Exploration and Application of R for Data Science

Alejandro Pachón, Santiago Meza, Alexander Morgan

2023-02-25

GitHub

Puedes visitar nuestro repositorio en internet, para más información : **Nuestro Repositorio**

Numeros primos

En este código, la línea for (x in 1:100){, identifica con la variable "x", desde el valor 1 al 100. Luego, se declara "Nprimo" como TRUE, para que en cuyo caso de que el número a valorar sea primo, el código posteriormente lo imprima sin necesidad de volver a preguntar el numero almacenado, después se abre un nuevo 'for' en donde se encuentran dos "if", para que luego de realizar la operación, identifique si es o no un numero primo. Conociendo que, si la división es menor al valor que este en el "for N", tomara la variable N el valor de x para salir del "for" y continuar a imprimir el número. De lo contrario, si el módulo es diferente de "0", Nprimo pasará a ser falso y no imprimirá el numero en la consola. Y así consecutivamente hasta llegar al número 100.

```
# Números primos

for (x in 1:100){
  Nprimo <- TRUE
  for(N in 2:x){
    mid=x/2
    if(N>mid){
      N=x
    }
    if(N!=x & x%%N==0){
      Nprimo <- FALSE
    }
  }
  if(Nprimo==TRUE & x!=1){
    print(x)
  }
}
```

[1] 2 [1] 3 [1] 5 [1] 7 [1] 11 [1] 13 [1] 17 [1] 19 [1] 23 [1] 29 [1] 31 [1] 37 [1] 41 [1] 43 [1] 47 [1] 53 [1] 59 [1] 61 [1] 67 [1] 71 [1] 73 [1] 79 [1] 83 [1] 89 [1] 97

2. Uso básico de la libreria Tidyverse

Los ejercicios propuestos en el documento se tomaron del apartado **Data transformation**, de la página *R for Data Science*.

5.2.4:

1. Encuentra todos los vuelos que:

- **Item 1:** Tuvieron un retraso de llegada de dos o más horas.

Primero identificamos la variable de retraso de llegada, en la base de datos es `arr_delay`, también identificamos que los datos que necesitamos son un conjunto de filas, por esto se toma la función `filter()` y se agrega la condición que necesitamos para tomar los vuelos que tuvieron un retraso de llegada de dos horas o más. De esta forma tendríamos que:

```
library(nycflights13)
library(dplyr)
filter(flights,arr_delay>119)
```

```
## # A tibble: 10,200 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      811        630     101    1047     830     137 MQ
##  2  2013     1     1      848       1835     853    1001    1950     851 MQ
##  3  2013     1     1      957        733     144    1056     853     123 UA
##  4  2013     1     1     1114        900     134    1447    1222     145 UA
##  5  2013     1     1     1505       1310     115    1638    1431     127 EV
##  6  2013     1     1     1525       1340     105    1831    1626     125 B6
##  7  2013     1     1     1549       1445      64    1912    1656     136 EV
##  8  2013     1     1     1558       1359     119    1718    1515     123 EV
##  9  2013     1     1     1732       1630      62    2028    1825     123 EV
## 10  2013     1     1     1803       1620     103    2008    1750     138 MQ
## # ... with 10,190 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

- **item 2:** Voló a Houston (IAH o HOU).

Se identifica la variable que indique el destino de los vuelos y esta se le da la condición a la cual queremos filtrar en este caso se define con 2 nombres diferentes al destino del vuelo, por lo tanto en la condición de `filter()` debe ser considerado para que tome ambas nomenclaturas, para esto se agrega la operación lógica `|`. De esta forma tendríamos que:

```
library(nycflights13)
library(dplyr)
filter(flights, dest=="IAH"|dest=="HOU" )
```

```
## # A tibble: 9,313 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      623        627      -4     933     932       1 UA
##  4  2013     1     1      728        732      -4    1041    1038       3 UA
##  5  2013     1     1      739        739       0    1104    1038      26 UA
##  6  2013     1     1      908        908       0    1228    1219       9 UA
##  7  2013     1     1     1028       1026       2    1350    1339      11 UA
##  8  2013     1     1     1044       1045      -1    1352    1351       1 UA
##  9  2013     1     1     1114        900     134    1447    1222     145 UA
## 10  2013     1     1     1205       1200       5    1503    1505      -2 UA
```

```
## # ... with 9,303 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

- **Item 3:** Fueron operados por United, American o Delta.

Lo primero es identificar las siglas de las aerolíneas que nos dan en este caso UA, AA y DL respectivamente, después tomamos la variable `carrier` y asignamos las siglas mediante una función `filter()`, de esta manera tendremos los vuelos operados por dichas aerolíneas, y como en el caso anterior se agrega la operación lógica `|` Para cumplir la condición de que se filtren cualquiera de las 3 aerolíneas. De esta forma tendríamos que:

```
       library(nycflights13)
       library(dplyr)
 filter(flights, carrier=="AA"|carrier=="DL"|carrier=="UA")
```

```
## # A tibble: 139,504 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      554        600      -6     812     837     -25 DL
##  5  2013     1     1      554        558      -4     740     728      12 UA
##  6  2013     1     1      558        600      -2     753     745       8 AA
##  7  2013     1     1      558        600      -2     924     917       7 UA
##  8  2013     1     1      558        600      -2     923     937     -14 UA
##  9  2013     1     1      559        600      -1     941     910      31 AA
## 10  2013     1     1      559        600      -1     854     902      -8 UA
## # ... with 139,494 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

- **Item 4**: Voló en verano (Julio, Agosto y Septiembre).

Para la implementación este ítem se identifica que los meses están definidos en la variable `month` y que los meses son almacenados por el orden numérico de estos, por esto se identifican los meses de verano como 7,8 y 9 correspondientes a julio, agosto y septiembre respectivamente, para la sintaxis de la condición se usa la función `filter()` y la función `%in%`, de esta manera mediante `%in%` se utiliza para verificar si los elementos de un conjunto de datos están presentes. De esta forma tendríamos que:

```
library(nycflights13)
library(dplyr)
filter(flights,month%in%c(7,8,9))
```

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     7     1        1       2029     212     236    2359     157 B6
##  2  2013     7     1        2       2359       3     344     344       0 B6
##  3  2013     7     1       29       2245     104     151       1     110 B6
##  4  2013     7     1       43       2130     193     322      14     188 B6
##  5  2013     7     1       44       2150     174     300     100     120 AA
##  6  2013     7     1       46       2051     235     304    2358     186 B6
```

```
##  7  2013     7     1       48       2001     287     308    2305     243 VX
##  8  2013     7     1       58       2155     183     335      43     172 B6
##  9  2013     7     1      100       2146     194     327      30     177 B6
## 10  2013     7     1      100       2245     135     337     135     122 B6
## # ... with 86,316 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

- **Item 5**: LLegó más de dos horas tarde, pero no se retraso.

Para este punto se toma la condición del Ítem 1 ya que se mencionan nuevamente los vuelos con más de 2 horas de retraso en llegar, y para la segunda condición se identifica la variable `dep_delay` como la que toma los datos de los vuelos retrasados, y se agrega el operador lógico `&` para combinar las condiciones y filtrar los vuelos que cumplan con las condiciones. De esta forma tendríamos que:

```
library(nycflights13)
library(dplyr)
filter(flights, arr_delay>119&dep_delay==0)
```

```
## # A tibble: 3 x 19
##    year month   day dep_time sched_dep~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##   <int> <int> <int>    <int>       <int>   <dbl>   <int>   <int>   <dbl> <chr>
## 1  2013    10     7     1350        1350       0    1736    1526     130 EV
## 2  2013     5    23     1810        1810       0    2208    2000     128 MQ
## 3  2013     7     1      905         905       0    1443    1223     140 DL
## # ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>, and abbreviated variable names 1: sched_dep_time,
## #   2: dep_delay, 3: arr_time, 4: sched_arr_time, 5: arr_delay
```

- **Item 6**: Se retrasaron al menos una hora, pero estuvieron más de 30 minutos en vuelo.

Con la lógica planteada en el Ítem 5, cambiamos las variables para cumplir con las condiciones, en este caso solo se necesitan los vuelos con un retraso de 1 hora, pero que también volaron por más de 30 minutos, para esto usamos la variable `air_time` y se asigna un tiempo menor o igual 30 minutos. De esta forma tendríamos que:

```
library(nycflights13)
library(dplyr)
filter(flights, arr_delay<=60 & air_time<=30)
```

```
## # A tibble: 1,194 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1     1318       1322      -4    1358    1416     -18 EV
##  2  2013     1     1     2000       2000       0    2054    2110     -16 9E
##  3  2013     1     1     2116       2110       6    2202    2212     -10 EV
##  4  2013     1     1     2302       2200      62    2342    2253      49 EV
##  5  2013     1     2      602        600       2     646     659     -13 US
##  6  2013     1     2      743        745      -2     858     857       1 9E
##  7  2013     1     2     1335       1322      13    1414    1416      -2 EV
##  8  2013     1     2     1606       1610      -4    1730    1729       1 9E
##  9  2013     1     2     2003       2015     -12    2102    2125     -23 9E
## 10  2013     1     2     2125       2110      15    2221    2212       9 EV
## # ... with 1,184 more rows, 9 more variables: flight <int>, tailnum <chr>,
```

```
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

- **Item 7**: Salió entre la medianoche y las 6 am.

Con la función `filter()`se definen un rango de valores entre las 0 (media noche) y 6 (6 de la mañana) e incluyendo los límites del rango, aplicamos esta lógica a la variable `hour`, así tendríamos los vuelos entre media noche y las 6 am.

```
library(nycflights13)
library(dplyr)
filter(flights, hour>=0 & hour<=6)
```

```
## # A tibble: 27,905 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # ... with 27,895 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

5.2.4

2. Otro ayudante de filtrado de dplyr útil es between(). ¿Qué hace? ¿Puede usarse para simplificar los códigos del anterior punto?

La función `between()` realiza un rango de valores partiendo de una variable, de esta forma puede cambiar la estructura de varios ejercicios vistos anteriormente, un ejemplo utilizando el Ítem 7 del punto anterior seria:

```
library(nycflights13)
library(dplyr)
filter(flights, between(hour, 0,6))
```

```
## # A tibble: 27,905 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
```

```
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # ... with 27,895 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

Dónde como se ve, al usar la función `between()` cambia totalmente la estructura interna de `filter()`, debido a que el uso de esta hace innecesario en uso de la condición de `<, >, => y <=`.

5.3.1:

1. ¿Cómo pudiste usar arrange() para ordenar todos los valores faltantes al principio?

`arrange()` funciona de manera similar a `filter()` excepto que en lugar de seleccionar filas, cambia su orden, y con el uso de `desc()`que permite ordenar columnas por orden descendente y de `is.na()` que comprueba si un valor es NA podemos creas la siguiente sintaxis que ordena al principio de una tabla los valores faltantes.

```
library(nycflights13)
library(dplyr)
arrange(flights, desc(is.na(dep_delay)))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1       NA       1630      NA      NA    1815      NA EV
##  2  2013     1     1       NA       1935      NA      NA    2240      NA AA
##  3  2013     1     1       NA       1500      NA      NA    1825      NA AA
##  4  2013     1     1       NA        600      NA      NA     901      NA B6
##  5  2013     1     2       NA       1540      NA      NA    1747      NA EV
##  6  2013     1     2       NA       1620      NA      NA    1746      NA EV
##  7  2013     1     2       NA       1355      NA      NA    1459      NA EV
##  8  2013     1     2       NA       1420      NA      NA    1644      NA EV
##  9  2013     1     2       NA       1321      NA      NA    1536      NA EV
## 10  2013     1     2       NA       1545      NA      NA    1910      NA AA
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

2. Ordena `flights` para encontrar los vuelos más retrasados. Y los vuelos que salieron antes.

Para ordenar `flights` de forma que podamos encontrar los vuelos más retrasados y los que despegaron lo más antes posible, tendremos que ordenar la variable `dep_delay` de forma ascendente y descendente, de esta forma se ordenan primero de forma decente para hallar el vuelo que más se retrasó.

```
library(nycflights13)
library(dplyr)
arrange(flights,desc(dep_delay))  # Más retrasado
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
```

```
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     9      641        900    1301    1242    1530    1272 HA
##  2  2013     6    15     1432       1935    1137    1607    2120    1127 MQ
##  3  2013     1    10     1121       1635    1126    1239    1810    1109 MQ
##  4  2013     9    20     1139       1845    1014    1457    2210    1007 AA
##  5  2013     7    22      845       1600    1005    1044    1815     989 MQ
##  6  2013     4    10     1100       1900     960    1342    2211     931 DL
##  7  2013     3    17     2321        810     911     135    1020     915 DL
##  8  2013     6    27      959       1900     899    1236    2226     850 DL
##  9  2013     7    22     2257        759     898     121    1026     895 DL
## 10  2013    12     5      756       1700     896    1058    2020     878 AA
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

Después eliminamos la función `desc()` para que el orden sea de manera ascendente y así el primer vuelo en la tabla será el vuelo que salió lo antes posible.

```
library(nycflights13)
library(dplyr)
arrange(flights,dep_delay)  # Más adelantado
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013    12     7     2040       2123     -43      40    2352      48 B6
##  2  2013     2     3     2022       2055     -33    2240    2338     -58 DL
##  3  2013    11    10     1408       1440     -32    1549    1559     -10 EV
##  4  2013     1    11     1900       1930     -30    2233    2243     -10 DL
##  5  2013     1    29     1703       1730     -27    1947    1957     -10 F9
##  6  2013     8     9      729        755     -26    1002     955       7 MQ
##  7  2013    10    23     1907       1932     -25    2143    2143       0 EV
##  8  2013     3    30     2030       2055     -25    2213    2250     -37 MQ
##  9  2013     3     2     1431       1455     -24    1601    1631     -30 9E
## 10  2013     5     5      934        958     -24    1225    1309     -44 B6
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

3. Ordenar flights para encontrar los vuelos más rápidos.

Para encontrar los vuelos más rápidos usamos `arrange()` con la variable `air_time`.

```
library(nycflights13)
library(dplyr)
arrange(flights,air_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1    16     1355       1315      40    1442    1411      31 EV
##  2  2013     4    13      537        527      10     622     628      -6 EV
```

```
##  3  2013    12     6      922        851      31    1021     954      27 EV
##  4  2013     2     3     2153       2129      24    2247    2224      23 EV
##  5  2013     2     5     1303       1315     -12    1342    1411     -29 EV
##  6  2013     2    12     2123       2130      -7    2211    2225     -14 EV
##  7  2013     3     2     1450       1500     -10    1547    1608     -21 US
##  8  2013     3     8     2026       1935      51    2131    2056      35 9E
##  9  2013     3    18     1456       1329      87    1533    1426      67 EV
## 10  2013     3    19     2226       2145      41    2305    2246      19 EV
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

4. ¿Qué vuelos viajaron más lejos? ¿Cuál viajó menos?.

Para encontrar los vuelos que viajaron más usamos `arrange()` con la variable `distance` y lo ordenamos con `desc()`.

```
library(nycflights13)
library(dplyr)
arrange(flights,desc(distance)) # Vuelo más largo
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      857        900      -3    1516    1530     -14 HA
##  2  2013     1     2      909        900       9    1525    1530      -5 HA
##  3  2013     1     3      914        900      14    1504    1530     -26 HA
##  4  2013     1     4      900        900       0    1516    1530     -14 HA
##  5  2013     1     5      858        900      -2    1519    1530     -11 HA
##  6  2013     1     6     1019        900      79    1558    1530      28 HA
##  7  2013     1     7     1042        900     102    1620    1530      50 HA
##  8  2013     1     8      901        900       1    1504    1530     -26 HA
##  9  2013     1     9      641        900    1301    1242    1530    1272 HA
## 10  2013     1    10      859        900      -1    1449    1530     -41 HA
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

Para encontrar el vuelo que viajaro menos, quitamos `desc()` del código anterior y así toman el orden normal.

```
library(nycflights13)
library(dplyr)
arrange(flights,distance) # Vuelo más largo
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     7    27       NA        106      NA      NA     245      NA US
##  2  2013     1     3     2127       2129      -2    2222    2224      -2 EV
##  3  2013     1     4     1240       1200      40    1333    1306      27 EV
##  4  2013     1     4     1829       1615     134    1937    1721     136 EV
##  5  2013     1     4     2128       2129      -1    2218    2224      -6 EV
```

```
##  6  2013     1     5     1155       1200      -5    1241    1306     -25 EV
##  7  2013     1     6     2125       2129      -4    2224    2224       0 EV
##  8  2013     1     7     2124       2129      -5    2212    2224     -12 EV
##  9  2013     1     8     2127       2130      -3    2304    2225      39 EV
## 10  2013     1     9     2126       2129      -3    2217    2224      -7 EV
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

5.4.1:

2. ¿Qué sucede si se incluye el nombre de una variable varias veces en select()?

Cualquier variable duplicada solo se incluye una vez. Para comprobar esto, tenemos el siguiente ejemplo.

```
library(nycflights13)
library(dplyr)
select(flights, year, day, year, year, day, dest, dep_delay)
```

```
## # A tibble: 336,776 x 4
##     year   day dest  dep_delay
##    <int> <int> <chr>     <dbl>
##  1  2013     1 IAH           2
##  2  2013     1 IAH           4
##  3  2013     1 MIA           2
##  4  2013     1 BQN          -1
##  5  2013     1 ATL          -6
##  6  2013     1 ORD          -4
##  7  2013     1 FLL          -5
##  8  2013     1 IAD          -3
##  9  2013     1 MCO          -3
## 10  2013     1 ORD          -2
## # ... with 336,766 more rows
```

3. Qué hace la función any_of()? ¿Por qué podría ser útil en conjunto con este vector?

`vars <- c("year", "month", "day", "dep_delay", "arr_delay")`

La función `one_of()` se utiliza para seleccionar solo las columnas del conjunto de datos que se encuentran en un vector, en este caso el vector "`vars`", también con esta función se puede simplificar la función `select()`.

```
library(nycflights13)
library(dplyr)
vars<-c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, one_of(vars))
```

```
## # A tibble: 336,776 x 5
##     year month   day dep_delay arr_delay
##    <int> <int> <int>     <dbl>     <dbl>
##  1  2013     1     1         2        11
##  2  2013     1     1         4        20
##  3  2013     1     1         2        33
##  4  2013     1     1        -1       -18
```

```
##  5  2013     1     1        -6       -25
##  6  2013     1     1        -4        12
##  7  2013     1     1        -5        19
##  8  2013     1     1        -3       -14
##  9  2013     1     1        -3        -8
## 10  2013     1     1        -2         8
## # ... with 336,766 more rows
```

4. ¿Te sorprende el resultado de ejecutar el siguiente código? `select(flights, contains("TIME"))`

Es sorprendente, es ingenioso que mediante una línea de código se podía realizar una "búsqueda" de esta forma. ya que realizar una búsqueda/filtración de variables mediante una sentencia es interesante.

```
library(nycflights13)
library(dplyr)
select(flights, contains("time"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##       <int>          <int>    <int>          <int>    <dbl> <dttm>
##  1      517            515      830            819      227 2013-01-01 05:00:00
##  2      533            529      850            830      227 2013-01-01 05:00:00
##  3      542            540      923            850      160 2013-01-01 05:00:00
##  4      544            545     1004           1022      183 2013-01-01 05:00:00
##  5      554            600      812            837      116 2013-01-01 06:00:00
##  6      554            558      740            728      150 2013-01-01 05:00:00
##  7      555            600      913            854      158 2013-01-01 06:00:00
##  8      557            600      709            723       53 2013-01-01 06:00:00
##  9      557            600      838            846      140 2013-01-01 06:00:00
## 10      558            600      753            745      138 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

5.5.2:

1. Convertir `dep_time` y `sched_dep_time` en una representación de minutos desde la media noche.

Para obtener los horarios de salida en minutos, se divide `dep_time` en 100 para obtener las horas desde la medianoche y multiplicar por 60 para tener los minutos, después sumar el resto de `dep_time` dividido por 100. como ejemplo podemos usar la ora 14:05 (1405 o 2:05Pm).

```
1405 %/% 100 * 60 + 1405 %% 100
```

```
## [1] 845
```

Sin embargo, para que todas las horas se puedan convertir se necita hacer la operación `&&` del resultado con `1440`. esto debido a que si se ingresa un valor de medida noche (2400) no se toma como el resultado de 0 como se espera, si no que da como resultado `1440`. Después se remplaza el valor ejemplo por la variable a convertir, sin embargo para realizar dicha conversión se requiere el uso de la función `mutate()`, adicionalmente, se crean dos nuevas columnas que representaran dicha conversión de los datos. Así se obtiene que:

```
library(nycflights13)
library(dplyr)
mutate(flights, dep_time_min=(dep_time%/%100*60+dep_time%%100)%%1440,
sched_dep_time_min=(sched_dep_time%/%100*60+sched_dep_time%%100)%%1440)
```

```
## # A tibble: 336,776 x 21
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # ... with 336,766 more rows, 11 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, dep_time_min <dbl>,
## #   sched_dep_time_min <dbl>, and abbreviated variable names 1: sched_dep_time,
## #   2: dep_delay, 3: arr_time, 4: sched_arr_time, 5: arr_delay
```

Y para facilitar la visualización de estás conversiones usamos una función `select()` con las nuevas variables.

```
library(nycflights13)
library(dplyr)
new_flights<-mutate(flights, dep_time_min=(dep_time%/%100*60+dep_time%%100)%%1440,
sched_dep_time_min=(sched_dep_time%/%100*60+sched_dep_time%%100)%%1440)
select(new_flights,dep_time_min, sched_dep_time_min)
```

```
## # A tibble: 336,776 x 2
##    dep_time_min sched_dep_time_min
##           <dbl>              <dbl>
##  1          317                315
##  2          333                329
##  3          342                340
##  4          344                345
##  5          354                360
##  6          354                358
##  7          355                360
##  8          357                360
##  9          357                360
## 10          358                360
## # ... with 336,766 more rows
```

2. Comparar `air_time` con `arr_time - dep_time`. ¿Qué esperas ver? ¿Qué ves? ¿Qué necesitas hacer para arreglarlo?

Espero que `air_time` sea la diferencia entre llegada y salida de los vuelos. visto de otra forma puede ser `air_time = arr_time - dep_time`. Para confirmar esto, se realiza un código parecido al anterior, esto para operar los valores de las variables. Una vez comparado `air_time` con `arr_time - dep_time` se puede afirmar que `air_time = arr_time - dep_time`.

```
library(nycflights13)
library(dplyr)
fl_air_time<-mutate(flights,dep_time=(dep_time%/%100*60+dep_time%%100)%%1440,
arr_time=(arr_time%/%100*60+arr_time%%100)%%1440,
air_time2=air_time-arr_time+dep_time)
select(fl_air_time, air_time2 )
```

```
## # A tibble: 336,776 x 1
##    air_time2
##        <dbl>
##  1        34
##  2        30
##  3       -61
##  4       -77
##  5       -22
##  6        44
##  7       -40
##  8       -19
##  9       -21
## 10        23
## # ... with 336,766 more rows
```

Pues no ya que, si esta declaración fuera correcta, todos los vuelos de `air_time2` deberían ser 0, pero esto no es así. así que para decir que `air_time = arr_time - dep_time` no es 0 debido la zona horaria de los vuelos ya que estas se pueden cruzar entre ellas hablando sobre vuelos internacionales, sin embargo, también hay casos en los que el vuelo es nacional y aun así el resultado de la operación no es 0. ¿A qué se debe esto? pues puede que los valores de `arr_time` y `dep_time` no sean valores que se toman desde el despegue o propio aterrizaje del vuelo ya que hay más variables a tener en cuenta de desde cuando sale el vuelo como el tiempo en pista antes de despegar o desembarcar.

5.6.7

1 Haga una lluvia de ideas sobre al menos 5 formas diferentes de evaluar las características típicas de retraso de un grupo de vuelos. Considere los siguientes escenarios:

1- Media aritmética: Calcula el promedio de los tiempos de llegada de los vuelos y compara con el horario previsto. Esta evaluación sería apropiada para todos los escenarios mencionados.

2- Desviación estándar: Calcula la variabilidad en los tiempos de llegada de los vuelos. Esta evaluación sería apropiada para los escenarios en los que los tiempos de llegada varían, como el primer y el tercer escenario.

3- Coeficiente de variación: Calcula la variabilidad relativa de los tiempos de llegada de los vuelos. Esta evaluación sería apropiada para los escenarios en los que los tiempos de llegada varían, como el primer y el tercer escenario.

4- Frecuencia de retraso: Calcula la cantidad de veces que los vuelos llegan tarde y compara con la cantidad de vuelos totales. Esta evaluación sería apropiada para los escenarios en los que hay retrasos, como el primer, segundo y tercer escenario.

5- Percentil 99: Calcula el tiempo de llegada del vuelo que ocurre en el percentil 99 y compara con el horario previsto. Esta evaluación sería apropiada para el cuarto escenario, donde el 1% de los vuelos llega dos horas tarde.

5.7.1

2 ¿Qué avión (tailnum) tiene el peor récord de puntualidad?

```
library(nycflights13)
library(tidyverse)
library(readr)
```

```
library(dplyr)
nycflights13::flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##    <int> <int> <int>    <int>      <int>   <dbl>   <int>   <int>   <dbl> <chr>
##  1  2013     1     1      517        515       2     830     819      11 UA
##  2  2013     1     1      533        529       4     850     830      20 UA
##  3  2013     1     1      542        540       2     923     850      33 AA
##  4  2013     1     1      544        545      -1    1004    1022     -18 B6
##  5  2013     1     1      554        600      -6     812     837     -25 DL
##  6  2013     1     1      554        558      -4     740     728      12 UA
##  7  2013     1     1      555        600      -5     913     854      19 B6
##  8  2013     1     1      557        600      -3     709     723     -14 EV
##  9  2013     1     1      557        600      -3     838     846      -8 B6
## 10  2013     1     1      558        600      -2     753     745       8 AA
## # ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

```
fl<-nycflights13::flights
time_record<-flights %>%
filter(!is.na(arr_delay)) %>%
group_by(tailnum) %>%
summarise(arr_delay = mean(arr_delay), n = n()) %>%
filter(n >= 20) %>%
filter(min_rank(desc(arr_delay)) == 1)
```

Es importante destacar que el registro de puntualidad de un avión puede verse afectado por una variedad de factores, incluyendo condiciones climáticas, problemas mecánicos, congestión del tráfico aéreo y otros imprevistos que pueden impactar el horario de vuelo. Por lo tanto, aunque este avión en particular tenga el peor record de puntualidad según el filtro aplicado, es posible que haya habido circunstancias atenuantes que hayan contribuido a su bajo desempeño. Es importante considerar estos factores antes de sacar conclusiones definitivas sobre el rendimiento de una aeronave.