BSM

Blockchain School
for Management

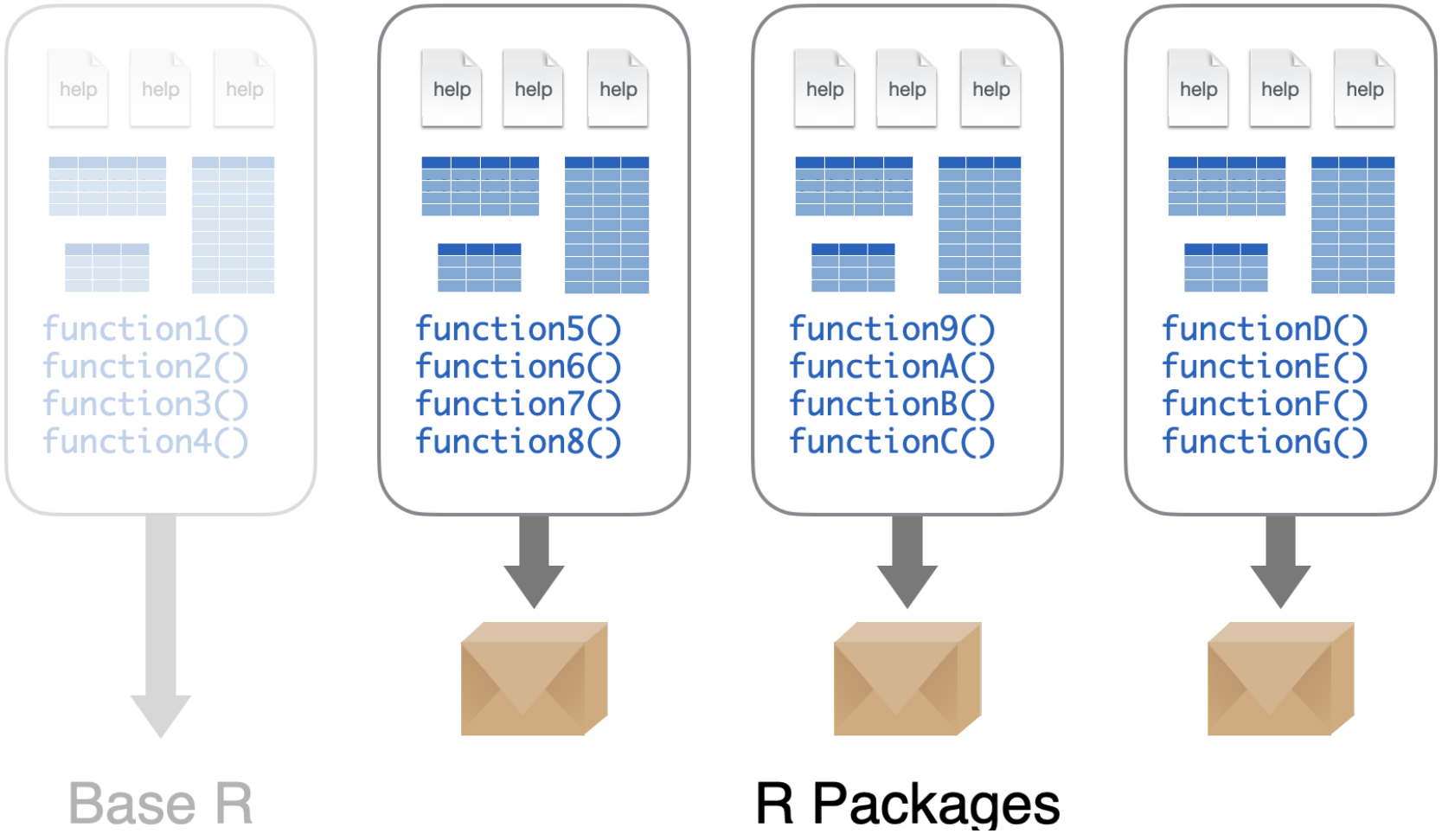
Tema 6

Módulo II: R. Manipulación de Datos I

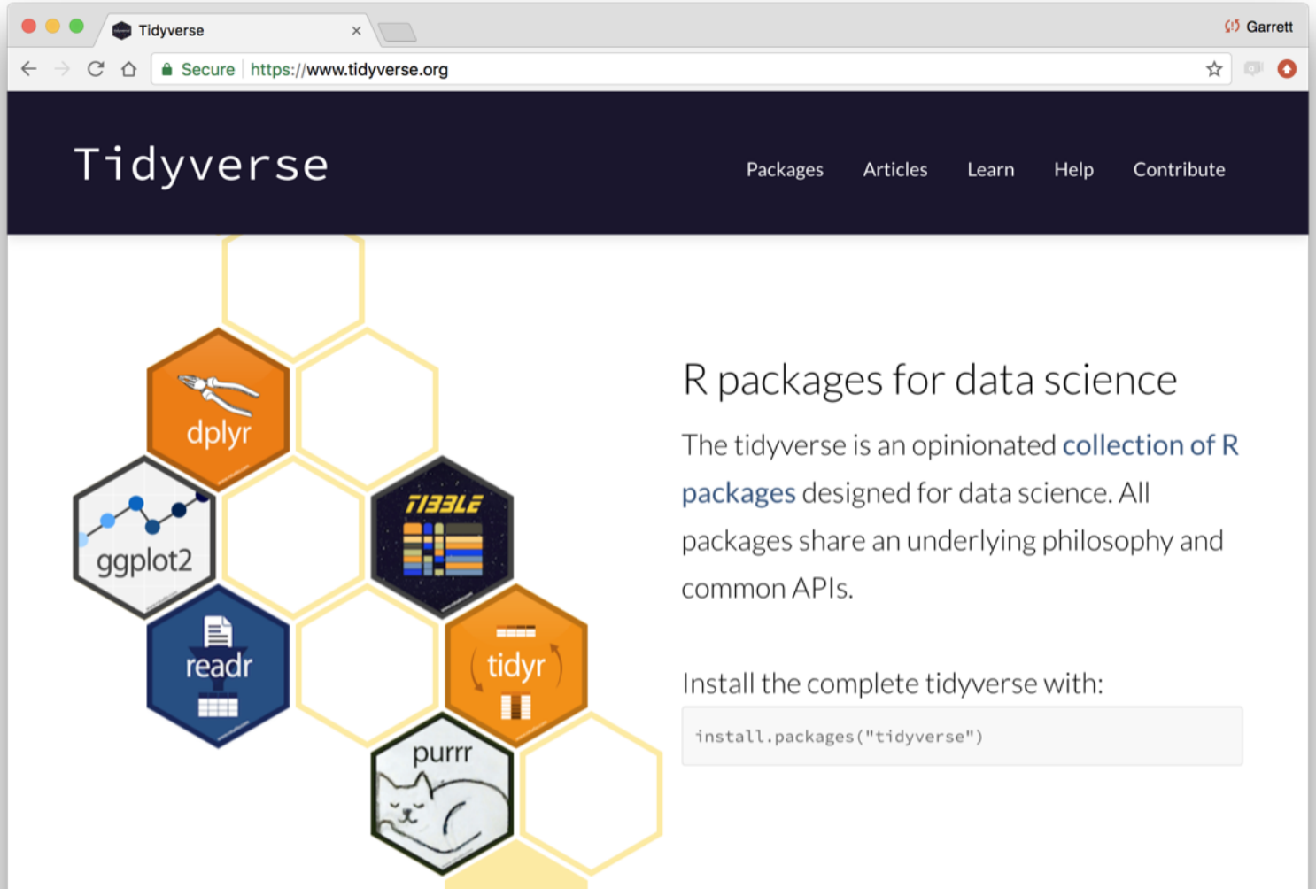
Nicolás Forteza

2022-11-15

Librerías


help
help
help
function1()
function2()
function3()
function4()
Base R
help
help
help
function5()
function6()
function7()
function8()
help
help
help
function9()
functionA()
functionB()
functionC()
R Packages
help
help
help
functionD()
functionE()
functionF()
functionG()

Transform Data with

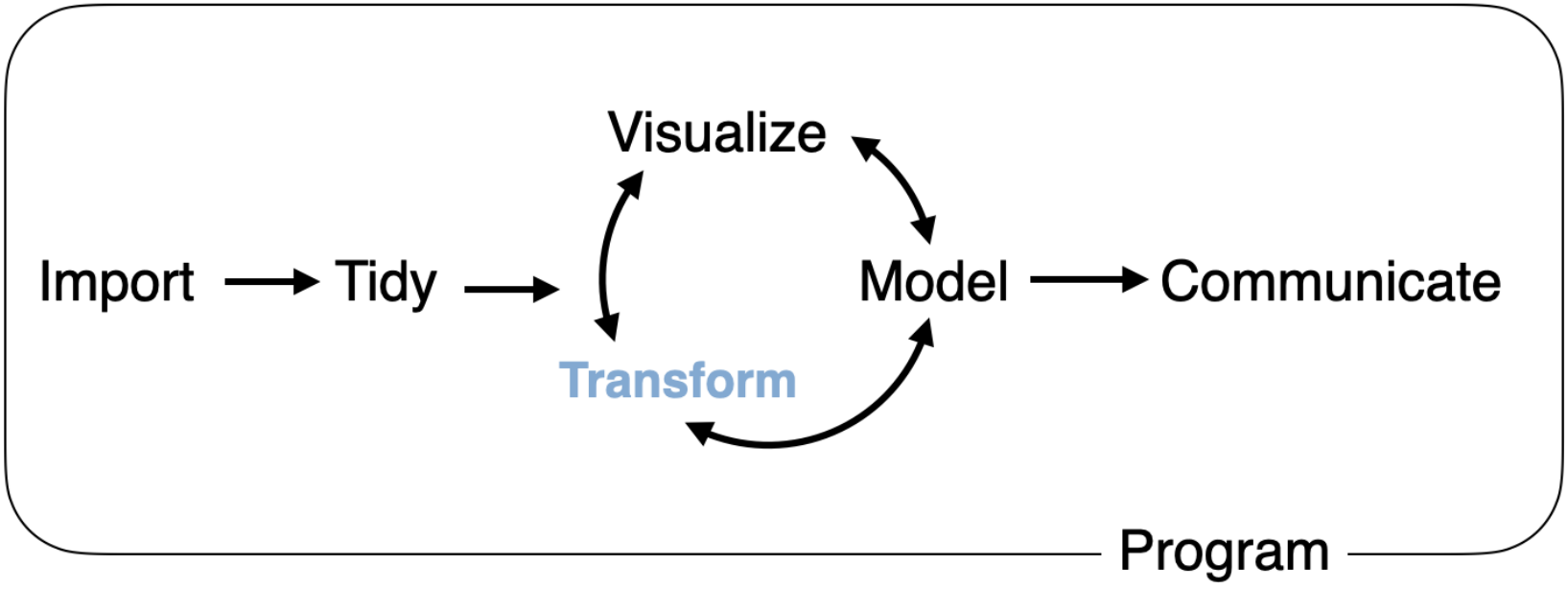

Tidyverse
Garrett
Secure
https://www.tidyverse.org
Tidyverse
Packages
Articles
Learn
Help
Contribute
dplyr
ggplot2
tibble
readr
tidyr
purrr
R packages for data science
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.
Install the complete tidyverse with:
install.packages("tidyverse")

Paquetes de Tidyverse

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("hms")
install.packages("stringr")
install.packages("lubridate")
install.packages("forcats")
install.packages("DBI")
install.packages("haven")
install.packages("httr")
install.packages("jsonlite")
install.packages("readxl")
install.packages("rvest")
install.packages("xml2")
install.packages("modelr")
```

Instalar Tidyverse

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


Visualize
Import
Tidy
Model
Communicate
Transform
Program

Transformar

Transformar los datos quiere decir todo lo relacionado con esto:

- Limpiar (quitar nulos, vacíos, etc)
- Manipular (seleccionar, ordenar, filtrar)
- Editar (nuevas variables, agrupar observaciones)

Data wrangling

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. The goal of data wrangling is to assure quality and useful data. Data analysts typically spend the majority of their time in the process of data wrangling compared to the actual analysis of the data.

Vamos a ver diferences formas de seleccionar columnas de una variable.

Vamos a usar el siguiente conjunto de datos:

```
library(tidyverse)
df <- msleep
```

```
glimpse(df)
```

```
Rows: 83
Columns: 11
$ name         <chr> "Cheetah", "Owl monkey", "Mountain bea
$ genus        <chr> "Acinonyx", "Aotus", "Aplodontia", "Bl
$ vore         <chr> "carni", "omni", "herbi", "omni", "her
$ order        <chr> "Carnivora", "Primates", "Rodentia", "
$ conservation <chr> "lc", NA, "nt", "lc", "domesticated",
$ sleep_total  <dbl> 12.1, 17.0, 14.4, 14.9, 4.0, 14.4, 8.7
$ sleep_rem    <dbl> NA, 1.8, 2.4, 2.3, 0.7, 2.2, 1.4, NA,
$ sleep_cycle  <dbl> NA, NA, NA, 0.1333333, 0.6666667, 0.76
$ awake        <dbl> 11.9, 7.0, 9.6, 9.1, 20.0, 9.6, 15.3,
$ brainwt      <dbl> NA, 0.01550, NA, 0.00029, 0.42300, NA,
$ bodywt       <dbl> 50.000, 0.480, 1.350, 0.019, 600.000,
```

La función `glimpse` permite ver las columnas de un modo vertical y no horizontal, a diferencia con el el método `print` o `head`.

Operadores: para usar las funciones de tidyverse, podemos usar el *pipe operator*.

Se hace de la siguiente forma:

```
df %>% summary()
```

Operador *pipe*: Se puede crear con un atajo de teclado - ctrl + shift + m (windows) - cmd + shift + m (mac)

Para seleccionar las columnas de un dataframe, simplemente pon los nombres de esas columnas en la función `select`. El orden en el que introduces las columnas importa.

```
df %>%
  select(name, genus, sleep_total, awake) %>%
  glimpse()
```

Podemos seleccionar desde una columna hasta otra, con simplemente poner dos columnas entre los dos puntos.

```
df %>%
  select(name:order, sleep_total:sleep_cycle) %>%
  glimpse()
```

Podemos *deseleccionar* columnas, es decir, coger todas menos las que le pongo un - delante. Esto aplica también a trozos de columnas como veíamos antes.

```
df %>%
  select(-conservation, -(sleep_total:awake)) %>%
  glimpse()
```

Es posible deseleccionar un cacho entero y luego añadir una columna o varias como se ve a continuación.

```
msleep %>%
  select(-(name:awake), conservation) %>%
  glimpse
```

Se pueden seleccionar columnas usando reglas de *matching* de strings (cadenas de caracter). Aquí vemos cómo seleccionamos las variables que empiezan por la palabra "sleep".

```
msleep %>%
  select(name, starts_with("sleep")) %>%
  glimpse
```

También podemos seleccionar columnas que terminan con una palabra, o contiene otra palabra (o trozo de palabra).

```
msleep %>%
  select(contains("eep"), ends_with("wt")) %>%
  glimpse
```

Se pueden usar, en consecuencia, expresiones regulares. Pero no nos vamos a parar de momento a ver cómo funcionan las expresiones regulares.

```
msleep %>%
  select(matches("o.+er")) %>%
  glimpse
```

Hay una opción que permite seleccionar columnas que están como caracter dentro de un vector como se puede ver a continuación.

```
classification <- c("name", "genus", "vore", "order", "cons

msleep %>%
  select(!!classification)
```

Podemos seleccionar variables que cumplan una condición. En este caso, que sean numéricas. Esto facilita mucho las cosas...

```
msleep %>%
  select_if(is.numeric) %>%
  glimpse
```

O bien podemos usar operadores lógicos para coger en este caso las variables que no son numéricas. Destacar que hay que usar el caracter ~ para asegurar de que dplyr lo entiende bien.

```
msleep %>%
  select_if(~!is.numeric(.)) %>%
  glimpse
```

No sólo podemos usar funciones que empiezan por `is`, si no que además podemos hacer que los datos cumplan una condición. Hay que añadir el caracter ~ de nuevo si hacemos esto. En este caso, filtrando por las columnas cuya media se encuentrar por encima de 10.

```
msleep %>%
  select_if(is.numeric) %>%
  select_if(~mean(., na.rm=TRUE) > 10)
```

Lo anterior se puede reescribir de la siguiente manera:

```
msleep %>%
  select_if(~is.numeric(.) & mean(., na.rm=TRUE) > 10)
```

Una función muy útil es la siguiente, que permite coger las columnas cuyos valores únicos se encuentran por encima o por debajo de un umbral.

```
msleep %>%
  select_if(~n_distinct(.) < 10)
```

Puedes seleccionar las columnas que quieres para que tengas un dataframe ordenado del modo que quieres.

```
msleep %>%
  select(conservation, sleep_total, name) %>%
  glimpse
```

Puedes además, coger las que quieres, y luego el resto con al función `everything`.

```
msleep %>%
  select(conservation, sleep_total, everything()) %>%
  glimpse
```

Puedes, cuando seleccionas columnas, renombrar las variables a tu gusto también.

```
msleep %>%
  select(animal = name, sleep_total, extinction_threat = co
  glimpse
```

O si quieres mantener todas las columnas y renombrar simplemente algunas, haz lo siguiente:

```
msleep %>%
  rename(animal = name, extinction_threat = conservation) %
  glimpse
```

La función `select_all` va a seleccionar todas las columnas. En este caso, selecciona todas y convierte los nombres de las mismas en letras mayúsculas

```
msleep %>%
  select_all(toupper)
```

Aquí por ejemplo, lo que estamos haciendo es cambiar los nombres sólamente de algunas variables.

```
msleep2 <- select(msleep, name, sleep_total, brainwt)
colnames(msleep2) <- c("name", "sleep total", "brain weight
```

Les estamos añadiendo una _.

```
msleep2 %>%
  select_all(~str_replace(., " ", "_"))
```

Puedes usar la función de seleccionar todas las variables junto con la de reemplazar strings para deshacerte de caracteres no deseados en los nombres de las columnas.

```
msleep2 %>%
  select_all(~str_replace(., "Q[0-9]+", "")) %>%
  select_all(~str_replace(., " ", "_"))
```

A veces, es interesante pasar los nombres de las filas a una columna.

```
mtcars %>%
  tibble::rownames_to_column("car_model") %>%
  head
```

Ejercicio

- Carga los datos `diamonds` en memoria.
- Cambiar el nombre de la variable depth por depth_%
- Cambia los nombres de las variables x, y, z por length, width, depth.
- Selecciona primero las variables que acabas de modificar y luego el resto.
- Selecciona la variable cut, pero previamente habiendo filtrado por aquellos diamantes cuyo precio es mayor o igual que 336.
- Selecciona las columnas de tipo caracter que tienen más de 4 categorías únicas.

Ahora vamos a ver otro tipo de operaciones muy usada que ya hemos visto. Se trata del uso de máscaras, filtros, etc. ... para devolver un conjunto de observaciones del dataframe que cumplen una o varias condiciones.

En Dplyr, el comando `filter()` será el encargado de realizar esto.

Puedes usar los operadores que ya hemos visto en lecciones anteriores, para usarlos sobre variables numéricas.

```
msleep %>%
  select(name, sleep_total) %>%
  filter(sleep_total > 18)
```

Si quieres seleccionar un rango de valores, puedes usar el comando `between()`.

```
msleep %>%
  select(name, sleep_total) %>%
  filter(between(sleep_total, 16, 18))
```

Otra función muy útil es la de `near()`, la cual devuelve las observaciones que se encuentran cerca de un valor. Tienes que especificar el parámetro o argumento `tol` de la función para indicar cuán lejos se encuentran los valores.

Por ejemplo, en el siguiente código, se devuelven las filas que se encuentran dentro de una desviación estándar de la variable que se está usando para filtar.

```
msleep %>%
  select(name, sleep_total) %>%
  filter(near(sleep_total, 17, tol = sd(sleep_total)))
```

Ejercicio

- Comprueba, dada la slide anterior, que efectivamente lo seleccionado se encuentra dentro de una desviación estándar de 17.

Para filtrar variables de tipo caracter o factor, puedes usar el operador ==.

```
msleep %>%
  select(order, name, sleep_total) %>%
  filter(order == "Didelphimorphia")
```

De manera similar, puedes usar usar operadores del siguiente modo

```
msleep %>%
  select(order, name, sleep_total) %>%
  # selecciona todo excepto "Rodentia"
  filter(order != "Rodentia")
```

Si quieres seleccionar más de una categoría o nivel, puedes usar el operador %in%.

```
msleep %>%
  select(order, name, sleep_total) %>%
  filter(order %in% c("Didelphimorphia", "Diprotodontia"))
```

También puedes realizar la operación justamente opuesta.

```
remove <- c("Rodentia", "Carnivora", "Primates")
msleep %>%
  select(order, name, sleep_total) %>%
  filter(!order %in% remove)
```

Para hacer uso de expresiones regulares, podemos usar varias librerías: `grepl`, que viene ya con R, o `stringr`. La última se suele usar más por su versatilidad.

```
msleep %>%
  select(name, sleep_total) %>%
  filter(str_detect(tolower(
    name), pattern = "mouse"))
```

Podemos usar como hemos dicho, múltiples condiciones, de la siguiente manera:

`filter(condition1, condition2)`: devuelve filas donde se cumplen ambas condiciones. `filter(condition1, !condition2)`: devuele filas donde se cumple una y la otra no. `filter(condition1 | condition2)`: devuelve filas donde una de las condiciones se cumpla. `filter(xor(condition1, condition2)`: devuelve filas donde sólo una de las condiciones se cumple, no las dos a la vez.

Se pueden combinar diferentes condiciones:

```
msleep %>%
  select(name, order, sleep_total:bodywt) %>%
  filter(bodywt > 100, (sleep_total > 15 |
                          order != "Carnivora"))
```

```
msleep %>%
  select(name, bodywt:brainwt) %>%
  filter(xor(bodywt > 100, brainwt > 1))
```

```
msleep %>%
  select(name, sleep_total, brainwt, bodywt) %>%
  filter(brainwt > 1, !bodywt > 100)
```

Para filtrar filas vacías, puedes realizar la negación con `is.na()`.

```
msleep %>%
  select(name, conservation:sleep_cycle) %>%
  filter(!is.na(conservation))
```

Otra funcionalidad más avanzada de Dplyr es poder filtrar múltiples columnas de la siguiente manera:

- `filter_all()`
- `filter_if()`
- `filter_at()`

Por ejemplo, aquí filtramos por "Ca" en todas las columnas, si "Ca" se encuentra en alguna de las columnas.

```
msleep %>%
  select(name:order, sleep_total, -vore) %>%
  filter_all(any_vars(str_detect(., pattern = "Ca")))
```

La funcion `any_vars()` es equivalente a la expresión lógica OR, y la función `all_vars()` equivale a AND.

```
msleep %>%
  select(name, sleep_total:bodywt, -awake) %>%
  filter_all(all_vars(. > 1))
```

La función filter_if funciona del siguiente modo.

```
msleep %>%
  select(name, sleep_total:bodywt, -awake) %>%
  filter_all(all_vars(. > 1))
```

Y la función filter_at.

```
msleep %>%
  select(name, sleep_total:sleep_rem, brainwt:bodywt) %>%
  filter_at(vars(sleep_total, sleep_rem), all_vars(.>5))
```

Ejercicio

Con los datos `diamond` del ejercicio anterior.

- Selecciona las variables numéricas de los diamantes con un corte que igual a ideal y premium.
- Filtra los diamantes que tienen un depth y table igual o mayor que 60.
- Deshazte de los NA del dataframes. Es decir, si hay alguno en alguna fila, borra esa fila.
- Filtra por los diamantes que tienen un caract mayor o igual que 0.25 o un precio mayor o igualque 340.

Ahora vamos a ver cómo crear nuevas variables gracias la función `mutate()`. Todo lo que puedes hacer con un vector, lo puedes hacer dentro de esta función.

```
msleep %>%
  select(name, sleep_total) %>%
  mutate(sleep_total_min = sleep_total * 60)
```

Podemos usar funciones que agregan para crear nuevas columnas.

```
msleep %>%
  select(name, sleep_total) %>%
  mutate(sleep_total_vs_AVG = sleep_total - round(
    mean(sleep_total), 1),
         sleep_total_vs_MIN = sleep_total - min(
           sleep_total))
```

Podemos hacer la media de variables columnas por ejemplo, con la función `rowwise()`.

```
msleep %>%
  select(name, contains("sleep")) %>%
  rowwise() %>%
  mutate(avg = mean(c(sleep_rem, sleep_cycle)))
```

Podemos aplicar, cómo no, un control de secuencias tipo `ifelse` dentro de un mutate.

```
msleep %>%
  select(name, brainwt) %>%
  mutate(brainwt2 = ifelse(brainwt > 4, NA, brainwt))
```

Y usarlo también con variables de tipo caracter.

```
msleep %>%
  select(name) %>%
  mutate(name_last_word = tolower(str_extract(name, patter
```

Como con la función `filter`, podemos transformar varias columnas a la vez. Por ejemplo, podemos transformar todas las variables de tipo caracter a minúscula:

```
msleep %>%
  mutate_all(tolower)
```

Un ejemplo con if:

```
msleep %>%
  select(name, sleep_total:bodywt) %>%
  mutate_if(is.numeric, round)
```

Para usar el mutate_at, necesitas:

- Primero, una selección de las variables donde quieres que se haga la transformación
- La función u operación a aplicar.

```
msleep %>%
  select(name, sleep_total:awake) %>%
  mutate_at(vars(contains("sleep")), ~(.*60))
```

Con `mutate()`, podemos recodificar datos. Recodificar quiere decir cambiar o reasignar valores de un conjunto de observaciones para que sean comparables entre sí.

```
msleep %>%
  mutate(conservation2 = recode(conservation,
                                "en" = "Endangered",
                                "lc" = "Least_Concern",
                                "domesticated" = "Least_Concern",
                                .default = "other")) %>%
  count(conservation2)
```

Podemos hacer lo anterior y devolver una variable de tipo factor.

```
msleep %>%
  mutate(conservation2 = recode_factor(conservation,
                                       "en" = "Endangered",
                                       "lc" = "Least_Concern",
                                       "domesticated" = "Least_Concern",
                                       .default = "other",
                                       .missing = "no data",
                                       .ordered = TRUE)) %>%
  count(conservation2)
```

Con estos ifelse, podemos crear columnas nuevas también, de 2 niveles.

```
msleep %>%
  select(name, sleep_total) %>%
  mutate(sleep_time = ifelse(sleep_total > 10, "long",
                             "short"))
```

Pone “long” donde se cumple la condición y “short” donde la condición es FALSE.

Podemos crear más de 2 niveles con la función `case_when()`.
"Cuando ocurre esto, pon este nivel".

```
levels <- c("short", "limited", "long", "very long")
msleep %>%
  select(name, sleep_total) %>%
  mutate(sleep_total_discr = case_when(
    sleep_total > 13 ~ "very long",
    sleep_total > 10 ~ "long",
    sleep_total > 7 ~ "limited",
    TRUE ~ "short")) %>%
  mutate(sleep_total_discr = factor(sleep_total_discr,
                                    levels = levels))
```

Podemos usar no sólo una columna, si no varias. Se puede usar para agrupar varias columnas con diferentes condiciones lógicas.

```
msleep %>%
  mutate(silly_groups = case_when(
    brainwt < 0.001 ~ "light_headed",
    sleep_total > 10 ~ "lazy_sleeper",
    is.na(sleep_rem) ~ "absent_rem",
    TRUE ~ "other")) %>%
  count(silly_groups)
```

Ejercicio

Con el dataset de diamonds de nuevo:

- Crea una nueva variable de tipo factor que dice "precio_alto" si el precio se encuentra por encima de la media y "precio_bajo" si se encuentra por debajo.
- Crea una nueva columna que se llama "indice" que es el resultado de multiplicar x * y * z.
- Crea una nueva variable usando la variable depth, usando la función case_when(). Utiliza los cortes 50, 55 y 60. Asigna valores de bajo, medio y alto.

Existen unas funciones que te permiten disponer de tus datos en otro formato.

Wide Format

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

Long Format

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

Puedes añadir el argumento `factor_key=T` para que la columna con los nombres de las variables sea de tipo factor. Esto será útil cuando empecemos a usar librerías de visualización de datos.

```
msleep_g <- msleep %>%
  select(name, contains("sleep")) %>%
  gather(key = "sleep_measure", value = "time", -name, fact
```

La función `spread()` transforma un dataframe de tipo long en uno de tipo wide.

```
msleep_g %>%
  spread(sleep_measure, time)
```

Por último, la función `na_if()` introduce NaNs donde se le indica.

```
msleep %>%
  select(name:order) %>%
  na_if("omni")
```

Vamos a ver ahora cómo hacer conteos de variables categóricas.

```
msleep %>%
  count(order, sort = TRUE)
```

Con la función `count()`!

Puedes añadir todas las columnas que quieras para añadir categorías cruzadas.

```
msleep %>%
  count(order, vore, sort = TRUE)
```

O añadir columnas con esta misma función, pero esta vez especificando la variable sobre la que queremos ver el conteo.

```
msleep %>%
  select(name:vore) %>%
  add_count(vore)
```

La función `sumarise` o `summarize` (da igual cómo la escribas), sirve, como su nombre indica, para resumir los datos.

```
msleep %>%
  summarise(n = n(),
            average = mean(sleep_total),
            maximum = max(sleep_total))
```

Lo más interesante suele ser hacer esto por una varible nuestra de interés, es decir, como si fuera un `group_by` de SQL.

```
msleep %>%
  group_by(vore) %>%
  summarise(n = n(),
            average = mean(sleep_total),
            maximum = max(sleep_total))
```

La función `summarise` en realidad funciona con cualquier función de agregación, y permite otras operaciones, como:

`n()` - Devuelve el número de observaciones `n_distinct(var)` - Devuelve el número de valores únicos de la variable `var`.

Otro ejemplo con `summarise`:

```
msleep %>%
  group_by(vore) %>%
  summarise(avg_sleep_day = mean(sleep_total)/24)
```

Al igual que con `filter`, tenemos summarises especiales. Por ejemplo, con `summarise_all` podemos hacer agrupaciones de todas las columnas por cada grupo de interés.

```
msleep %>%
  group_by(vore) %>%
  summarise_all(mean, na.rm=TRUE)
```

Ejercicio

- Calcular el precio medio por la variable cut
- Calcular la mediana por dos grupos: cut y color.
- Calcular otros estadísticos (cuartiles, por ejemplo) para las variables x, z e y, agrupando previamente por claridad.
- Usa las funciones `spread` y `gather` a tu gusto, con el dataset `diamonds`.

Forcats

La librería `forcats` nos ayuda a manipular las variables de tipo factor en R.

Todas las funciones de esta librería empiezan por `fct` (factor).

Mini ejercicio

- Transforma las variables del dataset diamonds que sean ordinales a factor.

Podemos reordenar los factores.

```
library(forcats)
df <- msleep %>% mutate_if(is.character, as.factor)
df %>%
  mutate(vore_fct = fct_infreq(vore, ordered = TRUE)) %>%
  count(vore_fct)
```

Podemos reordenar los factores.

```
library(forcats)
df <- msleep %>% mutate_if(is.character, as.factor)
df %>%
  mutate(vore_fct = fct_infreq(vore, ordered = TRUE)) %>%
  count(vore_fct)
```

Esto será muy útil en visualización

Podemos recodificar los factores (a parte de como vimos con Dplyr)

```
df %>%
 mutate(vore_rcd = fct_recode(vore, Carnivore = "carni", or
  select(vore_rcd)
```

Podemos usar forcats para buscar factores dentro de datasets muy grandes. En este caso hacemos un conteo.

Notáis algo raro?

```
fct_count(df %>% select(vore) %>% as.matrix)
```

Otro ejemplo de búsqueda

```
fct_match(df %>% select(vore) %>% as.matrix, 'omni')
```

Y otro más

```
fct_unique(df %>% select(vore))
```

Una buena documentación para `forcats` es la siguiente:

https://rpubs.com/phle/r-tutorial-forcats

Ejercicio

- Recodifica la variable corte del dataset diamonds y transforma los niveles al castellano.
- Usa 3 funciones escogidas por ti, de la libreria forcats, y aplícalas al dataset diamonds.

Existen 2 principales librerías que se usan:

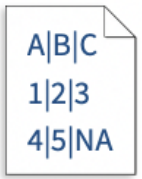
- Readr
- Readxl

Otras librerías:

- **haven** - SPSS, Stata, and SAS files
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)
- **readr::read_lines()** - text data

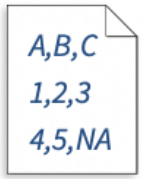
Read Tabular Data with readr

read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf, skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE**)** See **?read_delim**



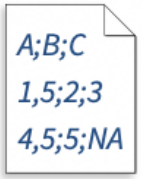
A	B	C
1	2	3
4	5	NA

read_delim("file.txt", delim = "|"**)** Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")



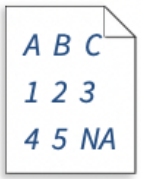
A	B	C
1	2	3
4	5	NA

read_csv("file.csv"**)** Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")



A	B	C
1.5	2	3
4.5	5	NA

read_csv2("file2.csv"**)** Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")



A	B	C
1	2	3
4	5	NA

read_tsv("file.tsv"**)** Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))**)** Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")

Lectura de datos con Tidyverse

USEFUL READ ARGUMENTS

A	B	C
1	2	3
4	5	NA

No header
read_csv("file.csv", col_names = FALSE)

x	y	z
A	B	C
1	2	3
4	5	NA

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

Read multiple files into a single table
read_csv(c("f1.csv", "f2.csv", "f3.csv"),
id = "origin_file")

1	2	3
4	5	NA

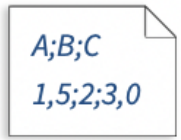
Skip lines
read_csv("file.csv", skip = 1)

A	B	C
1	2	3

Read a subset of lines
read_csv("file.csv", n_max = 1)

A	B	C
NA	2	3
4	5	NA

Read values as missing
read_csv("file.csv", na = c("1"))

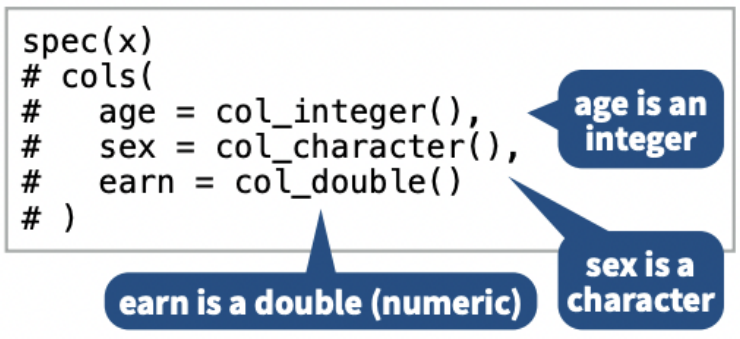


Specify decimal marks
read_delim("file2.csv", locale =
locale(decimal_mark = ","))

Column Specification with readr

Column specifications define what data type each column of a file will be imported as. By default readr will generate a column spec when a file is read and output a summary.

spec(x**)** Extract the full column specification for the given imported data frame.



USEFUL COLUMN ARGUMENTS

Hide col spec message
read_*(file, show_col_types = FALSE)

Select columns to import
Use names, position, or selection helpers.
read_*(file, col_select = c(age, earn))

Guess column types
To guess a column type, read_ *() looks at the first 1000 rows of data. Increase with **guess_max**.
read_*(file, guess_max = Inf)

Lectura de datos con Tidyverse

COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor(**levels, ordered = FALSE**)** - "f"
- **col_datetime(**format = ""**)** - "T"
- **col_date(**format = ""**)** - "D"
- **col_time(**format = ""**)** - "t"
- **col_skip()** - "-", "_"
- **col_guess()** - "?"

DEFINE COLUMN SPECIFICATION

Set a default type

```
read_csv(
  file,
  col_type = list(.default = col_double())
)
```

Use column type or string abbreviation

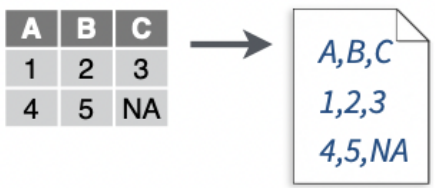
```
read_csv(
  file,
  col_type = list(x = col_double(), y = "l", z = "_")
)
```

Use a single string of abbreviations

```
# col types: skip, guess, integer, logical, character
read_csv(
  file,
  col_type = "_?ilc"
)
```

Save Data with readr

write_*(x, file, na = "NA", append, col_names, quote, escape, eol, num_threads, progress**)**



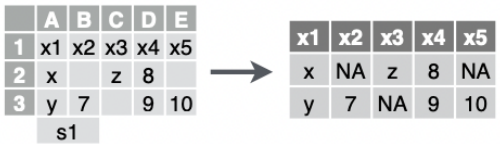
write_delim(x, file, delim = " "**)** Write files with any delimiter.

write_csv(x, file**)** Write a comma delimited file.

write_csv2(x, file**)** Write a semicolon delimited file.

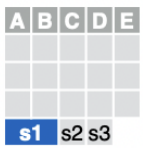
Import Spreadsheets with readxl

READ EXCEL FILES



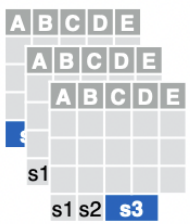
read_excel(path, sheet = NULL, range = NULL**)**
Read a .xls or .xlsx file based on the file extension. See front page for more read arguments. Also **read_xls()** and **read_xlsx()**.
read_excel("excel_file.xlsx")

READ SHEETS



read_excel(path, **sheet = NULL)** Specify which sheet to read by position or name.
read_excel(path, sheet = 1)
read_excel(path, sheet = "s1")

excel_sheets(path**)** Get a vector of sheet names.
excel_sheets("excel_file.xlsx")



To **read multiple sheets:**

1. Get a vector of sheet names from the file path.
2. Set the vector names to be the sheet names.
3. Use purrr::map_dfr() to read multiple files into one data frame.

READXL COLUMN SPECIFICATION

Column specifications define what data type each column of a file will be imported as.

Use the **col_types** argument of **read_excel()** to set the column specification.

Guess column types
To guess a column type, read_ excel() looks at the first 1000 rows of data. Increase with the **guess_max** argument.
read_excel(path, guess_max = Inf)

Set all columns to same type, e.g. character
read_excel(path, col_types = "text")

Set each column individually

```
read_excel(
  path,
  col_types = c("text", "guess", "guess","numeric")
)
```

COLUMN TYPES

logical	numeric	text	date	list
TRUE	2	hello	1947-01-08	hello
FALSE	3.45	world	1956-10-21	1

- skip
- guess
- logical
- numeric
- text
- date
- list

Ejercicio

- Lee directamente un csv de esta dirección web
- Haz una inspección de los datos para saber a lo que te enfrentas: dimensiones, nombre variables, etc. Pregúntate qué es cada observación.
- Calcula los 10 jugadores de la temporada con más:
  - puntos totales
  - puntos por partido
  - asistencias totales
  - asistencias por partido
  - faltas totales cometidas

Ejercicio

- Cuál es el equipo con más victorias?
- Cuál es el equipo que ha mantenido de media más diferencia de puntos en un partido contra su rival en la temporada?
- Cuál es el jugador que más puntos ha anotado en un partido? cuántos puntos anotó?