Taller 2: Exploración y validación de datos

José Daniel Conejeros

Magíster(c) Sociología UC - Licenciado en Ciencias Sociales

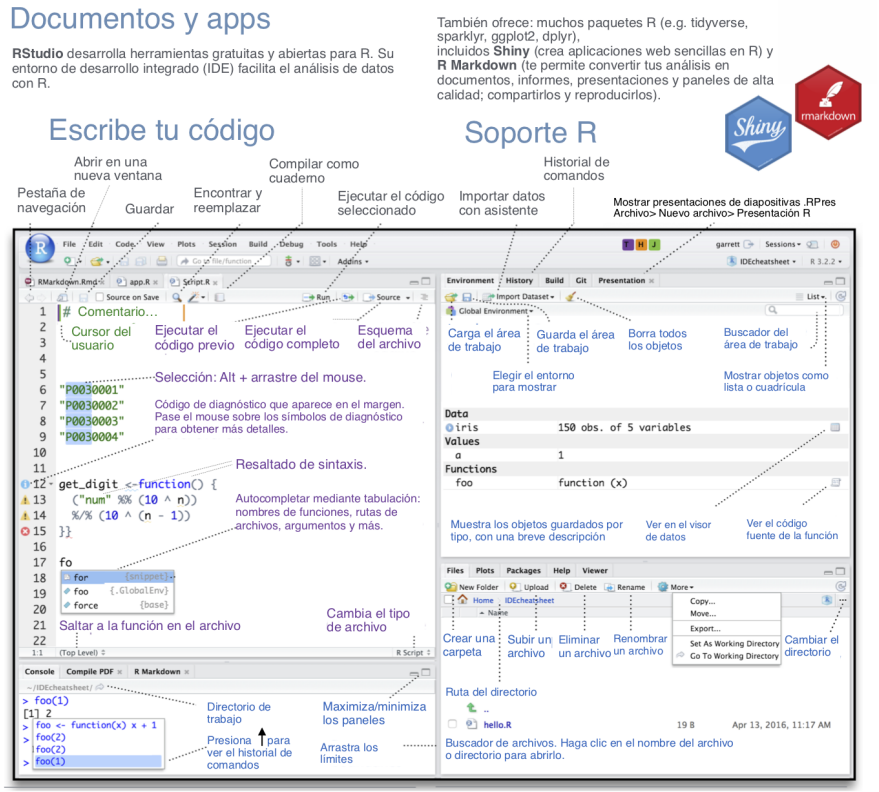
Junio 2020

Código del taller jdconejeros@uc.cl JDConejeros

joseconejerosp Jose_Conejeros

¿Qué hemos visto hasta el momento?

- Instalación de `R` y `RStudio`
- Explorar espacios de trabajo
- Comprender la lógica de objetos
- Explorar vectores, listas y matrices
- Estructura de funciones y argumentos
- Instalar y cargar librerías/paquetes

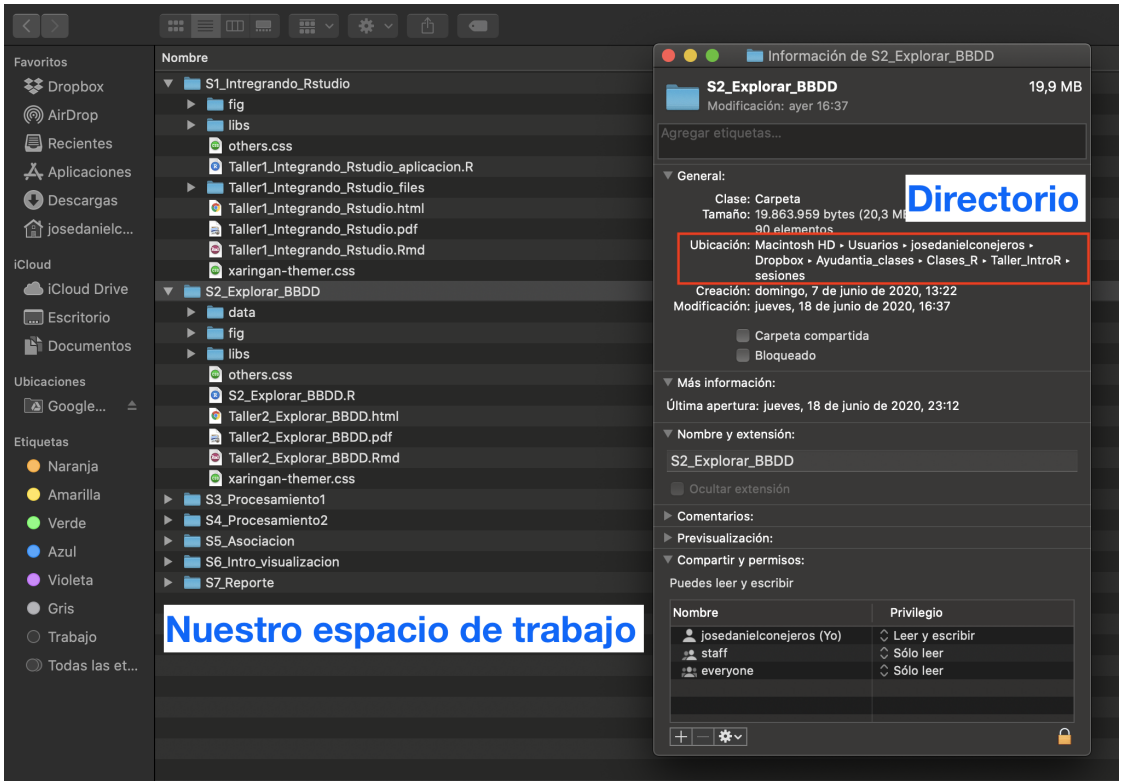


¿Alguna pregunta?

Hoy: Definir nuestro espacio de trabajo

```r
getwd() # Directorio de trabajo actual

setwd("ruta") # Establecer directorio de trabajo
```



Estructura de una base de datos

Las filas representan las observaciones - Las columnas representan las variables

```
id ← c(1, 2, 3, 4)
edad ← c(23, 45, 67, 89)
sexo ← c(1, 0, 1, 0)
peso ← c(80, 60, 70, 50)
altura ← c(180, 160, 200, 140)

data ← as.data.frame(cbind(id, edad, sexo,
                           peso, altura))

data
```

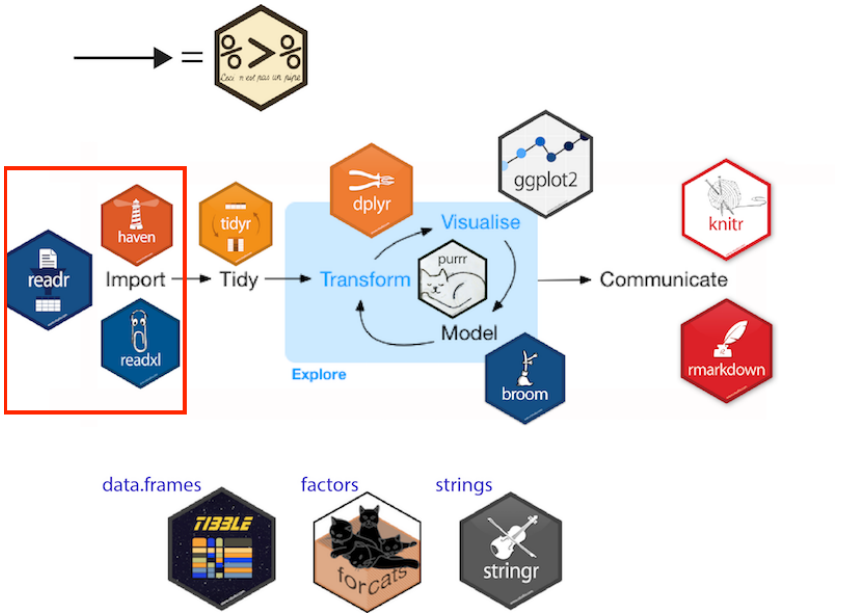
```
##   id edad sexo peso altura
## 1  1   23    1   80    180
## 2  2   45    0   60    160
## 3  3   67    1   70    200
## 4  4   89    0   50    140
```

Dim	Homogeneous	Heterogeneous
1d	Atomic Vector	Matrix
2d	Matrix	Data - Frame
nd	Array	

Mayor detalle pueden revisar el siguiente enlace

Importar una base de datos

```
install.packages(c("haven", "readxl", "readr"))
library(haven)
library(readxl)
library(readr)
```



Mayor detalle del paquete `readr` aquí

Mayor detalle del paquete `haven` aquí

Explorar nuestra base de datos


```
dim(data)   # Observaciones y variables
```

```
## [1] 4 5
```

```
names(data) # Nombre de nuestras variables
```

```
## [1] "id"     "edad"   "sexo"   "peso"   "altura"
```

```
str(data)   # Visor de nuestras variables
```

```
## 'data.frame':    4 obs. of  5 variables:
##  $ id    : num  1 2 3 4
##  $ edad  : num  23 45 67 89
##  $ sexo  : num  1 0 1 0
##  $ peso  : num  80 60 70 50
##  $ altura: num  180 160 200 140
```

Explorar nuestra base de datos


```
head(data)  # Primeras 6 observaciones
```

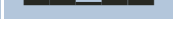
```
##   id edad sexo peso altura
## 1  1   23    1   80    180
## 2  2   45    0   60    160
## 3  3   67    1   70    200
## 4  4   89    0   50    140
```

```
tail(data)  # Últimas 6 observaciones
```

```
##   id edad sexo peso altura
## 1  1   23    1   80    180
## 2  2   45    0   60    160
## 3  3   67    1   70    200
## 4  4   89    0   50    140
```

Explorar nuestra base de datos

```
library(skimr)
skim(data)
```

skim_type	skim_variable	n_missing	numeric.mean	numeric.sd	numeric.p0	numeric.p25	numeric.p50	numeric.p75	numeric.p100	numeric.hist
numeric	id	0	2.5	1.2909944	1	1.75	2.5	3.25	4	
numeric	edad	0	56.0	28.4018779	23	39.50	56.0	72.50	89	
numeric	sexo	0	0.5	0.5773503	0	0.00	0.5	1.00	1	
numeric	peso	0	65.0	12.9099445	50	57.50	65.0	72.50	80	
numeric	altura	0	170.0	25.8198890	140	155.00	170.0	185.00	200	

¿Qué podemos ver de nuestras variables?

Validación de una base de datos

- Identificar observaciones duplicadas
- Revisar valores fuera de rango
- Explorar codificación de los casos pérdidos
- Chequear efectividad de los filtros (si es que los hay)

Exportar una base de datos

Podemos guardar nuestra base como texto plano:

```
write.table(data, file="data.txt", sep="\t")
```

Podemos guardar en un archivo separado por comas (`csv`):

```
# Separado por comas
write.csv(data, file="data.csv", row.names = FALSE)

# Otras opciones de separación
write.csv(data, file="data.csv", row.names = FALSE, sep = "-")
```

¿Qué ventaja tienen estos formatos?

Podemos guardar en un `excel` tradicional:

```
library(openxlsx)
write.xlsx(data, file="data.xlsx")
```

Exportar una base de datos

Podemos guardar en el formato de SPSS:

```
library(foreign)
write.foreign(data, datafile="data.txt", codefile="data.sps" package="SPSS")
```

Podemos guardar en el formato de SAS:

```
write.foreign(data, datafile="data.txt", codefile="data.sas" package="SAS")
```

Podemos guardar en el formato de Stata:

```
write.dta(data, datafile="data.dta")
```

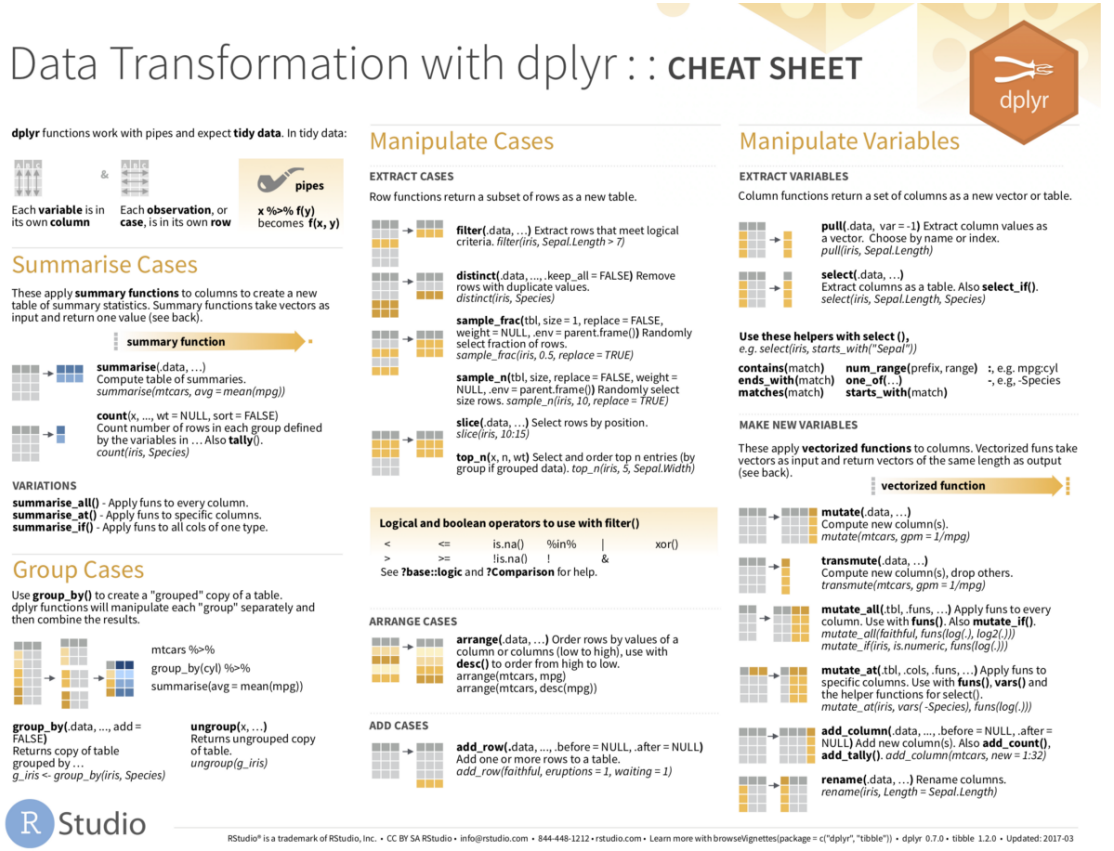
Podemos guardar en el formato de R:

```
save(rm, file="data.Rdata")
```

Próximo taller

- Encadenar operaciones: `pipe %>%`
- Filtrar y ordenar datos: `filter()` & `arrange`
- Seleccionar columnas: `select()`
- Generar nuevas variables: `mutate` & operador `$`
- Recodificar y etiquetar: `indexar code` & `dplyr`)

Próximo taller



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

Pueden adelantar revisando el siguiente enlace

Donación

Se donará el 25% de lo recaudado en el taller: $350.000.

Se está gestionando cajas de alimentos para personas con problemas económicos en La comuna de la Pintana, sector el castillo.

¡Se les avisará cuando se entreguen estas cajas! (en las próximas 2 semanas)

Sugerencias: ✉ Buzón anónimo

¡Gracias!

Presentación generada en:

jdconejeros@uc.cl joseconejerosp

JDConejeros Jose_Conejeros