Manipulación de datos

Curso básico sobre R para Investigación Social

Introducción

Hasta este punto hemos visto como importar bases de datos y a partir de las funciones básicas realizamos una primera exploración a las bases. Además de la primera exploración, uno de los puntos importantes luego de importar dataframes, es la manipulación de los datos que contiene el dataframe con el que estamos trabajando. Es decir, verificar si los datos han sido cargados correctamente y decidir de qué manera adecuarlo a nuestros intereses de la investigación, así como también poder realizar resúmenes con estos datos. Existe un paquete llamado `dplyr` el cual es muy útil y fue diseñado para manipular dataframes. Este paquete contiene una serie de funciones que nos permiten manipular los datos para el análisis de los mismos. A continuación se presenta una tabla que resume las funciones más utilizadas de este paquete.

Tabla 5

Función	Ejemplo de sintaxis
arrange()	arrange(variable1, variable2)
left_join()	dataframe1 %>% left_join(., dataframe2, by = "variable")
filter()	filter(condición lógica según valor de variable). Ejemplo, filter(variable == "valor")
select()	select(variable1, variable2)
mutate()	mutate(variable_nueva = operación entre variables). Ejemplo: mutate(var_nueva = variable1/variable2)
case_when	case_when(condicion1 ~ "Valor1", condicion2 ~ "Valor2", condicion3 ~ "Valor3")
rename()	rename(nombre_nuevo = nombre_viejo)
summarise()	summarise(nombre_col = medida(variable)
group_by()	group_by(variable1) %>% summarise(nombre_col = medida(variable)

Iremos viendo más adelante cada una de estas funciones.

Instalación

Podemos instalar el paquete Tidyverse que contiene al paquete dplyr entre otros paquetes útiles para el analisis de datos o directamente instalar dplyr individualmente.

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

# O sino instalar solo dplyr
install.packages("dplyr")
```

Pipe

Este paquete está diseñado para que todas las funciones trabajen encadenadamente mediante la utilización del **pipe** (tubería), el cual se escribe con el caracter **%>%** (se realiza combinando las teclas Ctrl + Shift + M o Cmd + Shift + M en Mac). La acción que realiza el pipe es llamar a un objeto que se encuentra a la izquierda y transformarlo mediante la función que se le indique a la derecha. El objeto que se indique a la izquierda no se verá afectado por cambios, a menos que se lo índique. Iremos viendo con más detalle en los ejemplos. Ahora vamos a ver ejemplos para cada una de las funciones de este paquete trabajando con las bases que importamos anteriormente.

Ordenar

La función `arrange` permite ordenar la base de datos en base a algún valor de una o más variables. Mediante esta función vamos a ordenar las bases de los hogares y de las personas por las variables Número identificador de vivienda (nocues) y Número identificador de hogar (nhog).

```
# Abrimos la libreria
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
# Ordenamos ambas bases con la funcion arrange
hogares <- hogares %>% arrange(nocues, nhog)
personas <- personas %>% arrange(nocues, nhog)
```

Emparejar

Existen cuatro tipos de uniones para bases de datos en el paquete dplyr y puede encontrar más información al respecto de cada una de ellas aquí. De estas cuatro maneras de emparejar bases, solamente nos vamos a centrar en una, es decir, en la que se realiza mediante la función `left_join`. Siendo así, respetando la cantidad de casos y de variables que tiene la base de personas, vamos a agregar las variables de la base de hogares. Esto le haremos de la siguiente manera:

```
# Emparejar bases
pisac <- personas %>% left_join(., hogares, by = c("nocues", "nhog"))
# el . llama al objeto que está antes del pipe
```

Entonces, hemos emparejado ambas bases y además también guardamos este resultado en un nuevo dataframe que se llama pisac. De esta manera, en esta nueva base incorporamos a la base de las personas todas las variables de las base de los hogares respetando las variables que funcionan como "key" o "llave" (nocues y nhog).

Filtrar y seleccionar

Con la función `filter` es posible filtrar la base de datos acorde al cumplimiento de condiciones lógicas y la función `select` permite especificar la serie de columnas o variables que se desea conservar o descartar de un dataframe. A continuación se presenta una tabla que resume las distintas condiciones lógicas que podemos utilizar para filtrar casos con la funcion `filter`.

Tabla 6

Símbolo	Significado	Símbolo	Significado
>	Mayor que	!=	Distinto a
<	Menor que	%in%	Dentro del grupo
==	Igual a	is.na	Es NA
>=	Mayor o igual a	!is.na	No es NA
<=	Menor o igual a	&, |	y, o

Fuente: https://derek-corcoran-barrios.github.io/Libro/_book/tidydata.html#filter

Entonces, a partir de esta nueva base pisac, que fue el resultado del emparejamiento de las bases personas

y hogares, vamos a filtrar solamente los casos de la región del NEA y a su vez le pediremos que seleccione solamente algunas variables del total de variables que contiene el dataframe.

```
# Filtrar la base emparejada por la region NEA y seleccionar variables
NEA <- pisac %>%
  filter(region.x == "NEA (Chaco, Corrientes, Formosa y Misiones)") %>%
  select(v108, v109, v116, t_hogar, nivel_ed, v134a,
estado, v179, nocues, nhog, f_calib3.x, region.x, aglo.x, v12, v14, v15, v16, v19, v235i,
v237, v259a, v260a, v213bi)
```

De esta manera, tendríamos creado un nuevo dataframe que se llama NEA y que contiene únicamente los casos correspondientes a la región del NEA y 23 variables que fueron seleccionadas del dataframe llamado pisac.

Renombrar

Podemos cambiar los nombres a las variables con la función `rename`. A modo de ejemplo modifiquemos solo tres nombres de las variable v108 a Edad, v109 a Sexo y v116 a Est_civil.

```
# Renombrar variables
NEA <- NEA %>% rename(Edad = v108,
                      Sexo = v109,
                      Est_civil = v116)
```

Resumir

La función `summarise` nos permite realizar tablas de resúmenes numéricos con la información original de una variable. Vamos a crear a modo de ejemplo una tabla que contenga algunas medidas para la variable *Ingreso total del hogar* (v235i).

```
# Resumen
Resumen_igreso <- NEA %>% summarise(Promedio_ing = mean(v235i),
                                    Mediana = median(v235i),
                                    Varianza = var(v235i),
                                    "Desvío estandar" = sd(v235i))

Resumen_igreso
```

```
##   Promedio_ing Mediana Varianza Desvío estandar
## 1     9925.995    7640 60203908        7759.118
```

Agrupar

La función `group_by` se complementa con la función `summarise` y permite generar resúmenes numéricos para cada valor categórico de una variable cualitativa. Por ejemplo, podemos generar la misma tabla de resumen anterior pero ahora agrupando las medidas para cada valor de la variable *Nivel educativo* (nivel_ed).

```
# Resumen por grupo
Ingreso_x_educ <- NEA %>%
  group_by(nivel_ed) %>%
  summarise(Promedio = mean(v235i),
```

```
            Mediana = median(v235i),
            Varianza = var(v235i),
            "Desvio estandar" = sd(v235i))

Ingreso_x_educ
```

```
## # A tibble: 11 x 5
##    nivel_ed                     Promedio Mediana  Varianza `Desvio estanda~
##    <fct>                           <dbl>   <dbl>     <dbl>            <dbl>
##  1 Menores de 5 años               9284.   6957     7.21e7            8492.
##  2 Sin instrucción (incluye nu~    6912.   5500     1.86e7            4310.
##  3 Primaria/EGB incompleto         8387.   6838     4.80e7            6932.
##  4 Primaria/EGB completo           7985.   6926.    2.69e7            5191.
##  5 Secundario/Polimodal incomp~    9215.   7000     4.18e7            6466.
##  6 Secundario/Polimodal comple~   11223.   9000     6.54e7            8090.
##  7 Terciario incompleto           10739.   9000     4.71e7            6866.
##  8 Terciario completo             13862.  11000     8.28e7            9098.
##  9 Universitario incompleto       13332.  11800     5.97e7            7729.
## 10 Universitario completo         20716.  18000     1.40e8           11827.
## 11 Educación especial              5249.   5700     6.38e6            2527.
```

Observe que en el caso del nombre de la columna Desvío estándar, el mismo está indicado entre comillas, esto permite crear nombres de columnas que estén separados con espacio.

Transformar

Utilizando la función `mutate` podemos agregar y crear nuevas variables a partir de operaciones con la misma o entre otras variables. Veamos cómo podemos recodificar una variable numérica realizando una operación matemática para crear y agregar al dataframe NEA la varibale *Ingreso total del hogar per cápita* (ITH_PC) que será resultado del cociente entre las variables *Ingreso total del hogar imputado* (v235i) y *Tamaño del hogar* (t_hogar).

```
# Recodificar variable
NEA <- NEA %>% mutate(ITH_PC = v235i/t_hogar)

# Observamos los cinco primeros valores
head(NEA$ITH_PC, 5)
```

```
## [1] 1273.714 1273.714 1273.714 1273.714 1273.714
```

Caso cuando

La función `case_when` permite crear una nueva variable generando los valores de la misma a partir de condiciones específicas. En caso de que no se cumpla las condiciones indicadas la variable tomará valores NA (valor faltante o valor perdido). Esta función se complementa con la función `mutate` y es útil para recategorizar valores de una variable cualitativa. Continuemos recategorizando los valores de la variable *Clase social de pertenencia* (v260a), agrupando los valores en Clase baja, Clase obrera, Clase media y Clase alta.

```r
# Primero observamos cuales son los valores de la variable
levels(NEA$v260a)
```

```
## [1] "Clase baja"       "Clase obrera"     "Clase media baja"
## [4] "Clase media"      "Clase media alta" "Clase alta"
## [7] "NS/NR"
```

```r
# Ahora recategorizamos los valores
NEA <- NEA %>% mutate(Clase_recod =
               case_when(v260a == "Clase baja" ~ "Clase baja",
                         v260a == "Clase obrera" ~ "Clase obrera",
                         v260a %in% c("Clase media baja",
                                      "Clase media",
                                      "Clase media alta") ~ "Clase media",
                         v260a == "Clase alta" ~ "Clase alta",
                         v260a == "NS/NR" ~ "NS/NR")) %>%
  mutate(Clase_recod = ordered(Clase_recod,
                               levels = c("Clase baja",     # indicamos que se trata de
                                          "Clase obrera",   # una variable ordinal
                                          "Clase media",    # y ordenamos los valores
                                          "Clase alta",
                                          "NS/NR")))
# Consultamos los valores
levels(NEA$Clase_recod)
```

```
## [1] "Clase baja"   "Clase obrera" "Clase media"  "Clase alta"
## [5] "NS/NR"
```

Valores NA (missing values)

En R los valores faltantes o valores perdidos se los conocen como `NA`, es común que cuando trabajemos con bases de datos nos encontremos con estos valores. Para saber si una variable contiene valores `NA` se utiliza la función is.na la cual devuelve un resultado lógico, es decir, si el objeto contiene valores perdidos el resultado será `TRUE`, de lo contrario será `FALSE`. Veamos un ejemplo con la variable *Condición de inactividad* (v179).

```r
# Observar valores NA para los primeros 5 casos
head(is.na(NEA$v179), 5)
```

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

Observe que de los cinco primeros casos, cuatro son NA. Ahora bien, en este caso ¿realmente son valores perdidos? Se debe tener en cuenta que no siempre se trata de valores perdidos como tales, ya que si observamos el formulario de encuesta del PISAC podemos notar que esta variable *Condición de inactividad* surge de una sucesión de preguntas, es decir, la condición de inactividad de la persona va a ser únicamente en el caso de que la persona se declare inactiva en la pregunta anterior. Por lo tanto, en esta variable los valores NA en realidad son valores que no corresponden y no valores perdidos. Esta aclaración es importante y se debe tener en cuenta al momento que nos encontremos con valores NA. Por otra parte, un inconveniente frecuente que suele suceder con estos valores surge cuando queremos realizar alguna medida de resumen numérica, por ejemplo el promedio. Si la variable tiene valores NA lo más probable es que el resultado sea "NA" y para resolver esto se utiliza el argumento na.rm. Observemos un ejemplo con la variable *Ingreso neto en ocupación principal* (v213bi).

```r
# Promedio de v213bi
mean(NEA$v213bi)
```

```
## [1] NA
```

```r
# Omitir valores NA
mean(NEA$v213bi, na.rm = TRUE)
```

```
## [1] 1695.497
```

Observe que el promedio de los ingresos de la ocupacion principal de las personas es considerablemente bajo debido a que ésta variable presenta muchos casos con ingresos de 0 pesos. Por lo tanto, para corregir este problema tendremos que filtrar los datos con solo aquellos que presenten ingresos mayores a 0. Entonces, procedemos a hacerlo de la siguiente manera:

```r
# Creamos la tabla filtrando
Ing_ocup_ppal <- pisac %>%
  filter(v213bi > 0 &
           !is.na(v213bi)) %>%  # filtramos tambien por los casos que no son NA
  select(v213bi) # y seleccionamos solo esta variable

# Calculamos ahora el promedio
mean(Ing_ocup_ppal$v213bi)
```

```
## [1] 6398.697
```

Otra caso particular con el que podemos encontrarnos al momento de manipular datos es la necesidad de convertir uno o más valores como NA. Por ejemplo, en nuestra base NEA la variable Edad se encuentra como tipo factor (categórica) y contiene dos valores "Menor de un año" y "99 años y más" que es necesario primero convertirlos en NA para poder hacer posteriormente el cambio de factor a numérica. Veamos el ejemplo:

```r
# Convertimos los valores en NA
NEA$Edad[NEA$Edad == "Menor de un año" |
         NEA$Edad == "99 años y más"] <- NA

# Pasamos primero de factor a character
NEA$Edad <- as.character(NEA$Edad)

# Por ultimo pasamos a numérica
NEA$Edad <- as.numeric(NEA$Edad)
```

Ahora, es posible recodificar la variable Edad como lo hicimos anteriormente con la funcion `case_when`, creando una nueva variable con categoría de edades.

```r
# Primero observamos los valores para definir categorías
summary(NEA$Edad)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1.00   14.00   26.00   31.43   47.00   97.00      34
```

```r
# Establecemos las categorías
NEA <- NEA %>%
  mutate(Edad_recod = case_when(Edad %in% c(1:14)  ~ "Hasta 14",
                                Edad %in% c(15:64) ~ "de 15 a 64",
                                Edad %in% c(65:97) ~ "65 y más")) %>%
  mutate(Edad_recod = ordered(Edad_recod,
                              levels = c("Hasta 14",
                                         "de 15 a 64",
                                         "65 y más")))
# Consultamos los valores
levels(NEA$Edad_recod)
```

```
## [1] "Hasta 14"   "de 15 a 64" "65 y más"
```

Referencias

Este post está basado en los siguientes aportes bobliográficos:

WICKHAM H., GROLEMUND G. "R for Data Science" (2017). Recuperado de: https://r4ds.had.co.nz/

MENDIZABAL S. "Taller de Introducción a R" (2017). Recuperado de: https://songeo.github.io/introduccion-r-bookdown/

WEKSLER G., KOZLOWSKI D., SHOKIDA N., "Curso de R para procesamiento de datos de la Encuesta Permanente de Hogares" (2018). Recuperado de: https://diegokoz.github.io/Curso_R_EPH_clases/