

# PrimerEntregable

Sebastian, William, Marlon

2022-08-28

## Solución al primer ClassWork

### Ejercicio de repaso

Se nos solicita que imprimamos los numeros del 1 al 100 con *while* y *for*. A continuación el codigo.

```
# while
i <- 1
while (i<=100) {
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
```

```
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
```

```
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
```

```
# for
for (x in 1:100) {
  print(x)
  x <- x + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
```

```
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
```

```
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
```

## 5.2.4 Ejercicios

### 1. Encuentra todos los vuelos

#### 1.1 Filtrado de los vuelos que tuvieron retraso de llegada de dos o más horas

Hemos usado la función *filter* el cual permite filtrar valores. El primer argumento es la tabla de los vuelos. El segundo argumento y los siguientes son las características para filtrar.

Importamos las librerías **tidyverse** y **nycflights13** para la práctica.

```
library(tidyverse)

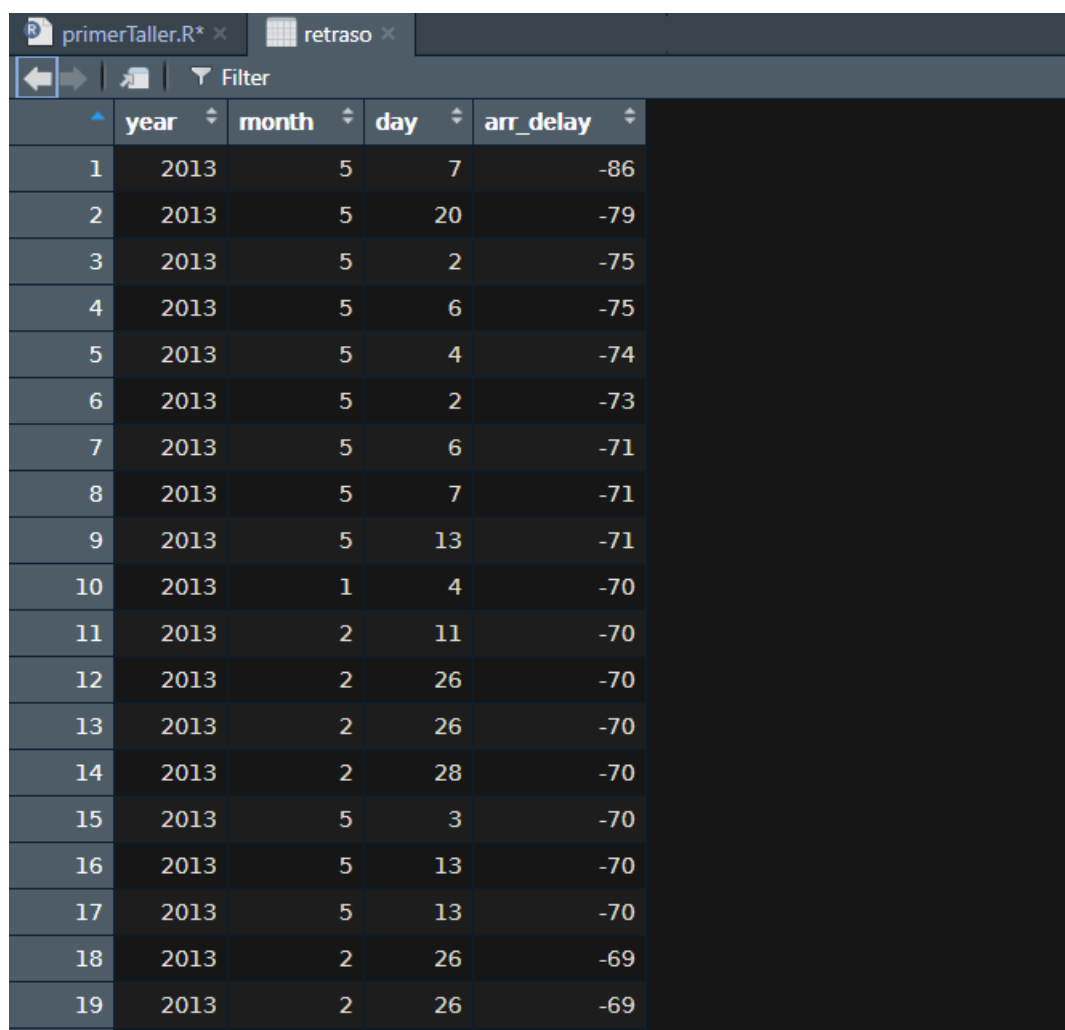
## -- Attaching packages ----- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.9
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(nycflights13)

vuelos <- flights
retraso = filter(flights, arr_delay >= 120)
```

Los datos 336766 fueron asignados en una variable llamada **VUELOS** y seguido creamos otra variable llamada **RETRASO**, donde asignamos los datos filtrados por columnas para encontrar los vuelos que tuvieron un retraso de dos o más horas

Aquí están los primeros 19 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



The screenshot shows an RStudio window with two tabs: 'primerTaller.R\*' and 'retraso'. The 'retraso' tab is active, displaying a data table. The table has four columns: 'year', 'month', 'day', and 'arr\_delay'. The data consists of 19 rows, all for the year 2013. The 'arr\_delay' values range from -86 to -69.

	year	month	day	arr_delay
1	2013	5	7	-86
2	2013	5	20	-79
3	2013	5	2	-75
4	2013	5	6	-75
5	2013	5	4	-74
6	2013	5	2	-73
7	2013	5	6	-71
8	2013	5	7	-71
9	2013	5	13	-71
10	2013	1	4	-70
11	2013	2	11	-70
12	2013	2	26	-70
13	2013	2	26	-70
14	2013	2	28	-70
15	2013	5	3	-70
16	2013	5	13	-70
17	2013	5	13	-70
18	2013	2	26	-69
19	2013	2	26	-69

Figure 1: foto tomada de los datos extraidos

## 1.2 Filtrado de vuelos a Houston (IAH - HOU)

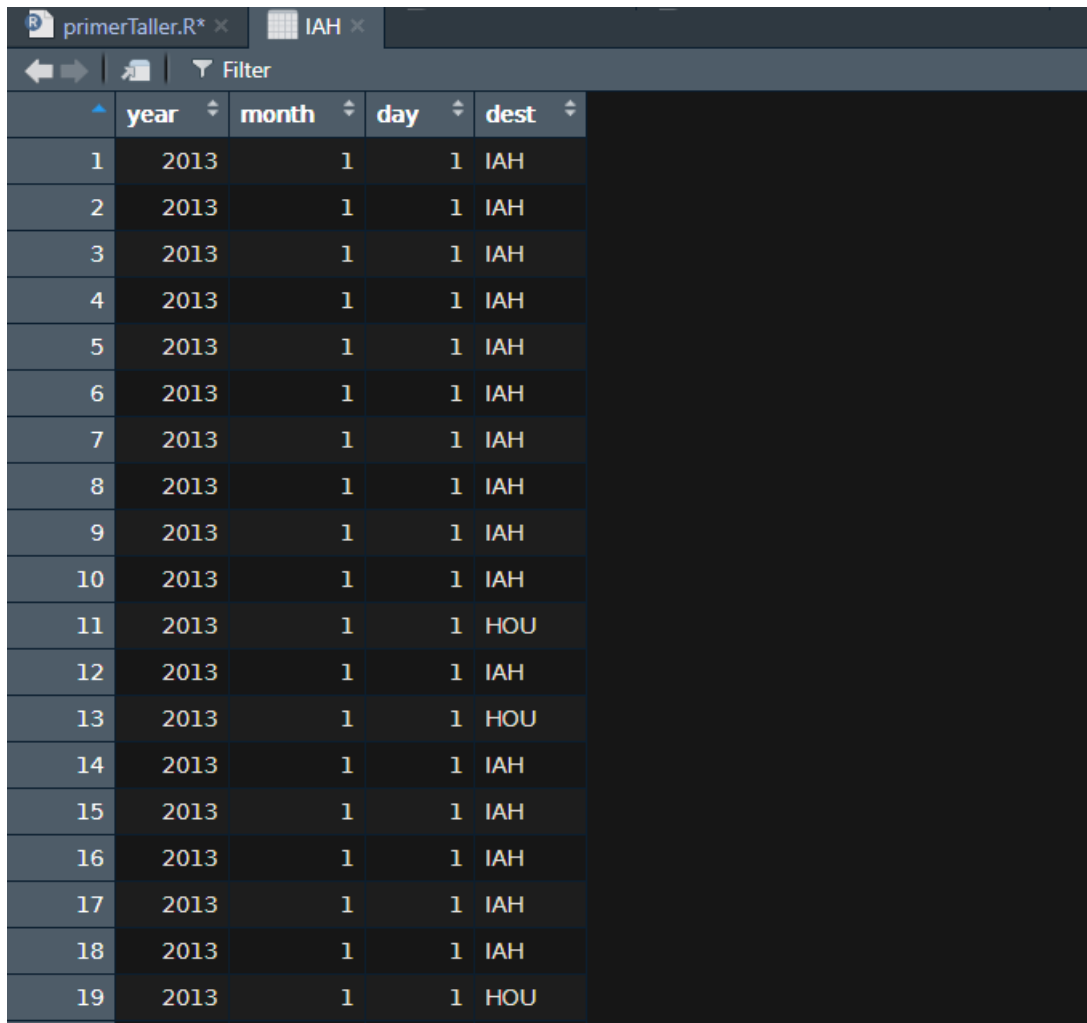
De la misma manera que realizamos el filtrado en el primer punto, lo hacemos en este apartado. Aquí el código

Importamos las librerías **tidyverse** y **nycflights13** para la práctica.

```
library(tidyverse)
library(nycflights13)
IAH <- vuelos %>%
  filter(dest == "IAH" | dest == "HOU") %>%
  select(year, month, day, dest) %>%
  arrange(year)
```

Los datos 336766 fueron asignados en una variable llamada **VUELOS** y seguido creamos otra variable llamada **IAH**, donde asignamos los datos filtrados por columnas para encontrar los vuelos a Houston.

Aquí están los primeros 19 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



	year	month	day	dest
1	2013	1	1	IAH
2	2013	1	1	IAH
3	2013	1	1	IAH
4	2013	1	1	IAH
5	2013	1	1	IAH
6	2013	1	1	IAH
7	2013	1	1	IAH
8	2013	1	1	IAH
9	2013	1	1	IAH
10	2013	1	1	IAH
11	2013	1	1	HOU
12	2013	1	1	IAH
13	2013	1	1	HOU
14	2013	1	1	IAH
15	2013	1	1	IAH
16	2013	1	1	IAH
17	2013	1	1	IAH
18	2013	1	1	IAH
19	2013	1	1	HOU

Figure 2: foto tomada de los datos extraídos

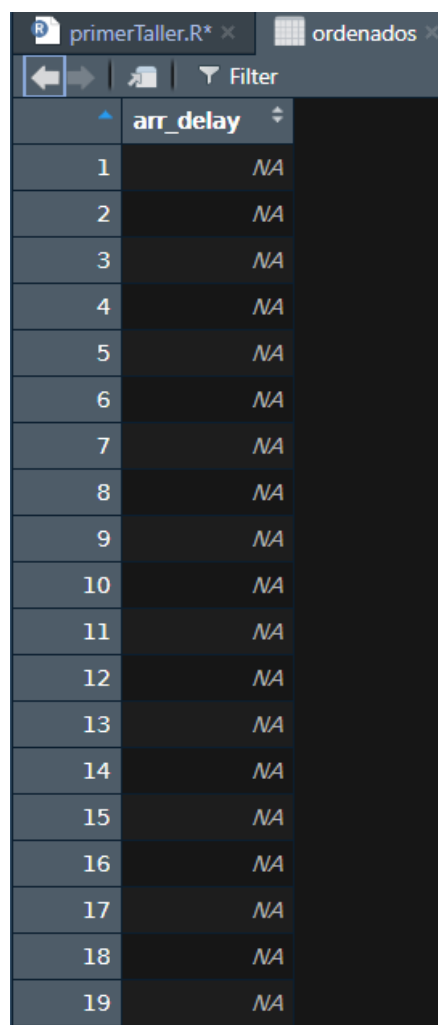
### 5.3.1 Ejercicios

## 2. Organiza todos los vuelos

### 2.1 ¿Cómo podría usar para ordenar todos los valores que faltan al principio?

Ordenamos los datos NA al principio, de manera descendente con una tabla creada de **ordenados** y lo almacenamos ahí.

```
ordenados <- arrange(vuelos, desc(is.na(arr_delay))) %>%  
  select(arr_delay)
```



	arr_delay
1	NA
2	NA
3	NA
4	NA
5	NA
6	NA
7	NA
8	NA
9	NA
10	NA
11	NA
12	NA
13	NA
14	NA
15	NA
16	NA
17	NA
18	NA
19	NA

Figure 3: foto tomada de los datos extraidos de vuelos retrasados

## 2.2 Ordene para encontrar los vuelos más retrasados. Encuentre los vuelos que salieron más temprano

La funcion **arrange** funciona en seleccionar filas, cambia su orden. Se necesita un marco de datos y un conjunto de nombres de columna (o expresiones más complicadas) para ordenar. Si proporciona más de



un nombre de columna, cada columna adicional se utilizará para romper los vínculos en los valores de las columnas anteriores, esta funcion la utilizaremos para el desarrollo de las actividades

Creamos una tabla llamada **retrasados** Guardamos los datos que se encuentran los vuelos mas atrasados.

Importamos las librerias **tidyverse** y **nycflights13** para la practica.

```
library(tidyverse)
library(nycflights13)
retrasados <- arrange(flights , desc(arr_delay)) %>%
  select(year, month, day, arr_delay ,flight)
```

Aquí están los primeros 21 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



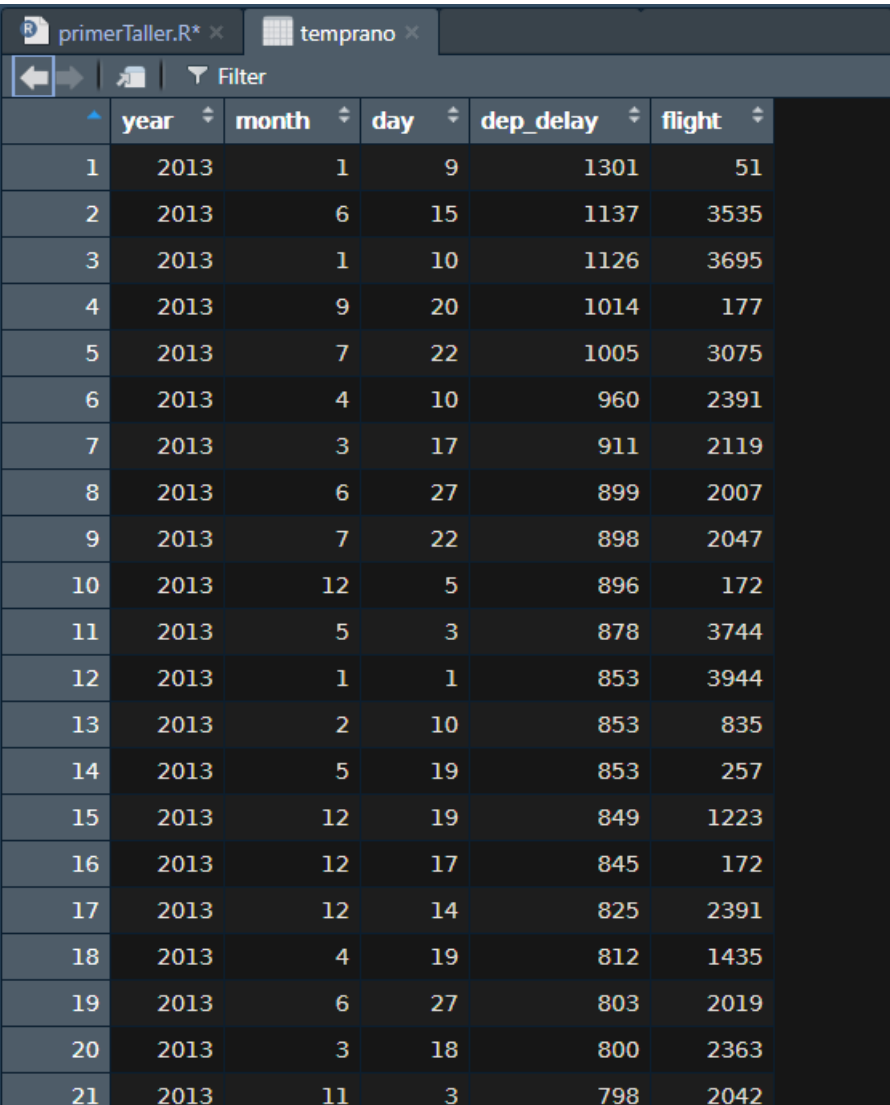
	year	month	day	arr_delay	flight
1	2013	1	9	1272	51
2	2013	6	15	1127	3535
3	2013	1	10	1109	3695
4	2013	9	20	1007	177
5	2013	7	22	989	3075
6	2013	4	10	931	2391
7	2013	3	17	915	2119
8	2013	7	22	895	2047
9	2013	12	5	878	172
10	2013	5	3	875	3744
11	2013	12	14	856	2391
12	2013	5	19	852	257
13	2013	1	1	851	3944
14	2013	6	27	850	2007
15	2013	12	19	847	1223
16	2013	12	17	846	172
17	2013	2	10	834	835
18	2013	4	19	821	1435
19	2013	6	27	802	2019
20	2013	11	3	796	2042
21	2013	3	18	784	2363

Figure 4: foto tomada de los datos extraidos de vuelos retrasados

Seguido de esto creamos otra tabla llamada **temprano** donde almacenaremos los datos de los vuelos que salieron temprano, ambos datos se ordenaron de manera descendente

```
library(tidyverse)
library(nycflights13)
temprano <- arrange(flights , desc(dep_delay))%>%
  select(year, month, day, arr_delay, flight)
```

Aquí están los primeros 21 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



	year	month	day	dep_delay	flight
1	2013	1	9	1301	51
2	2013	6	15	1137	3535
3	2013	1	10	1126	3695
4	2013	9	20	1014	177
5	2013	7	22	1005	3075
6	2013	4	10	960	2391
7	2013	3	17	911	2119
8	2013	6	27	899	2007
9	2013	7	22	898	2047
10	2013	12	5	896	172
11	2013	5	3	878	3744
12	2013	1	1	853	3944
13	2013	2	10	853	835
14	2013	5	19	853	257
15	2013	12	19	849	1223
16	2013	12	17	845	172
17	2013	12	14	825	2391
18	2013	4	19	812	1435
19	2013	6	27	803	2019
20	2013	3	18	800	2363
21	2013	11	3	798	2042

Figure 5: foto tomada de los datos extraidos de vuelos tempranos

## 2.3 Ordene para encontrar los vuelos más rápidos (de mayor velocidad).

Creamos una tabla llamada **masVeloces** Guardamos los datos que se encuentran los vuelos mas rapidos, se crearon dos formas para hacer la practica

Importamos las librerias **tidyverse** y **nycflights13** para la practica

```
library(tidyverse)
library(nycflights13)
masVeloces1.0 <- arrange(flights, air_time)%>%
  select(year, month, day, air_time, flight)
```

Aquí están los primeros 21 resultados de la primera opción, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



	year	month	day	air time	flight
1	2013	1			4368
2	2013	4	13	20	4631
3	2013	12	6	21	4276
4	2013	2	3	21	4619
5	2013	2	5	21	4368
6	2013	2	12	21	4619
7	2013	3	2	21	2132
8	2013	3	8	21	3650
9	2013	3	18	21	4118
10	2013	3	19	21	4276
11	2013	5	8	21	4276
12	2013	5	19	21	4276
13	2013	6	12	21	4276
14	2013	8	18	21	4577
15	2013	9	3	21	6062
16	2013	9	3	21	3847
17	2013	1	6	22	4619
18	2013	1	13	22	4368
19	2013	1	14	22	4368
20	2013	1	15	22	4368
21	2013	12	9	22	4276

Figure 6: foto tomada de los datos extraídos de vuelos mas veloces

Creemos la segunda opción.

```
library(tidyverse)
library(nycflights13)
masVeloces1.1 <- arrange(flights, hour)%>%
  select(year, month, day, hour, flight)
```

Aquí están los primeros 21 resultados, de la segunda opción, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.



	year	month	day	hour	flight
1	2013	7	27	1	1632
2	2013	1	1	5	1545
3	2013	1	1	5	1714
4	2013	1	1	5	1141
5	2013	1	1	5	725
6	2013	1	1	5	1696
7	2013	1	1	5	1806
8	2013	1	2	5	1030
9	2013	1	2	5	1453
10	2013	1	2	5	1141
11	2013	1	2	5	407
12	2013	1	2	5	725
13	2013	1	2	5	1676
14	2013	1	2	5	651
15	2013	1	3	5	1030
16	2013	1	3	5	1018
17	2013	1	3	5	1136
18	2013	1	3	5	1141
19	2013	1	3	5	725
20	2013	1	3	5	1570
21	2013	1	4	5	1030

Figure 7: foto tomada de los datos extraidos de vuelos mas veloces

## 2.4 ¿Qué vuelos viajaron más lejos? ¿Cuál viajó más corto?

Creemos una tabla llamada **masLejos** Guardamos los datos que se encuentran los vuelos que fueron mas lejos.

Importamos las librerias **tidyverse** y **nycflights13** para la practica

```
library(tidyverse)
library(nycflights13)
masCorto <- arrange(flights , distance) %>%
  select(year, month, day, distance, flight)
```

Aquí están los primeros 21 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.

	year	month	day	distance	flight
1	2013	7	27	17	1632
2	2013	1	3	80	3833
3	2013	1	4	80	4193
4	2013	1	4	80	4502
5	2013	1	4	80	4645
6	2013	1	5	80	4193
7	2013	1	6	80	4619
8	2013	1	7	80	4619
9	2013	1	8	80	4619
10	2013	1	9	80	4619
11	2013	1	10	80	3271
12	2013	1	11	80	4619
13	2013	1	12	80	4616
14	2013	1	13	80	4619
15	2013	1	14	80	4619
16	2013	1	15	80	4619
17	2013	1	16	80	4619
18	2013	1	17	80	3271
19	2013	1	18	80	4619
20	2013	1	19	80	4616
21	2013	1	20	80	4619

Figure 8: foto tomada de los datos extraidos de vuelos mas cortos

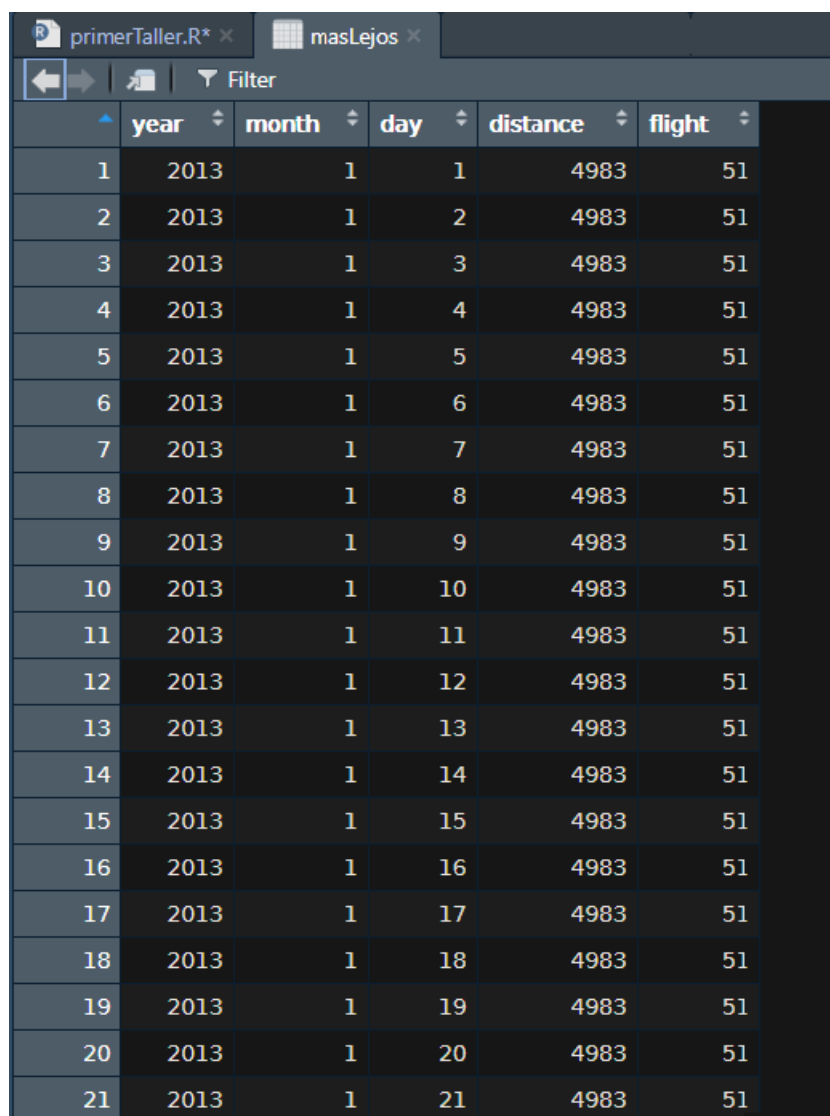
Seguido de esto creamos una tabla llamada **masCorto** de los vuelos que tuvieron un viaje mas corto.

```
library(tidyverse)
library(nycflights13)
masLejos <- arrange(flights , desc(distance)) %>%
  select(year, month, day, distance, flight)
```

Aquí están los primeros 21 resultados, ya que son muchos datos, sin embargo con el código puedes mostrarlos todos en el IDE, esta imagen es solo para efectos visuales.

#### 5.4.1

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



	year	month	day	distance	flight
1	2013	1	1	4983	51
2	2013	1	2	4983	51
3	2013	1	3	4983	51
4	2013	1	4	4983	51
5	2013	1	5	4983	51
6	2013	1	6	4983	51
7	2013	1	7	4983	51
8	2013	1	8	4983	51
9	2013	1	9	4983	51
10	2013	1	10	4983	51
11	2013	1	11	4983	51
12	2013	1	12	4983	51
13	2013	1	13	4983	51
14	2013	1	14	4983	51
15	2013	1	15	4983	51
16	2013	1	16	4983	51
17	2013	1	17	4983	51
18	2013	1	18	4983	51
19	2013	1	19	4983	51
20	2013	1	20	4983	51
21	2013	1	21	4983	51

Figure 9: foto tomada de los datos extraídos de vuelos mas lejos

```
65
66 select(flights, year)
67 |
68
```

67:1 (Top Level) ⌵

Console Render x Background

R 4.2.1 · ~/

```
> plot(exampleSelect)
> view(exampleSelect)
> select(flights, year,
# A tibble: 336,776 x
  year
  <int>
1 2013
2 2013
3 2013
4 2013
5 2013
6 2013
7 2013
```

No sucede nada y siempre va filtrar el campo desde que este indicado como en la tabla

### 3. ¿Qué hace la función `any_of()`? ¿Por qué podría ser útil junto con este vector?

Se usa el `any_of()` para permitir variables faltantes, también el `any_of()` es especialmente útil para eliminar variables de un marco de datos porque volver a llamarlo no provoque un error, en este vector podría ayudar haciendo un `select` y que me filtre si esta o no el campo requerido.

### 4. ¿Te sorprende el resultado de ejecutar el siguiente código? ¿Cómo tratan los ayudantes selectos el caso de forma predeterminada? ¿Cómo se puede cambiar ese valor predeterminado?

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

No me sorprende mucho ya que en el código está seleccionando los campos que contengan ese valor, se cambia este valor predeterminado en este caso cambiando la variable "TIME" por la que se requiera

## 5.5.2

1. Actualmente, `dep_time` y `sched_dep_time` son convenientes a la vista, pero difíciles de calcular porque en realidad no son números continuos. Conviértalos a una representación más conveniente de la cantidad de minutos desde la medianoche.

```
transmute(flights,
  dep_time,
  hour = dep_time %/% 100,
  minute = dep_time %% 100,
)
transmute(flights,
```

```

    sched_dep_time,
    hour = sched_dep_time %/% 100,
    minute = sched_dep_time %/% 100,
  )

```

**2. Compare air\_time con arr\_time - dep\_time. Que esperas ver? ¿Que ves? ¿Qué necesitas hacer para arreglarlo?**

```

transmute(flights,
  air_time = air_time,
  air_time_new = arr_time - dep_time
)

```

Segun la logica que hemos llevado se tiene que air\_time debe ser igual a la esa resta, veo que sales los datos desordenados y por ente es dificil ver esa percepcion, para qye esto se arregle se debe ordenar de los numeros para ver la similitud en los resultados

### Ejercicio 5.6.7 (Item 1)

Para este ejercicio en primera instancia se utilizó la funcion filter(), para poder filtrar los vuelos con llegada antes de 15 minutos, y los vuelos que llegaron 15 minutos tarde.

```

library(nycflights13)
library(tidyverse)

```

```

flights <- flights
minantes <- filter(flights, arr_delay == -15 | arr_delay == 15)

```

Para encontrar los vuelos que siempre llegan 10 minutos tarde

```

diezmin <- filter(flights, arr_delay == 10)

```

Para encontrar los vuelos que llegan 30 minutos antes y 30 minutos despues

```

terinmin <- filter(flights, arr_delay == -30 | arr_delay == 30 )

```

### Ejercicio 5.7.1 (Item 2)

Para este ejercicio debemos de encontrar el avión que tuvo el peor record de puntualidad, para esto nos vamos apoyar en las funciones de, arrange y group\_by.

```

#retraso3 <- arrange(flights, desc(dep_delay))
#group_by(tailnum)

```

Organizamos los datos según el dep\_delay que es la diferencia entre la duración pleada del vuelo y la real, seguido a esto lo agrupamos por tipo de avión.



## Descripción de dos función filter()

A continuación se realiza la explicación de la función filter(), la cual el primer argumento es la tabla a la que estamos accediendo como en este caso es la de “flights”, y los siguientes argumentos que se le pasan a la función son lo que deseamos buscar en este caso voy a agregar la libreria de tidyverse.

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

Utilizamos “message = FALSE”, para que no aparezca el código en la ejecución.

Luego de eso vamos a visualizar la tabla de los vuelos que fueron del mes de Septiembre, día 25, destino MIAMI, en la hora 15.

```
Vuelos_de_septiembre_25 <- filter(flights, month == 9, day == 25, dest == 'MIA', hour == 15)
```

También podemos realizar comparaciones en el código para este caso se filtra los vuelos en los meses de Junio o Julio

```
Vuelos_de_junio_julio <- filter(flights, month == 6 | month == 7)
```

## Descripción de función mutate()

Con esta función podemos agregar otra columna haciendo operaciones entre columnas ya existentes

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

```
## # A tibble: 336,776 x 9
##   year month day dep_delay arr_delay distance air_time gain speed
##   <int> <int> <int>   <dbl>   <dbl>   <dbl>   <dbl> <dbl> <dbl>
## 1  2013     1     1         2        11    1400    227    -9   370.
## 2  2013     1     1         4        20    1416    227   -16   374.
## 3  2013     1     1         2        33    1089    160   -31   408.
## 4  2013     1     1        -1       -18    1576    183    17   517.
## 5  2013     1     1        -6       -25     762    116    19   394.
## 6  2013     1     1        -4        12     719    150   -16   288.
## 7  2013     1     1        -5        19    1065    158   -24   404.
## 8  2013     1     1        -3       -14     229     53    11   259.
## 9  2013     1     1        -3        -8     944    140     5   405.
## 10 2013     1     1        -2         8     733    138   -10   319.
## # ... with 336,766 more rows
## # i Use 'print(n = ...)' to see more rows
```

En este caso vamos a agregar las columnas “gain” que será la resta entre la columna dep\_delay y arr\_delay, seguido se agrega la columna “speed” la cual es la división entre distance y air\_time multiplicado por 60.