

# R - Taller DESUC

## Manejo de base de datos

DESUC

21/01/2019

# Motivación

¿Por qué?

- ▶ Trabajamos con datos
- ▶ Invertir tiempo en lo que importa

# Motivación

¿Por qué?

- ▶ integración de principio a fin
- ▶ automatización, reportes y reproducibilidad
- ▶ gráficos
- ▶ paquetes y comunidad

## R: Lenguaje vs Software

¿Qué es lo que lo hace tan intimidante?

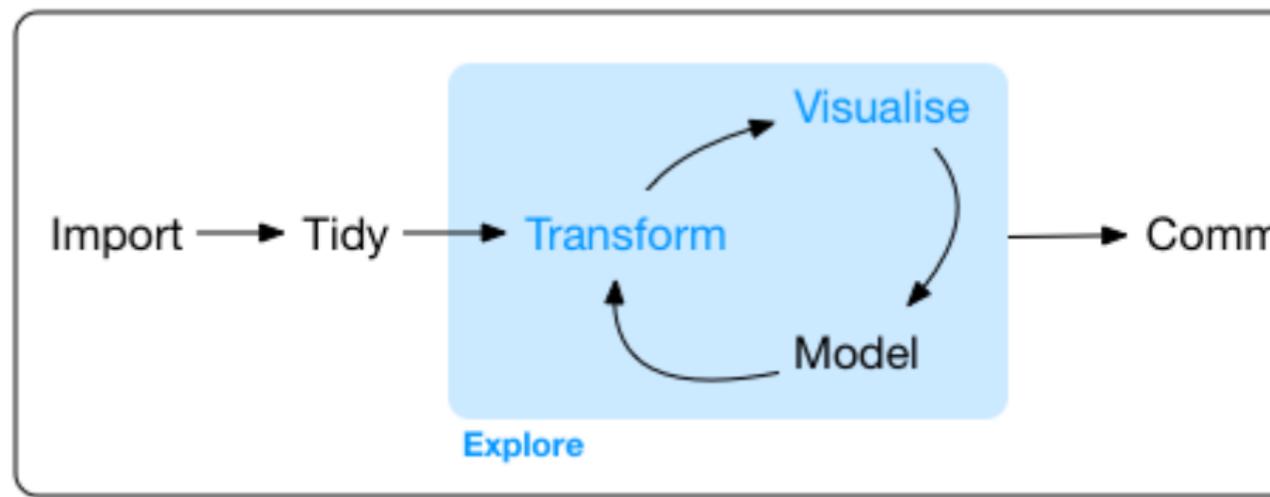
análisis <—> programación

## Definiciones

- ▶ tidyverse

*The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.*

```
include_graphics('images/data-science-explore.png')
```



# Definiciones

- ▶ tidy data

*Tidy data sets are arranged such that each variable is a column and each observation (or case) is a row*

```
not_tidy <- tibble(Año          = c(2017, 2018, 2019),  
                    Cualitativo = c(5, 4, 1),  
                    Cuantitativo = c(15, 16, 1),  
                    Organizacional = c(4, 3, 0))  
  
not_tidy
```

```
## # A tibble: 3 x 4  
##       Año Cualitativo Cuantitativo Organizacional  
##   <dbl>      <dbl>        <dbl>        <dbl>  
## 1  2017          5            15            4  
## 2  2018          4            16            3  
## 3  2019          1             1            0
```

## Definiciones

- ▶ tidy data

*Tidy data sets are arranged such that each variable is a column and each observation (or case) is a row*

```
tidy <- not_tidy %>%
  gather('Unidad', 'n', -Año)

tidy %>% head()
```

```
## # A tibble: 6 x 3
##       Año Unidad      n
##   <dbl> <chr>     <dbl>
## 1  2017 Cualitativo     5
## 2  2018 Cualitativo     4
## 3  2019 Cualitativo     1
## 4  2017 Cuantitativo   15
## 5  2018 Cuantitativo   16
## 6  2019 Cuantitativo     1
```

# Objetivo

Utilizar R y tidyverse para el manejo de base de datos:

- ▶ Lectura de bases de datos
- ▶ Manipulación de variables
- ▶ Uso y manejo de etiquetas

## Ejercicio

Uso de *directorio* y *matrícula* de alumnos de 2018, provista por el MINEDUC.

Queremos obtener la siguiente información según establecimiento:

- ▶ Categorizar establecimientos según áreas del Gran Santiago
- ▶ Establecimientos según si son CH y/o TP
- ▶ Establecimientos con enseñanza básica y/o media
- ▶ Número de hombres y mujeres por establecimiento
- ▶ Edad promedio de sus estudiantes NNA

# Paquetes

```
library(readxl) # Lectura de archivos excel  
library(haven) # Lectura de SPSS o Stata  
library(tidyverse) # Básico: dplyr, tidyr, purrr, stringr,  
library(janitor) # Arreglo de nombres y tablas  
  
library(sjlabelled) # Manejo de etiquetas  
library(sjmisc) # Funciones de ayuda  
  
library(knitr) # Creación de documentos mediante Rmarkdown
```

## Lectura de datos

## Lectura de datos: Excel

Datos de comunas que componen el Gran Santiago

- ▶ `xlsx`: `readxl::read_excel`

```
df_gran_santiago <- read_excel('..../data/gran_santiago.xlsx  
sheet = 1,  
trim_ws = TRUE)
```

Tips:

- ▶ Designar rangos específicos para leer información (`range` =)
- ▶ Saltar algunas filas en blanco (`skip` =)

## Lectura de datos: Excel

## Explorar base de datos:

¿Cómo se ve la base de datos que acabo de leer?

```
glimpse(df_gran_santiago)
```

```
## Observations: 34
## Variables: 3
## $ comuna <dbl> 13101, 13118, 13120, 13123, 13129, 13130
## $ ncomuna <chr> "Santiago", "Macul", "Ñuñoa", "Providencia"
## $ zonas <chr> "Centro", "Centro", "Centro", "Centro", "Centro",
```

## Lectura de datos: Bases en texto

Matrícula de alumnos 2018.

- ▶ **CSV**: `readr::read_csv` o `readr::read_csv2`.

Diferencia entre funciones es el delimitador de columnas. Para datos en Chile, en general, se usaría `read_csv2`.

```
df_mat_18 <- read_csv2('..../data/20181005_Matrícula_unica_2018')
```

Tips:

- ▶ Se puede leer csv comprimidos en zip
- ▶ Se puede configurar los parámetros si se quisiese (`read_delim`)

## Lectura de datos: Bases en texto

¿Cuáles son el nombre de las variables de la base de matrícula?

```
names(df_mat_18)
```

```
## [1] "AGNO"           "RBD"            "DGV_RBD"  
## [5] "COD_REG_RBD"   "COD_PRO_RBD"   "COD_COM_RBD"  
## [9] "COD_DEPROV_RBD" "NOM_DEPROV_RBD" "COD_DEPE"  
## [13] "RURAL_RBD"      "ESTADO_ESTAB"  "COD_ENSE"  
## [17] "COD_ENSE3"       "COD_GRADO"     "COD_GRADO2"  
## [21] "COD_JOR"         "COD_TIP_CUR"   "COD.Des_CUR"  
## [25] "GEN_ALU"         "FEC_NAC_ALU"   "EDAD_ALU"  
## [29] "COD_COM_ALU"    "NOM_COM_ALU"  "COD_SEC"  
## [33] "COD_RAMA"        "ENS"
```

¿Cuál es el tamaño de la base?

```
dim(df_mat_18)
```

```
## [1] 354062
```

34

# Lectura de datos: SPSS

## Directorio de establecimientos

- ▶ SPSS: `haven::read_sav`

```
df_dir_18 <- read_sav('..../data/20181005_Directorio_Oficial')
```

## Tips:

- ▶ Lee etiquetas de preguntas y variables.

## Lectura de datos: SPSS

¿Cuáles son las etiquetas de las variables en la base?

```
sjlabelled::get_label(df_dir_18)
```

## Lectura de datos: SPSS

¿Y las etiquetas de los niveles?

```
df_dir_18 %>%
  select(COD_DEPE) %>%
  sjlabelled::get_labels()
```

```
## $COD_DEPE
## [1] "Corporación Municipal"
## [2] "Municipal DAEM"
## [3] "Particular subvencionado"
## [4] "Participar pagado"
## [5] "Corporación de Administración Delegada"
## [6] "Servicio Local de Educación"
```

Análisis

## Análisis: herramienta

Para el manejo de base de datos, usaremos dplyr. En resumen:

- ▶ filter: filtra casos según una condición
- ▶ select: selecciona variables según nombre
- ▶ mutate: crea variables
- ▶ group\_by: agrupa casos según una variable
- ▶ summarise: crea resumen de variables

Todos estos verbos se enlazan con la pipa %>%

## Análisis: la pipa

Permite poner commandos según secuencia de ejecución.

Hace la lectura de código más sencilla y clara.

```
exp(sqrt(9))
```

```
## [1] 20.08554
```

```
9 %>%
  sqrt(.) %>%
  exp(.)
```

```
## [1] 20.08554
```

```
x <- 9
x <- sqrt(x)
exp(x)
```

```
## [1] 20.08554
```

# Análisis: tip

Hay muy buenos *cheatsheets* donde se pueden revisar diversas funciones existentes.

```
knitr:::include_graphics('dplyr.pdf')
```

## Data Transformation with dplyr :: CHEAT SHEET



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



### Summarise Cases

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

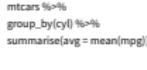
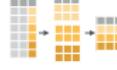
- **summary**(...) Compute table of summaries.  
`summarise(mtcars, avg = mean(mpg))`
- **count**(..., wt = NULL, sort = FALSE)  
Count number of rows in each group defined by the variables in .... Also **tally**.  
`count(mtcars, Species)`

#### 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.



Returns ungrouped copy

### Manipulate Cases

#### EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in \_ for non-standard evaluation friendly code.

- **filter**(data, ...)  
Extract rows that meet logical criteria. Also **filter\_**(...).  
`filter(iris, Sepal.Length > 7)`
- **distinct**(data, ..., keep\_all = FALSE)  
Remove rows with duplicate values. Also **distinct\_**(...).  
`distinct(iris, Species)`
- **sample\_frac**(tbl, size = 1, replace = FALSE, weight = NULL, env = parent.frame())  
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. Also **slice\_**(...).  
`slice(iris, 10:15)`
- **top\_n**(tbl, 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%      |  
>      >=      is.na()      !      &  
See ?base::logical and ?Comparison for help.

#### ARRANGE CASES

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

#### ADD CASES

→ **bind\_rows**(tbl1, ..., NUL = F, NUL = T)  
Bind multiple tables together.

Column functions return a set of columns as a new table. Use a variant that ends in \_ for non-standard evaluation friendly code.

- **select**(data, ...)  
Extract columns by name. Also **select\_**(...).  
`select(iris, Sepal.Length, Species)`

#### Use these helpers with select(...)

e.g. `select(iris, starts_with("Sepal"))`  
`contains(match)`    `num_range(prefix, range)`    e.g. `mpg:cyl`  
`ends_with(match)`    `one_of(...)`    e.g. `-Species`  
`matches(match)`    `starts_with(match)`

#### 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 **select**().  
`mutate_all(funs, log10(.), log2(.))`
- **mutate\_at**(tbl, cols, funs, ...)  
Apply funs to specific columns. Use with **fun**(), **vars**() and the helper functions for **select**.  
`mutate_at_if(cols, funs(log10(.)))`
- **mutate\_if**(tbl, .predicate, funs, ...)  
Apply funs to all columns of one type. Use with **fun**().  
`mutate_if(iris, is.numeric, funs(log(.)))`
- **add\_column**(data, before = NULL, after = NULL)

## Análisis Gran Santiago: casos

Utilizaré base de df\_gran\_santiago porque es más pequeña que las otras dos.

¿Cómo se ve la base de datos que acabo de leer?

- ▶ Ver primeras tres filas head().
- ▶ Ver últimas filas con tail()

```
df_gran_santiago %>% head(3)
```

```
## # A tibble: 3 x 3
##   comuna ncomuna zonas
##   <dbl> <chr>    <chr>
## 1 13101 Santiago Centro
## 2 13118 Macul     Centro
## 3 13120 Ñuñoa    Centro
```

```
df_gran_santiago %>% tail(3)
```

```
## # A tibble: 3 x 3
```

# Análisis Gran Santiago: filtrar casos

Usar función filter().

```
df_gran_santiago %>%
  filter(zonas == 'Norte')
```

```
## # A tibble: 6 x 3
##   comuna ncomuna      zonas
##   <dbl> <chr>        <chr>
## 1 13104 Conchalí    Norte
## 2 13107 Huechuraba  Norte
## 3 13108 Independencia Norte
## 4 13125 Quilicura   Norte
## 5 13127 Recoleta    Norte
## 6 13128 Renca       Norte
```

## Análisis Gran Santiago: ordenar casos

Ordenar la base en base al código comunal `arrange(comuna)`.

```
df_gran_santiago %>% arrange(comuna) %>% head(3)
```

```
## # A tibble: 3 x 3
##   comuna ncomuna     zonas
##   <dbl> <chr>       <chr>
## 1 13101 Santiago Centro
## 2 13102 Cerrillos Poniente
## 3 13103 Cerro Navia Poniente
```

## Análisis Gran Santiago: grupos

¿Cuál es la cantidad de comunas por zona? Agrupar por zona group\_by(zonas) y luego calcular dato por grupo summarise().

```
df_gran_santiago %>%  
  group_by(zonas) %>%  
  summarise(comunas_n = n())
```

```
## # A tibble: 5 x 2  
##   zonas     comunas_n  
##   <chr>     <int>  
## 1 Centro      6  
## 2 Norte       6  
## 3 Oriente     5  
## 4 Poniente    9  
## 5 Sur         8
```

# Análisis: Matrícula

¿Cuál es el número de alumnos según dependencia?

```
df_mat_18 %>%
  group_by(COD_DEPE2) %>%
  summarise(alumnos_n = n())
```

```
## # A tibble: 4 x 2
##   COD_DEPE2 alumnos_n
##       <dbl>     <int>
## 1 1           90117
## 2 2          123270
## 3 3          129913
## 4 4          10762
```

# Análisis: Matrícula

¿Podemos agregar la proporción?

```
df_mat_18 %>%
  group_by(COD_DEPE2) %>%
  summarise(alumnos_n = n()) %>%
  mutate(alumnos_prop = alumnos_n / sum(alumnos_n))
```

```
## # A tibble: 4 x 3
##   COD_DEPE2 alumnos_n alumnos_prop
##       <dbl>     <int>        <dbl>
## 1 1           90117      0.255
## 2 2          123270      0.348
## 3 3          129913      0.367
## 4 4           10762      0.0304
```

## Análisis: Matrícula

¿Y si queremos saber el número de alumnos por sexo?

- ▶ Alternativa *no-tidy*

```
df_mat_18 %>%
  group_by(COD_DEPE2) %>%
  summarise(alumnos_mas_n = sum(GEN_ALU == 1),
            alumnos_fem_n = sum(GEN_ALU == 2)) %>%
  mutate_at(vars(starts_with('alumnos')), funs(prop = ./sum))
```

```
## # A tibble: 4 x 5
##   COD_DEPE2 alumnos_mas_n alumnos_fem_n alumnos_mas_n_p
##   <dbl>        <int>        <int>        <dbl>
## 1 1             1        49402       40715     0.274
## 2 2             2        60457       62813     0.335
## 3 3             3        66029       63884     0.366
## 4 4             4        4626        6136      0.025
```

## Análisis: Matrícula

¿Y si queremos saber el número de alumnos por sexo? ¡Agrupar por la variable de sexo!

- ▶ Alternativa *tidy*

```
df_mat_18 %>%
  group_by(COD_DEPE2, GEN_ALU) %>%
  summarise(alumnos_n = n()) %>%
  mutate(alumnos_prop = alumnos_n / sum(alumnos_n))
```

```
## # A tibble: 8 x 4
## # Groups:   COD_DEPE2 [4]
##   COD_DEPE2 GEN_ALU alumnos_n alumnos_prop
##   <dbl>     <dbl>     <int>        <dbl>
## 1 1          1         1    49402        0.548
## 2 2          1         2    40715        0.452
## 3 2          2         1    60457        0.490
## 4 2          2         2    62813        0.510
## 5 3          1         1    66029        0.508
```

# Análisis: Matrícula

Hay funciones que facilitan la tarea

```
sjmisc::frq(df_mat_18, COD_DEPE2)
```

```
##  
## # COD_DEPE2 <numeric>  
## # total N=354062  valid N=354062  mean=2.17  sd=0.84  
##  
##   val     frq raw.prc valid.prc cum.prc  
##   1  90117    25.45      25.45    25.45  
##   2 123270    34.82      34.82    60.27  
##   3 129913    36.69      36.69    96.96  
##   4  10762     3.04      3.04    100.00  
## <NA>      0     0.00        NA        NA
```

# Análisis: Matrícula

Hay funciones que facilitan la tarea

```
df_mat_18 %>%
  group_by(GEN_ALU) %>%
  sjmisc::frq(COD_DEPE)
```

```
##
## Grouped by:
## GEN_ALU: 1
##
## # COD_DEPE <numeric>
## # total N=180514  valid N=180514  mean=2.99  sd=1.09
##
##   val    frq raw.prc valid.prc cum.prc
##   1 26961    14.94     14.94    14.94
##   2 22441    12.43     12.43    27.37
##   3 60457    33.49     33.49    60.86
##   4 66029    36.58     36.58   97.44
##   5 4626     2.56      2.56    100.00
```

# Etiquetas

## Agregar etiquetas

Agregar etiquetas a variable y niveles

```
df_mat_18 <- df_mat_18 %>%
  mutate(COD_DEPE2 = labelled(COD_DEPE2,
                               labels = c('Municipal' = 1,
                                         'Particular Subven',
                                         'Particular Pagado',
                                         'Administración De',
                                         'Servicio Local de',
                                         'Otro'),
                               label = 'Código de Dependencia'))
```

```
frq(df_mat_18, COD_DEPE2)
```

```
## # Código de Dependencia del Establecimiento (agrupado)
## # total N=354062 valid N=354062 mean=2.17 sd=0.84
## 
##   val                      label     frq raw.prc valid.prc
##   1 Municipal                90117    25.45      25.45
```

## Agregar etiquetas: multiples variables

```
etiquetas <- c('AGNO' = 'Año escolar',
              'RBD' = 'Rol base de datos del establecimiento',
              'NOM_RBD' = 'Nombre del Establecimiento')

set_label(df_mat_18[, names(etiquetas)]) <- etiquetas

df_mat_18 %>%
  select(1:2, NOM_RBD) %>%
  get_label()
```

```
##                                     AGNO
##                                     "Año escolar"
##                                     RBD
## "Rol base de datos del establecimiento"
##                                     NOM_RBD
## "Nombre del Establecimiento"
```

## Agregar niveles: multiples variables

```
niveles <- c('110' = 'Enseñanza Básica',
            '310' = 'Enseñanza Media H-C niños y jóvenes',
            '410' = 'Enseñanza Media T-P Comercial Niños y jóvenes',
            '510' = 'Enseñanza Media T-P Industrial Niños y jóvenes')

df_dir_18 <- set_labels(df_dir_18,
                        ENS_01, ENS_02,
                        labels = niveles)

df_dir_18 %>%
  filter(ENS_01 %in% names(niveles)) %>%
  frq(ENS_02)

## # ENS_02 <numeric>
## # total N=444  valid N=444  mean=237.38  sd=193.94
## 
## val          label frq
```

Integración

## Posición de escuelas en RM

Construir una base de datos con información geográfica a partir de longitud y latitud presente en la base.

```
df_dir_18_geo <- df_dir_18 %>%
  filter(!is.na(LONGITUD)) %>%
  st_as_sf(coords = c("LONGITUD", "LATITUD"))
```

# Posición de escuelas en RM

Ver la ubicación geográfica

```
df_dir_18_geo %>%  
  ggplot(aes(colour = as_factor(COD_DEPE2))) +  
  geom_sf(alpha = 0.3, size = 0.5)
```

Faltan las etiquetas

## Posición de escuelas en RM

Agregar etiquetas a variable COD\_DEPE2. La tenemos en la base de datos.

```
gg <- df_dir_18_geo %>%
  ggplot(aes(colour = as_label(COD_DEPE2))) +
  geom_sf(alpha = 0.3, size = 0.5) +
  scale_color_discrete(name = 'Dependencia')
```

## Posición de escuelas en RM: ggmap

```
library(ggmap)

gran_santiago <- c(left = -70.9634, bottom = -33.6558, right = -70.9000, top = -33.5500)
mapa_base <- get_stamenmap(gran_santiago, zoom = 11)

ggmap(mapa_base) +
  geom_point(data = df_dir_18, aes(x = LONGITUD, y = LATITUD),
             color = 'black', size = 100) +
  scale_colour_discrete(name = 'Dependencia')
```

## Posición de escuelas en RM: leaflet

```
library(leaflet)

leaflet() %>% # leaflet works with the pipe operator
  addTiles() %>% # setup the default OpenStreetMap map tiles
  addMarkers(lng = df_dir_18$LONGITUD, lat = df_dir_18$LATITUD)
```