Diplomado de Análisis Estadístico usando R

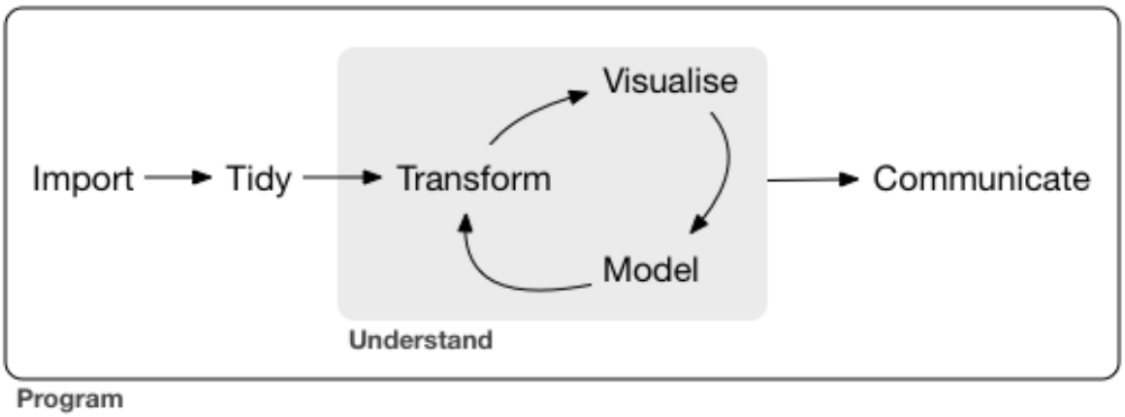
Módulo 3:
Exploración de datos

Profesor: *Víctor Macías E.*

> "Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone".
>
> *John Tukey*

1. Etapas de un proyecto usando datos

A continuación se presenta un modelo de las etapas de un proyecto relacionado a datos:



Fuente: https://r4ds.had.co.nz/index.html

2. Transformación de datos

Esta etapa incluye, entre otras tareas, crear variables; renombrar variables; colapsar datos a un valor que describe la media, desviación estándar y número de observaciones para distintos grupos, etc.

Para transformar los datos usaremos el paquete `dplyr`[1]. Entre las funciones que son utilizadas con mayor frecuencia se encuentran:

[1] Para una revisión más detallada se recomienda revisar las *vignettes* incluidas en la página https://cran.r-project.org/web/packages/dplyr/.

2.1. filter

Genera un subconjunto de los datos a partir de fitrar observaciones de acuerdo a un criterio determinado.

2.2. arrange

Ordena observaciones de acuerdo a los valores de una o más variables.

2.3. select

Genera un subconjunto de los datos a partir de una o más variables seleccionadas.

2.4. mutate

Añade una nueva columna que puede ser una función de otras columnas,

2.5. rename

Cambia el nombre de una variable.

2.6. summarise

Colapsa muchos valores a un solo valor que describe características de los datos como, por ejemplo, la media, desviación estándar, número de observaciones, etc.

Esta función es muy útil cuando se usa en conjunto con `group_by()`, lo que cambia la unidad de análisis del conjunto completo de datos a grupos individuales.

Entre las funciones que se pueden usar en conjunto con `summarise` se encuentran:

(a) Medidas de tendencia central

- Media aritmética: `mean(x)`
- Mediana: `median(x)`

(b) Medidas de dispersión

- Desviación estándar: `sd(x)`
- Rango intercuartil: `IQR(x)`
- Mediana de desviaciones absolutas: `mad(x)`

(c) Medidas de orden

- Mínimo: `min(x)`
- Máximo: `max(x)`
- Cuantiles: `quantile(x, 0.25)`. En este caso, se calcula el percentil 25, pero pueden especificarse otros cuantiles.

(d) Medidas de posición

- Primera observación: `first(x)`
- Posición n-ésima: `nth(x, 2)`
- Última observación: `last(x)`

(e) Medidas para contar el número de observaciones

- Número de observaciones: `n()`. Si no quieren contarse los `NAs`, se puede usar `sum(!is.na(x))`.
- Número de observaciones con valores únicos: `n_distinct(x)`
- Si lo único que se quiere obtener es el número de observaciones, usar `count()`.

2.7. Ejemplo:

A partir de los datos que se presentan a continuación, se presentan varios ejemplos aplicando lo visto anteriormente.

id_hogar	edad
A	40
A	40
A	NA
A	3
B	22
B	7
B	4

```
library(dplyr)
```

```
tb <- tibble::tibble(id_hogar = c("A", "A", "A", "A", "B", "B", "B"),
                     edad = c(40, 40, NA, 3, 22, 7, 4))
```

```
tb %>% group_by(id_hogar) %>% summarise(nobs1_grupo = n(),
                                        nobs2_group = sum(!is.na(edad)),
                                        nobs_distinct = n_distinct(edad),
                                        first_obs = first(edad),
                                        segunda_obs = nth(edad, 2),
                                        num_child = sum(edad < 10, na.rm = TRUE))
```

```
## # A tibble: 2 x 7
##   id_hogar nobs1_grupo nobs2_group nobs_distinct first_obs segunda_obs num_child
##   <chr>          <int>       <int>         <int>     <dbl>       <dbl>     <int>
## 1 A                  4           3             3        40          40         1
## 2 B                  3           3             3        22           7         2
```

A continuación se presenta el número de observaciones por grupo, usando `count()`:

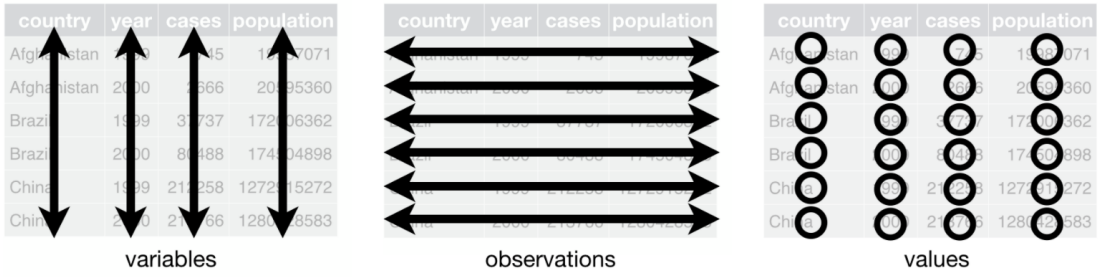
```r
tb %>% group_by(id_hogar) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   id_hogar [2]
##   id_hogar     n
##   <chr>    <int>
## 1 A            4
## 2 B            3
```

3. ¿Qué es un *tidy dataset*[2]?

Un conjunto de datos es *tidy* si:

1. Cada variable debe tener sus propia columna
2. Cada observación debe tener su propia fila
3. Cada valor debe tener su propia celda



Fuente: https://r4ds.had.co.nz/index.html

Disponer los datos de esta forma facilita el trabajo con éstos, ya que estandariza la organización de los datos.

Para estructurar los datos de esta forma, se pueden usar una serie de funciones disponibles en el paquete `tidyr` que al igual que `dplyr` es parte de *tidyverse*.

El primer paso de esta etapa es identificar las variables y observaciones. Luego, es importante ver si se tiene alguno de los siguientes problemas:

- Una variable puede estar distribuida en múltiples columnas.
- Una observación puede estar distribuida en varias filas.

Las dos funciones más importantes de `tidyr` para solucionar estos problemas, `pivot_longer()` y `pivot_wider()`, se describen a continuación:

```r
library(tidyr)
```

[2]Ver Wickham, H. 2014. Tidy data. Journal of Statistical Software. Vol.59, issue 10 http://www.jstatsoft.org/v59/i10/paper.

3.1. pivot_longer

Suponga que se dispone de los siguientes datos:

```
zona <- tibble(
  zona = c("A", "B", "C"),
  `2018` = c(100, 40, 120),
  `2019` = c(110, 60, 115)
)
zona
```

```
## # A tibble: 3 x 3
##   zona  `2018` `2019`
##   <chr>  <dbl>  <dbl>
## 1 A        100    110
## 2 B         40     60
## 3 C        120    115
```

En este caso, las columnas 2018 y 2019 representan valores de la variable `year`, los valores incluidos en dichas columnas son valores de la variable `num_casos` y cada fila contiene dos observaciones. Para llevar los datos a un formato *tidy* usaremos `pivot_longer()`.

```
zona %>% pivot_longer(
  cols = -zona,
  names_to = "year",
  values_to = "num_casos")
```

```
## # A tibble: 6 x 3
##   zona  year  num_casos
##   <chr> <chr>     <dbl>
## 1 A     2018        100
## 2 A     2019        110
## 3 B     2018         40
## 4 B     2019         60
## 5 C     2018        120
## 6 C     2019        115
```

3.2. pivot_wider

Suponga que se dispone de los siguientes datos:

```
val <- tibble(
  zona = c("A", "A", "A", "A", "B", "B", "B", "B"),
  year = c(2019, 2019, 2020, 2020, 2019, 2019, 2020, 2020),
  type = c("varA", "varB", "varA", "varB", "varA", "varB", "varA", "varB"),
  num_casos = c(115, 70, 124, 230, 145, 45, 54, 89)
)
```

```
val
```

```
## # A tibble: 8 x 4
##   zona   year type  num_casos
##   <chr> <dbl> <chr>     <dbl>
## 1 A      2019 varA        115
## 2 A      2019 varB         70
## 3 A      2020 varA        124
## 4 A      2020 varB        230
## 5 B      2019 varA        145
## 6 B      2019 varB         45
## 7 B      2020 varA         54
## 8 B      2020 varB         89
```

En este caso, una observación está definida por *zona* y *year* y se encuentra distribuida en 2 filas. Los nombres de las variables están en la columna *type* y los valores en *num_casos*. Para llevar los datos a un formato *tidy* usaremos `pivot_wider()`.

```
val %>%
    pivot_wider(names_from = type, values_from = num_casos)
```

```
## # A tibble: 4 x 4
##   zona   year  varA  varB
##   <chr> <dbl> <dbl> <dbl>
## 1 A      2019   115    70
## 2 A      2020   124   230
## 3 B      2019   145    45
## 4 B      2020    54    89
```

3.3. separate

Suponga que tiene los siguientes datos:

```
zona <- tibble::tibble(
  zona = c("A", "B", "C"),
  rate = c("100/160", "40/75", "120/130")
  )
zona
```

```
## # A tibble: 3 x 2
##   zona  rate
##   <chr> <chr>
## 1 A     100/160
## 2 B     40/75
## 3 C     120/130
```

En este caso, la columna *rate* incluye dos variables *num_casos* y *total*, las que se dejarán

en columnas separadas usando `separate`.

```
zona <- zona %>% separate(rate, into = c("num_casos", "total"),
                          convert = TRUE)
zona
```

```
## # A tibble: 3 x 3
##   zona  num_casos total
##   <chr>     <int> <int>
## 1 A           100   160
## 2 B            40    75
## 3 C           120   130
```

3.4. unite

`unite` es lo opuesto de `separate`, combinando múltiples columnas en una.

```
zona %>% unite(new, num_casos, total, sep = "/")
```

```
## # A tibble: 3 x 2
##   zona  new
##   <chr> <chr>
## 1 A     100/160
## 2 B     40/75
## 3 C     120/130
```

4. Combinando múltiples tablas de datos

Suponga que dispone de dos conjuntos de datos *data1* y *data2*, los cuales se combinarán usando la variable *id* que se denomina *key* y puede ser una o varias variables que identifican una observación y que se usan para conectar los datos de ambas tablas.

```
data1 <- tibble(
  id = c("a", "b", "c", "d"),
  x1 = c(10, 4, 7, 12),
)
data1
```

```
## # A tibble: 4 x 2
##   id       x1
##   <chr> <dbl>
## 1 a        10
## 2 b         4
## 3 c         7
## 4 d        12
```

```r
data2 <- tibble(
  id = c("a", "b", "c", "e"),
  x2 = c(3, 5, 13, 9),
)
data2
```

```
## # A tibble: 4 x 2
##   id       x2
##   <chr> <dbl>
## 1 a         3
## 2 b         5
## 3 c        13
## 4 e         9
```

A continuación se describen las siguientes funciones que existen para combinar datos de dos tablas:

- inner_join
- left_join
- right_join
- full_join

4.1. inner_join

Mantiene observaciones que se encuentran en ambas tablas.

```r
data1 %>% inner_join(data2, by = "id")
```

```
## # A tibble: 3 x 3
##   id       x1    x2
##   <chr> <dbl> <dbl>
## 1 a        10     3
## 2 b         4     5
## 3 c         7    13
```

4.2. left_join

Mantiene observaciones que se encuentran en la primera tabla.

```r
data1 %>% left_join(data2, by = "id")
```

```
## # A tibble: 4 x 3
##   id       x1    x2
##   <chr> <dbl> <dbl>
## 1 a        10     3
## 2 b         4     5
## 3 c         7    13
## 4 d        12    NA
```

4.3. right_join

Mantiene observaciones que se encuentran en la segunda tabla.

```
data1 %>% right_join(data2, by = "id")
```

```
## # A tibble: 4 x 3
##   id       x1    x2
##   <chr> <dbl> <dbl>
## 1 a        10     3
## 2 b         4     5
## 3 c         7    13
## 4 e        NA     9
```

4.4. full_join

Mantiene todas las observaciones que se encuentran en cada una de las tablas.

```
data1 %>% full_join(data2, by = "id")
```

```
## # A tibble: 5 x 3
##   id       x1    x2
##   <chr> <dbl> <dbl>
## 1 a        10     3
## 2 b         4     5
## 3 c         7    13
## 4 d        12    NA
## 5 e        NA     9
```

Otro tipo de match entre dos tablas de datos que no añade las variables presentes en ambas tablas es la que permiten:

- semi_join
- anti_join

4.5. semi_join

Mantiene las observaciones en *data1* que tienen un match en *data2*

```
data1 %>% semi_join(data2, by = "id")
```

```
## # A tibble: 3 x 2
##   id       x1
##   <chr> <dbl>
## 1 a        10
## 2 b         4
## 3 c         7
```

4.6. anti_join

Elimina las observaciones en *data1* que tienen un match en *data2*

```
data1 %>% anti_join(data2, by = "id")
```

```
## # A tibble: 1 x 2
##   id       x1
##   <chr> <dbl>
## 1 d        12
```