Herramientas Básicas de Minería de Datos

Juan C. Correa*

Cesa Business School, Bogotá, Colombia

En la minería de datos una de las primeras tareas por hacer es saber si las bases de datos que uno tiene están o no listas para usar. Veamos un ejemplo.

Algunas aplicaciones de la minería de datos

Aplicaciones de limpieza (tidyr)

Las primeras aplicaciones para hacer minería de datos con R dependen de la librería o paquete "tidyr" que forma parte del ecosistema de librerías "tidyverse".

```
library(tidyr)
library(tidyverse)
```

Ahora, vamos a ver cinco versiones de bases de datos que contienen los mismos datos, pero están dispuestos de forma diferente.

"Table 1" tiene cuatro columnas (country, year, cases, population) y seis filas.

```
data("table1")
head(table1)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

"Table 2" tiene 12 filas y cuatro columnas (country, year, type, count)

```
data("table2")
head(table2)
```

*Juan C. Correa can be contacted through: juan.correan@cesa.edu.co

```
## # A tibble: 6 x 4
##   country      year type           count
##   <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases          37737
## 6 Brazil       1999 population 172006362
```

"Table 3" tiene seis filas y tres columnas (country, year, rate)

```
data("table3")
head(table3)
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

"Table4a" tiene tres columnas (country, 1999, 2000) y tres filas.

```
data("table4a")
head(table4a)
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

"Table4b" tiene tres columnas (country, 1999, 2000).

```
data("table4b")
head(table4b)
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
##   <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

Si recordamos el slide 13 de la sesión 3B, será inmediatamente evidente que de todas estas tablas, la única que cumple con el principio de datos limpios es Table1, mientras que las demás son versiones sucias de Table1. Por fortuna, en R (y también en Python) existen sintaxis que le permiten limpiar bases de datos para dejarlas en formato limpio y listo para usar.

Aplicación 1: Gathering

Gathering (del verbo reunir en inglés) sirve para arreglar bases de datos que tienen columnas que en realidad muestran los valores de una variable no mencionada explícitamente. Ejemplo:

```
table4a %>%
gather(`1999`, `2000`, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Aplicación 2: Spreading

Spreading (del verbo esparcir en inglés) sirve para arreglar bases de datos que tienen observaciones o registros esparcidos en diversas filas (es la función opuesta a gathering). Ejemplo:

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Aplicación 3: Separate

Separate (del verbo separar en inglés) sirve para separar registros que muestran observaciones mezcladas de variables. Ejemplo:

```
table3 %>%
separate(rate, into = c("cases", "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Aplicación 4: Unite

Unite (del verbo unir en inglés) es el opuesto de separate. Ejemplo:

```
table5 %>%
unite(new, century, year)
```

```
## # A tibble: 6 x 3
##   country     new   rate
##   <chr>       <chr> <chr>
## 1 Afghanistan 19_99 745/19987071
## 2 Afghanistan 20_00 2666/20595360
## 3 Brazil      19_99 37737/172006362
## 4 Brazil      20_00 80488/174504898
## 5 China       19_99 212258/1272915272
## 6 China       20_00 213766/1280428583
```

Aplicaciones de datos relacionales

Las aplicaciones de datos relacionales en R dependen de la librería "dplyr" que también pertenece al ecosistema de librerías "tidyverse".

```
library(nycflights13)
```

Aplicación 5: Mutating joins

Mutating joins (del inglés unión mutante) sirve para combinar variables que provienen de dos tablas. En este ejemplo vamos a combinar variables dispuestas en la tabla flights con variables dispuestas en la tabla airlines, a través de la función "left_join".

```
flights2 <- flights %>%
select(year:day, hour, origin, dest, tailnum, carrier)
head(flights2)
```

```
## # A tibble: 6 x 8
##    year month   day  hour origin dest  tailnum carrier
##   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
## 1  2013     1     1     5 EWR    IAH   N14228  UA
## 2  2013     1     1     5 LGA    IAH   N24211  UA
## 3  2013     1     1     5 JFK    MIA   N619AA  AA
## 4  2013     1     1     5 JFK    BQN   N804JB  B6
## 5  2013     1     1     6 LGA    ATL   N668DN  DL
## 6  2013     1     1     5 EWR    ORD   N39463  UA
```

```
flights2 %>%
select(-origin, -dest) %>%
left_join(airlines, by = "carrier")
```

```
## # A tibble: 336,776 x 7
##     year month   day  hour tailnum carrier name
```

```
##    <int> <int> <int> <dbl> <chr>   <chr>   <chr>
##  1  2013     1     1     5 N14228  UA      United Air Lines Inc.
##  2  2013     1     1     5 N24211  UA      United Air Lines Inc.
##  3  2013     1     1     5 N619AA  AA      American Airlines Inc.
##  4  2013     1     1     5 N804JB  B6      JetBlue Airways
##  5  2013     1     1     6 N668DN  DL      Delta Air Lines Inc.
##  6  2013     1     1     5 N39463  UA      United Air Lines Inc.
##  7  2013     1     1     6 N516JB  B6      JetBlue Airways
##  8  2013     1     1     6 N829AS  EV      ExpressJet Airlines Inc.
##  9  2013     1     1     6 N593JB  B6      JetBlue Airways
## 10  2013     1     1     6 N3ALAA  AA      American Airlines Inc.
## # ... with 336,766 more rows
```

Para los próximos ejemplos, vamos a usar estas bases de datos de juguete:

```
x <- tribble(~key, ~val_x,
             1, "x1",
             2, "x2",
             3, "x3")

y <- tribble(~key, ~val_y,
             1, "y1",
             2, "y2",
             4, "y3")

x
```

```
## # A tibble: 3 x 2
##     key val_x
##   <dbl> <chr>
## 1     1 x1
## 2     2 x2
## 3     3 x3
```

```
y
```

```
## # A tibble: 3 x 2
##     key val_y
##   <dbl> <chr>
## 1     1 y1
## 2     2 y2
## 3     4 y3
```

Aplicación 6: Inner join

Inner join (del inglés unión interna) sirve para unir observaciones que compartan una misma variable clave (key).

```
x %>% inner_join(y, by = "key")
```

```
## # A tibble: 2 x 3
```

```
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

Aplicación 7: Outer join

Outer join (del inglés unión externa) sirve para mantener las observaciones o registros que aparecen en al menos una de las bases de datos. Hay tres tipos de outer joins: left_join (mantiene todas las observaciones de x), right_join (mantiene todas las observaciones de y), y full_join (mantiene todas las observaciones de x y de y). Estas funciones trabajan al añadir una observación virtual adicional en cada tabla, llenándola con un caso vacío o NA.

```
left_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
```

```
right_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     4 <NA>  y3
```

```
full_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
```