Análisis exploratorio de datos con `tidyverse`

Parte 1: Fundamentos de análisis exploratorio en `R`

Rodrigo Zepeda-Tello

2022-10-11

Mostramos cómo funciona `dplyr` para filtrar (`filter`), seleccionar (`select`), mutar (`mutate`), agrupar (`group_by`), y resumir (`summarise`) bases de datos en `R`

> **Note**
>
> Los datos están disponibles en el Github y en Dropbox

> **Warning**
>
> Si aún no cuentas con una instalación de `tidyverse` dentro de `R` corre la siguiente instrucción:
>
> ```
> install.packages("tidyverse")
> ```

El flujo de trabajo de trabajo con datos

En general el flujo de trabajo de un análisis de datos se divide en tres componentes principales:

1. **Preparación de los datos** lo que incluye la recolección (no discutida aquí), la **importación** de los datos y la **limpieza** inicial (por ejemplo el poner nombres a

las columnas u homologar mayúsculas y minúsculas) para generar una base de trabajo.

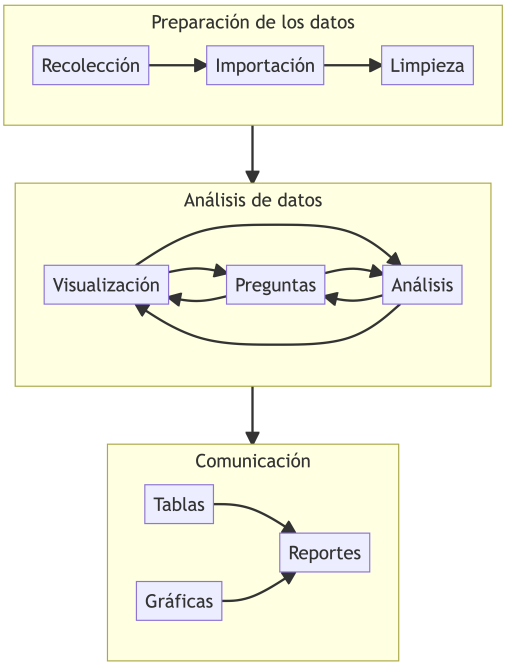
> ⚠ Warning
>
> Una vez recolectados la recomendación es que las bases de datos **no se toquen** una vez se tiene la información. Entre menos modifiquemos la base de datos hay menor probabilidad de cometer errores y accidentes que resulten en **pérdida de información**.

2. **Análisis de datos** incluye tres pasos que fluyen en cualquier orden:

a. **Formulación de preguntas** ¿qué me gustaría saber de mis datos?_
b. **Visualización de los datos** ¿puede una gráfica ayudarme a responder mi pregunta u orientarme hacia qué analizar?
c. **Análisis de los datos:** ¿qué resumen de la información (por ejemplo una tabla o un promedio) resulta eficaz para presentar lo que me interesa?

3. Una vez se tienen los datos analizados continuamos a la parte de **comunicación** donde buscamos **generar tablas, gráficas y reportes** (entre otros) que comuniquen nuestros datos al público.

El siguiente diagrama (traducido de aquí) intenta resumir el flujo de trabajo:



Armado de un proyecto

En `RStudio` para un proyecto de análisis de datos la recomendación es crear un `Project`. Ésta es una carpeta especial en la cual se almacena todo el código, las bases de datos e incluso los registros de las versiones de los paquetes que estás usando Con pasarle a otra persona tu proyecto (`Project`) ésta podrá reproducir absolutamente todo sin preocuparse por tener que acomodar las rutas a los archivos o instalar los paquetes que tú usaste.

Para armar un proyecto puedes ir a `File > New Project`. Esto abrirá una ventana como sigue:

Existen diferentes opciones: empezar a crear un proyecto desde cero (`New Directory`) o bien trabajar con algún folder que ya tenemos `Existing directory`. El control de versiones (`Version Control`) es una herramienta más avanzada de programación y no la discutiremos por ahora.

Elige la opción de `New Directory` y en la ventana que sigue elige `New R Proyect`. Notarás que esto sirve para muchas cosas

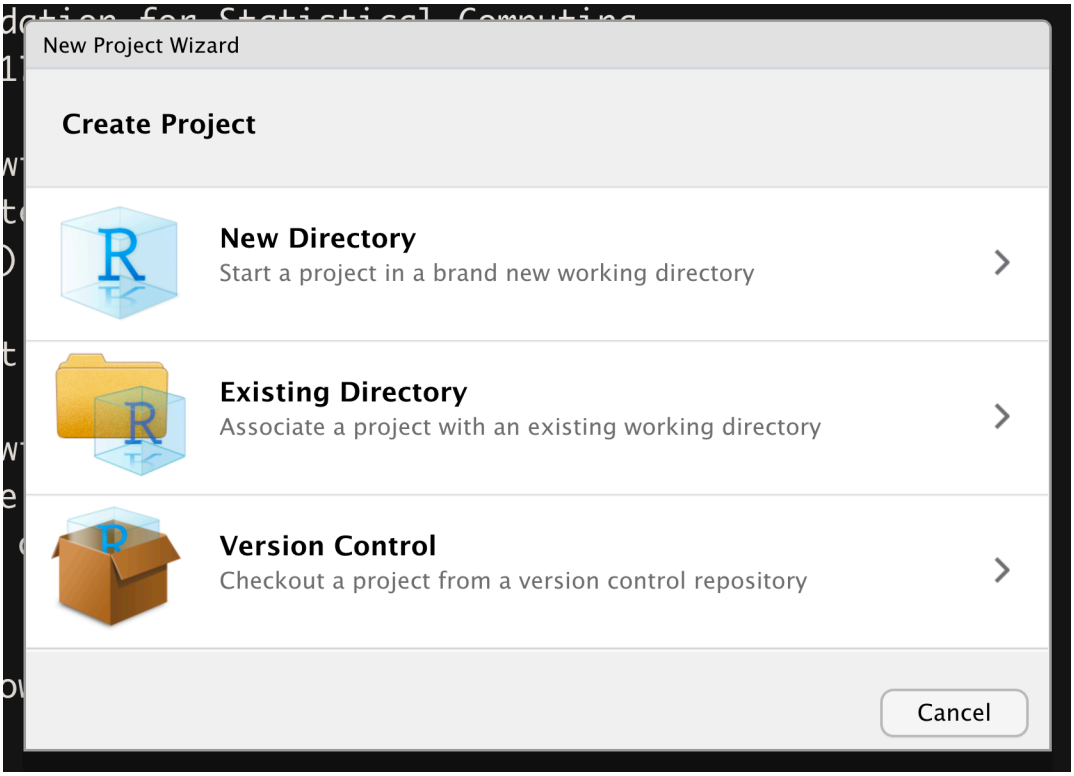


Figure 1: Opciones de proyecto incluyen nuevo directorio o existente así como control de versiones

entre ellas armar presentaciones y páginas web. Por ahora trabajaremos sólo con código normal.

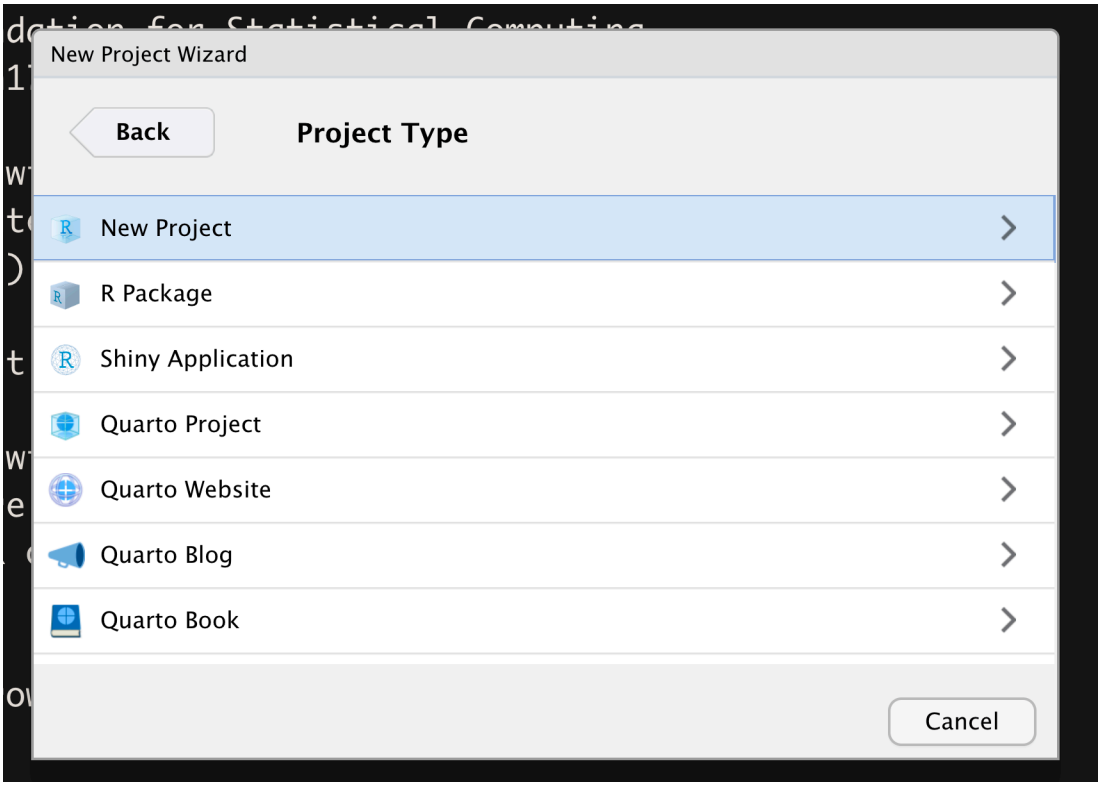


Figure 2: Las opciones de tipo de proyecto incluyen blogs, páginas web y libros. Por ahora comenzamos con `New Proyect` para hacer un proyecto nuevo sin opciones específicas de los otros tipos.

Finalmente en la tercer ventana elegimos el nombre del proyecto, el folder dentro de nuestros documentos donde habrá de guardarse y activamos las opciones:

- `renv` sirve para controlar los paquetes de `R` usados. Si eliges la opción `renv`, cualquier otra persona que use el proyecto se le instalarán los paquetes que tú usaste y las versiones específicas (digamos tu `ggplot2` es del 2021 entonces esa persona tendrá ese `ggplot2` cuando abra el proyecto).
- `git` sirve para guardar un historial de tu código. Expli-

caremos más adelante su funcionamiento principal pero la idea es que se guarde cada cambio que haces en el código (pues `ctrl Z` es limitado).

- `Open in new session` cierra la ventana actual de `RStudio` y te abre el proyecto en un nuevo lienzo en blanco para que comiences tu trabajo.

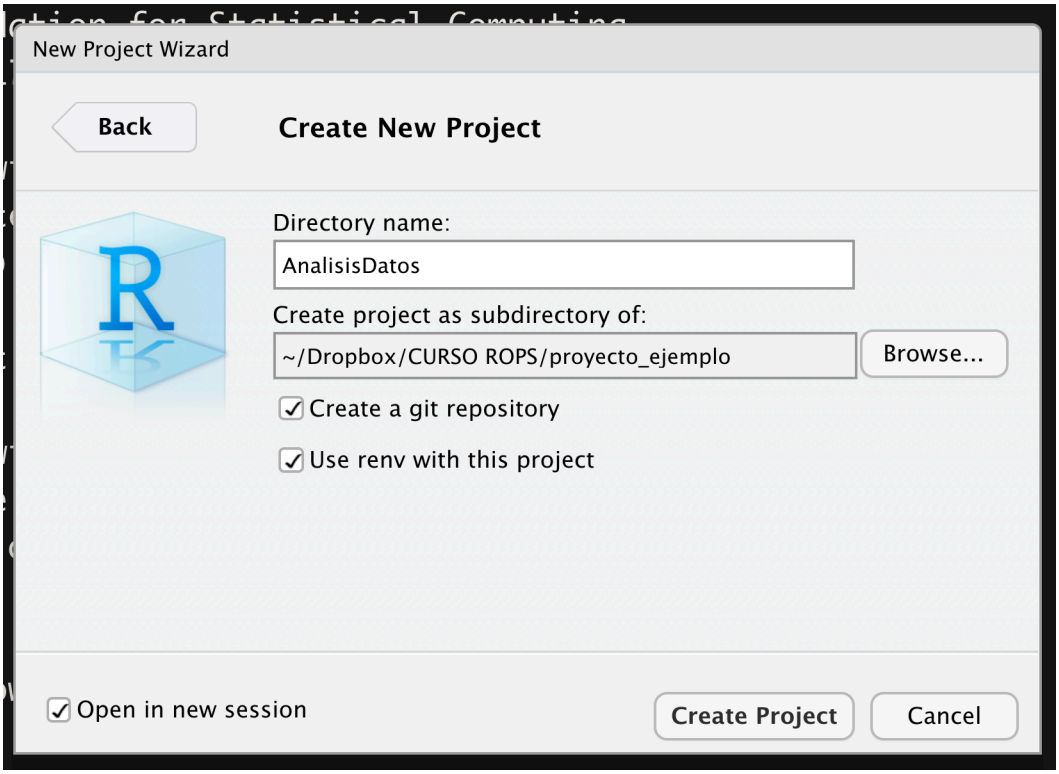


Figure 3: Un nuevo proyecto con nombre `AnalisisDatos`, en mi carpeta de `Dropbox` con las opciones de `git` y `renv` habilitadas. Se solicitó a `R` además abrir en una sesión nueva.

Una vez iniciado el proyecto nos aparecerá un mensaje similar a este en una nueva consola de 'R":

```
The version of R recorded in the lockfile will be updated:
- R      [*] -> [4.2.1]
```

```
* Lockfile written to '~/AnalisisDatos/renv.lock'.
```

¡Podemos empezar a usarlo!

> **Warning**
>
> Todas las bases de datos que utilicemos dentro de estas notas supondremos están almacenadas dentro del proyecto `AnalisisDatos` el cual está en tus documentos donde sea hayas elegido.
>
> Por favor copia todos los datos que uses **dentro del proyecto** para que así, a cualquier persona que le envíes el proyecto pueda usarlo.

Lectura de bases de datos

`R` sirve para abrir cualquier tipo de dato. En particular es posible leer bases de datos desde `Excel`, `csv`, `txt`, `Stata`. También (aunque no lo veremos) puede leer imágenes, videos, datos provenientes de documentos `pdf`, páginas web, mapas, etc. Al día de hoy no he encontrado un sólo tipo de dato que no pueda leer `R`.

Las bases de datos si ya están recolectadas **no se tocan**. Lo que haremos en `R` será leerlas, copiarlas en una base nueva y trabajar sobre la base nueva sin modificar la original. De esta forma, cualquier error que cometamos ¡y seguro cometeremos muchos! no afectará la original. Esto permite además tener un flujo de trabajo seguro (en el sentido de que estamos seguros que no borraremos nada).

Lectura de datos desde un archivo delimitado (`csv` ó `txt`).

Los documentos de datos, si son (relativamente) medianas, pueden ser compartidos en archivos de texto que se pueden abrir con el bloc de notas. Por ejemplo, el documento `casos_covid_agosto_2022.csv` se ve como sigue:

```
"FECHA_SINTOMAS","n"
"2020-01-01","287"
"2020-01-02","233"
"2020-01-03","247"
"2020-01-04","245"
```

por otro lado el documento `embarazo_adolescente.txt` es un archivo delimitado por tabs (a veces llamados `tsv`) donde cada columna ocurre después de un `tab` (espacio largo):

```
Location    PovPct  Brth15to17  Brth18to19  ViolCrime   TeenBrth
Alabama 20.1    31.5    88.7    11.2    54.5
Alaska  7.1 18.9    73.7    9.1 39.5
Arizona 16.1    35  102.5   10.4    61.2
Arkansas    14.9    31.6    101.7   10.4    59.9
```

finalmente, `covid_sinave_bc_bcs.txt` es un archivo donde las columnas son separadas por `|`:

```
SECTOR|ENTIDAD_UM|SEXO|ENTIDAD_NAC|ENTIDAD_RES|TIPO_PACIENTE|FECHA_INGRESO|FECHA_SINTOMAS|FECHA
4|03|2|12|03|1|2021-07-05T00:00:00Z|2021-07-04T00:00:00Z|NA|22|2|2|2|97|1|1|3|97
4|03|2|03|03|1|2021-07-12T00:00:00Z|2021-07-11T00:00:00Z|NA|52|2|1|2|97|1|2|7|97
4|03|2|03|03|1|2021-07-22T00:00:00Z|2021-07-19T00:00:00Z|NA|19|2|2|2|97|1|2|7|97
12|03|1|03|03|1|2021-07-15T00:00:00Z|2021-07-13T00:00:00Z|NA|10|2|2|2|97|1|1|3|97
```

Todos estos archivos son archivos de texto simple delimitados y se pueden leer con la librería `readr`.

Lectura simple

Para leer los archivos basta con `File > Import dataset` o bien con código:

```
casos_covid <- read_csv("casos_covid_agosto_2022.csv")
```

Para los archivos delimitados por espacios podemos usar `read_tsv`:

```
adolescentes <- read_tsv("embarazo_adolescente.txt")
```

Finalmente archivos con delimitadores que no son comas ni tabs (como los |) puedes usar `read_delim` y especificar el separador `delim`:

```
covid <- read_delim("covid_sinave_bc_bcs.txt", delim = "|")
```

Complicaciones

La base de datos `zapopan.csv` contiene las estimaciones de la población del municipio homónimo por parte del INEGI. Sin embargo, al momento de leerla pasan dos cosas:

1. Los acentos en algunos equipos no se leen bien
2. La primera fila no representa nada pues sólo dice **Datos**.

Aquí una muestra de cómo viene el archivo. Nota que en mi equipo los acentos y las ñ aparecen como el símbolo `?`:

Datos Poblacionales„„„„„„, Unidad,Nombre Unidad,Zona Coplademun,Habitantes,Hombres,Mujeres,Poblacin de 0 a 14 Aos,Poblacin de 15 a 64 Aos,Poblacin Mayor de 18 Aos,Viviendas Habitadas,Promedio de Ocupantes por Vivienda,Clave INEGI 1,12 de diciembre,7,“1,782”,“1,005”,953,759,“1,143”,“1,012”,394,5.221,02,27 de Septiembre,1B,“1,730”,824,906,469,“1,122”,“1,171”,478,4.141666667,001-F 3,“Abetos, Los”,1B,367,179,188,121,240,231,101,3.63,002-L

Para leer esta base es necesario saltarnos la primera fila pues **Datos Poblacionales** no es un nombre de columna. Para ello usamos `skip = 1` indicándole que se salte (`skip`) una fila (`= 1`). Por otro lado para los acentos usamos el encoding de `WINDOWS-1252`. Los más comunes para acentos en México son `UTF-8` y `WINDOWS-1252`. Cuando no leemos bien los acentos vale la pena probar ambos para hallar el correcto:

```
zapopan <- read_csv("zapopan.csv",
                    locale = locale(encoding = "WINDOWS-1252"),
                    skip = 1)
```

Lectura de datos desde un `Excel`

Lectura simple

Para leer datos desde un Excel podemos usar la `Import From > Excel` o bien `read_excel` dentro de la librería `readxl`:

```
library(readxl)
```

dado un archivo como `persona_cigarros.xlsx` podemos leerlo directamente:

```
cigarros <- read_excel("persona_cigarros.xlsx")
```

Complicaciones

Algunos archivos como `Poblacion_01.xlsx` contienen filas tanto al inicio como al final que funcionan como descriptores del archivo. Al momento de leerlo debemos ignorar estas partes:

Para ello en `read_excel` podemos especificar el rango de las celdas que nos interesa exportar indicando las dos esquinas del rectángulo de celdas:

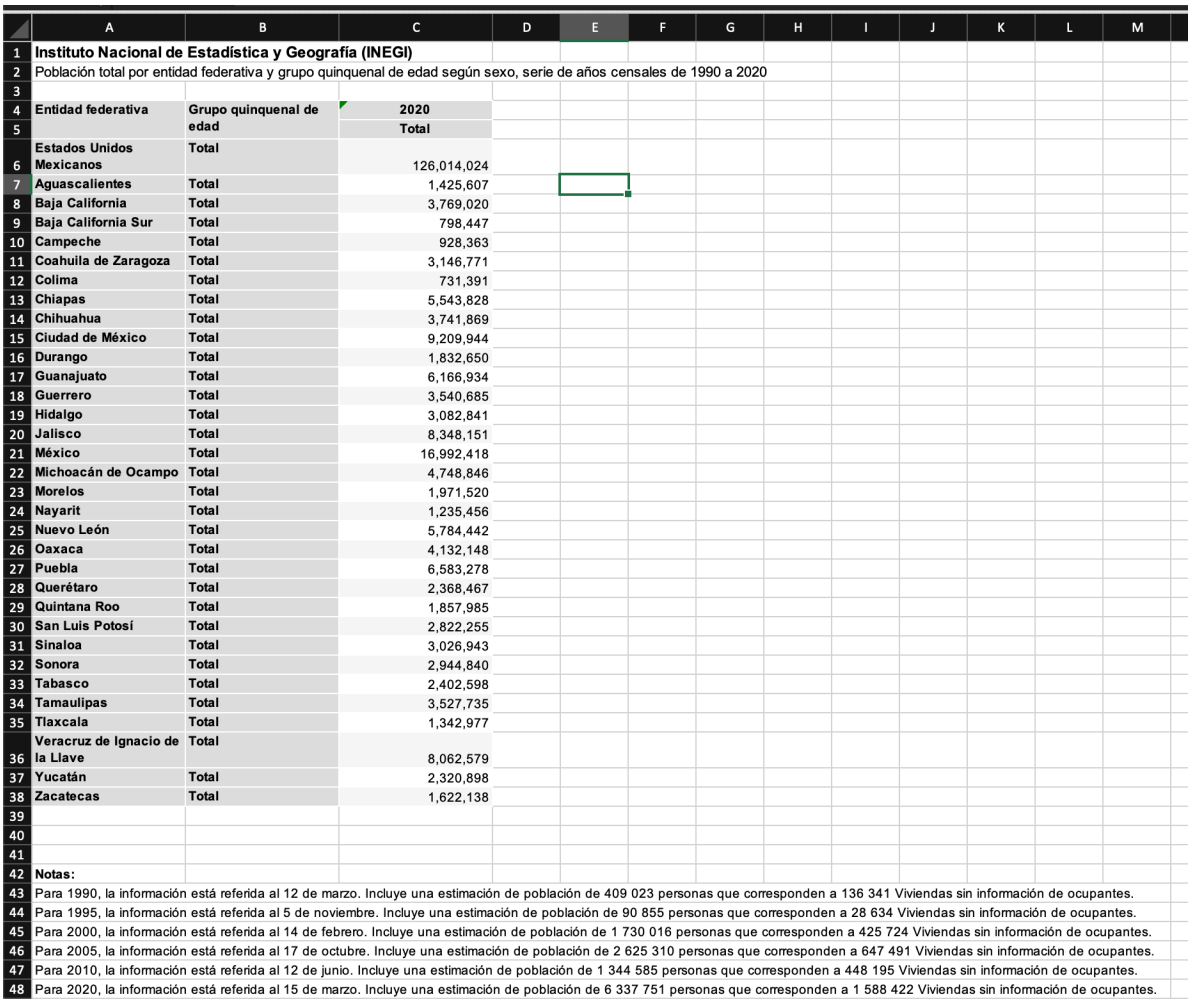
```
poblacion <- read_excel("Poblacion_01.xlsx", range = "A5:C38")
```

Ejercicios

1. Lee las bases de datos `ile-2019-2021.csv`, `IME_2020.xls` y `niños.txt`.

Análisis de bases de datos

Vamos a leer la base de datos `conjunto_de_datos_defunciones_generales_2017.csv` la cual contiene el registro de mortalidad en México para el año 2017 del INEGI:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Instituto Nacional de Estadística y Geografía (INEGI)**												
2	Población total por entidad federativa y grupo quinquenal de edad según sexo, serie de años censales de 1990 a 2020												
3													
4	**Entidad federativa**	**Grupo quinquenal de edad**	**2020**										
5			**Total**										
6	**Estados Unidos Mexicanos**	**Total**	126,014,024										
7	**Aguascalientes**	**Total**	1,425,607										
8	**Baja California**	**Total**	3,769,020										
9	**Baja California Sur**	**Total**	798,447										
10	**Campeche**	**Total**	928,363										
11	**Coahuila de Zaragoza**	**Total**	3,146,771										
12	**Colima**	**Total**	731,391										
13	**Chiapas**	**Total**	5,543,828										
14	**Chihuahua**	**Total**	3,741,869										
15	**Ciudad de México**	**Total**	9,209,944										
16	**Durango**	**Total**	1,832,650										
17	**Guanajuato**	**Total**	6,166,934										
18	**Guerrero**	**Total**	3,540,685										
19	**Hidalgo**	**Total**	3,082,841										
20	**Jalisco**	**Total**	8,348,151										
21	**México**	**Total**	16,992,418										
22	**Michoacán de Ocampo**	**Total**	4,748,846										
23	**Morelos**	**Total**	1,971,520										
24	**Nayarit**	**Total**	1,235,456										
25	**Nuevo León**	**Total**	5,784,442										
26	**Oaxaca**	**Total**	4,132,148										
27	**Puebla**	**Total**	6,583,278										
28	**Querétaro**	**Total**	2,368,467										
29	**Quintana Roo**	**Total**	1,857,985										
30	**San Luis Potosí**	**Total**	2,822,255										
31	**Sinaloa**	**Total**	3,026,943										
32	**Sonora**	**Total**	2,944,840										
33	**Tabasco**	**Total**	2,402,598										
34	**Tamaulipas**	**Total**	3,527,735										
35	**Tlaxcala**	**Total**	1,342,977										
36	**Veracruz de Ignacio de la Llave**	**Total**	8,062,579										
37	**Yucatán**	**Total**	2,320,898										
38	**Zacatecas**	**Total**	1,622,138										
39													
40													
41													
42	**Notas:**												
43	Para 1990, la información está referida al 12 de marzo. Incluye una estimación de población de 409 023 personas que corresponden a 136 341 Viviendas sin información de ocupantes.												
44	Para 1995, la información está referida al 5 de noviembre. Incluye una estimación de población de 90 855 personas que corresponden a 28 634 Viviendas sin información de ocupantes.												
45	Para 2000, la información está referida al 14 de febrero. Incluye una estimación de población de 1 730 016 personas que corresponden a 425 724 Viviendas sin información de ocupantes.												
46	Para 2005, la información está referida al 17 de octubre. Incluye una estimación de población de 2 625 310 personas que corresponden a 647 491 Viviendas sin información de ocupantes.												
47	Para 2010, la información está referida al 12 de junio. Incluye una estimación de población de 1 344 585 personas que corresponden a 448 195 Viviendas sin información de ocupantes.												
48	Para 2020, la información está referida al 15 de marzo. Incluye una estimación de población de 6 337 751 personas que corresponden a 1 588 422 Viviendas sin información de ocupantes.												

Figure 4: Opciones de proyecto incluyen nuevo directorio o existente así como control de versiones

ent_regis	mun_regis	ent_resid	mun_resid	tloc_resid	loc_resid	ent_ocurr	mun_ocurr	tloc_ocur
01	001	01	001	15	0001	01	001	1
01	009	01	009	1	0016	01	009	
01	001	01	001	15	0001	01	001	1
01	006	01	006	8	0001	01	006	
01	001	01	001	15	0001	01	001	1
01	001	01	001	15	0001	01	001	1

```
mortalidad_2017 <- read_csv("conjunto_de_datos_defunciones_generales_2017.csv")
```

Aquí sólo muestro las filas de la base:

La función `glimpse` nos permite darnos una idea de la composición de nuestros datos:

```
mortalidad_2017 %>% glimpse()
```

```
Rows: 703,047
Columns: 59
$ ent_regis  <chr> "01", "01", "01", "01", "01", "01", "01", "01", "01", "01",~
$ mun_regis  <chr> "001", "009", "001", "006", "001", "001", "001", "001", "00~
$ ent_resid  <chr> "01", "01", "01", "01", "01", "01", "01", "01", "01", "01",~
$ mun_resid  <chr> "001", "009", "001", "006", "001", "001", "001", "001", "00~
$ tloc_resid <dbl> 15, 1, 15, 8, 15, 15, 15, 15, 15, 5, 15, 1, 15, 15, 1, 15, ~
$ loc_resid  <chr> "0001", "0016", "0001", "0001", "0001", "0001", "0001", "00~
$ ent_ocurr  <chr> "01", "01", "01", "01", "01", "01", "01", "01", "01", "01",~
$ mun_ocurr  <chr> "001", "009", "001", "006", "001", "001", "001", "001", "00~
$ tloc_ocurr <dbl> 15, 1, 15, 8, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 1~
$ loc_ocurr  <chr> "0001", "0016", "0001", "0001", "0001", "0001", "0001", "00~
$ causa_def  <chr> "I679", "I64X", "E112", "X590", "I251", "N009", "E129", "M6~
$ lista_mex  <chr> "30Z", "30D", "20D", "51Z", "28Z", "38A", "20D", "37H", "09~
$ sexo       <dbl> 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1,~
$ edad       <dbl> 4068, 4090, 4088, 4096, 4051, 4078, 4070, 4079, 4061, 4069,~
$ dia_ocurr  <dbl> 26, 10, 12, 27, 1, 1, 19, 20, 7, 18, 18, 20, 1, 20, 28, 20,~
$ mes_ocurr  <dbl> 7, 11, 2, 12, 3, 7, 12, 10, 1, 1, 12, 1, 1, 1, 11, 1, 12, 1~
$ anio_ocur  <dbl> 1994, 2011, 2011, 2016, 2016, 2016, 2016, 2016, 2017, 2017,~
$ dia_regis  <dbl> 19, 30, 99, 10, 13, 10, 13, 4, 11, 25, 3, 24, 9, 25, 2, 24,~
$ mes_regis  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ anio_regis <dbl> 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017,~
```

```
$ dia_nacim  <dbl> 99, 31, 19, 20, 11, 8, 15, 22, 15, 1, 26, 18, 13, 28, 16, 4~
$ mes_nacim  <dbl> 99, 10, 11, 1, 11, 5, 3, 6, 5, 2, 2, 5, 9, 9, 6, 7, 6, 7, 1~
$ anio_nacim <dbl> 1926, 1921, 1922, 1920, 1964, 1938, 1946, 1937, 1955, 1947,~
$ ocupacion  <dbl> 11, 11, 4, 11, 9, 11, 2, 11, 2, 11, 2, 6, 11, 7, 6, 9, 2, 4~
$ escolarida <dbl> 3, 3, 3, 3, 1, 3, 10, 3, 10, 3, 9, 3, 4, 4, 1, 3, 9, 4, 1, ~
$ edo_civil  <dbl> 3, 5, 3, 3, 2, 5, 5, 5, 5, 5, 5, 5, 3, 5, 5, 2, 5, 5, 9, 5,~
$ presunto   <dbl> 8, 8, 8, 1, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,~
$ ocurr_trab <dbl> 8, 8, 8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,~
$ lugar_ocur <dbl> 88, 88, 88, 9, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, ~
$ necropsia  <dbl> 9, 2, 2, 9, 2, 2, 2, 2, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9, 2, 9,~
$ asist_medi <dbl> 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ sitio_ocur <dbl> 1, 11, 11, 1, 11, 3, 11, 11, 11, 3, 11, 3, 11, 3, 12, 3, 11~
$ cond_cert  <dbl> 2, 3, 3, 1, 3, 3, 1, 3, 1, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3, 1,~
$ nacionalid <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ derechohab <dbl> 1, 7, 2, 7, 99, 2, 2, 99, 2, 2, 1, 2, 1, 2, 2, 2, 99, 2, 3,~
$ embarazo   <dbl> 8, 8, 8, 8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 8, 8, 8, 8,~
$ rel_emba   <dbl> 8, 8, 8, 8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 8, 8, 8, 8,~
$ horas      <dbl> 8, 19, 7, 11, 14, 17, 10, 0, 20, 4, 19, 4, 10, 9, 16, 10, 1~
$ minutos    <dbl> 45, 0, 0, 10, 30, 15, 20, 20, 15, 40, 10, 7, 45, 15, 0, 15,~
$ capitulo   <dbl> 9, 9, 4, 20, 9, 14, 4, 13, 2, 11, 2, 10, 9, 2, 4, 11, 9, 2,~
$ grupo      <dbl> 7, 7, 2, 25, 4, 1, 2, 9, 2, 6, 2, 5, 7, 9, 2, 9, 4, 9, 5, 6~
$ lista1     <chr> "069", "069", "052", "103", "067", "085", "052", "083", "03~
$ gr_lismex  <chr> "30", "30", "20", "E51", "28", "38", "20", "37", "09", "35"~
$ vio_fami   <dbl> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,~
$ area_ur    <dbl> 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1,~
$ edad_agru  <chr> "18", "23", "22", "24", "15", "20", "19", "20", "17", "18",~
$ complicaro <dbl> 8, 8, 8, 8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 8, 8, 8, 8,~
$ dia_cert   <dbl> 99, 99, 99, 27, 1, 1, 19, 20, 7, 18, 19, 20, 1, 20, 28, 20,~
$ mes_cert   <dbl> 1, 1, 1, 12, 3, 7, 12, 10, 1, 1, 12, 1, 1, 1, 11, 1, 12, 1,~
$ anio_cert  <dbl> 2017, 2017, 2017, 2016, 2016, 2016, 2016, 2016, 2017, 2017,~
$ maternas   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ lengua     <dbl> 9, 9, 9, 2, 9, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,~
$ cond_act   <dbl> 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1,~
$ par_agre   <dbl> 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88,~
$ ent_ocules <dbl> 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88,~
$ mun_ocules <chr> "888", "888", "888", "888", "888", "888", "888", "888", "88~
$ loc_ocules <chr> "8888", "8888", "8888", "8888", "8888", "8888", "8888", "88~
$ razon_m    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ dis_re_oax <dbl> 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999,~
```

Hacer esto es lo mismo que hacerlo con el pipe nativo de `R`,

`|>`:

```
mortalidad_2017 |> glimpse()
```

o bien ponerlo dentro del `glimpse`:

```
glimpse(mortalidad_2017)
```

El comando `ncol` nos muestra el número de columnas en la base de datos y `tally` nos cuenta el número de filas en la base de datos.

```
mortalidad_2017 %>% ncol() #Columnas
```

```
[1] 59
```

```
mortalidad_2017 %>% tally() #Filas
```

```
# A tibble: 1 x 1
      n
  <int>
1 703047
```

El comando `nrow` hace lo mismo: conteo de filas. Pero no es tan recomendado como veremos más adelante.

```
mortalidad_2017 %>% nrow() #Filas
```

```
[1] 703047
```

Podemos ver los nombres de las columnas de la base de datos con `colnames`:

```
#Mostramos las columnas
mortalidad_2017 %>% colnames()
```

```
 [1] "ent_regis"  "mun_regis"  "ent_resid"  "mun_resid"  "tloc_resid"
 [6] "loc_resid"  "ent_ocurr"  "mun_ocurr"  "tloc_ocurr" "loc_ocurr"
[11] "causa_def"  "lista_mex"  "sexo"       "edad"       "dia_ocurr"
[16] "mes_ocurr"  "anio_ocur"  "dia_regis"  "mes_regis"  "anio_regis"
[21] "dia_nacim"  "mes_nacim"  "anio_nacim" "ocupacion"  "escolarida"
[26] "edo_civil"  "presunto"   "ocurr_trab" "lugar_ocur" "necropsia"
[31] "asist_medi" "sitio_ocur" "cond_cert"  "nacionalid" "derechohab"
[36] "embarazo"   "rel_emba"   "horas"      "minutos"    "capitulo"
[41] "grupo"      "lista1"     "gr_lismex"  "vio_fami"   "area_ur"
[46] "edad_agru"  "complicaro" "dia_cert"   "mes_cert"   "anio_cert"
[51] "maternas"   "lengua"     "cond_act"   "par_agre"   "ent_ocules"
[56] "mun_ocules" "loc_ocules" "razon_m"    "dis_re_oax"
```

Una descripción de las variables las puedes encontrar en el archivo `diccionario_datos_defunciones_generales_2017.csv`. Donde aparece como sigue:

Una vez tenemos una base de datos podemos analizar sus entradas usando el nombre de la base y corchetes. Por ejemplo:

```
#Obtengo todo el registro de la columna sexo
mortalidad_2017[ ,"sexo"]
```

```
# A tibble: 6 x 1
   sexo
  <dbl>
1     2
2     2
3     1
4     2
5     2
6     1
```

También podemos usar el número de columna:

```
#Me regresa las observaciones de la columna 2
#dada por mun_regis
#| eval: false
mortalidad_2017[ ,2]
```

NOMBRE_CAMPO	LONGITUD	TIPO	NEMÓNICO	CATÁLOGO	RANGO
Entidad Registro	2	N	ent_regis	decateml	01-32
Municipio Registro	3	N	mun_regis	decateml	001-570
Entidad Residencia	2	N	ent_resid	decateml	01-35, 99
Municipio Residencia	3	N	mun_resid	decateml	001-570,
Tamaño Localidad Residencia	2	N	tloc_resid	detamloc	01-17,99
Clave Localidad de Residencia	4	N	loc_resid	decateml	0001 - 69
Entidad Ocurrencia	2	N	ent_ocurr	decateml	01-35,99
Municipio Ocurrencia	3	N	mun_ocurr	decateml	001-570,
Tamaño de Localidad Ocurrencia	2	N	tloc_ocurr	detamloc	01-17,99
Clave Localidad de Ocurrencia	4	N	loc_ocur	decateml	0001 - 69
Causa Defunción (Lista Detallada)	4	A	causa_def	decatcausa	NA
Causa Defunción (Lista Mexicana)	3	A	lista_mex	delistamex	NA
Sexo	1	N	sexo	desexo	1-2,9
Edad	4	N	edad	deedad	1001-102
Día Defunción	2	N	dia_ocurr	dedias	01-31,99
Mes Defunción	2	N	mes_ocurr	demeses	01-12, 99
Año Defunción	4	N	anio_ocur	deaño	1900-Añ
Día Registro	2	N	dia_regis	dedias	01-31,99
Mes Registro	2	N	mes_regis	demeses	01-12
Año Registro	4	N	anio_regis	deaño	Año esta
Día Nacimiento	2	N	dia_nacim	dedias	01-31,99
Mes Nacimiento	2	N	mes_nacim	demeses	01-12,99
Año Nacimiento	4	N	anio_nacim	deaño	1900-Añ
Ocupación	2	N	ocupacion	deocupa	1 - 11, 9
Escolaridad	2	N	escolarida	deesco	1-10, 88,
Estado Conyugal	1	N	edo_civil	deedocony	1-6, 8, 9
Presunto	1	N	presunto	depresunto	1-5, 8
Ocurrió Trabajo	1	N	ocurr_trab	deocutrab	1-2, 8, 9
Sitio donde Ocurrio la Lesión	2	N	lugar_ocur	desitiolesion	0 - 9, 88
Necropsia	1	N	necropsia	denecrop	1-2, 9
Asistencia Médica	1	N	asist_medi	deasismed	1-2,9
Sitio donde Ocurrio la Defunción	2	N	sitio_ocur	desitiodefun	1-12,99
Certificada Por	1	N	cond_cert	deccertif	1-5, 8, 9
Nacionalidad	1	N	nacionalid	denacion	1-2, 9
Derechohabiencia	2	N	derechohab	dederech	1-9, 99
Condición de embarazo	1	N	embarazo	decondemba	1-6, 8, 9
Causas relacionadas con embarazo	1	N	rel_emba	derelemba	1,2, 8, 9
Hora de la defunción	2	N	horas	dehoradef	00-23, 99
Minuto de la defunción	2	N	minutos	demindef	00-59, 99
Capitulo	2	N	capitulo	decapgpo	NA
Grupo	2	N	grupo	decapgpo	NA
Lista1	3	A	lista1	delista1	NA
Grupos lista mexicana	3	A	gr_lismex	degpolisme	NA
Violencia Familiar	1	N	vio_fami	deviofam	1, 2, 8, 9
Area Urbano Rural	1	N	area_ur	deurbrur	1, 2, 9
Edad agrupada	2	C	edad_agru	deedadagrup	01-30
Complicaron el embarazo	1	N	complicaro	decomplicaemba	1, 2, 8, 9
Día de Certificación	2	N	dia_cert	dedias	01-31,99
Mes de Certificación	2	N	mes_cert	demeses	01-12, 99

```
# A tibble: 703,047 x 1
   mun_regis
   <chr>
 1 001
 2 009
 3 001
 4 006
 5 001
 6 001
 7 001
 8 001
 9 001
10 001
# ... with 703,037 more rows
```

```
#Me regresa las observaciones de la columna 2
#dada por mun_regis
head(mortalidad_2017[ , 2])
```

También podemos acceder a la base por renglones. Por ejemplo, así vemos el tercer renglón:

```
mortalidad_2017[3, ]
```

```
# A tibble: 1 x 6
  ent_regis mun_regis ent_resid mun_resid tloc_resid loc_resid
  <chr>     <chr>     <chr>     <chr>          <dbl> <chr>
1 01        001       01        001               15 0001
```

Finalmente, podemos combinar la primer entrada de la columna `lista1` se vería así:

```
mortalidad_2017[1, "lista1"]
```

```
# A tibble: 1 x 1
  lista1
  <chr>
1 069
```

En resumen, la notación siempre es de la siguiente forma:

$$\text{datos}[\underbrace{f}_{fila}, \overbrace{c}^{columna}]$$

Esta notación `notación matricial` es estándar en el mundo de computación y de las matemáticas. Siempre siempre primero es fila luego columna.

Otra forma de seleccionar una columna es con $ seguido del nombre de la columna. Por ejemplo:

```
mortalidad_2017$edo_civil
```

```
[1] 3 5 3 3 2 5
```

Por ciertos errores en los que se pueden incurrir al crear funciones, se recomienda que se utilice siempre la notación de doble corchete y se evite el signo de $. Pero ambos métodos funcionan. Sólo recuerda que la notación es así:

$$\text{datos\$Columna}[\underbrace{f}_{\text{Fila}}]$$

Una última opción es con `select`:

```
mortalidad_2017 %>% select(edo_civil)
```

```
# A tibble: 6 x 1
  edo_civil
      <dbl>
1         3
2         5
3         3
4         3
5         2
6         5
```

la cual nos regresa la columna de estado civil. El comando `select` es bastante útil si, por ejemplo, no recordamos exactamente el nombre de la columna. Por ejemplo, si sólo recordamos que tiene la palabra `"civil"` pero no exactamente cómo se escribe podemos usar `contains`:

```
#Podemos usar contains si recordamos que dice civil
#pero no el nombre completo
mortalidad_2017 %>% select(contains("civil"))
```

```
# A tibble: 703,047 x 1
   edo_civil
       <dbl>
 1         3
 2         5
 3         3
 4         3
 5         2
 6         5
 7         5
 8         5
 9         5
10         5
# ... with 703,037 more rows
```

Podemos preguntarnos, de igual manera por la 7a entrada de `edo_civil` haciendo:

```
mortalidad_2017$edo_civil[7]
```

```
[1] 5
```

Lo cual es similar (mas no equivalente) a la forma anterior:

```
mortalidad_2017[7, "edo_civil"]
```

```
# A tibble: 1 x 1
  edo_civil
      <dbl>
1         5
```

En el caso del `tidyverse` el equivalente a `select`, para elegir fila, el equivalente se conoce como `slice`:

```
  #Selecciona la fila 11
  mortalidad_2017 %>% slice(11)
```

```
# A tibble: 1 x 59
  ent_regis mun_regis ent_resid mun_re~1 tloc_~2 loc_r~3 ent_o~4 mun_o~5 tloc_~6
  <chr>     <chr>     <chr>     <chr>      <dbl> <chr>   <chr>   <chr>     <dbl>
1 01        001       01        001           15 0001    01      001          15
# ... with 50 more variables: loc_ocurr <chr>, causa_def <chr>,
#   lista_mex <chr>, sexo <dbl>, edad <dbl>, dia_ocurr <dbl>, mes_ocurr <dbl>,
#   anio_ocur <dbl>, dia_regis <dbl>, mes_regis <dbl>, anio_regis <dbl>,
#   dia_nacim <dbl>, mes_nacim <dbl>, anio_nacim <dbl>, ocupacion <dbl>,
#   escolarida <dbl>, edo_civil <dbl>, presunto <dbl>, ocurr_trab <dbl>,
#   lugar_ocur <dbl>, necropsia <dbl>, asist_medi <dbl>, sitio_ocur <dbl>,
#   cond_cert <dbl>, nacionalid <dbl>, derechohab <dbl>, embarazo <dbl>, ...
```

Podemos combinar múltiples argumentos con `%>%`[1]:

[1] En el mismo renglón debe estar el `%>%` que el último comando para no generar error

```
  mortalidad_2017 %>%
    select(sexo, edo_civil) %>%  #Selecciona edo civil y sexo
    slice(11:20) #Selecciona filas 11 a 20
```

```
# A tibble: 10 x 2
    sexo edo_civil
   <dbl>     <dbl>
 1     1         5
 2     1         5
 3     2         3
 4     1         5
 5     1         5
 6     2         2
```

```
 7     1          5
 8     1          5
 9     2          9
10     1          5
```

Esta forma de combinación con `pipes` (`%>%` ó `|>`) será bastante útil más adelante.

Para seleccionar múltiples filas o columnas, como vimos en el ejemplo anterior, hay que crear un vector usando `c`[2]. Un vector es una lista ordenada de variables del mismo tipo. Por ejemplo:

[2] Se llama `c` por `concatenate` (concatenar).

```
mi_vector <- c(1, 141, 12)
```

Es un vector de 3 entradas. De hecho, la forma de acceder a sus entradas es la misma que la de los `tibbles` con un agregado: ¡sólo hay filas, no hay columnas!

> **Note**
>
> Intuitivamente podemos pensar un vector como una columna de una base de datos. En ella todas las variables son del mismo tipo.

Para acceder a las entradas de un vector es de la siguiente forma:

```
#Para un vector sólo pongo el número de entrada ¡no hay columna!
mi_vector[2] #entrada 2
```

```
[1] 141
```

En general la notación es la siguiente:

$$\text{vector}[\underbrace{i}_{\text{Entrada}}]$$

Un vector que ya nos encontramos antes es el de columnas. Podemos, por ejemplo, ver el nombre de la cuarta columna de

la base combinando lo que sabemos de vectores con el comando `colnames`:

```
#Regresa el nombre de la 5a columna
colnames(mortalidad_2017)[5]
```

```
[1] "tloc_resid"
```

La pregunta también puede hacerse al revés: podemos pedirle a `R` que nos conteste cuál es el número de columna para una columna dada. Por ejemplo, ¿cuál es el número de columna para `edad`?

```
#Preguntamos a R cuál de las columnas se llama edad
which(colnames(mortalidad_2017) == "edad")
```

```
[1] 14
```

Nota que si preguntamos por una columna que no existe, `R` nos regresa lo siguiente:

```
#Preguntamos a R cuál de las columnas se llama edad
which(colnames(mortalidad_2017) == "Paraguas")
```

```
integer(0)
```

Esto significa que no hay ninguna columna con ese nombre. Finalmente, nota que el `which` puede regresar múltiples resultados si las cosas se repiten. Por ejemplo, el siguiente comando nos regresa las múltiples entradas del vector `nombres_de_amigos` donde hay un amigo que tiene el nombre de `Alejandro`:

```
#Creo un vector con nombres de mis amigos
nombres_de_amigos <- c("Alejandro", "Beatriz", "Alejandro", "Carla")

#Pregunto por cuáles entradas contienen a Alejandro
which(nombres_de_amigos == "Alejandro")
```

```
[1] 1 3
```

Finalmente, podemos usar vectores para seleccionar múltiples columnas y filas, por ejemplo si deseo seleccionar, de la base, las columnas de `sexo` y `edad`:

```
mortalidad_2017[ , c("sexo", "edad")]
```

```
# A tibble: 6 x 2
   sexo  edad
  <dbl> <dbl>
1     2  4068
2     2  4090
3     1  4088
4     2  4096
5     2  4051
6     1  4078
```

Finalmente si quiero seleccionar sólo algunos renglones puedo ponerlos en un vector:

```
#Selecciona sólo el renglón 1 y el 7
mortalidad_2017[c(1, 7), c("sexo", "edad")]
```

```
# A tibble: 2 x 2
   sexo  edad
  <dbl> <dbl>
1     2  4068
2     1  4070
```

Mientras que el `:` selecciona *del renglón 1 al renglón 7*

```
#Selecciona del 1 al 7
mortalidad_2017[1:7, c("sexo", "edad")]
```

```
# A tibble: 7 x 2
   sexo  edad
  <dbl> <dbl>
1     2  4068
2     2  4090
3     1  4088
4     2  4096
5     2  4051
6     1  4078
7     1  4070
```

El equivalente usando `select` y `slice` es:

```
mortalidad_2017 %>%
  select(sexo, edad) %>%
  slice(1:7)
```

```
# A tibble: 7 x 2
   sexo  edad
  <dbl> <dbl>
1     2  4068
2     2  4090
3     1  4088
4     2  4096
5     2  4051
6     1  4078
7     1  4070
```

```
#Selecciona renglones del 2 al 7
#y las columnas 5 a 9
mortalidad_2017[2:7, 5:9]
```

```
# A tibble: 6 x 5
  tloc_resid loc_resid ent_ocurr mun_ocurr tloc_ocurr
       <dbl> <chr>     <chr>     <chr>          <dbl>
1          1 0016      01        009                1
2         15 0001      01        001               15
3          8 0001      01        006                8
```

```
4          15 0001      01         001                15
5          15 0001      01         001                15
6          15 0001      01         001                15
```

Los `:` también funcionan para seleccionar rangos de columnas a través del `select`:

```
#Selecciona todas las columnas desde loc_resid hasta tloc_ocurr
mortalidad_2017 %>%
  select(loc_resid:tloc_ocurr)
```

```
# A tibble: 703,047 x 4
   loc_resid ent_ocurr mun_ocurr tloc_ocurr
   <chr>     <chr>     <chr>          <dbl>
 1 0001      01        001               15
 2 0016      01        009                1
 3 0001      01        001               15
 4 0001      01        006                8
 5 0001      01        001               15
 6 0001      01        001               15
 7 0001      01        001               15
 8 0001      01        001               15
 9 0001      01        001               15
10 1025      01        001               15
# ... with 703,037 more rows
```

Ejercicios

1. Lee la hoja `MCV2` del archivo `unicef_vacunas.xlsx` el cual contiene información sobre vacunas `MCV2` dada por la UNICEF. Responde las siguientes preguntas:

a. Determina los nombres de todas las columnas de las variables usando `R` (no vale ver el archivo).
b. Determina el número de renglones y el número de filas totales de la base.
c. Encuentra cuál fila contiene a México en la columna de `country`.
d. Regresa todas las mediciones para la fila de Colombia.

e. Regresa la 5a columna.
f. Obtén el nombre de la columna `9`.
g. Obtén la entrada en la columna `5` y fila `14`.

2. Considera la base de datos creada por el siguiente código:

```
base_inventada <- tibble(Tiempo = rexp(100),
                         Enfermo = rbinom(100, 83, 1/3))
```

Determina qué ocasionó los resultados siguientes:

a. ¿Por qué da error?

```
#Primer error
base_inventada[ , 200]
```

```
Error in `base_inventada[, 200]`:
! Can't subset columns past the end.
i Location 200 doesn't exist.
i There are only 2 columns.
```

b. ¿Por qué da `NA`?

```
#NA
base_inventada[1000, ]
```

```
# A tibble: 1 x 2
  Tiempo Enfermo
   <dbl>   <int>
1     NA      NA
```

c. ¿Por qué dice esto?

```
#Primer error
base_inventada[0,0]
```

```
# A tibble: 0 x 0
```

d. ¿Por qué da error‘?

```r
#NULL
base_inventada[2, c("tiempo")]
```

```
Error in `base_inventada[2, c("tiempo")]`:
! Can't subset columns that don't exist.
x Column `tiempo` doesn't exist.
```

e. Arregla este código para que no dé error

```r
base.inventada %>% select("Tiempo")
               %>% slice(2)
```

Limpieza de la base de datos

Seleccionar columnas con `select`

De la base de mortalidad comencemos por quedarnos sólo con las columnas `ent_regis`, `sexo`, `edad`, `dia_ocurr`, `mes_ocurr`, `anio_ocurr` y `causa_def`. Para ello seleccionamos múltiples columnas mediante un vector con el nombre de las columnas a seleccionar.

> **Warning**
>
> `R` trabaja con una copia de la base de datos. Todo lo que hacemos aquí nunca va a cambiar la base de datos a menos que explícitamente se lo pidamos. ¡Es genial porque nos evitamos el riesgo de perder información!

```r
#Reescribimos la variable sólo con las columnas que nos
#interesan
mortalidad_2017 <- mortalidad_2017 %>%
  select(ent_regis, sexo, edad, dia_ocurr, mes_ocurr, anio_ocur, causa_def)
```

Ojo que aquí estoy sobreescribiendo la copia de `R` de la base de `mortalidad_2017`. Es decir, estoy editando la base `mortalidad_2017` y guardándola en sí misma (para que se

queden los cambios). También lo pude haber guardado en otra variable:

```
#Reescribimos la variable sólo con las columnas que nos
#interesan
mortalidad_2017_pedazo <- mortalidad_2017 %>%
  select(ent_regis, sexo, edad, dia_ocurr, mes_ocurr, anio_ocur, causa_def)
```

El punto importante es guardarla porque de otra forma los cambios no son permanentes. Por ejemplo el siguiente comando de seleccionar sólo `sexo` y `edad` no edita la base pues no se asigna a la base:

```
#Reescribimos la variable sólo con las columnas que nos
#interesan
mortalidad_2017 %>%
  select(sexo, edad)
```

```
# A tibble: 703,047 x 2
    sexo  edad
   <dbl> <dbl>
 1     2  4068
 2     2  4090
 3     1  4088
 4     2  4096
 5     2  4051
 6     1  4078
 7     1  4070
 8     2  4079
 9     2  4061
10     1  4069
# ... with 703,037 more rows
```

Renombrar columnas con `rename`

Utilicemos `rename` para mejorar los nombres de las variables. El comando es:

```
rename(`Nuevo nombre` = `Viejo nombre`)
```

Por ejemplo:

```
mortalidad_2017 <- mortalidad_2017 %>%
  rename(Entidad = ent_regis)
```

También podemos renombrar varias columnas de golpe aplicando varios `rename` concatenados por el `%>%`:

```
mortalidad_2017 <- mortalidad_2017 %>%
  rename(Sexo = sexo) %>%
  rename(Edad = edad) %>%
  rename(Día = dia_ocurr) %>%
  rename(Mes = mes_ocurr) %>%
  rename(Año = anio_ocur) %>%
  rename(Causa = causa_def)
```

Filtrar filas con `filter`

Podemos reducir nuestra base quedándonos, por ejemplo, con aquellas observaciones que ocurrieron en el Año 2017. Nota que el número de filas cambia de:

```
#Antes de filtrar
mortalidad_2017 %>% tally()
```

```
# A tibble: 1 x 1
      n
  <int>
1 703047
```

```
#Filtro
mortalidad_2017 <- mortalidad_2017 %>%
  filter(Año == 2017)
```

```
#Después de filtrar
mortalidad_2017 %>% tally()
```

```
# A tibble: 1 x 1
       n
   <int>
1 688468
```

El filter opera poniéndole una condición adentro. Aquí te copio las instrucciones más comunes:

Table 1: Instrucciones comunes para el 'filter'

Nombre	Instrucción	Significado
Igualdad	`x == y`	`x` es igual a `y`
Mayor/menor	`x > y`	`x` es menor que `y`
Mayor/menor igual	`x >= y`	`x` es mayor o igual que `y`
Diferente	`x != y`	`x` es distinto de `y` ($x \neq y$)
Pertenencia	`x %in% y`	`x` está en `y` si `y` es un vector `c`
O	`A | B`	al menos una de las dos condiciones `A` ó `B` es verdadera
Y	`A & B`	ambas `A` y `B` son verdaderas
No	`!A`	lo contrario a la condición `A`
Búsqueda de palabras	`str_detect(Columna, "palabra")`	Devuelve aquellos casos donde la columna `Columna` contiene la palabra `"palabra"`
Identifica datos faltantes	`is.na`	`is.na(Columna)`

Podemos jugar más con el filter por ejemplo devolviendo aquellas defunciones en un mes después de febrero (mes `2`):

```
mortalidad_2017 %>%
  filter(Mes > 2)
```

```
# A tibble: 568,041 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          1  4079    22     3  2017 C61X
 2 01          2  4080     4     4  2017 M623
 3 01          1  4075     6     4  2017 E149
 4 01          2  1097    27     3  2017 P832
 5 01          2  4075     1     5  2017 E46X
 6 01          1  4085    30     4  2017 I500
 7 01          1  4092    17    11  2017 F102
 8 01          1  4061     2     5  2017 C189
 9 01          1  4024    21     4  2017 C959
10 01          2  4073    24     4  2017 E142
# ... with 568,031 more rows
```

Aquellas defunciones cuyo mes es antes de febrero (incluyendo febrero):

```
mortalidad_2017 %>%
  filter(Mes <= 2)
```

```
# A tibble: 120,427 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4061     7     1  2017 C259
 2 01          1  4069    18     1  2017 K631
 3 01          1  4091    20     1  2017 J441
 4 01          2  4076     1     1  2017 I698
 5 01          1  4070    20     1  2017 C61X
 6 01          2  4048    20     1  2017 K803
 7 01          1  4062    12     1  2017 C61X
 8 01          1  4089    13     1  2017 N390
 9 01          2  4042     8     1  2017 E117
10 01          2  4085    14     1  2017 I120
# ... with 120,417 more rows
```

Aquellas defunciones cuyos días estén entre el día 7 y el 9

```
  mortalidad_2017 %>%
    filter(Día %in% c(7,8,9)) #o bien 7:9 en lugar del c
```

```
# A tibble: 67,650 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4061     7     1  2017 C259
 2 01          2  4042     8     1  2017 E117
 3 01          1  4073     8     2  2017 J440
 4 01          1  4088     7     2  2017 I802
 5 01          1  4067     9     1  2017 E119
 6 01          1  4019     7     1  2017 Q070
 7 01          2  4062     9     1  2017 I678
 8 01          1  4057     8     5  2017 K703
 9 01          2  4052     7     1  2017 K274
10 01          1  4059     9     3  2017 J180
# ... with 67,640 more rows
```

Aquellas defunciones cuyo día **no** esté entre el día 7 y el 9

```
  mortalidad_2017 %>%
    filter(!(Día %in% c(7,8,9))) #o bien 7:9 en lugar del c
```

```
# A tibble: 620,818 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          1  4069    18     1  2017 K631
 2 01          1  4091    20     1  2017 J441
 3 01          2  4076     1     1  2017 I698
 4 01          1  4070    20     1  2017 C61X
 5 01          2  4048    20     1  2017 K803
 6 01          1  4062    12     1  2017 C61X
 7 01          1  4089    13     1  2017 N390
 8 01          2  4085    14     1  2017 I120
 9 01          2  4042    19     1  2017 K729
10 01          2  4067     3     1  2017 I120
# ... with 620,808 more rows
```

Aquellas defunciones en enero el día 14:

```
  mortalidad_2017 %>%
    filter(Día == 14 & Mes == 1) #o bien 7:9 en lugar del c
```

```
# A tibble: 2,035 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4085    14     1  2017 I120
 2 01          1  4083    14     1  2017 I698
 3 01          1  4089    14     1  2017 R54X
 4 01          1  4078    14     1  2017 E119
 5 01          1  4062    14     1  2017 E142
 6 01          1  4076    14     1  2017 I64X
 7 01          2  4085    14     1  2017 I210
 8 01          2  4078    14     1  2017 I219
 9 01          2  4087    14     1  2017 I119
10 01          1  4056    14     1  2017 E119
# ... with 2,025 more rows
```

O bien:

```
  mortalidad_2017 %>%
    filter(Día == 14) %>%
    filter(Mes == 1)
```

```
# A tibble: 2,035 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4085    14     1  2017 I120
 2 01          1  4083    14     1  2017 I698
 3 01          1  4089    14     1  2017 R54X
 4 01          1  4078    14     1  2017 E119
 5 01          1  4062    14     1  2017 E142
 6 01          1  4076    14     1  2017 I64X
 7 01          2  4085    14     1  2017 I210
 8 01          2  4078    14     1  2017 I219
 9 01          2  4087    14     1  2017 I119
10 01          1  4056    14     1  2017 E119
```

```
# ... with 2,025 more rows
```

Aquellas defunciones donde el mes sea enero o diciembre:

```
mortalidad_2017 %>%
  filter(Mes == 1 | Mes == 12)
```

```
# A tibble: 121,073 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4061     7     1  2017 C259
 2 01          1  4069    18     1  2017 K631
 3 01          1  4091    20     1  2017 J441
 4 01          2  4076     1     1  2017 I698
 5 01          1  4070    20     1  2017 C61X
 6 01          2  4048    20     1  2017 K803
 7 01          1  4062    12     1  2017 C61X
 8 01          1  4089    13     1  2017 N390
 9 01          2  4042     8     1  2017 E117
10 01          2  4085    14     1  2017 I120
# ... with 121,063 more rows
```

Aquellas defunciones donde el mes sea distinto de octubre:

```
mortalidad_2017 %>%
  filter(Mes != 10)
```

```
# A tibble: 630,907 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4061     7     1  2017 C259
 2 01          1  4069    18     1  2017 K631
 3 01          1  4091    20     1  2017 J441
 4 01          2  4076     1     1  2017 I698
 5 01          1  4070    20     1  2017 C61X
 6 01          2  4048    20     1  2017 K803
 7 01          1  4062    12     1  2017 C61X
 8 01          1  4089    13     1  2017 N390
```

```
 9 01          2  4042     8     1  2017 E117
10 01          2  4085    14     1  2017 I120
# ... with 630,897 more rows
```

Aquellas defunciones cuya causa contenga el caracter `C5`:

```
mortalidad_2017 %>%
  filter(str_detect(Causa,"C5"))
```

```
# A tibble: 14,701 x 7
   Entidad  Sexo  Edad   Día   Mes   Año Causa
   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1 01          2  4049     6     1  2017 C509
 2 01          2  4071    23     1  2017 C509
 3 01          2  4053    20     1  2017 C56X
 4 01          2  4066    18     2  2017 C56X
 5 01          2  4029     2     1  2017 C509
 6 01          2  4063     5    12  2017 C56X
 7 01          2  4059     6     1  2017 C56X
 8 01          2  4047    31     1  2017 C539
 9 01          2  4064    27     1  2017 C56X
10 01          2  4058    31     1  2017 C56X
# ... with 14,691 more rows
```

Ejercicio

En la base `unicef_vacunas.xlsx` abre la hoja `HEPBB` con información de la cobertura de dicha vacuna. Determina:

1. ¿Cuántos países están registrados para la región `ROSA`?
2. ¿Cuántos países en 2020 lograron una cobertura mayor al 90%?
3. Dentro de la región de Latinoamérica y el Caribe (`LACR`) ¿cuántos países lograron una cobertura menor a 80%?
4. ¿Cuál fue la cobertura de México, Colombia y Uruguay en 2020?
5. ¿Cuántos países no contienen datos para 1990?

Estadísticos de resumen con `summarise`

La función de `summarise` sirve para colapsar columnas en números. Por ejemplo: obtener el promedio de una columna, obtener el máximo, un cuartil, una suma o un conteo. Por ejemplo para obtener el promedio de la columna `Edad` basta con usar `mean` adentro de `summarise`:

```
mortalidad_2017 %>%
  summarise(`Edad promedio` = mean(Edad))
```

```
# A tibble: 1 x 1
  `Edad promedio`
            <dbl>
1           4000.
```

Nota que aquí la edad sale altísima y esto es porque el INEGI empieza a contar los años a partir del `4000` siendo `4001` un año, `4002` dos años etc. Por otro lado es necesario eliminar a aquellos con edad de `4998` pues corresponde a `Edad no especificada`. Para ello filtramos y luego calculamos la media de edad menos `4000`:

```
mortalidad_2017 %>%
  filter(Edad >= 4000 & Edad != 4998) %>%
  summarise(`Edad promedio` = mean(Edad - 4000))
```

```
# A tibble: 1 x 1
  `Edad promedio`
            <dbl>
1            65.5
```

Esto nos dice que en el `2017` el promedio de edad de las defunciones de más de un año fue de 65 años.

Observa que ese promedio es lo mismo que haber sumado todas las edades con `sum` y luego dividido entre el total de observaciones `n`:

```
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Suma de edades` = sum(Edad - 4000))
```

```
# A tibble: 1 x 1
  `Suma de edades`
             <dbl>
1         43216448
```

```
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Total de observaciones` = n())
```

```
# A tibble: 1 x 1
  `Total de observaciones`
                     <int>
1                   660152
```

pues:

```
  43216448/660152
```

```
[1] 65.46439
```

Podemos también obtener el máximo de edad con `max` (el mínimo sería con `min`)

```
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Edad promedio` = max(Edad - 4000))
```

```
# A tibble: 1 x 1
  `Edad promedio`
            <dbl>
1             120
```

O bien la mediana o algún cuantil:

```r
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Edad mediana` = median(Edad - 4000))
```

```
# A tibble: 1 x 1
  `Edad mediana`
           <dbl>
1             69
```

```r
  #Cuantil 0.25 = primer cuartil
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Edad (25%)` = quantile(Edad - 4000, 0.25))
```

```
# A tibble: 1 x 1
  `Edad (25%)`
         <dbl>
1           53
```

de hecho podemos aplicar el `summarise` de todos a la vez:

```r
  #Cuantil 0.25 = primer cuartil
  mortalidad_2017 %>%
    filter(Edad >= 4000 & Edad != 4998) %>%
    summarise(`Edad promedio` = mean(Edad - 4000),
              `Edad mediana`  = median(Edad - 4000),
              `Edad máxima`   = max(Edad - 4000))
```

```
# A tibble: 1 x 3
  `Edad promedio` `Edad mediana` `Edad máxima`
            <dbl>          <dbl>         <dbl>
1            65.5             69           120
```