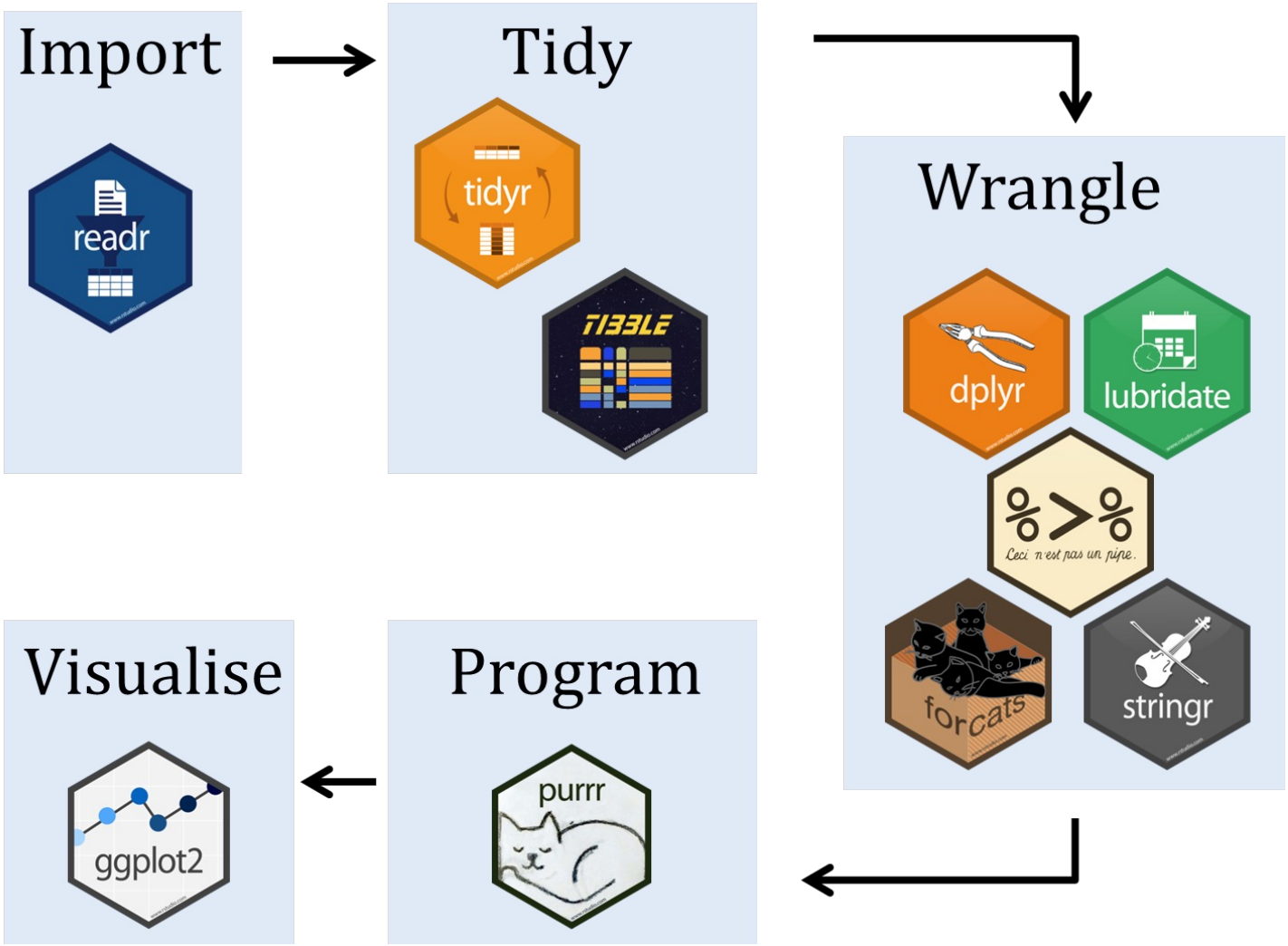
MANIPULATING AND CLEANING DATA

Coral del Val Muñoz

Dpt. Ciencias de la Computación e Inteligencia Artificial,
Universidad de Granada

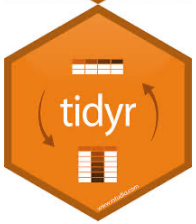
What is tidyverse?

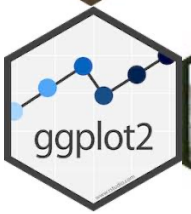
Colección de paquetes con una gramática, filosofía y estructura similar. Se basan en (Wickham and others 2014).



http://www.seec.uct.ac.za/r-tidyverse

What is tidyverse?





- **Data Import**: readr
- **Data Cleaning & Wrangling**: dplyr & tidyr
- **Time Series,** lubridate
- **Text:** stringr
- **Categorical Data**: forcats
- **Visualization**: ggplot2
- **Functions & Iteration:** purrr

Benefits

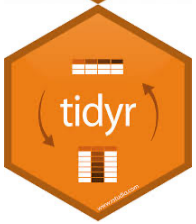
- ***consistent data structure*** - By imposing a uniform data structure, the cognitive load imposed on the analyst is minimized for each new project.

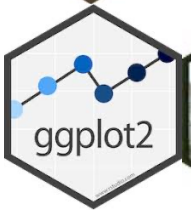
- ***Improves tool development-*** Software that all work within the tidy data framework can all work well with one another

- ***require only a small set of tools to be learned*** - using a consistent data format, tools can be reused from one project to the next.

- ***allow for datasets to be combined*** - Data are often stored in multiple tables or in different locations. By getting each table into a tidy format, combining across tables or sources becomes trivial.

Levels of Data Manipulation with tidyverse





- **Level 1**: Subsetting*
- **Level 2**: Transforming*
- **Level 3:** Joining*
- **Level 4:** Pivoting
- **Level 5**: Colwise*/Rowise transforming
- **Level 6**: Nesting
- **Level 7:** Mapping

readr

- Función: importar (leer) y exportar archivos.
- Mas rápido que la version de r base (approx. 10 veces)

```
# The easiest way to get readr is to install the whole tidyverse
install.packages("tidyverse")

# Alternatively, install just readr
install.packages("readr")

#Usage
library(tidyverse)

mtcars <- read_delim("mtcars.csv" ,";", escape_double = FALSE,
trim_ws = TRUE)
```

Pipe %>%, magrittr

- operador que sirve para realizar varias operaciones de forma secuencial sin recurrir a parentesis anidados o a sobrescribir bases de datos.

```
x <- c(1, 4, 6, 8)
y <- round(mean(sqrt(log(x))), 2)
```

Con %>%

```
x <- c(1, 4, 6, 8)
y <- x %>%
    log() %>%
    sqrt() %>%
    mean() %>%
    round(2)
```

What is dplyr()?

- The dplyr is a powerful R-package to manipulate, clean and summarize unstructured data.
- Makes data exploration and data manipulation easy and fast in R.

To install the dplyr package, type the following command.

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

To load dplyr package, type the command below

```
library(dplyr)
```

What is dplyr()?

- dplyr is a grammar of data manipulation,
- provides a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

filter()

- **Select rows in a** dataframe (df).
- Dataset `starwars`.
- Column `species`

```
> unique(starwars$species)
 [1] "Human"          "Droid"          "Wookiee"        "Rodian"         "Hutt"
 [6] "Yoda's species" "Trandoshan"     "Mon Calamari"   "Ewok"           "Sullustan"
[11] "Neimodian"      "Gungan"         NA               "Toydarian"      "Dug"
[16] "Zabrak"         "Twi'lek"        "Vulptereen"     "Xexto"          "Toong"
[21] "Cerean"         "Nautolan"       "Tholothian"     "Iktotchi"       "Quermian"
[26] "Kel Dor"        "Chagrian"       "Geonosian"      "Mirialan"       "Clawdite"
[31] "Besalisk"       "Kaminoan"       "Aleena"         "Skakoan"        "Muun"
[36] "Togruta"        "Kaleesh"        "Pau’an”
```

```
droids<-starwars %>% filter(species == "Droid")
```

filter()

```
#- Filtrado por cadenas
droids %>% filter(homeworld == “Naboo")


#- filas con valores menores que un valor
droids %>% filter(height < 100)
```

filter() ejercicios

- Instala y carga la librería tidyverse
- Usa el dataset starwars
- Calcula:

```
#- ¿Cuántos androides hay con una altura entre 96 y 200
#- ¿Cuántos androides hay con una altura mayor o igual a 96 que provengan de Naboo?
#- ¿Cuántos androides hay con una altura mayor o igual a 96 que provengan de Naboo o Tatooine?
```

filter()

```
#- filas con valores en un rango
droids %>% filter(height >= 96 , height < 200)
droids %>% filter(height >= 96 & height < 200)

#- filtrado con cadenas y números
droids %>% filter(height >= 96 & homeworld == “Naboo")

#- filtrado con cadenas (varias opciones) y números
droids %>% filter(height >= 96 & homeworld %in% c("Naboo",
"Tatooine") )
```

Other ways to filter rows: slice(), top_n()

slice(): filtra filas por su posición (física en el df)
top_n(): filtra filas por su ranking (según el valor de alguna columna)

```
#- selecciona las observaciones de la decima a la quinceava
starwars %>% slice(c(10:15))
#- selecciona las observaciones de la 12 a 14 Y de la 44 a
46, y las 4 últimas
starwars %>% slice( c(12:14, 44:46, n()-4:n()) )
#- selecciona las 5 filas con mayor valor de height
aa <- df %>% top_n(5, height)
#- selecciona las 4 filas con MENOR valor de birth_year
aa <- df %>% top_n(-4, birth_year)
```

select()

- Allows to **select variables** (columns) from the dataframe
- Probably our database is too big and we only need some of all available variables for our analysis
- We can select the columns that we need or we can exclude those that we do not need

```
# Select these columns
New_dataset<-iris %>% select(Petal.Length, Petal.Width)
```

```
New_dataset
    Petal.Length Petal.Width
1            1.4         0.2
2            1.4         0.2
3            1.3         0.2
4            1.5         0.2
5            1.4         0.2
```

```
# Exclude these columns
New_dataset<-iris %>% select(-Petal.Length, -Petal.Width)
```

select()

- **Select helper functions**

Helpers	Description
`starts_with()`	Starts with a prefix
`ends_with()`	Ends with a prefix
`contains()`	Contains a literal string
`matches()`	Matches a regular expression
`num_range()`	Numerical range like x01, x02, x03.
`one_of()`	Variables in character vector.
`everything()`	All variables.

```
# Select all columns with “color” in its name
Df<-iris %>% select(starts_with(”Sepal"))
```

select()

```
# Selecciona todas las variables que terminen con la palabra
color
# Selecciona todas las variables que no contengan la expresion
eye_color
# Selecciona todas las variables que no contengan la expresion
eye_color o gender
```

select()

```
# Select all columns with “color” in its name
starwars %>% select(name, ends_with("color"))

#- select all columns bur eye_color
starwars %>% select(-eye_color)

starwars %>% select(-c(gender, species))
```

arrange()

- **Reorder rows in a** dataframe (df).

```
#- order in ascendent order according to the variable height
starwars %>% arrange(height)

#- order in descendent order according to the variable height
starwars %>% arrange(desc(height))

#- order in ascendent order according to the variable height
#- if there are two rows with the same value resolve with
birth_year
starwars %>% arrange(height, birth_year)
```

rename()

- **Rename columns in a** dataframe (df).

```
#- changes the name of the variable hair_color
starwars %>% rename(hair = hair_color)


#-la función names() es muy útil.
aa<-starwars
names(aa) <- names(aa) %>% toupper
names(aa) <- names(aa) %>% tolower
```

mutate()

- **Create new columns in a** dataframe (df).

```
#- Creamos una variable nueva y modificamos una existente
install.packages("gapminder")
library(gapminder)
X<-gapminder %>%
    mutate(pop = pop / 1000000, gdp = gdpPercap*pop) %>%
    head()
```

We can modify existent columns (e.g. first case) or we can create new variables (columns) as in the case of gdp

mutate()

- **Create new columns in a** dataframe (df).

```
#- Crea la variable: indice de masa corporal con el dataset
starwars. Masa coporal es el peso/altura
```

```
#- Crea la variable: IMC
aa<-starwars %>% mutate(IMC = mass/height)
aa$IMC
```

summarise()

- **Allows to colapse/summarise rows in a** dataframe (df).

```
#- retornará un único valor: la media global de la v. "height"
aa <-starwars %>%
    summarise(media = mean(height, na.rm=TRUE))


#- retorna el número de filas
Nfilas<-starwars %>% summarise(NN = n())
```

summarise()

```
#- retornará  la desviación típica de  "height"
starwars %>% summarise(desviacion_tipica = sd(height, na.rm=TRUE))

#- retornará el máximo de la variable "mass"
starwars %>% summarise(max(mass, na.rm=TRUE))

#- retornará 2 valores: la media y sd de la v. "height"
starwars %>% summarise(mean(height, na.rm=TRUE), sd(height, na.rm=TRUE))

#- retornará 2 valores: las medias de "height" y "mass"
starwars %>% summarise(mean(height, na.rm=TRUE), mean(mass, na.rm=TRUE))
```

summarise()

```
starwars %>%
        group_by(species) %>%
        summarise(n = n(),
                  mass = mean(mass, na.rm = TRUE))
```

```
1 Aleena        1  15
 2 Besalisk      1 102
 3 Cerean        1  82
 4 Chagrian      1 NaN
 5 Clawdite      1  55
 6 Droid         5  69.8
 7 Dug           1  40
 8 Ewok          1  20  …..
```

Añade el código que necesites para
Ver solo aquellas especies que tengan mas de dos individuos y que la media de peso sea mayor a 50 kg

summarise()

```
starwars %>%
        group_by(species) %>%
        summarise(n = n(),
                  mass = mean(mass, na.rm = TRUE)) %>%
        filter(n > 2, mass > 50)
```

```
 # A tibble: 3 x 3
  species     n  mass
  <chr>   <int> <dbl>
1 Droid       5  69.8
2 Gungan      3  74
3 Human      35  82.8
```

summarise():

Useful functions to use with sumarise()

Center: mean(), median()

Spread: sd(), IQR(), mad()

Range: min(), max(), quantile()

Position: first(), last(), nth(),

Count: n(), n_distinct()

Logical: any(), all()

```
mtcars %>%
group_by(cyl) %>%
summarise(qs = quantile(disp, c(0.25, 0.75)), prob = c(0.25,
0.75))
```

Other auxiliar dplyr functions

```
dplyr::ntile(x, n) :
categorizes a vector of values into "ntiles" such as
quartiles if n = 4

dplyr::n_distinct(x):
counts unique values in a vector;
similar a length(unique(x))

#- crear una columna con el índice de rows
starwars %>% mutate(index = 1:n())

# recoge 5 datos aleatorios de starwars
sample_n(starwars, 5)

# recoge una muestra aleatoria de datos en una cantidad fija
sample_frac(starwars, 0.2) %>% head()
```

Todo junto

```
#- Crea una tabla basada en el dataframe starwars
en el que tengas por cada especie el numero de
individuos que hay y en cuantos planetas viven.
Ordenar los resultados de mayor a menor según el
número de individuos de la especie
```

sumarize()

```
#- cogemos df y lo agrupamos por "specie",
#- despues calculamos 2 cosas: el numero de observaciones o rows
#- y el número de mundos en los que vive cada especie (NN_countries)
starwars %>% group_by(species) %>%
        summarize(NN = n(), NN_countries = n_distinct(homeworld))%>%
        arrange(desc(NN_countries))
A tibble: 38 x 3
   species     NN NN_countries
   <chr>    <int>        <int>
 1 Human       35           16
 2 Droid        5            3
 3 NA           5            3
 4 Zabrak       2            2
 5 Aleena       1            1
```

sumarize()

- **Allows** to calculate operations for different groups in a dataframe (droids, humans...)

```
#- cogemos df y lo agrupamos por "specie",
#- despues calculamos 2 cosas: el numero de observaciones o rows
#- y el número de mundos en los que vive cada especie (NN_countries)
starwars %>% group_by(species) %>%
        summarize(NN = n(), NN_countries = n_distinct(homeworld))%>%
        arrange(desc(NN_countries))
A tibble: 38 x 3
   species     NN NN_countries
   <chr>    <int>        <int>
 1 Human       35           16
 2 Droid        5            3
 3 NA           5            3
 4 Zabrak       2            2
 5 Aleena       1            1
```

- The scoped variants of summarise() make it easy to apply the same transformation to multiple variables. There are three variants.
- summarise_all() affects every variable
- summarise_at() affects variables selected with a character vector or vars()
- summarise_if() affects variables selected with a predicate function

Summarise_all()

```
# The _at() variants directly support strings:
starwars %>%
summarise_at(c("height", "mass"), mean, na.rm = TRUE)

# You can supply selection helpers to _at() functions but
# quote them with vars():
starwars %>%
summarise_at(vars(height:mass), mean, na.rm = TRUE)

# The _if() variants apply a predicate function (a function that
# returns TRUE or FALSE) to determine the relevant subset of
# columns.
starwars %>%
summarise_if(is.numeric, mean, na.rm = TRUE)
```

Summarise_all()

```
# To apply multiple transformations, pass a list of
# functions.
by_species %>% summarise_all(list(min, max))

# the new variables include the function name, in order to
# keep things distinct. Passing purrr-style lambdas often creates #
better default names:
by_species %>% summarise_all(list(~min(.), ~max(.)))

# When that's not good enough, you can also supply the names
explicitly: by_species %>% summarise_all(list(min = min, max =
max))
# When there's only one function in the list, it modifies existing
# variables in place. Give it a name to create new variables
instead:
by_species %>% summarise_all(list(med = median))
by_species %>% summarise_all(list(Q3 = quantile), probs = 0.75)
```

Ordena de forma descendiente el numero de especies en el data frame starwars teniendo en cuenta solo aquellas que tienen valores

- Dataset starwars.
- Selecciona las filas en el dataframe (df) que tienen valores en esa columna
- Cuenta los elementos que hay de cada especie
- Ordenalos de forma descendente

count()

```
starwars %>%
filter(!is.na(species)) %>%
count(species, sort = TRUE)

#> # A tibble: 37 x 2
#> species n
#> <chr> <int>
 #> 1 Human 35
 #> 2 Droid 6
 #> 3 Gungan….
```

Condicionales: case_when()

```
x <- c(-2, -1, 0, 1, 2)
case_when(x < 0 ~ "Negative",
          x > 0 ~ "Positive",
          TRUE ~ "Zero")

#> [1] "Negative" "Negative" "Zero" "Positive"
"Positive"
```

Nos permite vectorizar, de forma similar a ifelse() en R base pero nos permite Dar como salida otro tipo de valores a parte de TRUE y FALSE

Condicionales: case_when()

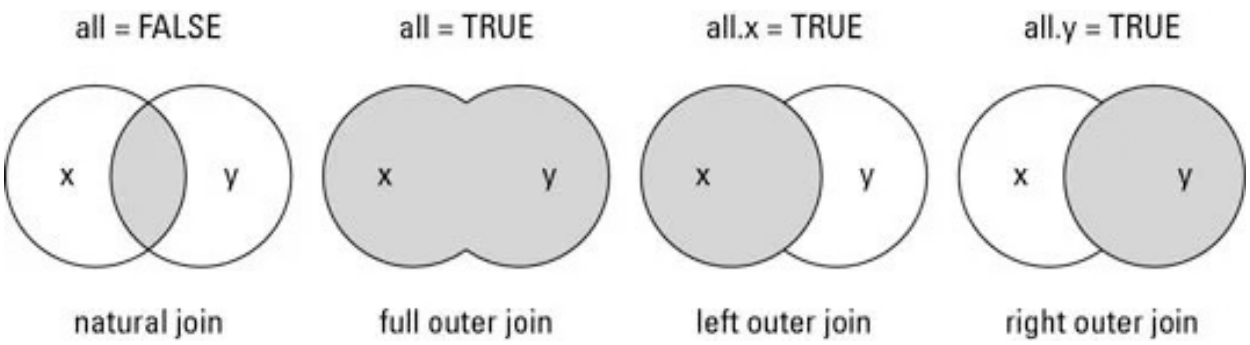
Su uso principal es para agrupar variables categóricas

```
murders %>%
 mutate(new_grouping = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "New England",
    abb %in% c("WA", "OR", "CA") ~ "West Coast",
    region == "South" ~ "South",
    TRUE ~ "Other")) %>%
 group_by(new_grouping) %>%
 summarize(rate = sum(total) / sum(population) * 10^5)
#> # A tibble: 4 × 2
#> group rate
#> <chr> <dbl>
#> 1 New England 1.72
#> 2 Other 2.71
#> 3 South 3.63
#> 4 West Coast 2.90
```

Data Manipulation

Implica cuatro funciones:

- right_join()
- left_join ()
- inner_join() -> intersection
- full_join() -> union
- Semi_join()
- Anti_join()



Based on Statistics Globe tutorials

Inner_join()

X

ID	Var1
1	x1
2	x2

Y

ID	Var2
2	z1
3	z2

ID	Var1	Var2
2	x2	z1

```
inner_join(x, y, by = "ID")
```

left_join()

ID	Var1
1	x1
2	x2

ID	Var2
2	z1
3	z2

ID	Var1	Var2
1	x1	NA
2	x2	z1

```
left_join(x, y, by = "ID")
```

Devuelve todas las flas de X y todas las columnas de X e Y. Si falta algun valor X en Y se introduce NA

right_join()

X

ID	Var1
1	x1
2	x2

Y

ID	Var2
2	z1
3	z2

ID	Var1	Var2
2	x2	z1
3	NA	z2

```
right_join(x, y, by = "ID")
```

Devuelve todas las flas de Y y todas las columnas de X e Y. Valores de Y sin presencia en X se pondra NA

full_join()

X

ID	Var1
1	x1
2	x2

Y

ID	Var2
2	z1
3	z2

ID	Var1	Var2
1	x1	NA
2	x2	z1
3	NA	z2

```
full_join(x, y, by = "ID")
```

Devuelve todas las filas y columnas de X e Y. Se pondra NA para valores no existentes. Union

semi_join()

X

ID	Var1
1	x1
2	x2

Y

ID	Var2
2	z1
3	z2

ID	Var1
2	x2

```
semi_join(x, y, by = "ID")
```

Devuelve todas las flas de X presentes en Y manteniendo solo las columnas de X. Extrae los valores de X presentes en Y

anti_join()

X

ID	Var1
1	x1
2	x2

Y

ID	Var2
2	z1
3	z2

ID	Var1
1	x1

```
semi_join(x, y, by = "ID")
```

Devuelve todas las flas de X NO presentes en Y manteniendo solo las columnas de X. Extrae los valores de X diferentes de Y

Useful Packages

Pre-modeling stage

Data visualization:
ggplot2, googleVis

Data Transformation:
plyr, dplyr, data.table

Missing value Imputations: Missforest, MissMDA

Outliers Detection: Outliers, EVIR

Gracias…

tally

- tally() is a convenient wrapper for summarise that will either call n() or sum(n) depending on whether you're tallying for the first time, or re-tallying. count() is similar but calls group_by() before and ungroup() after. If the data is already grouped, count() adds an additional group that is removed afterwards.
- add_tally() adds a column n to a table based on the number of items within each existing group, while add_count() is a shortcut that does the grouping as well. These functions are to tally() and count() as mutate() is to summarise(): they add an additional column rather than collapsing each group.

- Be consistent
- Choose good names for things
- Write dates as YYYY-MM-DD
- No empty cells
- Put just one thing in a cell
- Don’t use font color or highlighting as data
- Save the data as plain text files