

Informe Programacion en Rstudio

Santiago Avila, Andres Hernandez, Andres Montenegro, Brayan Zambrano, Nikolas Riapira

2023-02-25

INFORME - ELECTIVA AREA ELECTRONICA

1. Código básico en R

Números primos del 1 al 100

A través del siguiente código se muestran los números primos del 1 al 100

```
for (i in 2:100) {  
  x <- 0  
  for (j in 1:i) {  
    if (i %% j == 0) {  
      x <- x + 1  
    } else{  
  
    }  
  }  
  if (x <= 2) {  
    print(i)  
  }  
}
```

Obteniendo como resultado lo siguiente

```
## [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
```

Notese que el siguiente mensaje es debido a que el chunk de activacion de la libreria de tidyverse tiene como TRUE el valor para message, debido a esto se muestra el mensaje de carga de libreria en el documento

```
## -- Attaching packages ----- tidyverse 1.3.2 --
## v ggplot2 3.4.1      v purrr  1.0.1
## v tibble  3.1.8      v dplyr  1.1.0
## v tidyr   1.3.0      v stringr 1.5.0
## v readr   2.1.4      v forcats 1.0.0
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

2. Uso básico de Tidyverse

5.2.4.1 Ejercicios (filter())

A continuación se presentan los ejercicios básicos para la utilización de la función filter()

Item 1: Tuvieron un retraso de llegada de dos o mas horas

```
Arrival_Delay <- flights %>%
  filter(arr_delay >= 120)
View(Arrival_Delay)
```

Lo que genera la tabla flights unicamente con los vuelos con un retraso de llegada mayor a 120 minutos.

Item 2: Volaron a Houston

```
Houston_destiny <- flights %>%
  filter(dest == "HOU" | dest == "IAH")
view(Houston_destiny)
```

Este código muestra los vuelos con destino a Houston determinado por “HOU” o “IAH”

Item 3: Fueron operados por United, American o Delta

```
Operated_by <- flights %>%
  filter(carrier %in% c("UA", "AA", "DL"))
view(Operated_by)
```

Se filtran y muestran los vuelos correspondientes a las operaciones de las 3 aerolíneas mencionadas

Item 4: Vuelos que salieron en verano

```
Summer_dep <- flights %>%
  filter(month >= 7 & month <= 9)
view(Summer_dep)
```

Aca se filtran los vuelos que fueron operados en los 3 meses correspondientes a verano los cuales son Julio, Agosto y Septiembre

Item 5: Llegaron mas de dos horas tarde pero no partieron con retraso

```
LateArr_timeLeft <- flights %>%
  filter(arr_delay > 120 & dep_delay <= 0)
view(LateArr_timeLeft)
```

Se filtran dos variables diferentes siendo el retraso en la llegada superior a 2 horas y el retraso en la salida inferior a 0, es decir que salieron a tiempo o mucho antes del horario establecido

Item 6: Fueron retrasados por lo menos por una hora, pero recuperaron mas de 30min en vuelo

```
Delayed1H <- flights %>%  
  filter(dep_delay >= 60 & dep_delay - arr_delay > 30)  
view(Delayed1H)
```

Se muestran los vuelos que tuvieron un retraso en la salida de al menos una hora, pero que recuperaron al menos 30min de retraso durante el vuelo, por ello se hace la resta entre ambos retrasos y debería dar superior a los 30

Item 7: Salieron entre la medianoche y las 6 de la mañana

```
Midnight_to_6am <- flights %>%  
  filter(dep_time >= 0 & dep_time <= 600)  
view(Midnight_to_6am)
```

Igualmente se filtran los vuelos con salidas entre la medianoche (0) y las 6am (600)

5.2.4.2 Ejercicios (filter() + between())

A continuación se presentan los ejercicios anteriores utilizando la función between()

Item 1: Tuvieron un retraso de llegada de dos o mas horas

```
Arrival_Delay2 <- flights %>%  
  filter(between(arr_delay, 120, Inf))  
View(Arrival_Delay2)
```

Item 2: Volaron a Houston

```
Houston_destiny2 <- flights %>%  
  filter(between(dest, "HOU", "IAH"))  
view(Houston_destiny2)
```

Item 3: Fueron operados por United, American o Delta (No es posible utilizar between ya que se analizan 3 valores de la variable carrier)

```
Operated_by <- flights %>%  
  filter(carrier %in% c("UA", "AA", "DL"))  
view(Operated_by)
```

Item 4: Vuelos que salieron en verano

```
Summer_dep2 <- flights %>%  
  filter(between(month, 7, 9))  
view(Summer_dep2)
```

Item 5: Llegaron mas de dos horas tarde pero no partieron con retraso (No es posible utilizar between ya que se analizan 2 variables distintas)

```
LateArr_timeLeft2 <- flights %>%  
  filter(arr_delay > 120 & dep_delay <= 0)  
view(LateArr_timeLeft2)
```

Item 6: Fueron retrasados por lo menos por una hora, pero recuperaron mas de 30min en vuelo (No es posible utilizar between ya que se analizan 2 variables distintas)

```
Delayed1H2 <- flights %>%
  filter(dep_delay >= 60 & dep_delay - arr_delay > 30)
view(Delayed1H2)
```

Item 7: Salieron entre la medianoche y las 6 de la mañana

```
Midnight_to_6am2 <- flights %>%
  filter(between(dep_time, 0, 600))
view(Midnight_to_6am2)
```

5.3.1 Ejercicios (arrange()+is.na())

Item 1: ¿Como se podria utilizar la funcion arrange() para clasificar los valores vacios al inicio(Pista: Utilice la funcion is.na())?

```
# Cargar la librería dplyr
library(dplyr)
```

```
# Ordenar los valores faltantes al principio del marco de datos starwars
starwars_ordenado <- arrange(starwars, desc(is.na(mass)))
```

```
# Ver el marco de datos ordenado
head(starwars_ordenado)
```

```
## # A tibble: 6 x 14
##   name          height  mass hair_~1 skin_~2 eye_c~3 birth~4 sex  gender homew~5
##   <chr>          <int> <dbl> <chr>   <chr>   <chr>   <dbl> <chr> <chr> <chr>
## 1 Wilhuff Tar~    180    NA auburn~ fair    blue     64 male  mascu~ Eriadu
## 2 Mon Mothma     150    NA auburn  fair    blue     48 fema~ femin~ Chandr~
## 3 Arvel Crynyd    NA    NA brown  fair    brown    NA male  mascu~ <NA>
## 4 Finis Valor~    170    NA blond  fair    blue     91 male  mascu~ Corusc~
## 5 Rugor Nass     206    NA none   green   orange   NA male  mascu~ Naboo
## 6 Ric Olié       183    NA brown  fair    blue     NA <NA> <NA> Naboo
## # ... with 4 more variables: species <chr>, films <list>, vehicles <list>,
## #   starships <list>, and abbreviated variable names 1: hair_color,
## #   2: skin_color, 3: eye_color, 4: birth_year, 5: homeworld
```

```
View(starwars_ordenado)
```

Desarrollo: La funcion arrange por defecto, no genera nuevas tablas en un arreglo, sino toma datos ya existentes, ordenandolos de acuerdo a la sentencia del usuario, incluido la funcion donde los valores vacios (NA), se situan siempre debajo de la clasificacion de datos, para realizar lo contrario, se inspecciona la funcion is.na(), que examina los valores de un arreglo para hallar los valores vacios, donde por cada valor vacio encontrado, generara una respuesta de salida de tipo booleano si llega a ser verdadera o falsa la sentencia (Si x=NA =TRUE)(SI x != NA FALSE), en este caso, se toma el data frame starwars en la categoria masa, se examina si en los datos existe valores de tipo NA, se situa el valor de clasificacion de mayor a menor mediante la sentencia desc para mostrar los valores de tipo TRUE por encima de los FALSE, correspondientes a los valores de tipo NA en la parte superior de la tabla, y el restante de los valores diferentes de NA situados al final de la tabla, cabe resaltar que esta clasificacion es realizada unicamente para discriminar valores de tipo TRUE y FALSE consignados en la tabla, mas no ordena los valores numericos en un orden en especifico, cabe mencionar que la funcion head, esta encargada unicamente de desplegar los primeros datos de la tabla en el monitor de compilación

Item 2: Clasifique de el dataframe Flights los vuelos mas retrasados, de tal modo de querer observar cuales se retrasaron mas y cuales salieron antes de tiempo

```
#Se incluye la libreria dplyr
library(dplyr)
#Se incluye el dataframe nycflights2013
library(nycflights13)

#Se almacena en flight_most_delayed la clasificacion de vuelos con mayor retraso de manera descendente
flight_most_delayed <- arrange(flights,desc(dep_delay))
#Se almacena en flight_most_delayed la clasificacion de vuelos con menor retraso de manera ascendente (
flight_leftearliest <- arrange(flights,dep_delay)
#Se desea observar los primeros datos de la lista y desplegar en una pestaña la vista de la tabla
head(flight_most_delayed)
```

```
## # A tibble: 6 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     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
## # ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, and abbreviated variable names 1: sched_dep_time,
## #   2: dep_delay, 3: arr_time, 4: sched_arr_time, 5: arr_delay
```

```
head(flight_leftearliest)
```

```
## # A tibble: 6 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    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
## # ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, and abbreviated variable names 1: sched_dep_time,
## #   2: dep_delay, 3: arr_time, 4: sched_arr_time, 5: arr_delay
```

Desarrollo: Empleando la funcion `arrange()` en el dataset `flights`, en la categoria `dep_delay` (deploy delay), Se empleo en primera medida para hallar los vuelos con mayor retardo mediante la sentencia `arrange(flights,desc(dep_delay))` donde se muestran en primer encabezado los vuelos con mayor retardo de despegue, por el contrario al desear visualizar los de menor retardo se implementa igualmente la funcion `arrange()`, solo que en este caso es utilizado para mostrar los vuelos con menor retraso, cabe resaltar que los vuelos con valores negativos hacen referencia que los vuelon salieron dicha cantida de minutos antes de la hora de vuelo establecida

Item 3: Clasifique los vuelos con mayor velocidad de vuelo

```
#Cargar la libreria dplyr+
library(dplyr)
#Cargar la libreria nyc flights para obtener datos de vuelo
library(nycflights13)
```

```
#Se asigna los datos de arrange de velocidad a flights, calculando la velocidad=distancia/tiempo
flights <- mutate(flights,speed = distance /air_time*60)
#Se selecciona la categoria speed
select(flights,speed)
```

```
## # A tibble: 336,776 x 1
##   speed
##   <dbl>
## 1 370.
## 2 374.
## 3 408.
## 4 517.
## 5 394.
## 6 288.
## 7 404.
## 8 259.
## 9 405.
## 10 319.
## # ... with 336,766 more rows
```

```
#Se asigna el orden de mayor a menor de velocidad a flights
flights <- arrange(flights,desc(speed))
#Se despliegan los primeros valores de flights
head(flights)
```

```
## # A tibble: 6 x 20
##   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     5    25    1709         1700     9    1923    1937    -14 DL
## 2  2013     7     2    1558         1513    45    1745    1719     26 EV
## 3  2013     5    13    2040         2025    15    2225    2226     -1 EV
## 4  2013     3    23    1914         1910     4    2045    2043      2 EV
## 5  2013     1    12    1559         1600    -1    1849    1917    -28 DL
## 6  2013    11    17     650          655    -5    1059    1150    -51 DL
## # ... with 10 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, speed <dbl>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
```

```
View(flights)
```

Desarrollo: Empleando multiples variables para implementar el ejercicio, se asigna al mismo conjunto de datos flights la operacion mutate() para crear una nueva variable, equivalente al cociente entre la distancia de vuelo de los aviones, y el tiempo de vuelo de estos (el tiempo es expresado en minutos, por lo que es necesario multiplicar el valor por 60 para realizar conversion en segundos), despues se selecciona mediante select() la nueva categoria velocidad, para finalmente aplicar a flights, la clasificacion de los vuelos mas veloces en el encabezado mediante un ordenamiento de mayor a menor con arrange(flights, desc(speed))

Item 4: Cuales vuelos recorrieron la mayor distancia, y cuales viajaron la menor

```
#Cargar la libreria dplyr
library(dplyr)
#Cargar la libreria nyc flights para obtener datos de vuelo
library(nycflights13)
#Se asigna los datos de arrange de mayor distancia a flights farthest
```

```

flights_farthest <- arrange(flights, desc(distance))
#Se asigna los datos de arrange de menor distancia a shortest
flights_shortest <- arrange(flights, distance)
#Se asigna los head(primeros datos) en la salida
head(flights_farthest)

## # A tibble: 6 x 20
##   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     5     7     959         1000     -1    1401    1500    -59 HA
## 2  2013     6     6    1044         1000     44    1441    1435     6 HA
## 3  2013     9    29     957         1000     -3    1405    1445    -40 HA
## 4  2013     6     7     952         1000     -8    1354    1435    -41 HA
## 5  2013     6     8     951         1000     -9    1352    1435    -43 HA
## 6  2013     9     6     955         1000     -5    1359    1445    -46 HA
## # ... with 10 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, speed <dbl>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
#Se asigna los head(primeros datos) en la salida
head(flights_shortest)

```

```

## # A tibble: 6 x 20
##   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     7    27      NA         106      NA      NA      245      NA US
## 2  2013     2     3    2153         2129     24    2247    2224     23 EV
## 3  2013     2    12    2123         2130     -7    2211    2225    -14 EV
## 4  2013     1     6    2125         2129     -4    2224    2224     0 EV
## 5  2013     1    23    2128         2129     -1    2221    2224     -3 EV
## 6  2013     2    10    2127         2129     -2    2209    2224    -15 EV
## # ... with 10 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, speed <dbl>, and abbreviated variable names
## #   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
## #   5: arr_delay
#Se asigna los head(primeros datos) en la salida
View(flights_farthest)
#Se asigna los head(primeros datos) en la salida
View(flights_shortest)

```

Desarrollo: Al igual que el ítem planteado previamente, se realiza uso de la función `arrange()` nuevamente para clasificar de mayor a menor, los vuelos con mayor distancia recorrida, mediante la función `arrange(flights, desc(distance))`, reasignándose a la variable creada `flights_farthest`, y mediante una clasificación contraria (menor a mayor), se asigna a la variable `flights_shortest` los vuelos con menor recorrido de distancia, mediante la función `arrange(flights, distance)`

5.4.1 Ejercicios (`select()` ; `any_of()`)

Ítem 2: ¿Qué sucede si incluye el nombre de una variable varias veces en una función `select()`?

```

library(tidyverse)
library(nycflights13)

```

```
x <- flights %>%
  select(distance, distance, distance, distance, distance)
view(x)
```

Respuesta: Al ejecutar el anterior código se puede observar que el resultado es un data-frame donde la variable distancia aparece con todos sus valores como una única columna, no importa cuantas veces se escriba la variable “distancia” siempre se va a obtener la columna antes mencionada sin ninguna alteración.

Item 3: ¿Que hace la función “any_of()”?

```
#missing variables
w <- flights %>%
  select(any_of("distance"))
view(w)

#hide the variable
u <- flights %>%
  select(-any_of("distance"))
view(u)

#3.1) Why might it be helpful in conjunction with this vector?

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

#missing variables
z <- flights %>%
  select(any_of(vars))
view(z)

#Hide the variables

f <- flights %>%
  select(-any_of(vars))
view(f)
```

Respuesta: La función “any_of()” en el anterior ítem se usa para catalogar las variables tanto las que estén dentro de ella como variables no faltantes (línea 39 de código), como las que estén dentro de ellas como variables faltantes con el cambio de signo en la función (línea 44); al correr el anterior código si la función no tiene el signo negativo las variables que estén dentro del parentesis serán las únicas que aparecerán en el data-frame resultante, de otra forma si el signo negativo es colocado las variables dentro del parentesis serán quitadas del data-frame final y aparecerán el resto de variables.

3.1) ¿Por qué podría ser útil la sentencia “any_of()” junto con el siguiente vector?

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

Respuesta: La sentencia que toma el valor de la variable “vars” funciona o complementa la función “any_of()” al hacer que esta última tome más valores para hacerlos variables faltantes o para omitir en el data frame las variables que no estén en la función; esto conlleva que al correr el código del ítem 3 se tengan muchas más columnas de varios tipos de variables (int, carácter etc).

Item 4: ¿Que puede concluir al ejecutar el siguiente código?

```
#4Does the result of running the following code surprise you?
#4.1How do the select helpers deal with case by default?
```



```
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> <dtm>
## 1    1709        1700        1923        1937        65 2013-05-25 17:00:00
## 2    1558        1513        1745        1719        93 2013-07-02 15:00:00
## 3    2040        2025        2225        2226        55 2013-05-13 20:00:00
## 4    1914        1910        2045        2043        70 2013-03-23 19:00:00
## 5    1559        1600        1849        1917       105 2013-01-12 16:00:00
## 6     650         655       1059       1150       170 2013-11-17 06:00:00
## 7    2355       2358         412         438       172 2013-02-21 23:00:00
## 8     759         800       1212       1255       175 2013-11-17 08:00:00
## 9    2003        1925         17         36       173 2013-11-16 19:00:00
## 10   2349       2359        402        440       173 2013-11-16 23:00:00
## # ... with 336,766 more rows
```

Respuesta: A simple vista puede parecer que la linea de codigo necesita mas especificaciones para funcionar pero se concluye que de por si la funcion “contains()” es bastante util ya que abarca bastantes variables que tengan una relacion entre ellas y como se ve posteriormente en el item 4.2 la forma de obtener el mismo resultado que ocurre con el anterior codigo es mucho menos efectiva.

4.1) ¿Cómo tratan los “select helpers” el caso de forma predeterminada?

Respuesta: En esta forma predeterminada la sentencia “contains(“TIME”)” es muy util ya que afecta el funcionamiento al hacer que se muestren solo las columnas cuyas variables tengan en el nombre o cadena de caracteres “time”, como curiosidad parece que no afecta si se usan las mayusculas y de paso esta sentencia funcionaria como un filtro mas efectivo de las variables.

4.2) ¿Cómo puede cambiar ese valor predeterminado?

#4.2 How can you change that default?

```
#solution with any_of

library(tidyverse)
library(nycflights13)
vars <- c("dep_time", "sched_dep_time", "arr_time", "sched_arr_time", "air_time", "time_hour")
a <- flights %>%
  select(any_of(vars))
view(a)
```

Respuesta: Se interpreto que usando la sentencia de “any_of()” podria darse un resultado igual al que se obtiene con “contains()” aunque si bien cumple con el objetivo el codigo de la parte superior, como se dijo anteriormente este no seria muy eficaz si se tiene un dataset con muchas columnas que necesiten ser filtradas; situacion que por otro lado con la sentencia “contains()” solo tomara una linea de codigo.

```
library(readr)
library(dplyr)
library(tidyverse)

myData <- nycflights13::flights
```

5.5.2 Ejercicos (mutate()+functions)

Item 1: Las categorias deploy time y scheduled deploy time se encuentran en formato de tipo horario, lo cual las hace sencillas de interpretar analiticamente, pero no de forma computacional, conviertalos a minutos

desde la medianoche

```
library(dplyr)
library(nycflights13)

flights_enminutos <- mutate(flights, dep_time_minutos=(dep_time %/% 100 * 60 + dep_time %% 100) %% 1440,

select(flights_enminutos, dep_time_minutos, sched_dep_time_minutos)

## # A tibble: 336,776 x 2
##   dep_time_minutos sched_dep_time_minutos
##           <dbl>           <dbl>
## 1             1029             1020
## 2              958              913
## 3             1240             1225
## 4             1154             1150
## 5              959              960
## 6              410              415
## 7             1435             1438
## 8              479              480
## 9             1203             1165
## 10            1429             1439
## # ... with 336,766 more rows
```

Desarrollo: El formato en defecto de las categorias deploy time y scheduled deploy time se encuentran por defecto en formato de hora militar (24 horas), de modo que es sencillo intepretar el tiempo exacto, mas no una cantidad cuantificable de expresar dicho dato, se emplea en el caso la asignacion de flights_enminutos la creacion de una nueva variable llamada dep_time_minutos, donde se va separar el formato horario, con los operadores %/% (division aritmetica) y %%(modulo), donde se toma la categoria de horas y es convertida en minutos, y el remanente de minutos en la funcion, es adicionado a la conversion realizada anteriormente, es importante anotar la presencia de la condicion de conteo de minutos desde la medianoche, donde si llegase a ser el caso, la hora de salida fuese a las 24 horas, no deberia establecerse como 1440 minutos, como 0 correspondientemente , mediante el operador mod, limitara el dato, si llega a ser igual, tendra el cuenta dicha condicion, respetandola en las variables deploy time y scheduled deploy time

Item 2: Compare el tiempo en el aire con el tiempo de llegada y el tiempo de despegue, Que se espera observar, que se muestra, y que se requiere arreglar?

```
library(dplyr)
library(nycflights13)

flights_comparacionreal <- mutate(flights, arr_time=((arr_time %/% 100*60)+(arr_time %%100))
%%1440, dep_time=((dep_time %/% 100*60)+(dep_time %%100))%%1440, difference=arr_time-dep_time, comparation=

head(flights_comparacionreal)

## # A tibble: 6 x 22
##   year month day dep_time sched_dep~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
##   <int> <int> <int>   <dbl>       <int>   <dbl>   <dbl>   <int>   <dbl> <chr>
## 1  2013     5   25     1029         1700     9     1163    1937    -14 DL
## 2  2013     7    2      958         1513    45     1065    1719     26 EV
## 3  2013     5   13     1240         2025    15     1345    2226     -1 EV
## 4  2013     3   23     1154         1910     4     1245    2043      2 EV
## 5  2013     1   12      959         1600    -1     1129    1917    -28 DL
## 6  2013    11   17      410          655    -5      659    1150    -51 DL
```

```
## # ... with 12 more variables: flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>, speed <dbl>, difference <dbl>, comparation <dbl>, and
## #   abbreviated variable names 1: sched_dep_time, 2: dep_delay, 3: arr_time,
## #   4: sched_arr_time, 5: arr_delay
```

```
View(flights_comparacionreal)
```

Desarrollo: En el rango de datos flights, se puede deducir que el tiempo en el aire, es igual a la diferencia entre el tiempo de despegue y el tiempo de llegada, por lo que se puede concluir que si se desea ver idealmente el resultado sea cero, este no es el caso para este arreglo de datos donde entran factores como el tiempo de retraso de despegue y llegada, por lo que sera comun observar diferencias de valores negativos, aspecto que en primera instancia se suele ver, para realizar la codificacion, se implementa nuevamente una conversion de formato horario en 24 horas a minutos, para realizar la diferencia entre los valores dados como en el anterior item, se crean las nuevas variables la reasignacion de arr_time y dep_time, junto con la categoria difference, que es la resta entre arr_time y dep_time, y la diferencia entre el tiempo de vuelo y la resta anterior en la variable comparation, se puede llegar a solucionar esto, tomando en cuenta los factores externos, como lo son el tiempo de retraso de despegue y llegada, para esperar que la diferencia sea cero

5.6.7 Ejercicios

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.

Item 1: Un vuelo llega 15 minutos antes el 50% del tiempo y 15 minutos tarde el 50% del tiempo.

-Vuelos con llegada 15 minutos antes

```
library(knitr)
ANTES <- myData%>%
  filter (arr_delay== (-15))
#kable(head(ANTES))
```

-Vuelos con llegada 15 minutos despues

```
DESPUES <- myData%>%
  filter (arr_delay== 15)
```

-tiempo de vuelo de 30 minutos

```
MITAD_TIEMPO <- myData%>%
  filter (air_time<=30)
```

-combinacion 15 minutos antes de llegada con 30 minutos de vuelo

```
ANTES_MITAD_TIEMPO<- myData%>%
  select( carrier,flight, tailnum, origin , dest,arr_delay,air_time)%>%
  filter (arr_delay== (-15),air_time==30)
kable(head(ANTES_MITAD_TIEMPO))
```

carrier	flight	tailnum	origin	dest	arr_delay	air_time
EV	4309	N12922	EWR	ALB	-15	30
US	2126	N957UW	LGA	BOS	-15	30
9E	2946	N8672A	JFK	PHL	-15	30
EV	3807	N14568	EWR	PVD	-15	30
US	1911	N955UW	LGA	PHL	-15	30

-combinacion 15 minutos despues de llegada con 30 minutos de vuelo

```
DESPUES_MITAD_TIEMPO<- myData%>%
  select( carrier,flight, tailnum, origin , dest,arr_delay,air_time)%>%
  filter (arr_delay== (15),air_time==30)
kable(head(DESPUES_MITAD_TIEMPO))
```

carrier	flight	tailnum	origin	dest	arr_delay	air_time
EV	4404	N17196	EWR	PVD	15	30
EV	4404	N14905	EWR	PVD	15	30
EV	4125	N14204	EWR	ALB	15	30
US	2055	N713UW	LGA	PHL	15	30

Item 2: Un vuelo siempre llega 10 minutos tarde.

```
SIEMPRE_TARDE <- myData%>%
  filter (arr_delay== 10)
```

Item 3: Un vuelo llega 30 minutos antes el 50% del tiempo y 30 minutos tarde el 50% del tiempo.

-Vuelos con llegada 30 minutos antes

```
ANTES_30 <- myData%>%
  filter (arr_delay== (-30))
```

-Vuelos con llegada 30 minutos despues

```
DESPUES_30 <- myData%>%
  filter (arr_delay== 30)
```

-tiempo de vuelo de 60 minutos

```
MITAD_TIEMPO_60 <- myData%>%
  filter (air_time<=60)
```

-combinacion 30 minutos antes de llegada con 60 minutos de vuelo

```
ANTES_MITAD_30_TIEMPO_60<- myData%>%
  select( carrier,flight, tailnum, origin , dest,arr_delay,air_time)%>%
  filter (arr_delay== (-30),air_time==60)
kable(head(ANTES_MITAD_30_TIEMPO_60))
```

carrier	flight	tailnum	origin	dest	arr_delay	air_time
EV	5378	N716EV	LGA	PIT	-30	60
9E	3320	N935XJ	JFK	BUF	-30	60
DL	1235	N315NB	LGA	PIT	-30	60
DL	1129	N301NB	LGA	PIT	-30	60
9E	3455	N919XJ	JFK	PIT	-30	60
B6	6	N304JB	JFK	BUF	-30	60

-combinacion 30 minutos despues de llegada con 60 minutos de vuelo

```
DESPUES_MITAD_30_TIEMPO_60<- myData%>%
  select( carrier,flight, tailnum, origin , dest,arr_delay,air_time)%>%
  filter (arr_delay== (30),air_time==60)
kable(head(DESPUES_MITAD_30_TIEMPO_60))
```

carrier	flight	tailnum	origin	dest	arr_delay	air_time
9E	3846	N8894A	JFK	ORF	30	60
9E	2906	N604LR	JFK	BUF	30	60
MQ	4418	N852MQ	JFK	DCA	30	60
UA	1638	N36247	EWB	CLE	30	60

Item 4: El 99% de las veces un vuelo es puntual. El 1% de las veces llega 2 horas tarde.

-vuelos puntuales aproximadamente el 99% del total menos a 120 min

```
PUNTUAL <- myData %>%
  filter(arr_delay< 120)
```

-vuelos que llegan 2 horas o mas tarde aproximadamente El 1% del total

```
DOS_HORAS_TARDE <- myData %>%
  filter(arr_delay>= 120)
```

¿Qué es más importante: el retraso en la llegada o el retraso en la salida?

Consideramos que si sabemos la hora de salida del vuelo y hay un retraso de llegada según la hora programada puede generar incertidumbre ya que pudo haber pasado algo en el vuelo o en la logística de transporte aéreo. Si embargo un retraso en la salida genera molestias para el usuario, pero no hay inseguridad para él, por tal motivo es más importante el retraso en la llegada que en la salida en termino de seguridad para quien usan el servicio y sus servidores.

5.7.1 Ejercicio

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

```
RECORD_INPUNTUAL <- myData %>%
select( carrier,flight, tailnum, origin , dest,arr_delay)%>%
  filter(arr_delay>1200)
kable(head(RECORD_INPUNTUAL))
```

carrier	flight	tailnum	origin	dest	arr_delay
HA	51	N384HA	JFK	HNL	1272

Reporting with Rmarkdown

Function: Mutate ()

La función Mutate permite crear, modificar y eliminar variables existentes en un conjunto de datos, un ejemplo básico de su uso es el siguiente:

```
ConjuntoN <- mutate(ConjuntoDD, Nvariable = Calculo)
```

Donde “**ConjuntoN**” es el conjunto al cual llegaran la modificación, “**ConjuntoDD**” es el conjunto de datos de donde se toman originalmente, “**Nvariable**” es el nombre asignado a la variable creada y por último “**Cálculo**” las expresiones que van a definir como se obtiene la variable a crear.

A continuación se muestra el código al crear un dataframe básico con tres columnas, los primeros 5 números pares, los primeros 5 números impares y los primeros 5 números primos.

```

Numeros_pares <- c(2, 4, 6, 8, 10)
Numeros_impares <- c(1, 3, 5, 7, 9)
Numeros_primos <- c(2, 3, 5, 7, 11)

Numeros <- data.frame(Numeros_pares,
                      Numeros_impares,
                      Numeros_primos)

```

Lo que genera el siguiente dataframe

```

##   Numeros_pares Numeros_impares Numeros_primos
## 1             2             1             2
## 2             4             3             3
## 3             6             5             5
## 4             8             7             7
## 5            10             9            11

```

Para modificar a través de mutate se realiza la suma de el valor 2 a cada uno de los números pares, para crear a través de mutate se se genera una nueva columna donde se hace la multiplicación de las filas entre ambas columnas del dataframe Números y adicional a ello, para eliminar a través de mutate, se remueve la columna de los números primos.

```

Numeros <- Numeros %>%
  mutate(
    Numeros_pares = Numeros_pares + 2, #Modificación
    MultParesImpares = Numeros_pares * Numeros_impares, #Creación
    Numeros_primos = NULL #Eliminación
  )

```

Donde como resultado se obtiene los nuevos numeros pares en la misma columna, una nueva columna en el dataframe Números con el resultado de la multiplicación y se observa que se eliminó la columna de los números primos

```

##   Numeros_pares Numeros_impares MultParesImpares
## 1             4             1             4
## 2             6             3            18
## 3             8             5            40
## 4            10             7            70
## 5            12             9           108

```

Function: Select ()

La funcion select() como su nombre lo indica funciona para distinguir una variable por su nombre o reasignarla a otra previamente declarada para, en este caso usarla en mas operaciones como lo son el any_of() o simplemente para mostrar esa variable independientemente del resto que se encuentra en el dataset correspondiente.

Ejemplo:

```

example <- starwars %>%
  select(name, height, mass, skin_color)
head(example)

```

```

## # A tibble: 6 x 4
##   name          height  mass skin_color
##   <chr>         <int> <dbl> <chr>
## 1 Luke Skywalker   172    77 fair
## 2 C-3PO           167    75 gold
## 3 R2-D2            96    32 white, blue

```

```
## 4 Darth Vader      202   136 white
## 5 Leia Organa      150    49 light
## 6 Owen Lars        178   120 light
```

Segun el anterior ejemplo se tiene el dataset de nombre “starwars” el cual se le asigna la variable de nombre “example” para posteriormente visualizar los datos de nombre, altura, masa y color de piel; en este caso solo se busca separa variables o distinguir estas del resto y visualizar las cuatro columnas en una grafica diferente.

Respecto al chunk “error”, puede decirse que al haber un error debajo de el este se visualiza en la parte inferior del codigo en una linea en caso de estar en true y del chunk “echo” si este esta en true lo que este en ese espacio (codigo) no se vera en el posterior archivo previsualizado.