Transformar y seleccionar variables

Sesión N° 4

29 agosto 2021

Análisis de datos estadísticos en R

Profesora Valentina Andrade de la Horra
Ayudantes Dafne Jaime y Nicolás Godoy

Universidad Alberto Hurtado

Contenidos Sesión 4

Básicos en manipulación

Operadores y tipos de datos

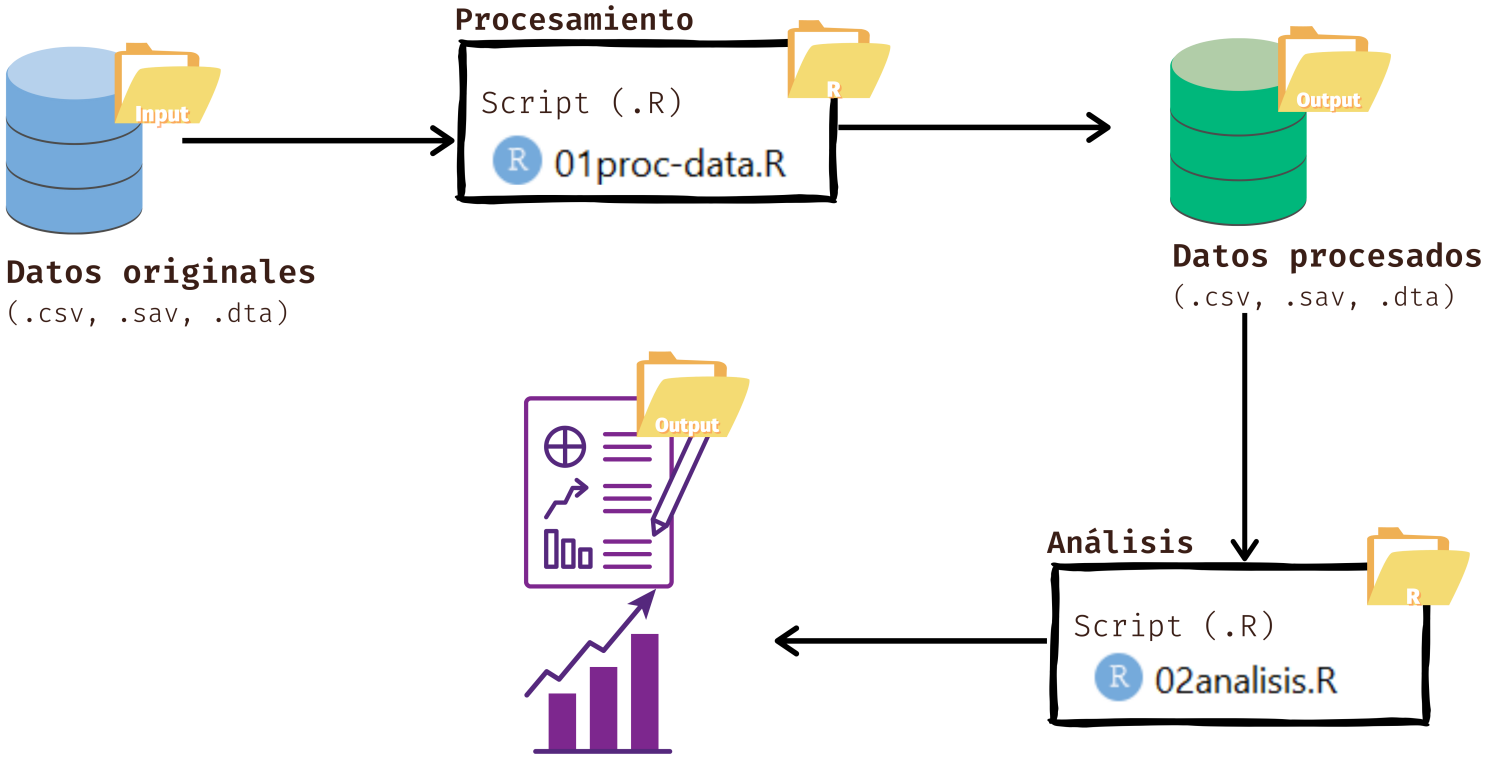
Seleccionar variables

Filtrar

Crear variables

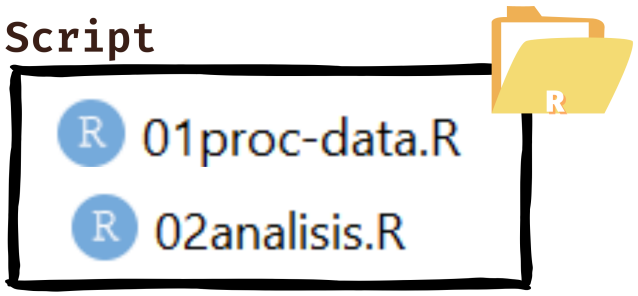
1: Flujo del Rproject

Etapas del flujo



- *Hoy* nosotras/os finalizaremos la parte de **procesamiento**.

Orden de un script de procesamiento




```
# 1. Cargar paquetes -----------------------------------

# 2. Cargar datos --------------------------------------

# 3. Procesar datos ------------------------------------

# 4. Guardar datos -------------------------------------
```

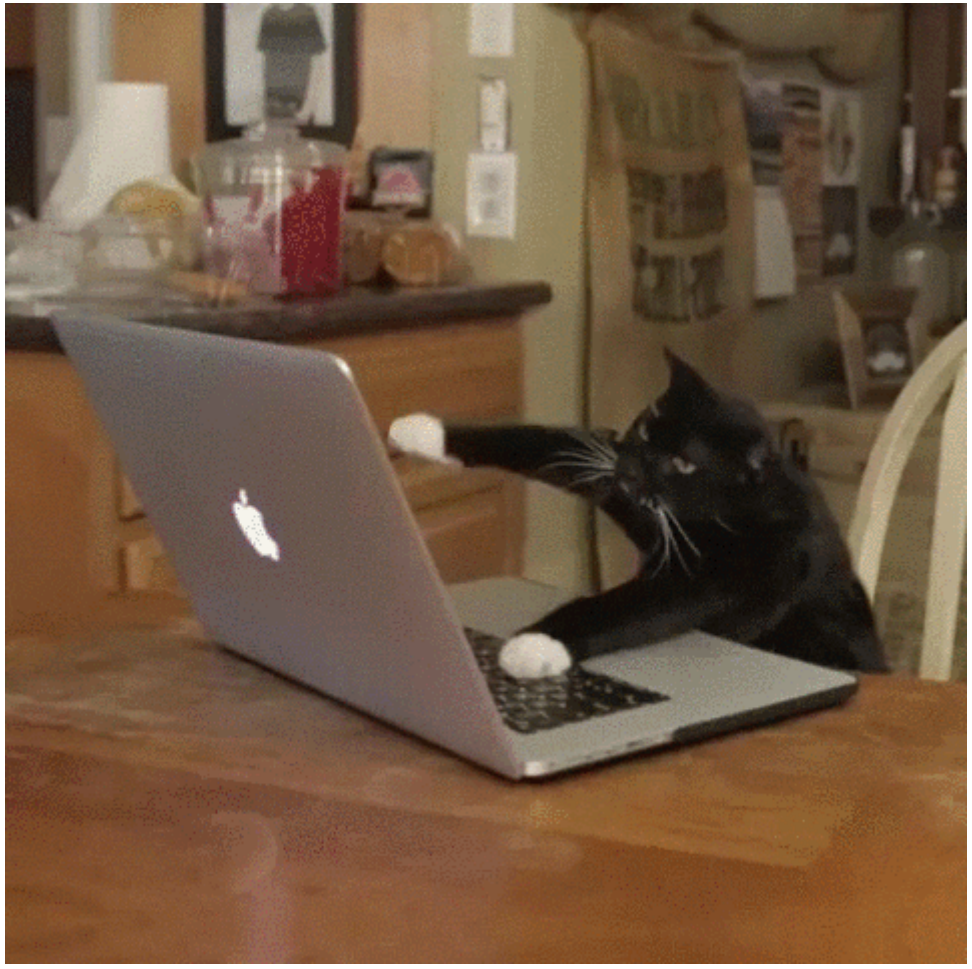


Figura 1: Estudiantes de Análisis de datos en R haciendo el **paso 1 y 2.**

Paso 1: Cargar paquetes

Paso 1: Cargar paquetes

```
pacman::p_load(tidyverse,
               magrittr,
               car,
               sjmisc)
```

Paso 2: Importar datos

Consideraciones antes de importar datos

Para **importar** los datos en R debemos tener en consideración tres cosas:

1. 2. 3.

Consideraciones antes de importar datos

Para **importar** los datos en R debemos tener en consideración tres cosas:

1. Cómo se llaman los datos (en nuestro caso Casen en Pandemia 2020 STATA)
2. El formato de nuestros datos (en nuestro caso .sav)
3. El lugar de donde están alojados nuestros datos.

Paso 2: Importar datos

```
datos <- read_dta("../Rproject/input/Casen en Pandemia 2020 ST
```

Como resultado

Nuevo objeto en el Enviroment

Paso 3: Procesar datos

Procesar datos

Es un proceso **iterativo**

explorar ⏭ **transformar**

transformar ⏭ **explorar**

Por ello, si bien para manipular datos estaremos utilizando constantemente paquetes de `tidyverse`, también utilizaremos el paquete `base` y `sjmisc` para explorar nuestros datos

Explorar datos para procesar

```r
# 1. Base

dim(datos) # Nos entrega las dimensiones, es decir el numero d
View(datos) # Visualizar objetos
names(datos) # entrega los nombres de las variables que compon
head(datos) # muestra las primeras filas presentes en el marco

# 2. sjmisc

find_var(datos, "concepto") # Encontrar variables
```

¡Partamos el paso 3: procesar datos!

¿Dónde?

Descargar el zip del sesión 4 el sitio del curso

1. Recursos de la práctica

- Datos: *Encuesta de Caracterización Socioeconómica (CASEN)* (2020).
- Para ello, deben dirigirse al siguiente enlace y descargar los zip.
- **Libro de códigos** antes de trabajar una base de datos.

Al explorar nuestros datos

Pudimos notar cierta información sobre las columnas que de a poco debemos asimilar

Tipos de datos

- Las que principalmente nos interesarán son los "vectores"
- En nuestro "idioma", son las **variables**.
- En general, una combinación de vectores da origen a una matriz (o **data frame** o "base de datos" [^1])

[^1]: *Un error omitido por muchas/os*

Tipos de datos: columnas

Relación entre clase y nivel de medición de la variable

- `numeric`
- `character`
- `factor`
- hay varios más...

Operadores

- Símbolos que no son de uso exclusivo en R ¡probablemente los conoces desde tus cursos de matemática! (*la suma, la resta*)
- Ahora bien, no todos tienen el mismo significado que en otros softwares.
- Tendrán distintos objetivos: relacionar, condicionar, excluir, repetir etc. Lo importante: buscan darle un **sentido** a la "orden" que le estamos dando a R.
- Los utilizaremos cuando **filtremos** nuestros datos para personas de ciertas categorías, cuando **calculemos variables** nuevas (de

Operadores relacionales

Se usan para hacer comparaciones. Cuando en la *Tabla 1* nos referimos a `un valor`, esto refiere tambien a `variables`

Símbolo	Función
<	Un valor es menor que otro
>	Un valor es mayor que otro
==	Un valor es igual que otro [^1]
<=	Un valor es menor o igual que otro
>=	Un valor es mayor o igual que otro

Tips

- ¡Atención! Fíjate bien que `==` y `=` son distintos. En R `==` es indicar *"igual a"*, mientras que `=` es *asignar* (sinónimo de `<-`)
- El operador `%in%` es **muy utilizado**, sirve para indicar que algo está dentro de una cadena de valores.

Operadores aritméticos

Realizan operaciones, como la suma, resta, división, entre otros.

Símbolo	Función
+	Suma
-	Resta
*	Multiplicación
/	División
^	Elevado

Operadores de asignación

Hay dos formas de asignar:

- `objetoA <- objetoB`
- `objetoA = objetoB`.

Ambas implican que lo que se este realizando en el *objetoB* implica que eso va a producir o generar al *objetoA*.

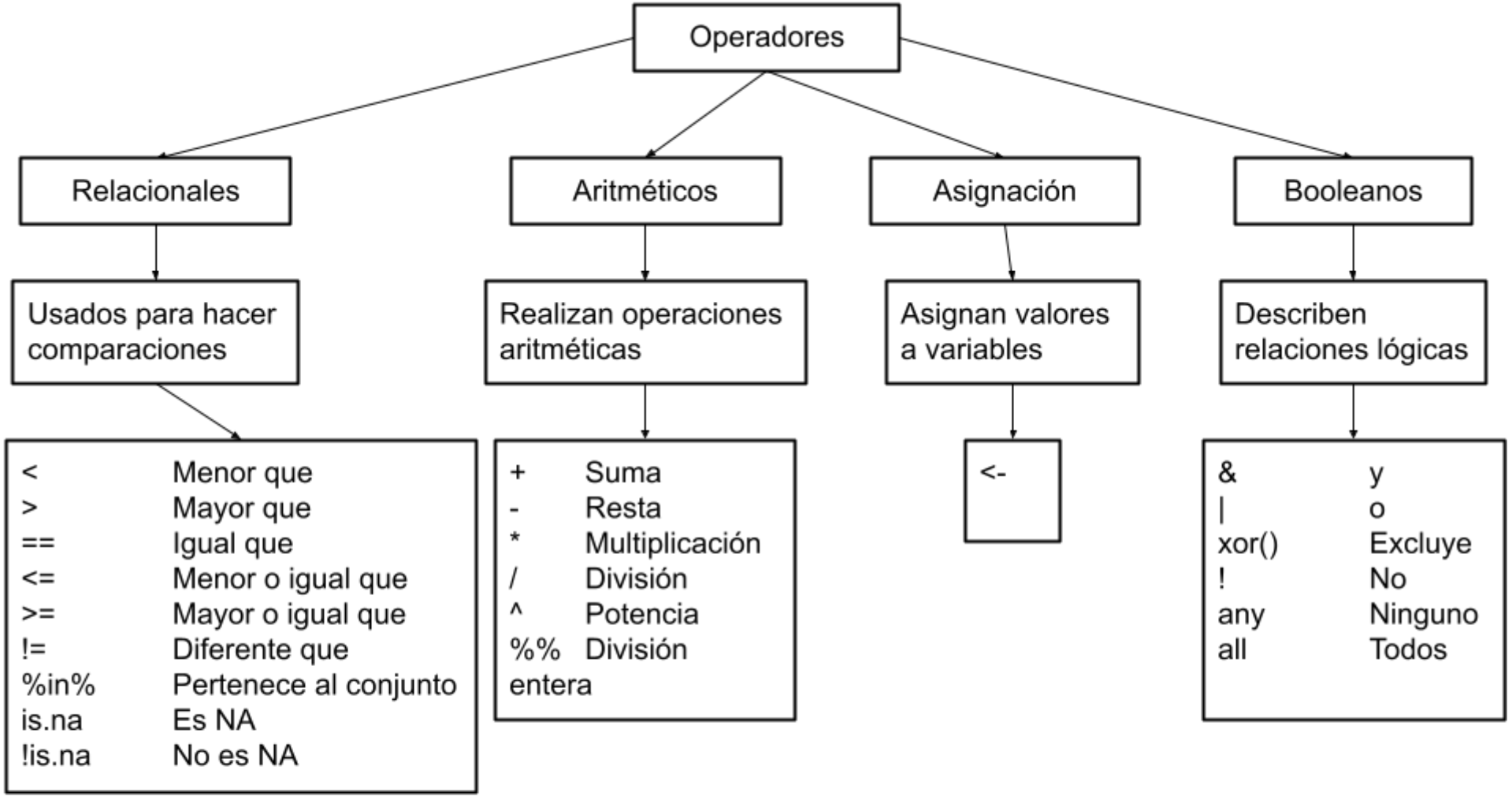
La diferencia está en que la segunda opción es más utilizada dentro de las funciones pues su significado más bien es *"es"*

Operadores booleanos

Describen relaciones **lógicas** o **condicionales**

Símbolo	Función
`&`	Indica un *y* lógico
`|`	Indica un *o* lógico
`xor()`	Excluye la condición
`!`	Distinto de ...
`any`	Ninguna de las condiciones serán utilizadas

Resumen


Operadores
Relacionales
Aritméticos
Asignación
Booleanos
Usados para hacer comparaciones
Realizan operaciones aritméticas
Asignan valores a variables
Describen relaciones lógicas
< Menor que
> Mayor que
== Igual que
<= Menor o igual que
>= Mayor o igual que
!= Diferente que
%in% Pertenece al conjunto
is.na Es NA
!is.na No es NA
+ Suma
- Resta
* Multiplicación
/ División
^ Potencia
%% División entera
<-
& y
| o
xor() Excluye
! No
any Ninguno
all Todos

Redoble de tambores, el más importante de la fiesta 

Operador pipeline %>%

- `%>%` (llamado `pipe`) no es un operador que esté contenido en las funciones base del lenguaje R (origen `magrittr` de `tidyverse`)
- Es de los operadores **más útiles y utilizados** en R.

En el teclado

- `Ctrl + shift + M` Para Windows
- `⌘ + shift + M` Para Mac

Operador pipeline %>%

- Función: **concatenar** múltiples funciones y procesos.
- ¿Cómo es eso? 🤨
- ¡Tranquila/o! Ya lo veremos 🧘

¡Ya queremos ir a practicar!

Transformación y selección de variables

1. `select()` para seleccionar variables
2. `filter()` para filtrar observaciones
3. `mutate()` crear nuevas variables
    - `recode()`, `if_else()`, `case_when()`

`select()` para manipular variables

- Para **seleccionar variables** ocuparemos `select()`.

Si queremos incluir las variables `variable1`, `variable2` y `variable3`

```
select(datos, variable1, variable2, variable3)
```

Si queremos excluir anteponemos un menos `-variable1`

```
select(datos, -variable1)
```

Formas de hacer `select()`

1. Por indexación

```
select(datos, 1,2) # la primera y la segunda columna
select(datos, 1:4) # la primera hasta la cuarta columna
```

1. por nombre de columna

```
select(datos, edad, sexo, o1)
```

Trucos de `select()`

1. Renombrar en el mismo proceso de selección indicando `nuevo_nombre = nombre_original` en el proceso de selección

```
select(datos, edad, sexo, ocupacion = o1)
```

1. Reordenar variables

- `everything()` nos indica que ponga "todo el resto".

```
select(datos, id_persona, sexo, edad, everything())
```

Formas de hacer `select()`

1. Con patrones de texto: prefijos, sufijos o partes de *cómo están nombradas las variables*.

- Poner textos y expresiones regulares entre **comillas**.
- `starts_with()`: prefijo
- `ends_with()`: sufijo
- `contains()` : contiene una cadena de texto literal
- `matches()` : coincide con una expresión regular

Formas de hacer `select()`

```
select(datos, starts_with("a"), ends_with("preg"))

# También se pueden combinar con operadores logicos

select(datos, starts_with("y1")&ends_with("preg"))
select(datos, contains("pobre")|contains("vivienda"))
select(datos, matches("pobreza_|vivienda"))
```

Formas de hacer `select()`

1. Con condiciones lógicas

- `select()` + `where()`: *"seleciona donde"*, ese *donde* responde a una condición que cumple cierta variable.
- Por ejemplo, queremos seleccionar todas las variables que son carácteres (`is.character`):

```
select(datos, where(is.character))
```

- Luego de la exploración de datos mediante a funciones como `find_var()` de `sjmisc` decidimos trabajar con las siguientes variables.

- `edad`

- `sexo`
- `s13`: previsión de salud
- `tot_per`: número de personas en el hogar
- `ytoth`: ingresos totales del hogar
- `o1`: ocupación
- `y26d_total`: Monto del IFE
- `y26d_hog`: ¿Alguien recibió el IFE?

Nuevo data set

- Buena práctica trabajar solo con las columnas que utilizaremos para el análisis, principalmente pues disminuye el *uso de memoria*

```
datos_proc <- select(datos, edad, sexo, prev = 592, ocupacion
```

¡Ejercicio en grupos!

- Seleccionen una variable de nivel educacional, region, sexo, la variable 700 y todas las que refieran a ingresos.
- Indiquen cómo hicieron ese procedimiento
- Con estos cambios, creen un nuevo objeto llamado `ejercicio`

¿Qué pasa si quiero trabajar con un

`filter()` para manipular observaciones

- La función `filter()` de `dplyr` escoge o extrae filas basados en sus valores, subdivide un data frame (*subset*)
- Uso de los **operadores**

```
filter(datos, condicion_para filtrar)
```

Imaginemos que queremos filtrar valores mayores o iguales 3 **respecto a la variable1**

```
filter(datos, variable1 >= 3)
```

Formas de `filter()`

1. Con números

Imaginémos que queremos una base con las personas mayores de 15 años.

```
filter(datos_proc, edad >= 15)
filter(datos_proc, edad >= 15 & tot_per <7)
```

Pero también que pertenezcan a hogares con menos de 7 personas.

```
filter(datos_proc, edad >= 15 & tot_per <7)
```

Formas de `filter()`

1. Con carácteres

- Importancia de explorar los datos

  1. Fijarse bien cómo están escritos
  2. Clase (ver si efectivamente son carácteres)

- R es *sensible* a cómo está escrito el texto (*key sensitive*)

Truco

- Cuando haya problemas aplicar `as_factor()` que permite conservar los niveles pero definiendo sus categorías de respuesta en base a la etiqueta que traen (el `lbl`)
- Consejo útil sobre todo en bases que provienen de SPSS y STATA

Formas de `filter()`

```
datos_proc$sexo <- as_factor(datos_proc$sexo)
```

```
filter(datos_proc, sexo == "Mujer")
filter(datos_proc, sexo != "Hombre")
```

Un clásico, y la solución: %in%

- ¿Cómo se seleccionan dos condiciones en carácter? Con el operador `%in%`

```
datos_proc$prev <- as_factor(datos_proc$prev)

filter(datos_proc, prev %in% c("Sistema Público FONASA", "ISAP
```

Ejercicio

- Con su data frame `ejercicio` filtren
  - Excluyan a las personas sin estudios
  - Filtren a la persona con máximos ingresos del hogar
  - Conserven a las personas de la RM y Valparaíso.

¿Y si quiero crear variables nuevas?

`mutate()`

`mutate()` para transformación de

- `mutate()` permite hacer operaciones para crear nuevas variables o transformar las ya existentes.

```
mutate(datos, nueva_variable = cálculo o condición)
```

Formas de hacer `mutate()`

1. En base a cálculo

```
mutate(datos_proc, nueva_variable = 3+2)
mutate(datos_proc, nueva_variable = 3+2,
       ingreso_percapita = ytoth/tot_per)
```

Aquí mucha mucha atención...
Viene la aplicación de %>%

¿Qué pasa si queremos, luego de calcular nuestras nuevas variables, filtrar un ingreso per cápita menor o igual a $1.000.000

%>%

```
datos %>%
  mutate(., nueva_variable = calculo ) %>%
  filter(., nueva_variable <= valor)
```

- Básicamente, el %>% permite "ingresar" nuestra base de datos como argumento para cada función e ir operándola en proceso

%>%

```
datos_proc %>%
  mutate(ingreso_percapita = ytoth/tot_per) %>%
  filter(ingreso_percapita <= 1000000)
```

En el teclado

- `Ctrl + shift + M` Para Windows
- `⌘ + shift + M` Para Mac

Crear nuevas variables utilizando *además* otras funciones

mutate(variable_nueva = alguna_funcion_genial)

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde el objeto que era anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

```
datos_proc %$%
```

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.
- Uno/a puede ser *ingenioso* y ocuparla como **validador**

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

```
datos_proc %>%
```

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

  ¡Ahora que estamos seguras/os sobre-escribimos la base!

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

Opciones para recodificar.

- `dplyr::recode()` y `car::recode()`
- Con `dplyr::recode()`: recodificamos las categorías de respuesta de Mujer a Femenino y de Hombre a Masculino

```
datos_proc %>%
  mutate(sexo = dplyr::recode(sexo, "Mujer" = "Femenino", "Hom
```

- El problema de `recode()` que se utiliza dentro de `dplyr` es que si recodifico se pierde la etiqueta anterior.

En síntesis

Básicos en manipulación

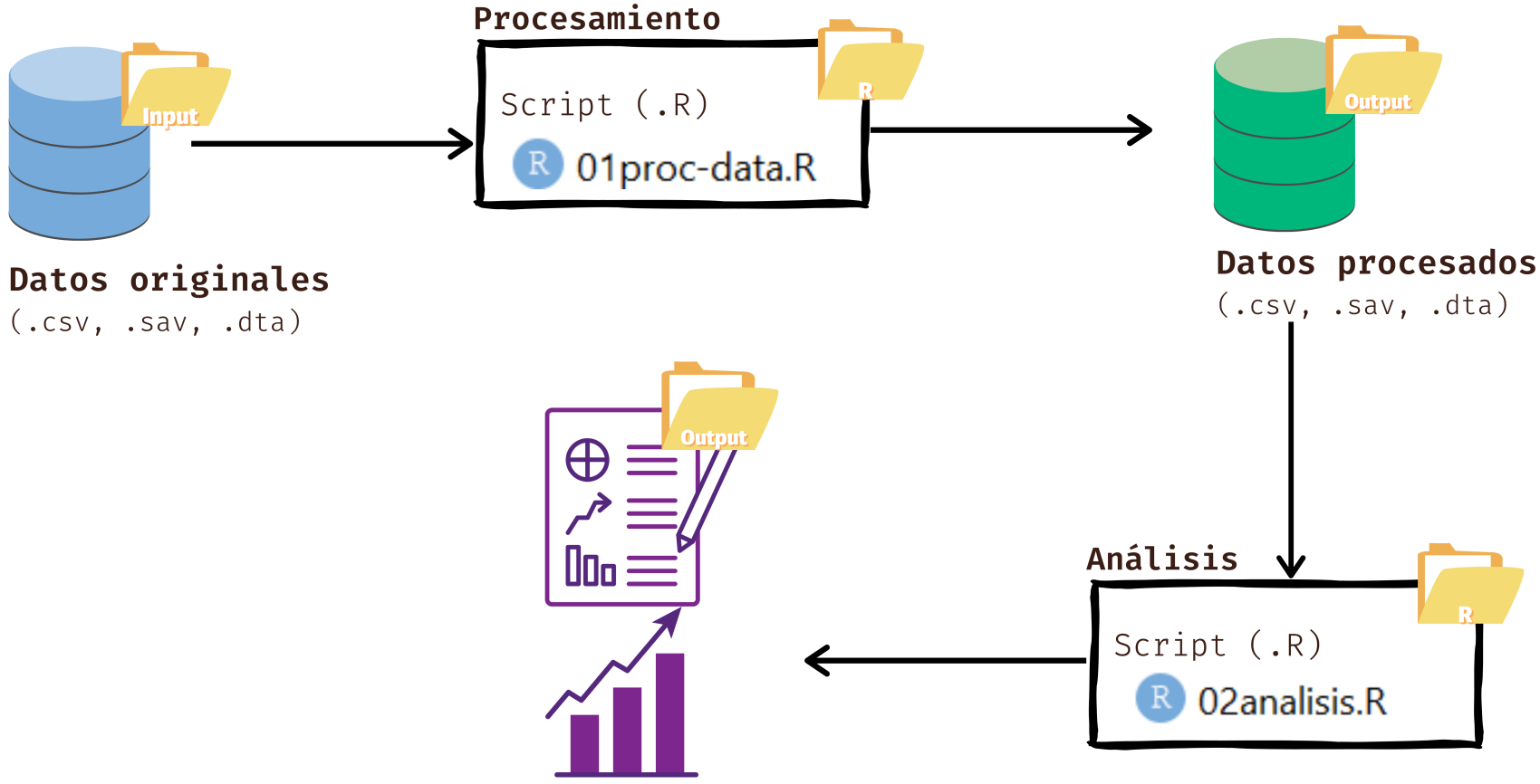
Operadores y tipos de datos

Seleccionar variables

Filtrar

¡Y a no olvidar el flujo para el análisis!

Nos permite hacernos amigas/os más rápido del programa


Procesamiento
Input
Script (.R)
01proc-data.R
R
Output
Datos originales
(.csv, .sav, .dta)
Datos procesados
(.csv, .sav, .dta)
Output
Análisis
Script (.R)
02analisis.R
R

¿Y eso era?

¡Ahora si que si! Nos vemos el próximo viernes en la última sesión

Transformar y seleccionar variables

Sesión N° 4

29 agosto 2021

Análisis de datos estadísticos en R

Profesora Valentina Andrade de la Horra
Ayudantes Dafne Jaime y Nicolás Godoy

Universidad Alberto Hurtado