Capacitación en R y herramientas de productividad

Proyecto Estratégico Servicios Compartidos para la Producción Estadística

Procesamiento de bases de datos

Agosto 2020

Contenidos de hoy

El objetivo de esta sesión es aprender a manipular *data frames*.

Para cumplir con el objetivo, revisaremos los siguientes temas:

- Un breve resumen de lo visto en la sesión anterior.
- Importación de datos a R (.csv, .xlsx, .dta, .sav, .RData).
- Funciones del paquete `dplyr` para la manipulación de *data frames*.
- Encadenamiento de instrucciones: uso de *pipes*.
- Agregación de unidades (`group_by`).

Breve resumen

Antes de avanzar, una recopilación de los puntos vistos en la primera sesión y que serán necesarios en esta sesión

- **¿Qué es un script?**

Corresponde a un documento de texto en el cual se escribe una serie de instrucciones para R.

- **¿Qué es un paquete?**

Corresponde a una colección de `funciones` diseñadas para atender distintas tareas. Los paquetes complementan a las funciones de base existentes en R. Para poder ejecutar estas funciones, es necesario instalar el `paquete` y cargar las `librerías`.

- **¿Qué se puede entender por función?**

Corresponde a un conjunto de instrucciones que permite automatizar rutinas largas y/o complicadas.

```
round(x = c(4.5, 4.4444449), digits = 1)
```

```
## [1] 4.5 4.4
```

Breve resumen

- **¿Cómo se asigna un valor a un elemento?**

```
# Utilizando la expresión "<-" se asigna un valor.
z <- 1
# El objeto con el valor asignado se puede usar en funciones.
zz <- z + 3
# print() se utiliza para mostrar el valor del objeto en la consola.
print(zz)
```

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

- **¿Qué es un vector?**:

Es el objeto más básico en R. Este objeto es una forma de almacenar datos.

```
x <- c(1, 2, 5)
print(x)
```

```
## [1] 1 2 5
```

Breve resumen

- **Tipos de vectores**

```
character = c("gato", "perro")
numeric   = c(8, 15.9, -2.1)  # números reales
integer   = c(2L, 4L) # L indica que son enteros
logical   = c(TRUE, FALSE, TRUE)
complex   = 3 + 4i
```

Los vectores son atómicos. **¿Qué significa esto?**

```
v <-  c("gato", 1)
class(v)
```

```
## [1] "character"
```

```
v <-  c(1L, 2.2)
class(v)
```

```
## [1] "numeric"
```

Breve resumen

- **Operadores matemáticos**

```
x + y   # suma
x - y   # resta
x * y   # multiplicación
x / y   # división
x ^ y   # exponenciación
x %% y  # módulo
x %/% y # cociente
```

Al realizar una operación matemática sobre dos vectores, la operación se realiza entre los componentes que ocupan la misma `posición` en ambos vectores.

```
x <- c(0, 1, 2)
y <- c(0, -2, 1)
x + y
```

```
## [1]  0 -1  3
```

Breve resumen

- **Operadores relacionales**

```
x < y      # x menor que y
x > y      # x mayor que y
x <= y     # x menor o igual que y
x >= y     # x mayor o igual que y
x == y     # x igual a y
x != y     # x distinto a y
```

Al realizar una operación relacional entre dos vectores, la **relación** se realiza entre los componentes que ocupan la misma `posición` en ambos vectores.

El resultado de una operación relacional es **TRUE** o **FALSE** en cada componente del nuevo vector.

```
x <- c(0, 1, 2)
y <- c(0, -2, 1)
x == y
```

```
## [1]  TRUE FALSE FALSE
```

Breve resumen

- **Operadores lógicos (booleanos)**

```
x | z     # x o z es verdadero
x & z     # x y z son verdaderos
!x        # x no es verdadero (negación)
isTRUE(x) # x es verdadero (afirmación)
```

Las operaciones lógicas analizan el valor de verdad de dos sentencias.

```
(24 > 5) & (-1 * -1 > 0)
```

```
## [1] TRUE
```

```
length(c(1:10)) == length(seq(from = 1, to = 10, by = 2))
```

```
## [1] FALSE
```

- **Indexación**

```
c("a", "a,b", "c")[3] # El tercer elemento
```

```
## [1] "c"
```

Breve resumen

- **¿Qué es un data frame?**

Es una estructura de datos de dos dimensiones (rectangular) que puede contener datos de diferentes tipos, por lo tanto, es heterogénea. Esta estructura de datos es la más usada para realizar análisis de datos, y es la estructura con la cual trabajaremos en esta sección.

- **Operador $**

Se utiliza para vincular una variable con un objeto.

- **Función table()**

Se utiliza para tabular un resultado.

```
x <- data.frame(nombres = c("Ignacio", "Klaus", "Juan", "Ignacio"))
# Tabulación
table(x$nombres)
```

```
##
## Ignacio    Juan   Klaus
##       2       1       1
```

Importación de datos

Importación de datos

R tiene distintos paquetes para importar datos, diferenciando la extensión de los archivos.

- Para importar archivos delimitados (.csv, .txt, .tab), existe el paquete `readr`. Dentro de las funciones de ese paquete, la función `read_csv()` permite la importación de archivos con la extensión "**.csv**".

```
# Para usar el paquete primero es necesario cargarlo
library(readr)
# Al usar la función, asignamos el archivo a un valor
base_csv  <-  read_csv(file = "ejemplo.csv")
```

- El paquete `readxl` nos permite importar archivos de Excel. Una de sus funciones es `read_excel()`.

```
# Para usar el paquete primero es necesario cargarlo
library(readxl)
# Al usar la función, asignamos el archivo a un valor
base_excel  <-  read_excel(path = "ejemplo_excel.xlsx",
                           sheet="Hoja1")
```

Importación de datos

Tanto las funciones de `readr` como `readxl` tienen argumentos que permiten setear distintas opciones para la importación.

`readr`

- Cambiar nombres de columnas
- Saltarse líneas al momento de importar
- Recodificar valores perdidos

`readxl`

- Leer un rango específico de la hoja de cálculo
- Cambiar nombres de columnas
- Saltarse líneas al momento de importar
- Recodificar valores perdidos

Importación de datos

Para importar archivos guardados desde Stata o SPSS existe el paquete `haven`.

Al igual que los paquetes `readxl` o `readr` tiene funciones bastante explícitas respecto de lo que hacen:

- **SPSS**: `read_sav()` permite importar archivos ".sav".
- **Stata**: `read_dta()` permite importar archivos ".dta".


```
# Cargar librería
library(haven)
esi <-  read_dta("data/esi_2018_personas.dta")
```

La función `read_dta()` permite cargar bases de datos desde sitios web.

Importación de datos

Los archivos de R tienen la extensión `.RData`.

Para importar estas bases se usa el comando `load()`.

La importación de un archivo en R no requiere una asignación de valor.

Estos archivos puede incluir más de un objeto.

```
load(file = "data/Censo_area.RData")
```

Comencemos con nuestro ejercicio...

Importemos el archivo de Excel "**`Censo_area.xlsx`**"...

¿Qué función debemos utilizar?

Importación de datos

Debemos utilizar la función `read_excel` de la librería `readxl`.

Esta función requiere indicar el nombre de la base a importar, así como la ruta en donde se encuentra guardada.

IMPORTANTE: al escribir la ruta, se debe utilizar el símbolo "**/**".

```
library(readxl)
base <- read_excel(path = "data/Censo_area.xlsx")
```

¿Cómo se puede revisar la estructura de la base de datos?

```
str(base)
```

```
## tibble [6,464 x 5] (S3: tbl_df/tbl/data.frame)
##  $ CÓDIGO_REGIÓN: chr [1:6464] "15" "15" "15" "15" ...
##  $ EDAD         : chr [1:6464] "0" "1" "2" "3" ...
##  $ N            : num [1:6464] 135 120 123 129 133 133 151 147 146 132 ...
##  $ AREA         : num [1:6464] 2 2 2 2 2 2 2 2 2 2 ...
##  $ SEXO         : num [1:6464] 2 2 2 2 2 2 2 2 2 2 ...
```

Ahora que importamos la base de datos, trabajaremos con la manipulación de los data frames.

Transformación de datos

El paquete `dplyr`

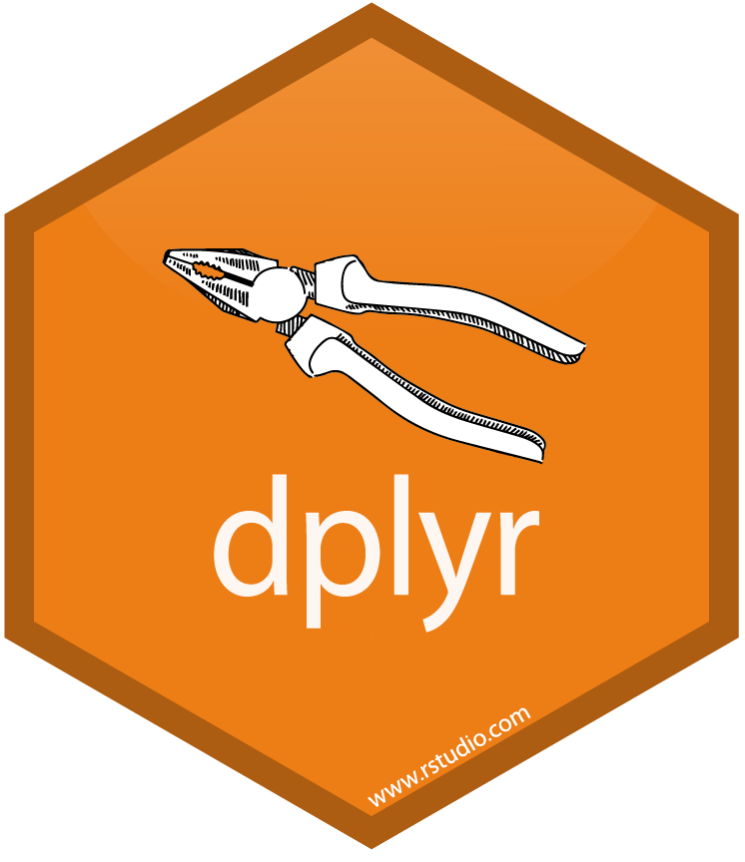
Para la manipulación de los data frames existe un paquete llamado **`dplyr`**.

Este paquete fue desarrollado por Hadley Wickham de RStudio.

Es una versión optimizada de un paquete anterior llamado `plyr`.

El paquete `dplyr` proporciona una "gramática" (particularmente verbos) para la manipulación y operaciones con data frames.

Los paquetes `dplyr`, `readr`, `readxl` y `haven` forman parte del universo de `tidyverse`.



Transformación de datos

Durante esta sesión veremos siete funciones del paquete **`dplyr`** que se utilizan para la manipulación de data frames.

- **select**: selecciona y devuelve un conjunto de columnas.
- **filter**: selecciona y devuelve un conjunto de filas según una o varias condiciones lógicas.
- **arrange**: reordena filas de un data frame.
- **rename**: renombra variables en un data frame.
- **mutate**: añade nuevas variables o transforma variables existentes.
- **group_by**: agrupa filas de un data frame.
- **summarise**: genera resúmenes de datos.

Estas funciones serán algunas de nuestras mejores compañeras y siempre nos acompañarán en nuestro camino de programación.

Transformación de datos

Todas estas funciones tienen en común una serie de argumentos:

- El primer argumento es el data frame a manipular.
- Los otros argumentos describen qué hacer con el data frame especificado en el primer argumento.
- El valor de retorno de la función es un nuevo data frame.

Como veremos en esta sesión, una de las principales ventajas de estas funciones es que podemos referirnos a las columnas en el data frame directamente sin utilizar el operador "`$`", es decir, solo con el nombre de la variable.

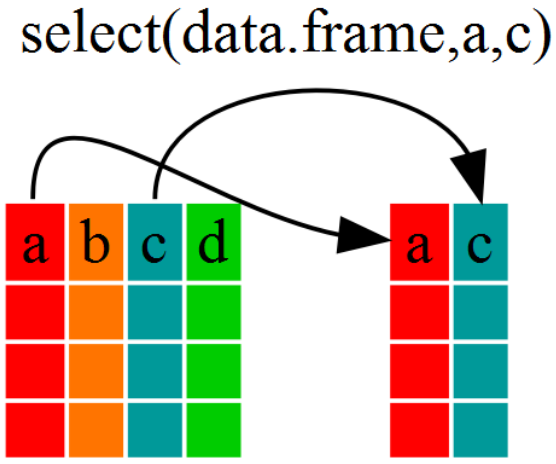
Transformación de datos

Función `select()`

La primera función que veremos es la función `select()`.

Esta función es utilizada para seleccionar columnas dentro de un data frame.

El resultado de esta función corresponde a un data frame que solo incluye las variables seleccionadas.



Transformación de datos

Función `select()`

El orden definido en la selección de variables, es el orden que tendrán las columnas en el resultado.

La selección puede ser realizada...

...según nombre de la variable.

```
library(dplyr)
head(select(base, EDAD, N))
```

```
## # A tibble: 6 x 2
##   EDAD      N
##   <chr> <dbl>
## 1 0       135
## 2 1       120
## 3 2       123
## 4 3       129
## 5 4       133
## 6 5       133
```

...según índice de la variable.

```
library(dplyr)
head(select(base, 3, 2))
```

```
## # A tibble: 6 x 2
##       N EDAD
##   <dbl> <chr>
## 1   135 0
## 2   120 1
## 3   123 2
## 4   129 3
## 5   133 4
## 6   133 5
```

Función `select()`

Otra forma de seleccionar consiste en definir qué columnas no se seleccionarán. Para esto es necesario incluir el signo menos ("`-`") antes de las columnas a seleccionar

Esta manera de seleccionar se puede entender como "la eliminación de columnas".

Realicemos un ejercicio...

En un objeto llamado `ejemplo`, seleccionar solo las primeras 3 columnas del objeto `base`.

```
head(select(base, -4, -5), n = 4)
```

```
## # A tibble: 4 x 3
##   CÓDIGO_REGIÓN EDAD      N
##   <chr>         <chr> <dbl>
## 1 15            0       135
## 2 15            1       120
## 3 15            2       123
## 4 15            3       129
```

Función `select()`

Realicemos algunos ejercicios...

Creemos un data frame:

```
df <- data.frame(x1 = c(1:3),
                 x2 = c(1,7,2),
                 y3 = c("a","b","c"),
                 x4 = c(5,-1,8),
                 x5 = c("perro","gato","conejo"),
                 y1 = c(1, "perro", "gato"))
```

Ahora que creamos el data frame, realicemos algunas selecciones:

- Seleccionemos todas las columnas excepto x4 y x5.
- Reordenemos las columnas en el siguiente orden: x1, y1, x5, y3, x2 y x4.

Función `select()`

Cuando trabajamos con bases de datos, a veces nos encontramos con variables cuyo nombre sigue cierto patrón, por ejemplo:

- La primera letra es la misma (a1, a2, a3).
- La variable posee un mismo sufijo (a1_otro, a2_otro, a3_otro)

Existen funciones auxiliares a `select()`, las cuales permiten trabajar con ciertos patrones.

Una que utilizaremos reiteradas veces dentro de las siguientes secciones es `starts_with()`. Esta función auxiliar se utiliza para identificar las variables que comienzan con un cierto prefijo.

Por ejemplo, seleccionemos las variables que comienzan con la letra "**x**".

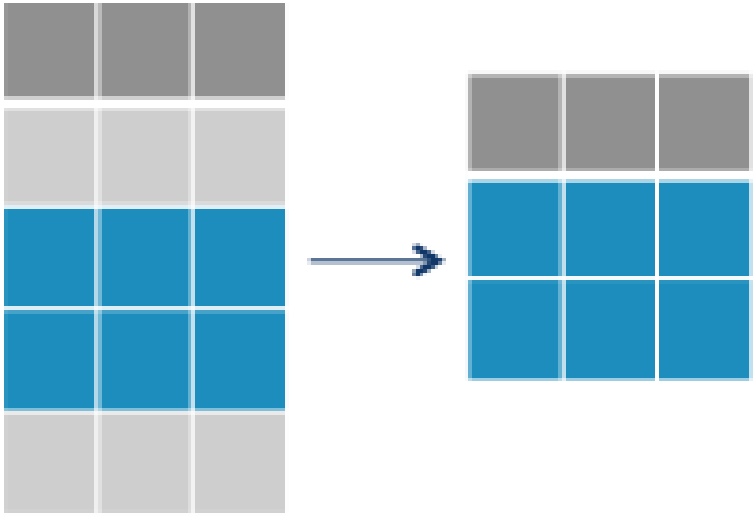
```
head(select(df, starts_with("x")))
```

```
##   x1 x2 x4     x5
## 1  1  1  5  perro
## 2  2  7 -1   gato
```

Función `filter()`

Esta función se utiliza para filtrar un data frame según alguna condición a definir.

Así como la función `select` es utilizada para seleccionar columnas de un data frame, la función `filter` puede ser entendida como la selección de filas de un data frame.



Transformación de datos

Función `filter()`

¿Es posible filtrar en base a más de una condición?

SI, es posible utilizando operadores lógicos.

Por ejemplo, si queremos filtrar nuestra base, y seleccionar a las mujeres de la región 6 y de edad 35, debemos hacer lo siguiente...

```
library(dplyr)
filter(base, SEXO == 2 & CÓDIGO_REGIÓN == 6 & EDAD == 35)
```

```
## # A tibble: 2 x 5
##   CÓDIGO_REGIÓN EDAD      N  AREA  SEXO
##   <chr>         <chr> <dbl> <dbl> <dbl>
## 1 6             35     1431     2     2
## 2 6             35     5245     1     2
```

Ejercicio: filtrar el data frame "df", según los criterios: x1 > 2 y x5 es distinto de "perro".

Transformación de datos

Función `arrange()`

Como se vio en la sesión anterior, esta función se utiliza para reordenar las filas de un data frame:

- El orden se puede realizar según una o más columnas.
- El reordenamiento se realiza según el orden de las columnas seleccionadas.
- Por defecto, el orden es ascendente.
- Para ordenar de manera descendente se debe utilizar la función auxiliar `desc()`.

```
head(arrange(base,desc(SEXO),N),n=3)
```

```
## # A tibble: 3 x 5
##   CÓDIGO_REGIÓN EDAD      N  AREA  SEXO
##   <chr>         <chr> <dbl> <dbl> <dbl>
## 1 15            96        0     2     2
## 2 2             95        0     2     2
## 3 2             96        0     2     2
```

Función `rename()`

Si queremos **`renombrar`** una variable (columna) de un data frame, existen múltiples formas y varias muy complejas.

La función `rename` permite modificar el nombre de una variable de una manera sencilla, sin editar el resto de variables.

Para obtener el nombre de las variables en un data frame se utiliza la función `names()`.

```
# Función para obtener nombres de las columnas
names(df)
```

```
## [1] "x1" "x2" "y3" "x4" "x5" "y1"
```

Función `rename()`

La función **`rename`** permite renombrar una variable o más de un variable cada vez.

```
df <- rename(df, var1 = x1, var5 = x5)
head(df)
```

```
##   var1 x2 y3 x4   var5    y1
## 1    1  1  a  5  perro     1
## 2    2  7  b -1   gato perro
## 3    3  2  c  8 conejo  gato
```

La función **`rename_all`** permite renombrar todas las variables dentro de un data frame, y combinado con la función `tolower` permite pasar a minúscula todos los nombres.

```
base <- rename_all(base, tolower)
names(base)
```

```
## [1] "código_región" "edad"          "n"             "area"
## [5] "sexo"
```

Función `mutate()`

La función `mutate()` tiene la finalidad de realizar transformaciones sobre los valores de un data frame.

Algunas de las transformaciones posibles son:

- Cambiar el formato de una variable.

```
# Cambiar el formato de una variable
class(df$x4)
```

```
## [1] "numeric"
```

```
df <- mutate(df, x4 = as.character(x4))
class(df$x4)
```

```
## [1] "character"
```

Transformación de datos

Función `mutate()`

- Modificar el valor de alguna variable.

En este caso se requiere la utilización de una función auxiliar llamada `ifelse()`. Esta función trabaja con los siguientes argumentos: *test*, *yes* y *no*.

```
# Modificar la edad "100 o más" por 100
base <- mutate(base,
               edad = ifelse(test = edad == "100 o más",
                             yes = "100",
                             no = edad))
table(base$edad == "100")
```

```
##
## FALSE  TRUE
##  6400    64
```

Transformación de datos

Función `mutate()`

- Permite utilizar variables del data frame para transformar otras.

Un mini ejercicio: utilizando el objeto **`base`**, generar una variable llamada `tramos_edad`, la cual agrupe a las personas en 3 tramos (0-30, 31-60, 61-100).

```
# Cambiar el formato de una variable
base <- mutate(base,
               edad = as.numeric(edad))
base <- mutate(base,
               tramos_edad = ifelse((edad %in% 0:30),
                                    1,
                                    ifelse((edad %in% 31:60),
                                           2,
                                           3)))
table(base$tramos_edad, useNA = "always")
```

```
##
##    1    2    3 <NA>
## 1984 1920 2560    0
```

Transformación de datos

Función `mutate()`

- Permite modificar el valor de más de una variable. La modificación se realiza de izquierda a derecha.
- Permite crear una variable que no existe

A continuación un pequeño ejercicio...

Utilizando el data frame `df`, generar una variable `x1` con los mismos valores de `var1`. Luego, generar una variable `x6` cuyos valores sean igual a `x1 + x2`. Asignar el resultado de la función al data frame `df`.

```
df <- mutate(df,
             x1 = var1,
             x6 = x1 + x2)
head(df)
```

```
##   var1 x2 y3 x4   var5    y1 x1 x6
## 1    1  1  a  5  perro     1  1  2
## 2    2  7  b -1   gato perro  2  9
## 3    3  2  c  8 conejo  gato  3  5
```

Transformación de datos

Hasta ahora hemos visto cómo `renombrar` variables, cómo `seleccionar` columnas, cómo `filtrar` filas, cómo `reordenar` las filas y cómo `crear/transformar` variables en un data frame.

Realicemos un ejercicio aplicando lo aprendido:

- Utilizando el data frame `base`, renombrar la variable `código_región` y asignarle el nombre `region`. El resultado guardarlo en el objeto `resultado`.

Las siguientes transformaciones realizarlas sobre el objeto `resultado`.

- Reordenar las columnas de forma tal que la columna `n` corresponda a la última columna del data frame.
- Cambiar el formato de las variables `region` y `edad`, y convertirlas en variables numéricas (usar `as.numeric()`).
- Filtrar el objetivo, y solo quedar con los registros que correspondan a mujeres (sexo == 2) con región distinta a la 1 (region == 6).
- Reordenar las filas de acuerdo a: `region` y `edad`.

Transformación de datos

El resultado es el siguiente:

```
# Renombrar
resultado <- rename(base, region = código_región)
# Seleccionar
resultado <- select(resultado, c(1,2,4,5,3))
# Cambiar el formato
resultado <- mutate(resultado, region = as.numeric(region),
                    edad = as.numeric(edad))
# Cambiar el formato
resultado <- filter(resultado, (sexo == 2 & region == 6))
# Reordenar filas
resultado <- arrange(resultado, region, edad)
# Miremos el resultado
head(resultado, n = 3)
```

```
## # A tibble: 3 x 5
##   region  edad  area  sexo     n
##    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1      6     0     2     2  1264
## 2      6     0     1     2  4246
## 3      6     1     2     2  1367
```

¿Es posible encadenar todas estas instrucciones? Si, utilizando el operador **%>%** (llamado **pipe**)

Uso de pipes

Uso de pipes

El operador **%>%** nos permite tomar el resultado de una función y mandarlo directamente a la siguiente función, concatenando acciones.

Se puede leer como "luego" o "a continuación".

Este operador nos ayudará enormemente a mejorar la legibilidad de un código.

Utilizando pipes, el resultado del ejercicio anterior es posible escribirlo de la siguiente manera:

```
library(dplyr)
resultado <- base %>% # data frame a manipular
  rename(region = código_región) %>% # renombrar
  select(1,2,4,5,3) %>% # seleccionar columnas
  mutate(region = as.numeric(region),
         edad = as.numeric(edad)) %>% # cambiar formatos
  filter(sexo == 2 & region == 6) %>% # filtrar data frame
  arrange(region, edad) # ordenar filas
```

Agrupación de unidades

Resumen de información

Para la generación de tablas de resumen de información se utilizan dos funciones:

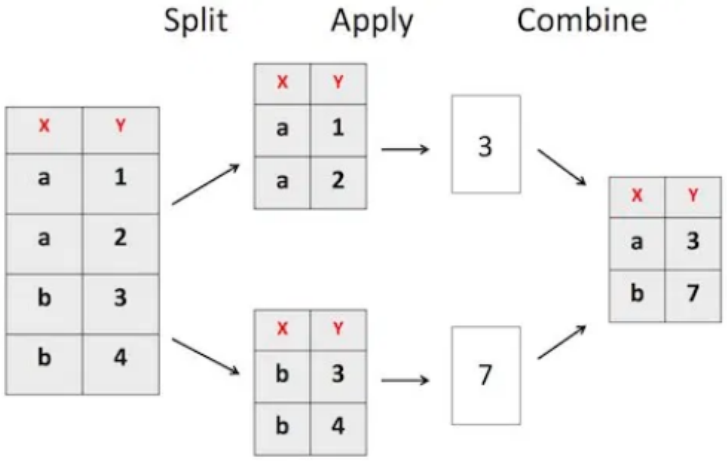
- `group_by()`
- `summarise()`

La función `group_by()` permite agrupar a las observaciones según una categoría de una variable y aplicarles alguna función.

Esto resulta muy útil pues muchos nuestros análisis estadísticos se aplican sobre grupos definidos por una variable o una combinación de variables (sexo, región, comuna, etc).

Gran parte del análisis de datos que realizamos involucra la aplicación de una estrategia que en inglés se acuña como *split-apply-combine*.

Resumen de información



- Primero `group_by()` divide el *data frame*.
- Luego opera una función con la data fragmentada.
- Finalmente `group_by()` vuelve a combinar los resultados.

¿Qué funciones podemos aplicar sobre la data fragmentada?

Todas las funciones que hemos visto durante esta sesión pueden ser combinadas con `group_by()`, pero una que funciona perfectamente es `summarise()`.

Resumen de información

La función `summarise()` crea una o más escalares resumiendo información de variables existentes en un data frame.

En combinación con otras funciones, es una manera de tabular información.

Algunas funciones que se pueden operar dentro de `summarise()` son:

- Tendencia central: `mean()`, `median()`.
- Rango: `min()`, `max()`. `quantile()`.
- Posición: `first()`, `last()`.
- Conteo: `n()`, `n_distinct()`.

Resumen de información

Ahora que ya conocemos estas funciones: ¿cuántas hombres hay por tipo de área y región?

Para resolver a la pregunta, tomamos el objeto **resultado**, filtramos a los hombres, agrupamos por tipo de área y región, y obtenemos la suma de "n".

```
base %>%
  rename(region = código_región) %>%
  filter(sexo == 1 & region == 12) %>%
  group_by(area, tramos_edad) %>%
  summarise(total = sum(n)) %>%
  ungroup()
```

```
## # A tibble: 6 x 3
##    area tramos_edad total
##   <dbl>       <dbl> <dbl>
## 1     1           1 33501
## 2     1           2 31558
## 3     1           3 11056
## 4     2           1  3553
## 5     2           2  4406
## 6     2           3  1175
```

Resumen de información

Revisemos el ejercicio bonus de la tarea 1: ¿cuál es el nombre más usado en Chile en todos los tiempos?

Lo primero que tenemos que hacer es cargar la librería **guaguas**, luego asignar la base de guaguas a un objeto llamado guaguas, por último, debemos generar el cuadro resumen.

```
library(guaguas)
guaguas <- guaguas
guaguas %>%
  group_by(nombre) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total)) %>%
  slice(1) %>%
  select(nombre)
```

```
## # A tibble: 1 x 1
##   nombre
##   <chr>
## 1 María
```

Ejercicio para la 🏠

Vamos a trabajar con una base real de la **Encuesta Nacional Urbana de Seguridad Ciudadana (ENUSC)**, que levanta el INE año a año. Utilizaremos la última versión publicada (2019).

Para la ENUSC, un **hogar victimizado** es aquel en el cual durante el último año **alguno de sus miembros fue víctima de al menos uno de los siguientes delitos:**

- robo con intimidación (**A1_1_1**)
- robo por sorpresa (**B1_1_1**)
- robo con fuerza en la vivienda (**C1_1_1**)
- hurto (**D1_1_1**)
- lesiones (**E1_1_1**)
- robo de vehículo (**G1_1_1**)
- robo desde vehículo (**H1_1_1**)

Es decir, basta con tener un valor 1 (sí) en cualquiera de estas variables para que este hogar sea **victimizado (VA_DC == 1)**.

Ejercicio para la 🏠

Ahora sí el ejercicio:

En primer lugar debes descargar la base de datos 2019 desde el sitio de la ENUSC en la página del INE. Puedes descargarla en formato .csv o en .sav (SPSS). Crea o escoge un fichero en tu computador y descomprimer el archivo ahí. Si deseas, renombra la base de datos para facilitar el trabajo con ella.

1- Utilizando las funciones de **importación** aprendidas, carga la base de datos en tu entorno de trabajo de **R**.

2- ¿Cuántas observaciones y cuantas variables tiene la base?

3- La base es un poco **grande** y los nombres de las variables **poco intuitivos** 😳. Simplifiquemos la base:

- filtrando solo a los informantes de la encuesta (Kish == 1).

- seleccionando solo las variables que que usaremos (`enc_region`, `VA_DC` y las variables de victimización indicadas en la lámina anterior)

- y renombrando estas variables para hacerlas más entendibles (sugerencia: region, intimidacion, sorpresa, fuerza_viv, hurto, lesiones, de_vehiculo, desde_vehiculo)

Ejercicio para la 🏠

Puedes hacer todo lo anterior en una sola instrucción usando `pipes` (`%>%`). Te recomendamos guardarla en un objeto diferente, por ejemplo: `enusc_2`.

4- ¿Cuántas observaciones y cuantas variables tiene tu base ahora?

5- Ahora, Genera una **variable dicotómica con valores 0 y 1** que indique, a partir de las variables de delitos, si un hogar ha sido víctima de al menos un delito. Crearla dentro de la base de datos y ponerle como nombre `VA_DC_2`.

6- Compara la victimización agregada que calculamos (`VA_DC_2`) con la que ya viene construida en la base original (`VA_DC`). ¿Son iguales?

7- Crear una tabla que muestre la victimización agregada para cada región (**Sin usar factores de expansión**).

Ejercicio para la 🏠

hint 1 👀:

Puedes usar `ifelse()`, a través de mutate, para generar variables dicotómicas.

```
enusc_2 <- enusc_2 %>%
  mutate(intimidacion = ifelse(intimidacion==1,1,0))
```

hint 2 🕵:

Existe una función llamada `rowSums()`. que te podría servir para sumar un total fila sobre un set de variables.

hint 3 □:

Los valores `NA` son muy contagiosos. Puedes evitar su influencia usando el argumento `na.rm = TRUE`.

Referencias

Nada de esto sería posible sin:

- `R fot Data Science, de Hadley Wickham`
- `Advanced R, de Hadley Wickham`
- `Data wrangling, exploration, and analysis with R, de Jenny Bryan`
- `Introduction to R, de Data Carpentry`
- `Xaringan: Presentation Ninja, de Yihui Xie`. Para generar esta presentación con la planilla ninja ⚔

R for Data Science tiene una traducción al español realizada por la comunidad hispana de R:

- `R para ciencia de datos, de Hadley Wickham`

Capacitación en R y herramientas de productividad

Proyecto Estratégico Servicios Compartidos para la Producción Estadística

Procesamiento de bases de datos

Agosto 2020