Exploración y Aplicación de R para Data Science

Daniela Ducuara Javier Morales Miguel Angel Chavez Diego Palacios
Fernando Garcia

2023-02-27

Contents

1. Codificación básica en R. 1
2. Paquete Tidyverse 1
3. Informes con Rmarkdown 13
Summarise() 13
Group_by() 13
Arrange() 14

1. Codificación básica en R.

1. Realizamos un Script en R en donde se imprimen los números primos del 1 al 100. A continuación se muestra el desarrollo del código:

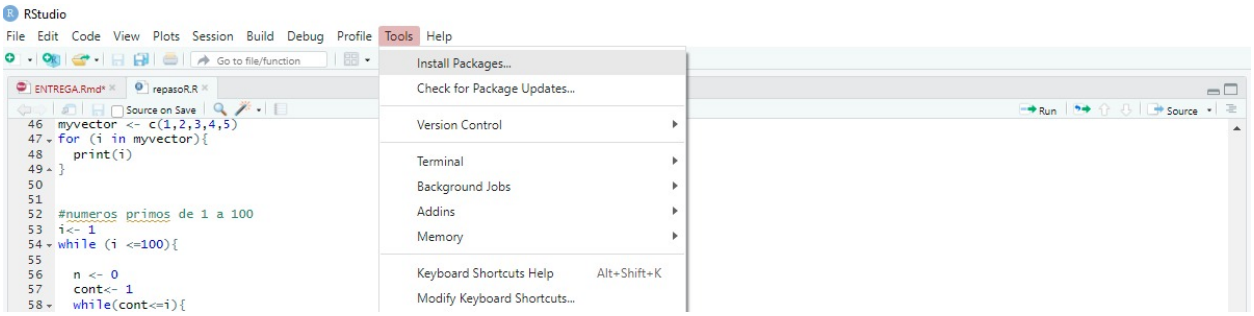
```
num <- 100L
  prime_nums <- c()
  for (i in 2:num)
    {
    is_prime <- TRUE
    for (j in prime_nums) {
      if (j > sqrt(i)) {
        break
      }
      if (i %% j == 0) {
        is_prime <- FALSE
        break
      }
    }
    if (is_prime) {
      (prime_nums <- c(prime_nums, i))

    }
  }
  print(prime_nums)
```

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

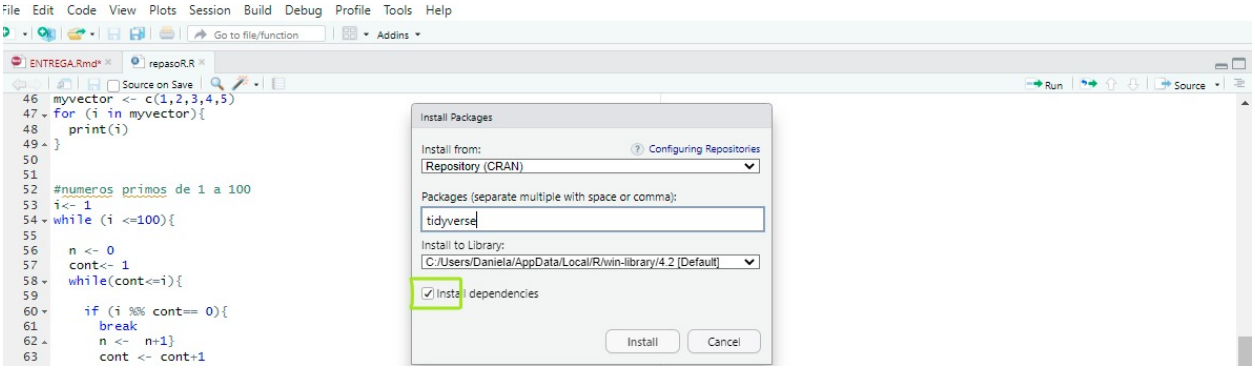
2. Paquete Tidyverse

Tidyverse es un paquete en R, que esta diseñado para ayudar en el proceso de transformación de datos. Esto es algo muy útil en la ciencia de datos ya que nos ayuda a realizar un análisis y manipulación de los datos cuando se requiera.(Rafa, 2020).

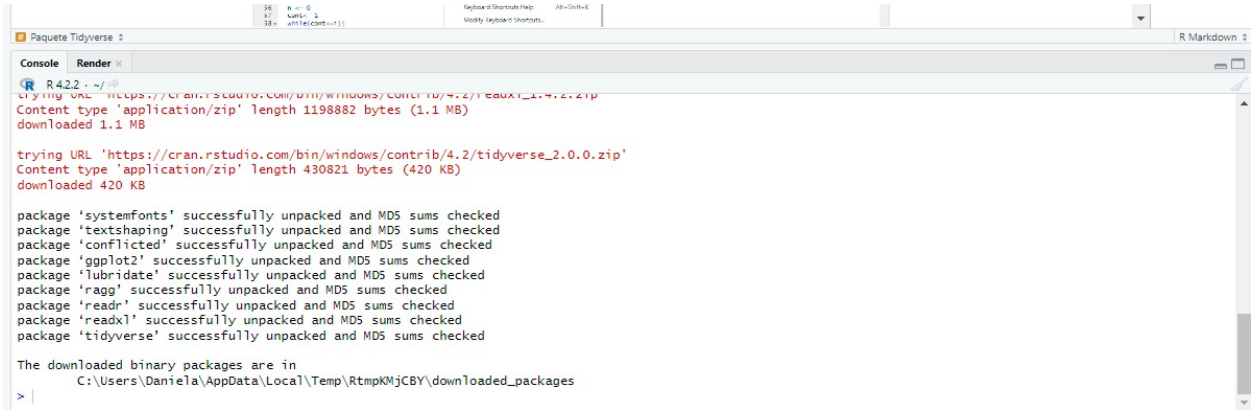
Ahora vamos a mostrar como se debe realizar la instalación del paquete en RStudio. Primero debemos ir a Tools y en la opción de Install Packages como se muestra en la imagen:



Después de seleccionar esta opción nos sale la siguiente ventana, y en la casilla debemos escribir Tidyverse, y tener seleccionada la casilla de install Dependencies, y le indicamos en Install.



Cuando se haya instalado nos aparecera un mensaje en la consola:



Una vez instalado el paquete, en el nuevo Script debemos cargar la libreria para que podamos usar las funciones del paquete.

Ya cargada la librería podemos empezar a usar las funciones. Para este caso vamos a usar el paquete dplyr, que es otro miembro central de Tidyverse. Se carga también el paquete de nycflights13 en donde nos muestra un marco de base de datos de 336.776 vuelos que se registraron en 2013 que partieron desde la ciudad de New York. Con este marco de datos empezaremos a usar las funciones que son claves para realizar la manipulación de los datos mostrando algunos ejemplos.(Wickham,H. Grolemund,G).

```
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
```

1. **Función Filter**, Usando la función de filtrer realizar los siguientes ejercicios:

- Encontrar los vuelos que tuvieron retraso de llegada de dos o mas horas

```
library(tidyverse)
library(nycflights13) #se cargan las librerias
filter(flights,arr_delay>=120)
```

```
## # 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
```

Para este punto lo que se realiza es un filtrado de los vuelos que tuvieron un retraso de llegada de dos o mas horas, por eso se coloca el mayor o igual y como la data esta en minutos por eso se coloca 120 minutos.

- Vuelos con destino a (IAH o HOU)

```
filter(flights,dest =="IAH" | dest== "HOU")%>%
  select(dest)   #Colocamos la función select para que nos aparezca solo la columna que acabamos de
```

```
## # A tibble: 9,313 x 1
##    dest
##    <chr>
##  1 IAH
##  2 IAH
```

```
##  3 IAH
##  4 IAH
##  5 IAH
##  6 IAH
##  7 IAH
##  8 IAH
##  9 IAH
## 10 IAH
## # ... with 9,303 more rows
```

```
#filtrar anteriormente
```

Podemos ver que los vuelos se filtran con los destinos corrrespondientes, en este caso para cada destino se utilizan esas abreviaturas y se usa la operación de "|" para que filtre el destino de los dos.

- Fueron operados por United, American o Delta

```
filter (flights, carrier =="DL" | carrier=="UA" | carrier =="AA" )%>%
select(carrier)
```

```
## # A tibble: 139,504 x 1
##    carrier
##    <chr>
##  1 UA
##  2 UA
##  3 AA
##  4 DL
##  5 UA
##  6 AA
##  7 UA
##  8 UA
##  9 AA
## 10 UA
## # ... with 139,494 more rows
```

En este punto se realiza el filtrado de las operadoras correspondientes, por lo que se utiliza la función "|" que es una ccondición de "o" para que nos seleccione alguna de las tres operadoras.

- Salida en verano (julio, agosto y septiembre)

```
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
```

Se realiza el filtrado de los meses correspondiente al mes de julio, agosto y septiembre, en este caso podemos ver que usamos el comando %in%, a lo que se refiere es que hace una coincidencia del valor, toma los datos de la varibale month correspondientes a los meses que se filtraron.

- Llegó más de dos horas tarde, pero no se fueron tarde.

```
filter (flights, dep_delay <=0 & arr_delay>120)
```

```
## # A tibble: 29 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    27     1419       1420      -1    1754    1550     124 MQ
##  2  2013    10     7     1350       1350       0    1736    1526     130 EV
##  3  2013    10     7     1357       1359      -2    1858    1654     124 AA
##  4  2013    10    16      657        700      -3    1258    1056     122 B6
##  5  2013    11     1      658        700      -2    1329    1015     194 VX
##  6  2013     3    18     1844       1847      -3      39    2219     140 UA
##  7  2013     4    17     1635       1640      -5    2049    1845     124 MQ
##  8  2013     4    18      558        600      -2    1149     850     179 AA
##  9  2013     4    18      655        700      -5    1213     950     143 AA
## 10  2013     5    22     1827       1830      -3    2217    2010     127 MQ
## # ... with 19 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
```

En este caso realizamos el filtrado de los vuelos que llegaron dos horas tarde pero adicional a eso los que no se fueron tarde, por eso es necesario utilizar el operador "&" para que se filtre las dos condiciones.

- Se retrasaron al menos una hora, pero recuperaron más de 30 minutos en vuelo

```
filter(filter(flights, dep_delay>=60 & arr_time-arr_delay>30))
```

```
## # A tibble: 22,806 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      826        715      71    1136    1045      51 AA
##  3  2013     1     1      848       1835     853    1001    1950     851 MQ
##  4  2013     1     1      957        733     144    1056     853     123 UA
##  5  2013     1     1     1114        900     134    1447    1222     145 UA
##  6  2013     1     1     1120        944      96    1331    1213      78 EV
##  7  2013     1     1     1301       1150      71    1518    1345      93 MQ
##  8  2013     1     1     1337       1220      77    1649    1531      78 B6
##  9  2013     1     1     1400       1250      70    1645    1502     103 EV
## 10  2013     1     1     1505       1310     115    1638    1431     127 EV
## # ... with 22,796 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
```

- Salida entre la medianoche y las 6 a. m.

```
filter (flights, hour>=0 & hour<=6)%>%
  select(flight:minute)
```

```
## # A tibble: 27,905 x 8
##    flight tailnum origin dest  air_time distance  hour minute
##     <int> <chr>   <chr>  <chr>    <dbl>    <dbl> <dbl>  <dbl>
##  1   1545 N14228  EWR    IAH        227     1400     5     15
##  2   1714 N24211  LGA    IAH        227     1416     5     29
##  3   1141 N619AA  JFK    MIA        160     1089     5     40
##  4    725 N804JB  JFK    BQN        183     1576     5     45
##  5    461 N668DN  LGA    ATL        116      762     6      0
##  6   1696 N39463  EWR    ORD        150      719     5     58
##  7    507 N516JB  EWR    FLL        158     1065     6      0
##  8   5708 N829AS  LGA    IAD         53      229     6      0
##  9     79 N593JB  JFK    MCO        140      944     6      0
## 10    301 N3ALAA  LGA    ORD        138      733     6      0
## # ... with 27,895 more rows
```

En este caso lo que se realiza es que se filtra la hora, que es la hora programada que tenía las salidas de los vuelos, se filtra entre las 0 horas y las 6 horas.

- Otro útil ayudante de filtrado de dplyr es between(). ¿Qué hace? ¿Puedes usarlo para simplificar el código necesario para responder a los desafíos anteriores?

```
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
```

```
filter(flights,between(month,7,9))
```

```
## # 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
```

La función between sirve cuando queremos realizar un filtrado de una variable entre algunos valores, lo que se le da son limites hacia la derecha o izquierda, como en el ejemplo, en el que se filtra la variable hour, entre las 0 y 6 horas y el mes entre el mes de Julio y septiembre.

2. **Función Arrange**, realizar los siguientes ejercicios:

- ¿ Cómo podría utilizar arrange()para ordenar todos los valores que faltan al principio? (Sugerencia: use is.na()).

```
arrange(flights,desc(is.na(dep_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     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
```

En esta función con el desc(), lo que realiza es que toma primero los valores mas altos al inicio, en este caso se ordena la variable dep_time, y adicional a eso nos indica que se deben colocar al inicio los que le faltan este valor, por eso se utilia is.na(), para que coloque los que no tienen algun valor en la variable.

- Ordenar flights para encontrar los vuelos más retrasados. Encuentra los vuelos que salieron antes.

```
arrange(flights, desc(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     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
```

Se ordenan los datos de forma descendente para que nos muestre primero los vuelos que tuvieron mas retraso en la salida.

```
arrange(flights, 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    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
```

En este caso, solo se coloca la función arrange y nos va a mostrar los vuelos que salieron antes de la hora programda.

- Ordenar flights para encontrar los vuelos más rápidos (velocidad más alta).

```
arrange (flights, desc(distance/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     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
##  7  2013     2    21     2355       2358      -3     412     438     -26 B6
##  8  2013    11    17      759        800      -1    1212    1255     -43 AA
##  9  2013    11    16     2003       1925      38      17      36     -19 DL
## 10  2013    11    16     2349       2359     -10     402     440     -38 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
```

En este caso se realiza la división entre la distancia y el tiempo de vuelo ya que esto correspondería a la velocidad con la que viajo cada vuelo y con la función arrange nos organiza de forma ascendente el vuelo que tuvo mas velocidad.

- ¿Qué vuelos viajaron más lejos?, ¿Cuál viajó menos?

```
arrange(flights, desc(distance))
```

```
## # 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
```

```
arrange(flights, distance)
```

```
## # 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
```

3. **Función Select**, Realizar los siguientes ejercicios

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

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

```
## # A tibble: 336,776 x 1
##    dest
```

```
##    <chr>
##  1 IAH
##  2 IAH
##  3 MIA
##  4 BQN
##  5 ATL
##  6 ORD
##  7 FLL
##  8 IAD
##  9 MCO
## 10 ORD
## # ... with 336,766 more rows
```

Lo que pasa es que la función select no duplica la misma variable las veces que se coloque en la función, solo la deja una vez ya que se esta selecionando la misma.

- Qué hace la any_of()función? ¿Por qué podría ser útil junto con este vector?

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, any_of(c("dep_time", "dep_delay", "arr_time", "arr_delay")))
```

```
## # A tibble: 336,776 x 4
##    dep_time dep_delay arr_time arr_delay
##       <int>     <dbl>    <int>     <dbl>
##  1      517         2      830        11
##  2      533         4      850        20
##  3      542         2      923        33
##  4      544        -1     1004       -18
##  5      554        -6      812       -25
##  6      554        -4      740        12
##  7      555        -5      913        19
##  8      557        -3      709       -14
##  9      557        -3      838        -8
## 10      558        -2      753         8
## # ... with 336,766 more rows
```

Podemos ver que la función any_off lo que hace es guardar en una variable, en este caso el vector, solo las varibales que se necesitan guardar y usarlos mas adelante en alguna función de select.

- ¿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"))
```

```
## # 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
```

Se observa es que este codigo lo que hace es seleccionar todas las varibales que en su mombre contenga la palabra TIME.

4. **Función Mutate y Transmutate**, Realizar lo siguientes ejercicios

- Actualmente dep_timey 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,
  minutos_deptime = (dep_time %/% 100 * 60 + dep_time %% 100) %% 1440,
  sched_dep_time,
  minutos_sched = (sched_dep_time %/% 100 * 60 +
    sched_dep_time %% 100) %% 1440
  )
```

```
## # A tibble: 336,776 x 4
##    dep_time minutos_deptime sched_dep_time minutos_sched
##       <int>           <dbl>          <int>         <dbl>
##  1      517             317            515           315
##  2      533             333            529           329
##  3      542             342            540           340
##  4      544             344            545           345
##  5      554             354            600           360
##  6      554             354            558           358
##  7      555             355            600           360
##  8      557             357            600           360
##  9      557             357            600           360
## 10      558             358            600           360
## # ... with 336,766 more rows
```

La función mutate lo que realiza es que se pueden realizar operaciones entre las variables y ese resultado se puede asignar a una variable nueva, y esa varibale se va a ver al final de todas las variables.

Para la función transmute hace dos funciones en una sola, ya que funciona como un select, selecciona las variables que queremos que queden y asi mismo podemos manipular esas variables para crear una nueva, que sería la función que cumple el mutate.

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

```
transmute(flights, air_time_diff = air_time - arr_time + dep_time)
```

```
## # A tibble: 336,776 x 1
##    air_time_diff
##            <dbl>
##  1           -86
##  2           -90
##  3          -221
##  4          -277
##  5          -142
##  6           -36
##  7          -200
##  8           -99
##  9          -141
## 10           -57
```

```
## # ... with 336,766 more rows
```

Lo que se esperaría ver de la resta de estas dos variables es que debería dar el tiempo que duro el vuelo.

5. **Función Summarise**, 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:

- Un vuelo llega 15 minutos antes el 50% del tiempo y 15 minutos tarde el 50% del tiempo.
- Un vuelo siempre llega 10 minutos tarde.
- Un vuelo llega 30 minutos antes el 50% del tiempo y 30 minutos tarde el 50% del tiempo.
- El 99% de las veces un vuelo es puntual. El 1% de las veces llega 2 horas tarde.

6. **Mutaciones agrupadas (y filtros)**, Realizar el siguiente ejercicio: ¿Qué avión (tailnum) tiene el peor récord de puntualidad?

```
table <- flights %>%
    filter(!is.na(tailnum)) %>%
    mutate(on_time = !is.na(arr_time) & (arr_delay <= 0)) %>%
    group_by(tailnum) %>%
    summarise(on_time = mean(on_time), n = n()) %>%
    filter(min_rank(on_time) == 1)
    print(table)
```

```
## # A tibble: 110 x 3
##    tailnum on_time     n
##    <chr>     <dbl> <int>
##  1 N121DE        0     2
##  2 N136DL        0     1
##  3 N143DA        0     1
##  4 N17627        0     2
##  5 N240AT        0     5
##  6 N26906        0     1
##  7 N295AT        0     4
##  8 N302AS        0     1
##  9 N303AS        0     1
## 10 N32626        0     1
## # ... with 100 more rows
```

```
    part_1 <-flights %>%
    filter(!is.na(tailnum), is.na(arr_time) | !is.na(arr_delay)) %>%
    mutate(on_time = !is.na(arr_time) & (arr_delay <= 0)) %>%
    group_by(tailnum) %>%
    summarise(on_time = mean(on_time), n = n()) %>%
    filter(n >= 20) %>%
    filter(min_rank(on_time) == 1)
    print(part_1)
```

```
## # A tibble: 1 x 3
##   tailnum on_time     n
##   <chr>     <dbl> <int>
## 1 N988AT    0.189    37
```

```
    #part(2)
    part_2 <- 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)
    print(part_2)
```

```
## # A tibble: 1 x 3
##   tailnum arr_delay     n
##   <chr>       <dbl> <int>
## 1 N203FR       59.1    41
```

3. Informes con Rmarkdown

Como sabemos el paquete dplyr es una librería que se utiliza para la manipulación y transformación de datos de una manera eficiente. Para poder comenzar a utilizar estos verbos, tendremos que tener un conjunto de datos a los que podamos acceder a ellos, es por esto que empezaremos instalando y llamando a la libreria "nycflight13"; después de esto procederemos a cargar la librería "tidyverse". Esta librería es la que almacena los verbos.

```
library(nycflights13)             #cargamos el paquete con los datos
library(tidyverse)                #cargamos el paquete tidyverse

data("flights")                   #cargamos los datos al entorno
head("flights")                   #generamos la tabla de los datos
```

```
## [1] "flights"
```

Luego de haber instalado y llamado a las librerias procederemos a cargar y generar la tabla "flights", esta es un compilado de datos acerca de los vuelos que se realizaron en el año 2013; pero no se utilizaran todos los datos de todo el año, nos basaremos en los datos del segundo mes (febrero) de el dia 16 y el antepenultimo mes del año (noviembre) de el dia 16.

```
feb16 <- filter(flights,month ==2, day == 16)
nov16 <- filter(flights,month ==11, day ==16)
```

Summarise()

Es una función que se utiliza para agregar datos en un conjunto de datos mas pequeños. Es utilizado para calcular estadísticas de resumen para diferentes grupos en un conjunto de datos, su sintaxis se muestra a continuación:

summarise(data, summary_statist = function(variable))

en donde "data" es el conjunto de datos que se desea resumir, la sección "summary_statistic" es la estadística de resumen que se desea calcular(por ejemplo, la media, mediana, etc) y la "variable" es la variable para la que se desea calcular la estadística del resumen.

Group_by()

Para poder utilizar este verbo también tendremos que utilizar otro verbo para que nos ayude y es el group_by, el cual su función es para agrupar un conjunto de datos por una o más variables, una vez se ha agrupado el conjunto de datos , se puede aplicar la funcion de resumen a cada grupo por separado, es decir la función summarise(), su sintaxis es la siguiente:

group_by(data, variable)

donde "data" encontramos el conjunto de datos que queremos agrupar y "variable" son las variables que se desean agrupar.

Arrange()

Esta función se utiliza para ordenar filas de un conjunto en base a una o varias columnas, su sintaxis general de esta función es:

arrange(data, columna1, columna2. . . .)

Donde "data" es el conjunto de datos que se desea ordenar y el grupo "columnas", son las columnas por las que se desea ordenar.

despues de tener este abrebocas explicando las anteriores funciones, podremos ver en ejemplos de como funciona. En este ejemplo que se realizó de las tablas de (feb16 y nov16) solo hicimos uso de algunas columnas esto para poder organizar de una forma descendente la distancia de cada vuelo.

```
feb16_2 <- select(feb16,month,day,flight,distance,hour) #escoge solo las columnas escritas dentro del s
arrange(feb16_2, desc(distance)) #organiza los valores de distancia de mayor a menor del 16 de febrero
```

```
## # A tibble: 738 x 5
##    month   day flight distance  hour
##    <int> <int>  <int>    <dbl> <dbl>
##  1     2    16     51     4983     9
##  2     2    16     15     4963    13
##  3     2    16    303     2586     6
##  4     2    16   1865     2586     7
##  5     2    16     11     2586     7
##  6     2    16    643     2586     7
##  7     2    16    309     2586     7
##  8     2    16     59     2586     7
##  9     2    16    179     2586    10
## 10     2    16    642     2586    11
## # ... with 728 more rows
```

```
ejem_arran <- arrange(feb16_2,desc(distance)) #crea la tabla
```

```
nov16_2 <- select(nov16,month,day,flight,distance,hour) #escoge solo las columnas escritas dentro del s
```

```
arrange(nov16_2, desc(distance))  #organiza los valores de distancia de mayor a menor del 16 de febrero
```

```
## # A tibble: 714 x 5
##    month   day flight distance  hour
##    <int> <int>  <int>    <dbl> <dbl>
##  1    11    16     51     4983    10
##  2    11    16     15     4963     9
##  3    11    16    303     2586     6
##  4    11    16    430     2586     7
##  5    11    16    397     2586     8
##  6    11    16     59     2586     7
##  7    11    16     11     2586     7
##  8    11    16    505     2586     9
##  9    11    16    469     2586    10
## 10    11    16    642     2586    10
## # ... with 704 more rows
```

```
ejem_arran <- arrange(nov16_2,desc(distance))
```

4.GitHub – Integración de paquetes R

Entrar al siguiente enlace para ingresar al repositorio.

https://github.com/Khallavan/classworkPackage