

R: Transformación de datos

Transformación de datos

Esta sección resume algunas de las funciones existentes para **transformar** datos en R. En particular, se utiliza la estrategia **SAC** (*split-apply-combine*) implementada en el paquete **dplyr** (**dplyr**). En la figura 1 podemos ver la etapa del análisis de datos correspondiente.

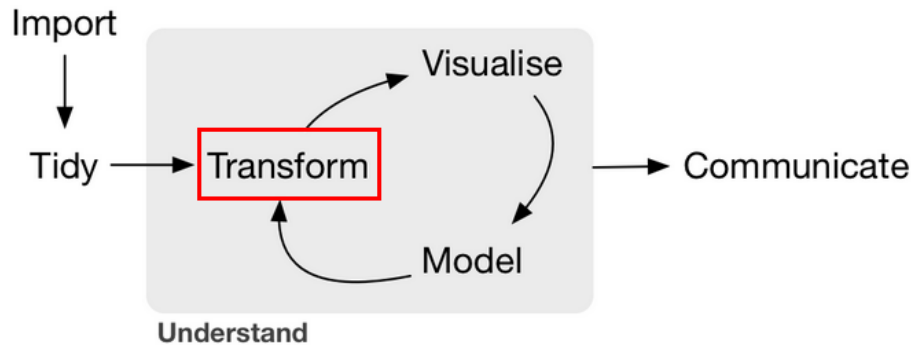


Figura 1: Transformación de datos **grolemund2016r**

Existen muchas maneras de transformar los datos y una gran cantidad de paquetes que implementan distintas funciones útiles para realizar esta tarea. En particular, resaltamos **dplyr** y **data.table**.

dplyr

El material que se presenta aquí está basado en los ejemplos y código presentado por Wickham en su tutorial de **useR** en el 2014.

5 verbos fundamentales implementados en dplyr

- **filter**: seleccionamos filas de acuerdo a los valores de las variables
- **select**: seleccionamos un subconjunto de las columnas utilizando los nombres de las variables
- **arrange**: reordenamos filas
- **mutate**: generar nuevas variables a partir de las variables originales
- **summarise**: reducir variables a valores

Todos estos verbos funcionan de la misma manera (tienen la misma estructura):

- El primer argumento de la función es un *data.frame*
- Los argumentos subsecuentes indican qué es lo que se debe hacer a ese *data.frame*
- Siempre regresa un *data.frame*

Para ejemplificar el uso de cada uno de los verbos, utilizaremos los siguientes conjuntos de datos de muestra, todos disponibles en el paquete **nycflights13** (**flights**).

Nota como utilizamos la función del paquete **readr** **read_csv**. Esta es una nueva implementación de **read.csv** pero mucho más rápida.

```
flights <- read_csv("data/flights.csv")
flights
```

```
## # A tibble: 227,496 × 14
##       date hour minute  dep  arr dep_delay arr_delay
##       <dtm> <int> <int> <int> <int>      <int>      <int>
## 1 2011-01-01 12:00:00   14     0 1400 1500         0        -10
## 2 2011-01-02 12:00:00   14     1 1401 1501         1         -9
## 3 2011-01-03 12:00:00   13    52 1352 1502        -8         -8
## 4 2011-01-04 12:00:00   14     3 1403 1513         3          3
## 5 2011-01-05 12:00:00   14     5 1405 1507         5         -3
## 6 2011-01-06 12:00:00   13    59 1359 1503        -1         -7
## 7 2011-01-07 12:00:00   13    59 1359 1509        -1         -1
## 8 2011-01-08 12:00:00   13    55 1355 1454        -5        -16
## 9 2011-01-09 12:00:00   14    43 1443 1554         43         44
## 10 2011-01-10 12:00:00  14    43 1443 1553         43         43
## # ... with 227,486 more rows, and 7 more variables: carrier <chr>,
## #   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
## #   dist <int>
```

```
# weather <- read_csv("data/weather.csv")
# weather
```

```
planes <- read_csv("data/planes.csv")
planes
```

```
## # A tibble: 2,853 × 9
##   plane year      mfr      model no.eng no.seats speed
##   <chr> <int>      <chr>      <chr> <int>   <int> <int>
## 1 N576AA 1991 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 2 N557AA 1993      MARZ BARRY   KITFOX IV     1      2   NA
## 3 N403AA 1974      RAVEN      S55A     NA      1   60
## 4 N492AA 1989 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 5 N262AA 1985 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 6 N493AA 1989 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 7 N477AA 1988 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 8 N476AA 1988 MCDONNELL DOUGLAS DC-9-82(MD-82)     2    172   NA
## 9 N504AA   NA AUTHIER ANTHONY P   TIERRA II     1      2   NA
## 10 N565AA 1987 MCDONNELL DOUGLAS DC-9-83(MD-83)     2    172   NA
## # ... with 2,843 more rows, and 2 more variables: engine <chr>, type <chr>
```

```
airports <- read_csv("data/airports.csv")
airports
```

```
## # A tibble: 3,376 × 7
##   iata      airport      city state country    lat
##   <chr>      <chr>      <chr> <chr>   <chr>    <dbl>
## 1 00M      Thigpen      Bay Springs  MS     USA 31.95376
## 2 00R Livingston Municipal Livingston TX     USA 30.68586
## 3 00V      Meadow Lake Colorado Springs CO     USA 38.94575
## 4 01G      Perry-Warsaw      Perry NY     USA 42.74135
## 5 01J      Hilliard Airpark Hilliard FL     USA 30.68801
## 6 01M      Tishomingo County Belmont MS     USA 34.49167
## 7 02A      Gragg-Wade      Clanton AL     USA 32.85049
## 8 02C      Capitol      Brookfield WI     USA 43.08751
## 9 02G      Columbiana County East Liverpool OH     USA 40.67331
## 10 03D      Memphis Memorial Memphis MO     USA 40.44726
## # ... with 3,366 more rows, and 1 more variables: long <dbl>
```

filter

Ya habíamos visto muchas maneras de extraer datos específicos de una base de datos de acuerdo a condiciones lógicas impuestas en los valores de las filas de columnas específicas. `filter` nos permite poner tantas condiciones como queramos de manera muy fácil y entendible por cualquiera que lea nuestro código.

Ejemplos

Busquemos todos los vuelos hacia SFO o OAK

```
filter(flights, dest == "SFO" | dest == "OAK")
```

```
## # A tibble: 3,508 × 14
##       date      hour minute  dep   arr dep_delay arr_delay
##       <dtm> <int>  <int> <int> <int>    <int>    <int>
## 1 2011-01-31 12:00:00     8    51  851 1052         1       -27
## 2 2011-01-31 12:00:00    11    29 1129 1351         4         1
## 3 2011-01-31 12:00:00    14    32 1432 1656         7         5
## 4 2011-01-31 12:00:00    17    48 1748 2001         3        -4
## 5 2011-01-31 12:00:00    21    43 2143 2338        50        24
## 6 2011-01-31 12:00:00     7    29  729 1002        -1         2
## 7 2011-01-31 12:00:00    15    58 1558 1812        -2        -8
## 8 2011-01-30 12:00:00     9    35  935 1203        45        49
## 9 2011-01-30 12:00:00    11    43 1143 1359        18        14
## 10 2011-01-30 12:00:00    14    59 1459 1715        34        24
## # ... with 3,498 more rows, and 7 more variables: carrier <chr>,
## #   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
## #   dist <int>
```

Los vuelos con retraso mayor a 5 horas

```
filter(flights, arr_delay > 5)
```

```
## # A tibble: 77,848 × 14
##       date      hour minute  dep   arr dep_delay arr_delay
##       <dtm> <int>  <int> <int> <int>    <int>    <int>
## 1 2011-01-09 12:00:00    14    43 1443 1554         43        44
## 2 2011-01-10 12:00:00    14    43 1443 1553         43        43
## 3 2011-01-11 12:00:00    14    29 1429 1539         29        29
## 4 2011-01-17 12:00:00    15    30 1530 1634         90        84
## 5 2011-01-20 12:00:00    15     7 1507 1622         67        72
## 6 2011-01-31 12:00:00    14    41 1441 1553         41        43
## 7 2011-01-13 12:00:00     7    22  722  841          2         6
## 8 2011-01-16 12:00:00     7    43  743  843         23         8
## 9 2011-01-17 12:00:00     7    24  724  842          4         7
## 10 2011-01-24 12:00:00     7    31  731  904         11        29
## # ... with 77,838 more rows, and 7 more variables: carrier <chr>,
## #   flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
## #   dist <int>
```

Podemos juntar las preguntas: vuelos con retraso mayor a 5 horas con destino a SFO o OAK

```
filter(flights, dest == "SFO" | dest == "OAK", arr_delay > 5)
```

```
## # A tibble: 1,581 × 14
##       date      hour minute  dep   arr dep_delay arr_delay
##       <dtm> <int>  <int> <int> <int>    <int>    <int>
## 1 2011-01-31 12:00:00    21    43 2143 2338        50        24
```

```
## 2 2011-01-30 12:00:00 9 35 935 1203 45 49
## 3 2011-01-30 12:00:00 11 43 1143 1359 18 14
## 4 2011-01-30 12:00:00 14 59 1459 1715 34 24
## 5 2011-01-30 12:00:00 17 49 1749 2011 4 6
## 6 2011-01-30 12:00:00 19 31 1931 2159 41 41
## 7 2011-01-30 12:00:00 21 0 2100 2320 10 9
## 8 2011-01-29 12:00:00 8 52 852 1126 2 12
## 9 2011-01-29 12:00:00 14 42 1442 1655 17 9
## 10 2011-01-29 12:00:00 18 9 1809 2021 14 11
## # ... with 1,571 more rows, and 7 more variables: carrier <chr>,
## # flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
## # dist <int>
```

select

Podemos ahora, mas facilmente, quedarnos con unicamente ciertas variables. `select` esta implementado de tal manera que funciona *nombrando* las variables que se quieren utilizar.

```
select(flights, flight, dest)
```

```
## # A tibble: 227,496 × 2
##   flight dest
##   <int> <chr>
## 1    428 DFW
## 2    428 DFW
## 3    428 DFW
## 4    428 DFW
## 5    428 DFW
## 6    428 DFW
## 7    428 DFW
## 8    428 DFW
## 9    428 DFW
## 10   428 DFW
## # ... with 227,486 more rows
```

Tambien podemos especificar que queremos todas las variables *menos* algunas.

```
select(flights, -date, -hour, -minute, -dep, -arr, -carrier, -flight)
```

```
## # A tibble: 227,496 × 7
##   dep_delay arr_delay dest plane cancelled time dist
##   <int> <int> <chr> <chr> <int> <int> <int>
## 1      0     -10 DFW N576AA      0    40  224
## 2      1      -9 DFW N557AA      0    45  224
## 3     -8      -8 DFW N541AA      0    48  224
## 4      3       3 DFW N403AA      0    39  224
## 5      5      -3 DFW N492AA      0    44  224
## 6     -1      -7 DFW N262AA      0    45  224
## 7     -1      -1 DFW N493AA      0    43  224
## 8     -5     -16 DFW N477AA      0    40  224
## 9     43     44 DFW N476AA      0    41  224
## 10    43     43 DFW N504AA      0    45  224
## # ... with 227,486 more rows
```

Podemos pedir las variables que empiezan con algun caracter.

```
select(flights, starts_with("d"))
```

```
## # A tibble: 227,496 × 5
##       date    dep dep_delay dest  dist
##       <dtm> <int>    <int> <chr> <int>
## 1 2011-01-01 12:00:00 1400      0 DFW   224
## 2 2011-01-02 12:00:00 1401      1 DFW   224
## 3 2011-01-03 12:00:00 1352     -8 DFW   224
## 4 2011-01-04 12:00:00 1403      3 DFW   224
## 5 2011-01-05 12:00:00 1405      5 DFW   224
## 6 2011-01-06 12:00:00 1359     -1 DFW   224
## 7 2011-01-07 12:00:00 1359     -1 DFW   224
## 8 2011-01-08 12:00:00 1355     -5 DFW   224
## 9 2011-01-09 12:00:00 1443     43 DFW   224
## 10 2011-01-10 12:00:00 1443     43 DFW   224
## # ... with 227,486 more rows
```

O las que contienen algun patron

```
select(flights, contains("dep"))
```

```
## # A tibble: 227,496 × 2
##       dep dep_delay
##       <int>    <int>
## 1    1400         0
## 2    1401         1
## 3    1352        -8
## 4    1403         3
## 5    1405         5
## 6    1359        -1
## 7    1359        -1
## 8    1355        -5
## 9    1443        43
## 10   1443        43
## # ... with 227,486 more rows
```

arrange

`order` habiamos visto que es la implementacion del base para ordenar vectores o en su defecto, dataframes de acuerdo a valores de vectores en esta. Sin embargo, es engorrosa la manera de llamarlo.

Podemos arreglar los valores de las tablas, facilmente con `arrange`. Por ejemplo, podemos ver los 5 vuelos con mayor retraso de llegada.

```
head(arrange(flights, desc(arr_delay)), n=5)
```

```
## # A tibble: 5 × 14
##       date    hour minute  dep  arr dep_delay arr_delay carrier
##       <dtm> <int>  <int> <int> <int>    <int>    <int>    <chr>
## 1 2011-12-12 12:00:00     6    50  650  808      970      978    AA
## 2 2011-08-01 12:00:00     1    56  156  452      981      957    CO
## 3 2011-11-08 12:00:00     7    21  721  948      931      918    MQ
## 4 2011-06-21 12:00:00    23    34 2334  124      869      861    UA
## 5 2011-05-20 12:00:00     8    58  858 1027      803      822    MQ
## # ... with 6 more variables: flight <int>, dest <chr>, plane <chr>,
```

```
## # cancelled <int>, time <int>, dist <int>
```

O los 5 con menor atraso de llegada

```
head(arrange(flights, arr_delay), n=5)
```

```
## # A tibble: 5 × 14
##       date   hour minute  dep   arr dep_delay arr_delay carrier
##       <dtm> <int>  <int> <int> <int>    <int>    <int>    <chr>
## 1 2011-07-03 12:00:00    19    14 1914 2039      -1      -70      XE
## 2 2011-12-25 12:00:00     7    41  741  926      -4      -57      00
## 3 2011-08-21 12:00:00     9    35  935 1039     -10      -56      00
## 4 2011-08-31 12:00:00     9    34  934 1039     -11      -56      00
## 5 2011-08-26 12:00:00    21     7 2107 2205      -3      -55      00
## # ... with 6 more variables: flight <int>, dest <chr>, plane <chr>,
## # cancelled <int>, time <int>, dist <int>
```

Podemos arreglar primero por destino y luego por retraso de llegada.

```
arrange(flights, dest, arr_delay)
```

```
## # A tibble: 227,496 × 14
##       date   hour minute  dep   arr dep_delay arr_delay
##       <dtm> <int>  <int> <int> <int>    <int>    <int>
## 1 2011-11-25 12:00:00    12    55 1255 1344      0      -26
## 2 2011-01-12 12:00:00     8    56  856 1000     -14      -25
## 3 2011-12-25 12:00:00    19    50 1950 2045      -5      -25
## 4 2011-03-16 12:00:00    17    39 1739 1841      -8      -24
## 5 2011-03-17 12:00:00    11    17 1117 1214      -3      -24
## 6 2011-12-30 12:00:00    17    27 1727 1828      -3      -24
## 7 2011-01-29 12:00:00    17    32 1732 1837      -3      -23
## 8 2011-05-29 12:00:00    18    11 1811 1902      -4      -23
## 9 2011-02-13 12:00:00    17    34 1734 1843      -1      -22
## 10 2011-04-17 12:00:00    17    18 1718 1818      -7      -22
## # ... with 227,486 more rows, and 7 more variables: carrier <chr>,
## # flight <int>, dest <chr>, plane <chr>, cancelled <int>, time <int>,
## # dist <int>
```

mutate

Muchas veces lo que se desea es generar nuevas variables utilizando funciones sobre las variables de la tabla.

Por ejemplo, queremos saber cual fue el vuelo mas rapido. Para esto queremos calcular la velocidad promedio del vuelo.

```
select(arrange(mutate(flights, velocidad = dist/time), desc(velocidad)),
       flight, dest, velocidad)
```

```
## # A tibble: 227,496 × 3
##   flight dest velocidad
##   <int> <chr>    <dbl>
## 1  1646  AUS  12.72727
## 2  5229  MEM  11.16667
## 3   944  CLT  10.74118
## 4 4634  HOB  10.65957
## 5   500  IND  10.30488
## 6   106  EWR  10.14493
```

```
## 7      644   CLE  10.10185
## 8     1074   CLE  10.10185
## 9     1054   EWR  10.07194
## 10    1424   DCA  10.06667
## # ... with 227,486 more rows
```

Esta manera de transformar a los datos (utilizando varios de los verbos) es confusa y difícil de leer. Es mas sencillo utilizar el operador pipe de R implementado en el paquete `magrittr`, es decir, `%>%`.

```
flights2 <- mutate(flights, velocidad = dist/time) %>%
  arrange(., desc(velocidad)) %>%
  select(., flight, dest, velocidad)

flights2
```

```
## # A tibble: 227,496 × 3
##   flight dest velocidad
##   <int> <chr>      <dbl>
## 1    1646  AUS  12.72727
## 2    5229  MEM  11.16667
## 3     944  CLT  10.74118
## 4    4634  HOB  10.65957
## 5     500  IND  10.30488
## 6     106  EWR  10.14493
## 7     644  CLE  10.10185
## 8    1074  CLE  10.10185
## 9    1054  EWR  10.07194
## 10   1424  DCA  10.06667
## # ... with 227,486 more rows
```

La lectura es mucho mas sencilla de esta forma. Recuerden, muchas veces los lectores de su código serán ustedes en el futuro.

summarise

Ahora, si queremos saber el promedio de velocidad de los vuelos por destino, podemos calcularlo facilmente con `group_by` y `summarise`.

```
flights2 %>% group_by(dest) %>%
  summarise(vel_prom = mean(velocidad, na.rm = T))
```

```
## # A tibble: 116 × 2
##   dest vel_prom
##   <chr>      <dbl>
## 1  ABQ  6.878846
## 2  AEX  5.587412
## 3  AGS  7.970874
## 4  AMA  6.656453
## 5  ANC  8.150118
## 6  ASE  6.672732
## 7  ATL  7.373678
## 8  AUS  4.919712
## 9  AVL  7.718733
## 10 BFL  7.489401
## # ... with 106 more rows
```

Joins

Muchas veces la información se tiene repartida entre diferentes tablas pero es necesario juntar las variables de las diferentes observaciones en una sola tabla para modelarlas o describirlas. Es muy estándar, en el lenguaje SQL, el tipo de joins que se pueden utilizar. La figura 2 muestra un resumen del tipo de joins que pueden realizarse.

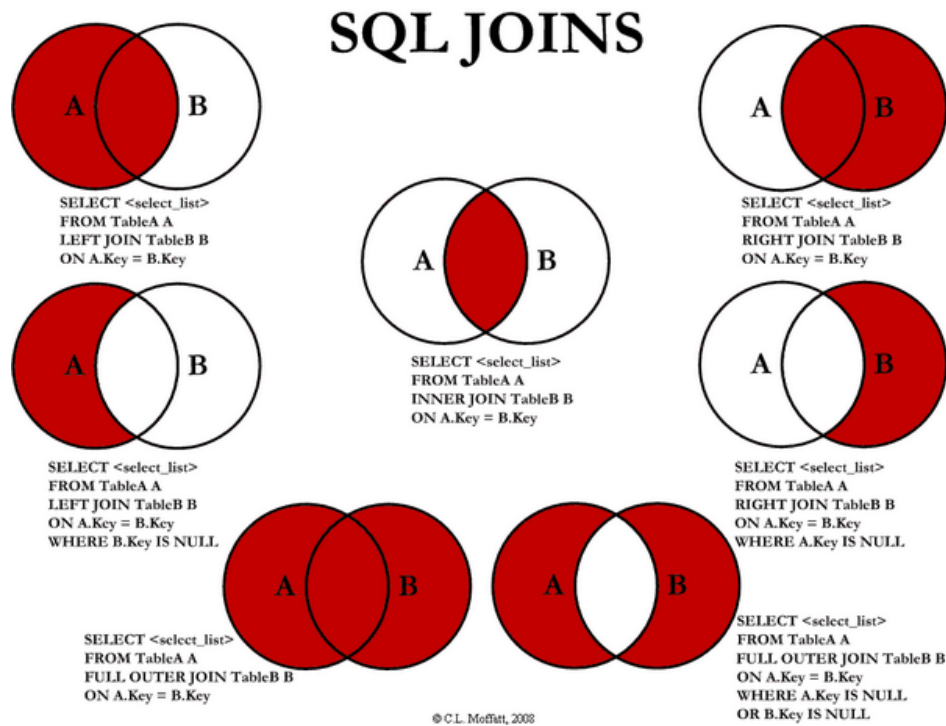


Figura 2: Joins en el lenguaje SQL

El paquete `dplyr` implementa estos joins de manera natural, utilizando la lógica de SQL.

- `inner_join`: regresa todas las filas de x en donde hay valores correspondientes para y, junto con todas las columnas.
- `left_join`: regresa todas las filas de x, rellenando con NA para valores que no encuentro en y.
- `right_join`: regresa todas las filas de y, rellenando con NA para valores que no encuentro en x.
- `full_join`: regresa todas las filas y todas las columnas para x y y. Donde no hay valores en alguno de los dos, rellena con NA.
- `semi_join`: regresa todas las filas de x para las que hay valores en y regresando únicamente las columnas de x.
- `anti_join`: regresa todas las filas de x donde no hay valores en y, manteniendo solo las columnas de x.

Ahora, supongamos que queremos saber la velocidad promedio de los aviones que tenemos en nuestros datos para todos sus vuelos.

```
# base de aviones con velocidad

vel_aviones <- flights %>% group_by(plane) %>%
  summarise(vel_prom = mean(dist/time, na.rm = T))

inner_join(
  planes,
```



```

    vel_aviones
  ) %>%
    select(plane, year, vel_prom) %>%
    arrange(desc(vel_prom))

```

```

## # A tibble: 2,853 × 3
##   plane  year vel_prom
##   <chr> <int>   <dbl>
## 1 N653JB  2007  9.333333
## 2 N709UW  1999  9.316327
## 3 N3744F  2001  9.070175
## 4 N3769L  2002  9.065789
## 5 N623JB  2005  9.037975
## 6 N607JB  2005  8.936351
## 7 N580JB  2003  8.869907
## 8 N658JB  2007  8.869565
## 9 N589JB  2004  8.814815
## 10 N760JB 2008  8.788441
## # ... with 2,843 more rows

```

Ahora, queremos saber los destinos con mayores retrasos.

```

destinos <- flights %>% group_by(dest) %>%
  summarise(retraso = mean(arr_delay, na.rm = T))

inner_join(
  airports,
  destinos,
  by = c("iata" = "dest")
) %>%
  arrange(desc(retraso))

```

```

## # A tibble: 114 × 8
##   iata      airport      city state
##   <chr>      <chr>      <chr> <chr>
## 1 ANC Ted Stevens Anchorage International Anchorage AK
## 2 CID      Eastern Iowa Cedar Rapids IA
## 3 DSM      Des Moines International Des Moines IA
## 4 SFO San Francisco International San Francisco CA
## 5 BPT Southeast Texas Regional Beaumont/Port Arthur TX
## 6 GRR Kent County International Grand Rapids MI
## 7 DAY James M Cox Dayton Intl Dayton OH
## 8 VPS Eglin Air Force Base Valparaiso FL
## 9 SAV Savannah International Savannah GA
## 10 RIC Richmond International Richmond VA
## # ... with 104 more rows, and 4 more variables: country <chr>, lat <dbl>,
## #   long <dbl>, retraso <dbl>

```

¿Cuáles son los aeropuertos que SI estan en la base de destinos? ¿Cuáles son los aeropuertos que NO estan en la base de destinos?