petal.Length, Species)
```

Resumir

Intenten esto: `summarise(iris, mean(Sepal.Length))`

`summarise_all()` lo hace para todas las columnas



Intenten esto:

`summarise_all(iris, mean)`

Resumir

```
> summarise_all(iris, mean)
  Sepal.Length  Sepal.Width   Petal.Length   Petal.Width  Species
1    5.843333    3.057333       3.758          1.199333     NA
Warning message:
In mean.default(Species) : argument is not numeric or logical: returning NA
```

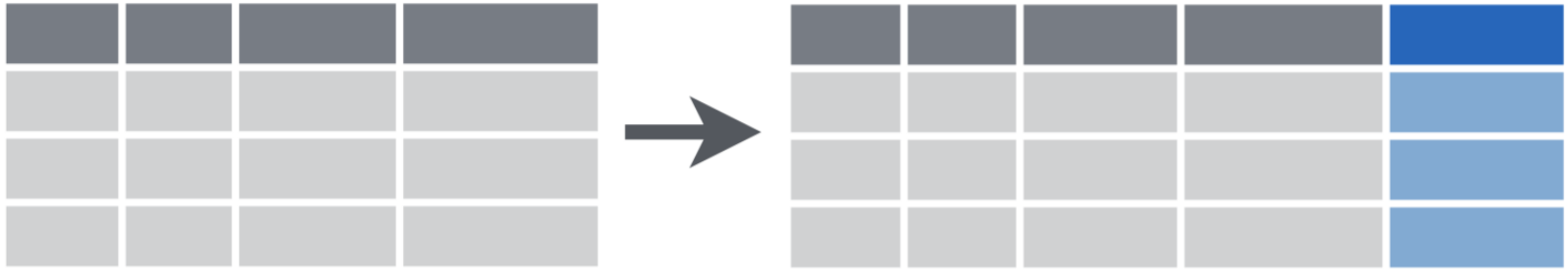
```
> str(iris)
data.frame':          150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Resumir

- `mean()`
- `sd()`
- `min()`
- `max()`
- `sum()`

Crear nuevas variables

`mutate()` para crear nuevas columnas



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

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	sepal
1	5.1	3.5	1.4	0.2	setosa	8.6
2	4.9	3.0	1.4	0.2	setosa	7.9
3	4.7	3.2	1.3	0.2	setosa	7.9
4	4.6	3.1	1.5	0.2	setosa	7.7
5	5.0	3.6	1.4	0.2	setosa	8.6
6	5.4	3.9	1.7	0.4	setosa	9.3

Crear nuevas variables

`between()` genera un vector lógico:

Intenten esto:

```
mutate(iris, sepal_m = between(Sepal.Width,3,4))
```

**creates new column named Sepal_m of T or F*

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	sepal_m
1	5.1	3.5	1.4	0.2	setosa	TRUE
2	4.9	3.0	1.4	0.2	setosa	TRUE
3	4.7	3.2	1.3	0.2	setosa	TRUE
4	4.6	3.1	1.5	0.2	setosa	TRUE
5	5.0	3.6	1.4	0.2	setosa	TRUE
6	5.4	3.9	1.7	0.4	setosa	TRUE

Combinar

(columnas)

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

x1	x2	x3
A	1	T
B	2	F
C	3	NA

Prioritize data in a

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

Use x1 to join by!

Join matching rows from b to a.

Combinar

(columnas)

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

x1	x2	x3
A	1	T
B	2	F
C	3	NA

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

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

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

Prioritize data in b

Join matching rows from a to b.

Combinar

(columnas)

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

x1	x2	x3
A	1	T
B	2	F
C	3	NA

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

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

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

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

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

Join data. Retain only rows in both sets.

Combinar

(columnas)

a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

x1	x2	x3
A	1	T
B	2	F
C	3	NA

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

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

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

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

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

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

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

Join data. Retain all values, all rows.

Tiene herramientas para:

- Reordenar
- Subconjuntos
- Resumir
- Crear nuevas variables
- Combinar

Piping

Agrupamiento

%>%

- Se llama ”piping" y hace que su código sea más legible.

`%>%` Pasa el objeto del lado izquierdo para que sea el primer argumento de la función del lado derecho.

```
data %>% function()
```

Intenten esto:

```
summarise(iris, mean(Sepal.Width))
```

```
iris %>% summarise(mean(Sepal.Width))
```

Piping

Agrupamiento

group_by

dplyr::**group_by(iris, Species)**

Group data into rows with the same value of Species.

iris %>% group_by(Species) %>% summarise(…)

Compute separate summary row for each group.

Intenten esto:

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

```
  Species    `mean(Sepal.Width)`
* <fct>                    <dbl>
1 setosa                    3.43
2 versicolor                2.77
3 virginica                 2.97
```

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Tidy Data - A foundation for wrangling in R

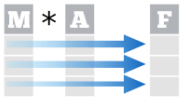
In a tidy data set:

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

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



M * A

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

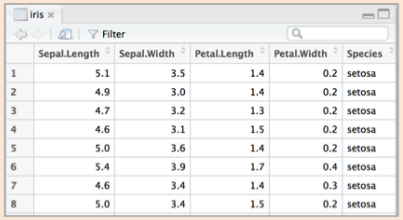
```
Source: local data frame [150 x 5]

   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)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



iris

Filter

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

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

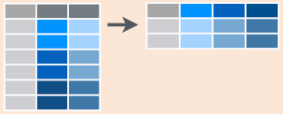
"Piping" with %>% makes code more readable, e.g.

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

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

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

Combine vectors into data frame (optimized).

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

Order rows by values of a column (low to high).

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

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

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

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

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

Randomly select fraction of rows.

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

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

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

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
# of values in a vector.

dplyr::**n_distinct**
# of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

sd
Standard deviation of a vector.

Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.

Make New Variables

dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**
Copy with values shifted by 1.

dplyr::**lag**
Copy with values lagged by 1.

dplyr::**dense_rank**
Ranks with no gaps.

dplyr::**min_rank**
Ranks. Ties get min rank.

dplyr::**percent_rank**
Ranks rescaled to [0, 1].

dplyr::**row_number**
Ranks. Ties got to first value.

dplyr::**ntile**
Bin vector into n buckets.

dplyr::**between**
Are values between a and b?

dplyr::**cume_dist**
Cumulative distribution.

dplyr::**cumall**
Cumulative `all`

dplyr::**cumany**
Cumulative `any`

dplyr::**cummean**
Cumulative `mean`

cumsum
Cumulative `sum`

cummax
Cumulative `max`

cummin
Cumulative `min`

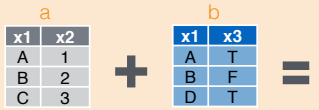
cumprod
Cumulative `prod`

pmax
Element-wise `max`

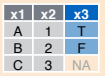
pmin
Element-wise `min`

iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.

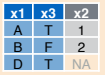
Combine Data Sets



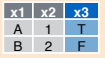
Mutating Joins



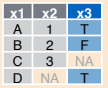
dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.



dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.



dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

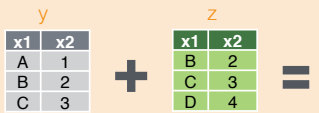


dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Filtering Joins

dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.

dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.



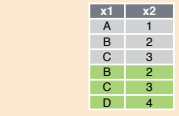
Set Operations

dplyr::**intersect(y, z)**
Rows that appear in both y and z.

dplyr::**union(y, z)**
Rows that appear in either or both y and z.

dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding



dplyr::**bind_rows(y, z)**
Append z to y as new rows.

dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

Lo que aprendimos!

- Cómo instalar paquetes externos
- Que es tidyverse
- Cómo organizar datos usando dplyr