

Medición y diseño de investigación

Clase 2: Manipulación de datos

FCS-UdelaR - Martín Opertti y Fabricio Carneiro

June 27, 2023

Directorios de trabajo

Directorios de trabajo

- Para abrir en R un archivo guardado en tu computadora, debes especificar en qué carpeta está guardado. Hay muchas opciones, una de ellas es fijar un directorio por defecto (es decir, especificarle a R la carpeta en la que vamos a trabajar):

```
# Puedo fijar el directorio de trabajo con la función setwd()
# Fijar la carpeta donde vamos a importar y exportar los archivos:
setwd("micompu/micarpeta")
getwd() # Con esta función puedo consultar el directorio
```

```
# Ahora, si quiero leer un archivo que esté en "micompu/micarpeta" simplemente
# escribo su nombre dentro de la función, en el lugar del "path".
```

```
# Supongamos que tengo dentro de la carpeta "micarpeta" un excel con datos
# de desempleo en Uruguay:
```

```
library(readxl)
desempleo_uru <- read_excel("data/desempleo.xlsx")
head(desempleo_uru, 4)
```

```
## # A tibble: 4 × 2
##   Year  tasa
##   <dbl> <dbl>
## 1  1990   8.5
## 2  1991   8.9
## 3  1992    9
## 4  1993   8.3
```

Crear una carpeta

En la computadora en la que estés trabajando debes crear una carpeta (dentro de la carpeta que prefieras o en el escritorio), con las siguientes carpetas dentro:

- **Data** Descargar de EVA la carpeta data dentro de la sección laboratorio y pegar el contenido aquí.
- **Scripts** Descargar los distintos scripts (archivos de código) del EVA y guardarlos en esta carpeta (código de la clase, ejercicios, etc.)
- **graficos** Crear una carpeta gráficos que por el momento esté vacía, vamos a ir incluyendo guardando archivos ahí
- **resultados** Crear una carpeta resultados que por el momento esté vacía, vamos a ir incluyendo guardando archivos ahí

Carpeta del curso

Deberían ver en su carpeta algo así:

| | | | | |
|--|--------------------------|------------|------------------|-------------|
| ← → ▾ ↑ > Dropbox > cursos > Medición y Diseño > Laboratorio | | | | |
| > Quick access | <input type="checkbox"/> | Name ^ | Date modified | Type |
| | | Data | 08/06/2023 12:08 | File folder |
| | | graficos | 08/06/2023 12:08 | File folder |
| | | resultados | 08/06/2023 12:08 | File folder |
| | | Scripts | 08/06/2023 12:08 | File folder |
| | | | | |
| > Dropbox | | | | |
| > OneDrive - Duke University | | | | |
| > OneDrive - Personal | | | | |
| > This PC | | | | |
| > Network | | | | |

Directorio de R

Al comienzo de cada script **deben** especificar el directorio donde está su carpeta del curso, de lo contrario los códigos les van a devolver errores.

En este caso, por ejemplo, creé la carpeta "cacrpeta_del_curso" dentro de una carpeta llamada "Medicion y diseño" dentro de "cursos". Para acceder al directorio completo vayan a donde esta su carpeta hagan click derecho en la carpeta, seleccionen "propiedades" y luego ahí bajo dirección pueden copiar el directorio.

```
setwd("C:/Usuario/Martin/Dropbox/cursos/Medicion y diseño/cacrpeta_del_curso")
```

De esta forma, por ejemplo, si queremos leer una base de datos llamada "datauru.xlsx" que está guardada en la carpeta data dentro de la carpeta laboratorio, simplemente tenemos que especificar la siguiente dirección (porque R ya lo va a buscar dentro de la carpeta laboratorio)

```
datauru <- read_excel("data/datauru.xlsx")
```

Importar y exportar datos

Importar datos

- Hasta ahora trabajamos principalmente con datos ingresados manualmente con las funciones `c()` y `data.frame()`
- Normalmente cuando trabajamos con datos solemos utilizar datos ya creados guardados en los formatos de otros programas (ej. Excel, Stata, SPSS)
- Existen varios paquetes que permiten importar y exportar datos desde distintos formatos. Algunos de los más utilizados son `readr`, `haven`, `readxl` y `utils`
- El primer paso es siempre identificar la extensión del archivo que queremos importar. Por ejemplo, las planillas excel (dependiendo de la versión) suelen tener extensión `.xlsx`. Pueden consultar la extensión de un archivo con botón derecho + propiedades
- Luego deben identificar en qué carpeta está guardada esa base de datos.

Importar datos

Distintas funciones nos sirven para importar datos a R desde distintos formatos. Veamos algunos ejemplos:

```
# Con la función read_csv() del paquete readr importamos archivos .csv
library(tidyverse)
gapminder_csv <- read_csv("data/gapminder.csv")

# Con la función read_excel() del paquete readxl importamos archivos excel
library(readxl)
gapminder_excel <- read_excel("data/gapminder.xlsx")
```

```
# Vemos que los dataframes son iguales, tienen la mismas filas y columnas
dim(gapminder_csv)
```

```
## [1] 1704    6
```

```
dim(gapminder_excel)
```

```
## [1] 1704    6
```

Importar datos

Algunos paquetes incluyen datos, por ejemplo, gapminder. En la documentación del paquete se encuentra el nombre de los datos. Con una simple asignación los podemos cargar

```
#install.packages("gapminder")
```

```
library(gapminder)
```

```
data_gapminder <- gapminder
```

```
head(data_gapminder)
```

```
## # A tibble: 6 × 6
```

| ## | country | continent | year | lifeExp | pop | gdpPercap |
|------|-------------|-----------|-------|---------|----------|-----------|
| ## | <fct> | <fct> | <int> | <dbl> | <int> | <dbl> |
| ## 1 | Afghanistan | Asia | 1952 | 28.8 | 8425333 | 779. |
| ## 2 | Afghanistan | Asia | 1957 | 30.3 | 9240934 | 821. |
| ## 3 | Afghanistan | Asia | 1962 | 32.0 | 10267083 | 853. |
| ## 4 | Afghanistan | Asia | 1967 | 34.0 | 11537966 | 836. |
| ## 5 | Afghanistan | Asia | 1972 | 36.1 | 13079460 | 740. |
| ## 6 | Afghanistan | Asia | 1977 | 38.4 | 14880372 | 786. |

Importar datos

También es posible importar datos guardados en los formatos de otros softwares estadísticos como SPSS o Stata. Para esto usaremos el paquete `haven`.

```
library(haven)

# SPSS
gapminder_spss <- read_spss("data/gapminder.sav")

# STATA
gapminder_stata <- read_stata("data/gapminder.dta")
```

O podríamos llamar a la función y paquete dado que generalmente solo utilizamos una función de los paquetes que cargan datos (depende del caso obviamente)

```
# SPSS
gapminder_spss <- haven::read_spss("data/gapminder.sav")

# STATA
gapminder_stata <- haven::read_stata("data/gapminder.dta")
```

Importar datos

R también cuenta con sus propios formatos de almacenamiento de datos (`.rds` y `.Rdata` o `.rda`). Este enfoque es poco práctico si queremos usar los datos almacenados en otro programa, pero muy útil si solamente usaremos R dado que mantiene la información tal cual estaba en R (por ej. tipos de variables o atributos):

```
# Para esto no necesitamos cargar paquetes.  
# Guardar un objeto como .rds:  
saveRDS(object = data_gapminder,  
         file = "resultados/data_gapminder.rds")  
  
# Leemos un archivo .rds  
miobjeto_rds <- readRDS(file = "resultados/data_gapminder.rds")  
  
# Con .rda se pueden guardar varios objetos al mismo tiempo!  
# Exportamos un archivo .Rdata  
save(data_gapminder, miobjeto_rds,  
      file = "resultados/dos_dataframes.Rdata")  
  
# Importamos un archivo .Rdata  
load("resultados/dos_dataframes.Rdata")
```

Exportar datos

- También podemos guardar archivos desde R en otros formatos.
- Con **readr** podemos exportar archivos en formato .csv
- Con **writexl** podemos exportar directamente un excel.
- Con **haven** podemos exportar archivos en formato .dta (Stata) y .sav (SPSS)

```
# Guardar .csv
library(gapminder)
data_gapminder <- gapminder
write_excel_csv(data_gapminder, "resultados/gapminder.csv")

# Guardar excel
library(writexl)
write_xlsx(data_gapminder, "resultados/gapminder.xlsx")

# Guardar .dta (Stata)
library(haven)
write_dta(data_gapminder, "resultados/gapminder.dta")

# Guardar .sav (SPSS)
write_sav(data_gapminder, "resultados/gapminder.sav")

# Guardar .sas (SAS)
write_sas(data_gapminder, "resultados/gapminder.sas")
```

Explorar Dataframes

Resumen de un dataframe

```
dim(data_gapminder) # Número de filas y columnas
```

```
## [1] 1704    6
```

```
names(data_gapminder) # Nombre de variables
```

```
## [1] "country" "continent" "year" "lifeExp" "pop" "gdpPercap"
```

```
head(data_gapminder, 3) # Imprime primeras filas (3 en este caso)
```

```
## # A tibble: 3 × 6  
##   country    continent  year lifeExp      pop gdpPercap  
##   <fct>      <fct>    <int>  <dbl>    <int>    <dbl>  
## 1 Afghanistan Asia      1952   28.8  8425333    779.  
## 2 Afghanistan Asia      1957   30.3  9240934    821.  
## 3 Afghanistan Asia      1962   32.0 10267083    853.
```

Resumen de un dataframe

Una de las funciones más útiles para resumir un dataframe es `glimpse()` del paquete `dplyr` o `tidyverse`. Es particularmente útil debido a que permite un vistazo al nombre, tipo y primeros valores de **todos** las variables de un dataframe.

```
# Resumen más completo:  
glimpse(gapminder)
```

```
## Rows: 1,704  
## Columns: 6  
## $ country   <fct> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan", ...  
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ...  
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ...  
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8...  
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 12...  
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134, ...
```


Tablas simples

En R Base la función para obtener frecuencias es `table()` junto con `prop.table()` y `addmargins()`

```
# Para obtener una tabla de frecuencias de una variable usamos la función
# table() de R Base
tabla_1 <- table(data_gapminder$continent) # Frecuencia simple
tabla_1
```

```
##
##   Africa Americas      Asia  Europe Oceania
##    624      300      396    360     24
```

```
prop.table(tabla_1) # Proporciones
```

```
##
##   Africa  Americas      Asia  Europe  Oceania
## 0.36619718 0.17605634 0.23239437 0.21126761 0.01408451
```

```
addmargins(tabla_1) # Totales
```

```
##
##   Africa Americas      Asia  Europe Oceania      Sum
##    624      300      396    360     24    1704
```

```
addmargins(prop.table(tabla_1)) # Proporciones y totales
```

```
##
##   Africa  Americas      Asia  Europe  Oceania      Sum
## 0.36619718 0.17605634 0.23239437 0.21126761 0.01408451 1.00000000
```

Estadística descriptiva

Medidas de tendencia central

```
mean(data_gapminder$lifeExp) # Media
```

```
## [1] 59.47444
```

```
median(data_gapminder$lifeExp) # Mediana
```

```
## [1] 60.7125
```

```
sd(data_gapminder$lifeExp) # Desvío estandar
```

```
## [1] 12.91711
```

```
range(data_gapminder$lifeExp) # Rango
```

```
## [1] 23.599 82.603
```

```
max(data_gapminder$lifeExp)
```

```
## [1] 82.603
```

```
min(data_gapminder$lifeExp)
```

```
## [1] 23.599
```

Histogramas

También podemos graficar los datos rápidamente. Por ejemplo, un histograma:

```
hist(data_gapminder$lifeExp,  
main = "Distribución de expectativa de vida (Gapminder)")
```

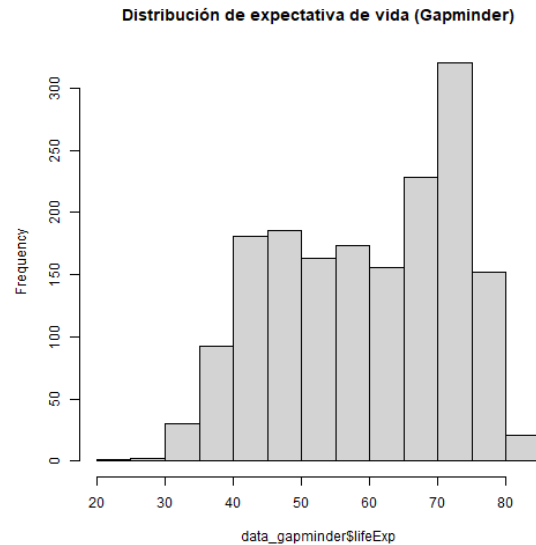
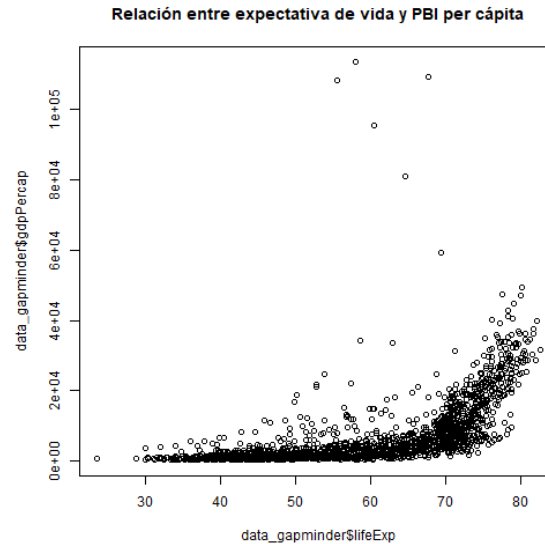


Gráfico de dispersión (scatterplot)

```
plot(data_gapminder$lifeExp, data_gapminder$gdpPerCap,  
main = "Relación entre expectativa de vida y PBI per cápita")
```



Ejercicio

- (1) Importar el archivo excel en la carpeta data llamado "datauru"*
- (2) Explorar las variables y el tipo de cada variable con la función glimpse()*
- (3) Calcular la media de una variable numérica*
- (4) Crear una tabla de proporciones con la distribución de la variable partido*
- (5) Crear un gráfico de dispersión con las variables inflación (eje de las x) y aprobación (eje de las y)*

Transformar datos

Transformar datos con dplyr

El paquete dplyr contiene funciones muy útiles para la transformación de dataframes (tibbles). Todas las funciones tienen en común que su primer argumento es un dataframe y que devuelven un dataframe. Algunas de las funciones que vamos a ver:

- `filter()`: filtrar observaciones en base a valores
- `select()`: filtrar variables
- `mutate()`: crear o recodificar variables
- `group_by()`: define grupos de valores utilizar las otras funciones
- `summarise()`: colapsa valores según alguna fórmula (sumar, número de casos, media, etc.)

Filtrar

Una de las tareas más comunes en el análisis de datos es filtrar observaciones en base a condiciones. Existen muchas maneras de filtrar datos en R, la función `filter()` de dplyr es una de las más sencillas de utilizar. El primer argumento es el dataframe y el segundo la condición por la que queremos filtrar.

```
# Tenemos datos de muchos años:  
table(gapminder$year)
```

```
##  
## 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007  
##  142  142  142  142  142  142  142  142  142  142  142  142
```

```
# Filtremos para con los datos a partir de 2007  
gapminder_07 <- filter(gapminder, year == 2007)  
table(gapminder_07$year)
```

```
##  
## 2007  
##  142
```

Utilizando operadores lógicos podemos filtrar de formas más complejas:

```
# Todas los años menos 2007
gapminder_pre07 <- filter(gapminder, year != 2007)
table(gapminder_pre07$year)
```

```
##
## 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002
##   142   142   142   142   142   142   142   142   142   142   142
```

```
# Solo los siguientes años: 1952, 1992 y 2007
años_especificos <- c(1952, 1992, 2007)
gapminder_esp <- filter(gapminder, year %in% años_especificos)

table(gapminder_esp$year)
```

```
##
## 1952 1992 2007
##   142   142   142
```

Seleccionar variables

Con `select()` podemos seleccionar las variables (columnas) que queremos mantener en un dataframe. Podemos nombrarlas, seleccionar cuáles queremos eliminar y referirnos por su orden:

```
# Seleccionar un conjunto de variables (país, año, población)  
select(gapminder, country, year, pop)  
  
# Seleccionar todas las variables menos las especificadas  
select(gapminder, -continent)  
  
# Seleccionar un rango de variables según orden  
select(gapminder, country:lifeExp)  
select(gapminder, 1:3) # Orden numérico
```

Pipeline %>%

Cuando queremos realizar más de una operación a un dataframe podemos utilizar el pipeline. Como vimos, la mayoría de las funciones de dplyr que se aplican a un dataframe tienen como primer argumento el dataframe al que le queremos aplicar la función.

Con el pipeline especificamos el dataframe solamente una vez al principio, y luego todas las funciones que vamos utilizando no necesitan especificación. De esta forma nos enfocamos en la transformación y no en el objeto.

```
gapminder_07_america <- gapminder %>%  
  filter(year == 2007 & continent == "Americas") %>%  
  select(-continent)  
  
print(gapminder_07_america)
```

```
## # A tibble: 25 × 5  
##   country      year lifeExp      pop gdpPercap  
##   <fct>      <int>   <dbl>    <int>    <dbl>  
## 1 Argentina    2007    75.3  40301927  12779.  
## 2 Bolivia      2007    65.6   9119152   3822.  
## 3 Brazil       2007    72.4 190010647   9066.  
## 4 Canada       2007    80.7  33390141  36319.  
## 5 Chile        2007    78.6  16284741  13172.  
## 6 Colombia     2007    72.9  44227550   7007.  
## 7 Costa Rica   2007    78.8   4133884   9645.  
## 8 Cuba         2007    78.3  11416987   8948.  
## 9 Dominican Republic 2007    72.2   9319622   6025.  
## 10 Ecuador     2007    75.0  13755680   6873.  
## # ... with 15 more rows
```

Pipeline %>%

- Una de las ventajas del Tidyverse es la facilidad con la que se puede leer e interpretar el código. Un elemento fundamental para esto es el pipeline (%>%). Es muy útil para expresar una secuencia de muchas operaciones.
- Habíamos visto varias formas de realizar esto: sobrescribir el mismo objeto, con objetos intermedios o anidando funciones.
- El pipeline del paquete `magrittr` hace más fácil modificar operaciones puntuales dentro de conjunto de operaciones, hace que sea más fácil leer (evitando leer de adentro hacia afuera) entre otras ventajas.
- Es recomendable evitar usar el pipeline cuando queremos trabajar más de un objeto a la vez
- `x %>% f == f(x)`
- Se puede leer como un "y entonces"

Otras funciones de dplyr muy útiles

- `arrange()` ordenar los datos según una o más variables
- `rename()` cambiar el nombre de las variables de un dataframe
- `pull()` y `distinct()`: con `distinct()` es posible identificar los valores distintos de una variable y con `pull()` los podemos extraer como un vector
- `slice_min()` y `slice_max()`: filtrar n observaciones de mayor o menor valor según variable. En general, la familia de funciones `slice` permite filtrar observaciones en función de su posición.
- `count()` contar observaciones por grupo
- `relocate()` cambiar el orden de columnas

Crear y recodificar variables

Crear variables con mutate()

El paquete **dplyr** contiene la función `mutate()` para crear nuevas variables. `mutate()` crea variables al final del dataframe.

```
data_gapminder <- gapminder

# Variable de caracteres
data_gapminder <- mutate(data_gapminder, var1 = "Valor fijo")

# Variable numérica
data_gapminder <- mutate(data_gapminder, var2 = 7)
head(data_gapminder, 3)
```

```
## # A tibble: 3 × 8
##   country      continent year lifeExp      pop gdpPercap var1      var2
##   <fct>      <fct>    <int>   <dbl>    <int>    <dbl> <chr>    <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779. Valor fijo      7
## 2 Afghanistan Asia      1957    30.3  9240934    821. Valor fijo      7
## 3 Afghanistan Asia      1962    32.0 10267083    853. Valor fijo      7
```

```
## Podemos escribir lo mismo de distinta manera:
data_gapminder <- mutate(data_gapminder, var1 = "Valor fijo",
                          var2 = 7)
```

Recodificar variables con mutate()

Con `mutate()` también podemos realizar operaciones sobre variables ya existentes:

```
## Podemos recodificar usando variables y operadores aritméticos
# Calculemos el pbi total (pbi per capita * población)
d_gap <- mutate(gapminder, gdp = gdpPercap * pop)
head(d_gap, 3)
```

```
## # A tibble: 3 × 7
##   country      continent year lifeExp      pop gdpPercap      gdp
##   <fct>      <fct>    <int>   <dbl>    <int>    <dbl>    <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779. 6567086330.
## 2 Afghanistan Asia      1957    30.3  9240934    821. 7585448670.
## 3 Afghanistan Asia      1962    32.0 10267083    853. 8758855797.
```

```
# Podemos calcular el logaritmo
d_gap <- mutate(d_gap, gdp_log = log(gdp))
head(d_gap, 2)
```

```
## # A tibble: 2 × 8
##   country      continent year lifeExp      pop gdpPercap      gdp gdp_log
##   <fct>      <fct>    <int>   <dbl>    <int>    <dbl>    <dbl> <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779. 6567086330.    22.6
## 2 Afghanistan Asia      1957    30.3  9240934    821. 7585448670.    22.7
```

Transformaciones de tipo

Al igual que hacíamos con los vectores, podemos transformar el tipo de una variable

```
# Exploro tipo de variables
glimpse(d_gap)
```

```
## Rows: 1,704
## Columns: 3
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, ...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8...
```

```
# Variable continente a caracteres y año a factor
d_gap <- d_gap %>%
  mutate(continent = as.character(continent),
         year = as.factor(year))

glimpse(d_gap)
```

```
## Rows: 1,704
## Columns: 3
## $ continent <chr> "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asi...
## $ year      <fct> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, ...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40.8...
```

Recodificaciones condicionales

Recodificaciones condicionales

- Muchas veces transformar los datos implica recodificar una variable de forma condicional, esto es, asignar distintos valores en función de los valores de una o más variables.
- Para esto podemos utilizar las funciones; `ifelse()` (R Base), `mutate()`, `recode()` y `case_when()` (Tidyverse)
- Nosotros veremos recodificaciones condicionales con `case_when()` y `mutate()`

Recodificación condicional con `case_when` y `mutate`

Podemos crear variables condicionales con `case_when()` del paquete `dplyr`. Esencialmente, con `ifelse()` (R Base) podemos lograr lo mismo que con `case_when()` (Tidyverse). `case_when()` puede resultar más sencilla de utilizar al no haber necesidad de anidar la función cuando establecemos múltiples condiciones.

Cuando trabajamos con dataframes `case_when()` se utiliza dentro de `mutate()`. `case_when()` testea condiciones en orden (esto es importante cuando pasamos condiciones no excluyentes). `case_when()` lista condiciones para las que asigna un valor en caso de que sean verdaderas, y permite pasar múltiples condiciones. `TRUE` refiere a las condiciones no listadas. La estructura de `case_when()` es:

```
data %>%  
  mutate(var_nueva = case_when(var_original == "Valor 1" ~ "Valor A",  
                                var_original == "Valor 2" ~ "Valor B",  
                                TRUE ~ "Otros"))
```

Recodificación condicional con case_when y mutate

```
d_gap <- gapminder  
  
# Creemos una variable que indique si el país es Uruguay o no  
d_gap <- d_gap %>%  
  mutate(uruono = case_when(  
    country == "Uruguay" ~ "Si",  
    TRUE ~ "No")  
  )  
  
table(d_gap$uruono)
```

```
##  
##      No      Si  
## 1692     12
```

Recodificación condicional con case_when y mutate

Podemos establecer varias condiciones fácilmente:

```
d_gap <- gapminder  
d_gap <- d_gap %>%  
  mutate(mercosur = case_when(country == "Uruguay" ~ 1,  
                               country == "Argentina" ~ 1,  
                               country == "Paraguay" ~ 1,  
                               country == "Brazil" ~ 1,  
                               TRUE ~ 0))  
  
table(d_gap$mercosur)
```

```
##  
##      0      1  
## 1656    48
```


Recodificación condicional con case_when y mutate

También podríamos usar operadores para simplificar esto:

```
d_gap <- d_gap %>%
  mutate(mercosur = case_when(
    country %in% c("Argentina", "Paraguay", "Brazil", "Uruguay") ~ 1,
    TRUE ~ 0)
)

d_gap <- d_gap %>%
  mutate(mercosur2 = case_when(
    country == "Argentina" | country == "Paraguay" |
    country == "Brazil" | country == "Uruguay" ~ 1,
    TRUE ~ 0)
)

identical(d_gap$mercosur, d_gap$mercosur2)
```

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

Recodificación condicional con case_when y mutate

Recodificación con valores numéricos. Supongamos que queremos crear una nueva variable pob_rec, que clasifica a los países en población grande (más de 20 millones), mediana (entre 5 y 20) o pequeña (menos de 5)

```
d_gap <- d_gap %>%  
  mutate(pob_rec = case_when(  
    pop >= 20000000 ~ "Grande",  
    pop >= 5000000 & pop < 20000000 ~ "Mediana",  
    pop < 5000000 ~ "Pequeña",  
    TRUE ~ "Error")  
  )  
  
table(d_gap$pob_rec)
```

```
##  
##  Grande Mediana Pequeña  
##    419      580      705
```

Recodificación condicional con `case_when` y `mutate`

`case_when()` sirve también para recodificar una variable con condiciones basadas en múltiples variables.

Supongamos que queremos una variable que indique los países-año con expectativa de vida mayor a 75 o pbi per cápita mayor a 20.000

```
d_gap <- d_gap %>%  
  mutate(var1 = case_when(gdpPercap > 20000 ~ 1,  
                           lifeExp > 75 ~ 1,  
                           TRUE ~ 0))  
  
table(d_gap$var1)
```

```
##  
##      0      1  
## 1493  211
```

Ejercicio

(1) Abran un script nuevo y tomen el dataframe de gapminder (importarlo de la carpeta data o descargarlo desde el paquete gapminder como debajo, y filtrar para la base para quedarnos solamente con las observaciones América y Oceanía

```
library(gapminder)  
gapminder <- gapminder
```

(2) En ese mismo dataframe con solamente las observaciones de América y Oceanía seleccionar solamente tres variables: country y lifeExp

(3) En ese mismo dataframe, crear una nueva variable que divida la expectativa de vida en 3 categorías: "Alta" (mayor o igual a 75 años); "Media" (mayor o igual a 65 y menor a 75 y "Baja" (menor a 65 años)

(4) Asegurarse de hacer los primeros 3 pasos en un mismo pipeline

Tablas cruzadas

Resumir datos: tablas simples

Con `count()` podemos hacer lo mismo que con `table()`, pero en el contexto del tidyverse. Con un simple `mutate()` podemos transformar la frecuencia simple en proporción o porcentaje

```
# Frecuencia simple
d_gap %>%
  count(continent)
```

```
## # A tibble: 5 × 2
##   continent     n
##   <fct>      <int>
## 1 Africa      624
## 2 Americas    300
## 3 Asia        396
## 4 Europe      360
## 5 Oceania      24
```

```
# Frecuencia proporción
d_gap %>%
  count(continent) %>%
  mutate(per = n/sum(n)*100) %>% # Porcentaje
  mutate(prop = n/sum(n)) # Proporción
```

```
## # A tibble: 5 × 4
##   continent     n    per    prop
##   <fct>      <int> <dbl> <dbl>
## 1 Africa      624  36.6  0.366
## 2 Americas    300  17.6  0.176
## 3 Asia        396  23.2  0.232
## 4 Europe      360  21.1  0.211
## 5 Oceania      24   1.41  0.0141
```

Resumir datos: tablas cruzadas

Cuando usamos `count()` con dos variables vemos que la salida está en formato largo. Eso nos puede servir para graficar si usamos `ggplot2` y para otros paquetes, pero no es como generalmente queremos leer una tabla para interpretarla.

```
# Tabla cruzada en formato largo
d_gap %>%
  count(continent, pob_rec)
```

```
## # A tibble: 15 × 3
##   continent pob_rec     n
##   <fct>      <chr> <int>
## 1 Africa    Grande    86
## 2 Africa    Mediana  208
## 3 Africa    Pequeña  330
## 4 Americas Grande    72
## 5 Americas Mediana   98
## 6 Americas Pequeña  130
## 7 Asia      Grande   168
## 8 Asia      Mediana  111
## 9 Asia      Pequeña  117
## 10 Europe   Grande    92
## 11 Europe   Mediana  152
## 12 Europe   Pequeña  116
## 13 Oceania  Grande     1
## 14 Oceania  Mediana   11
## 15 Oceania  Pequeña   12
```

Resumir datos: tablas cruzadas

Para pasar a formato ancho podemos utilizar la función `spread()`. El primer argumento dentro de `spread()` es la variable que queremos que pase a ser columnas y el segundo argumento la variable que contiene los valores.

```
# Tabla cruzada en formato ancho
d_gap %>%
  count(continent, pob_rec) %>%
  spread(pob_rec, n)
```

```
## # A tibble: 5 × 4
##   continent Grande Mediana Pequeña
##   <fct>      <int>   <int>   <int>
## 1 Africa         86     208     330
## 2 Americas        72      98     130
## 3 Asia          168     111     117
## 4 Europe          92     152     116
## 5 Oceania         1      11      12
```


Resumir datos: tablas cruzadas

Cuando analizamos tablas cruzadas, muchas veces queremos ver porcentajes en lugar de frecuencias. Para ello podemos transformar la frecuencia en porcentaje.

```
# Porcentaje total
d_gap %>%
  count(continent, pob_rec) %>%
  mutate(n = n/sum(n)*100) %>%
  spread(pob_rec, n)
```

```
## # A tibble: 5 × 4
##   continent Grande Mediana Pequeña
##   <fct>      <dbl>   <dbl>   <dbl>
## 1 Africa     5.05    12.2    19.4
## 2 Americas  4.23     5.75    7.63
## 3 Asia       9.86     6.51    6.87
## 4 Europe     5.40     8.92    6.81
## 5 Oceania    0.0587   0.646   0.704
```

Resumir datos: tablas cruzada

Cuando trabajamos con porcentajes en tablas cruzadas, tenemos que considerar el total del %. En el ejemplo anterior el porcentaje era sobre el total, muchas veces queremos calcularlo sobre cada fila o cada columna

```
# % de países en cada tamaño de población por continente
d_gap %>%
  count(continent, pob_rec) %>%
  mutate(n = n/sum(n)*100, .by = continent) %>%
  spread(pob_rec, n)
```

```
## # A tibble: 5 × 4
##   continent Grande Mediana Pequeña
##   <fct>      <dbl>   <dbl>   <dbl>
## 1 Africa      13.8     33.3    52.9
## 2 Americas    24       32.7    43.3
## 3 Asia       42.4     28.0    29.5
## 4 Europe     25.6     42.2    32.2
## 5 Oceania     4.17    45.8    50
```

```
# % de continente por tamaño de población
d_gap %>%
  count(continent, pob_rec) %>%
  mutate(n = n/sum(n)*100, .by = pob_rec) %>%
  spread(pob_rec, n)
```

```
## # A tibble: 5 × 4
##   continent Grande Mediana Pequeña
##   <fct>      <dbl>   <dbl>   <dbl>
## 1 Africa     20.5     35.9    46.8
## 2 Americas   17.2     16.9    18.4
## 3 Asia       40.1     19.1    16.6
## 4 Europe     22.0     26.2    16.5
## 5 Oceania     0.239    1.90    1.70
```

Tablas con estadísticas descriptivas

Resumir datos

Resumir datos con estadísticos descriptivos es una de las partes fundamentales del análisis de datos. Para ello utilizaremos la función `summarise()` o `summarize()`, muchas veces en conjunto con `group_by()`.

Escencialmente `summarise()` resume un dataframe en una fila según una estadística especificada. Por ejemplo, calculando la media de una variable

```
gapminder %>%  
  summarise(media = mean(lifeExp, na.rm=T))
```

```
## # A tibble: 1 × 1  
##   media  
##   <dbl>  
## 1  59.5
```

```
# Por ahora no hay mucha diferencia con  
mean(gapminder$lifeExp, na.rm = TRUE)
```

```
## [1] 59.47444
```

Resumir datos

Hasta ahora `summarise()` no nos es de gran utilidad, la utilidad de `summarise()` es su uso conjunto con `group_by()`, para estimar diferentes estadísticas según grupos específicos.

Cuando utilizamos `group_by()` en un pipeline cambiamos la unidad de análisis desde todo el dataframe a niveles de una variable. Retomando el ejemplo, podemos ver el promedio de expectativa de vida según año:

```
gapminder %>%  
  group_by(year) %>%  
  summarise(media = mean(lifeExp, na.rm = T))
```

```
## # A tibble: 12 × 2  
##   year media  
##   <int> <dbl>  
## 1  1952  49.1  
## 2  1957  51.5  
## 3  1962  53.6  
## 4  1967  55.7  
## 5  1972  57.6  
## 6  1977  59.6  
## 7  1982  61.5  
## 8  1987  63.2  
## 9  1992  64.2  
## 10 1997  65.0  
## 11 2002  65.7  
## 12 2007  67.0
```

Resumir datos

Algunas de las operaciones más utilizadas para resumir datos:

- `mean()`: media
- `median()`: mediana
- `sd()`: desvío estandar
- `sum()`: suma
- `n()`: número de observaciones
- `n_distinct()`: número de valores únicos
- `min()` y `max()`: mínimo y máximo
- `first()`: primer valor

Resumir datos

Podemos utilizar más de una variable dentro de `group_by()`. Por ejemplo, calculemos la media de expectativa de vida por año comparando America y Europa para 1997, 2002 y 2007:

```
resumen_1 <- gapminder %>%  
  filter(continent %in% c("Americas", "Europe")) %>%  
  filter(year >= 1997) %>%  
  group_by(continent, year) %>%  
  summarise(media = mean(lifeExp, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'continent'. You can override using the  
## `.groups` argument.
```

```
resumen_1
```

```
## # A tibble: 6 × 3  
## # Groups:   continent [2]  
##   continent year media  
##   <fct>     <int> <dbl>  
## 1 Americas  1997  71.2  
## 2 Americas  2002  72.4  
## 3 Americas  2007  73.6  
## 4 Europe    1997  75.5  
## 5 Europe    2002  76.7  
## 6 Europe    2007  77.6
```

Resumir datos

Una de las grandes ventajas de `summarise()` es que podemos resumir muy fácilmente varias estadísticas en un solo dataframe.

```
resumen_2 <- gapminder %>%  
  filter(continent %in% c("Americas", "Europe")) %>%  
  filter(year == 2007) %>%  
  group_by(continent) %>%  
  summarise(media = mean(gdpPercap),  
            desvio = sd(gdpPercap),  
            suma = sum(gdpPercap),  
            max = max(gdpPercap),  
            min = min(gdpPercap),  
            paises = n())
```

```
resumen_2
```

```
## # A tibble: 2 × 7  
##   continent  media desvio   suma   max   min paises  
##   <fct>      <dbl> <dbl>   <dbl> <dbl> <dbl> <int>  
## 1 Americas  11003.  9713. 275076. 42952. 1202.    25  
## 2 Europe    25054. 11800. 751634. 49357. 5937.    30
```


Exportar tablas

Las tablas y resúmenes de salida de tidyverse son dataframes. Como tales, los podemos exportar fácilmente a otros tipos de archivos. Veamos como exportar una tabla con la función `write_xlsx()` del paquete `writexl`. Ya habíamos creado la tabla "resumen_2", entonces con una sola línea de código la podemos guardar. La función `write_xlsx()` tiene dos argumentos principales, el dataframe a ser guardado y el directorio donde ser guardado junto con el nombre del archivo que vamos a crear. Tener en cuenta que si el directorio y nombre de un archivo coincide con el de otro archivo ya existente en la computadora, lo sobreescribirá.

```
library(writexl)
writexl::write_xlsx(resumen_2, "resultados/mi_primera_tabla.xlsx")
```

Ejercicio

- (1) En el mismo script que el ejercicio anterior, utilizar count() para crear una tabla con el % de casos en cada categoría de la recodificación de expectativa de vida*
- (2) Ahora crear una tabla cruzada entre country y la variable recodificada de expectativa de vida (solo con el número de casos, no %)*
- (3) Hay algún país con observaciones en las tres categorías de nuestra variable?*
- (4) Recrear la tabla debajo (tabla cruzada de continente y expectativa de vida recodificada, con % por fila):*
- (5) Por último crear una tabla resumen con la media de la expectativa de vida y la cantidad de casos según continente*