Exercícios Cap 05

Rodrigo Giannotti

Capitulo 05

Inicialização

```r
library(tidyverse)
library(magrittr) # mais pipes, como %<>%
library(lubridate) # melhor manejo de datas
```

Para o capitulo 5 também utilizaremos a biblioteca de voos de NYC

```r
library(nycflights13)
# ?flights
# View(flights)
head (flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Exercícios

5.2 filter()

5.2.1

Find all flights that:

a Had an arrival delay of two or more hours

```
flights %>% filter(
  arr_delay >= 120
  )
```

```
## # A tibble: 10,200 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      811            630       101     1047            830
##  2  2013     1     1      848           1835       853     1001           1950
##  3  2013     1     1      957            733       144     1056            853
##  4  2013     1     1     1114            900       134     1447           1222
##  5  2013     1     1     1505           1310       115     1638           1431
##  6  2013     1     1     1525           1340       105     1831           1626
##  7  2013     1     1     1549           1445        64     1912           1656
##  8  2013     1     1     1558           1359       119     1718           1515
##  9  2013     1     1     1732           1630        62     2028           1825
## 10  2013     1     1     1803           1620       103     2008           1750
## # ... with 10,190 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

b Flew to Houston (IAH or HOU)

```r
flights %>% filter(
  dest %in% c("IAH","HOU")
  )
```

```
## # A tibble: 9,313 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      623            627        -4      933            932
##  4  2013     1     1      728            732        -4     1041           1038
##  5  2013     1     1      739            739         0     1104           1038
##  6  2013     1     1      908            908         0     1228           1219
##  7  2013     1     1     1028           1026         2     1350           1339
##  8  2013     1     1     1044           1045        -1     1352           1351
##  9  2013     1     1     1114            900       134     1447           1222
## 10  2013     1     1     1205           1200         5     1503           1505
## # ... with 9,303 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

c Were operated by United, American or Delta

```
flights %>% filter(
  carrier %in% c("UA", "AA", "DL")
  )
```

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

d Departed in the summer (July, August and September)

```
summer <- c(7:9)
flights %>% filter(
  month %in% summer
  )
```

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     1        1           2029       212      236           2359
##  2  2013     7     1        2           2359         3      344            344
##  3  2013     7     1       29           2245       104      151              1
##  4  2013     7     1       43           2130       193      322             14
##  5  2013     7     1       44           2150       174      300            100
##  6  2013     7     1       46           2051       235      304           2358
##  7  2013     7     1       48           2001       287      308           2305
##  8  2013     7     1       58           2155       183      335             43
##  9  2013     7     1      100           2146       194      327             30
## 10  2013     7     1      100           2245       135      337            135
## # ... with 86,316 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

e Arrived more than two hours late, but didn't leave late

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

```
## # A tibble: 29 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    27     1419           1420        -1     1754           1550
##  2  2013    10     7     1350           1350         0     1736           1526
##  3  2013    10     7     1357           1359        -2     1858           1654
##  4  2013    10    16      657            700        -3     1258           1056
##  5  2013    11     1      658            700        -2     1329           1015
##  6  2013     3    18     1844           1847        -3       39           2219
##  7  2013     4    17     1635           1640        -5     2049           1845
##  8  2013     4    18      558            600        -2     1149            850
##  9  2013     4    18      655            700        -5     1213            950
## 10  2013     5    22     1827           1830        -3     2217           2010
## # ... with 19 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

f Were delayed by at least an hour, but made up over 30 minutes in flight

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

```
## # A tibble: 2,074 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1     1716           1545        91     2140           2039
##  2  2013     1     1     2205           1720       285       46           2040
##  3  2013     1     1     2326           2130       116      131             18
##  4  2013     1     3     1503           1221       162     1803           1555
##  5  2013     1     3     1821           1530       171     2131           1910
##  6  2013     1     3     1839           1700        99     2056           1950
##  7  2013     1     3     1850           1745        65     2148           2120
##  8  2013     1     3     1923           1815        68     2036           1958
##  9  2013     1     3     1941           1759       102     2246           2139
## 10  2013     1     3     1950           1845        65     2228           2227
## # ... with 2,064 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

g Departed between 00:00 and 6:00 (inclusive)

```
flights %>%  filter(
  dep_time <= 600 | dep_time == 2400
  )
```

```
## # A tibble: 9,373 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 9,363 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.2.2

Another useful dplyr filtering helper is between(). What does it do? Can you use it to simplify the code needed to answer the previous challenges?

```
# ?between
```

Como dito na ajuda, “This is a shortcut for x >= left & x <= right” ou seja, é uma maneira de testar se valores dentro de um vetor estão dentro de dois limites.

Isso só seria útil para simplificar a questão dos meses do verão

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

```
## # A tibble: 86,326 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7     1        1           2029       212      236           2359
##  2  2013     7     1        2           2359         3      344            344
##  3  2013     7     1       29           2245       104      151              1
##  4  2013     7     1       43           2130       193      322             14
##  5  2013     7     1       44           2150       174      300            100
##  6  2013     7     1       46           2051       235      304           2358
##  7  2013     7     1       48           2001       287      308           2305
##  8  2013     7     1       58           2155       183      335             43
##  9  2013     7     1      100           2146       194      327             30
## 10  2013     7     1      100           2245       135      337            135
## # ... with 86,316 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.2.3

How many flights have a missing dep_time? What other variables are missing? What might these rows represent?

```
flights %>% filter(
  is.na(dep_time)
  )
```

```
## # A tibble: 8,255 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1       NA           1630        NA       NA           1815
##  2  2013     1     1       NA           1935        NA       NA           2240
##  3  2013     1     1       NA           1500        NA       NA           1825
##  4  2013     1     1       NA            600        NA       NA            901
##  5  2013     1     2       NA           1540        NA       NA           1747
##  6  2013     1     2       NA           1620        NA       NA           1746
##  7  2013     1     2       NA           1355        NA       NA           1459
##  8  2013     1     2       NA           1420        NA       NA           1644
##  9  2013     1     2       NA           1321        NA       NA           1536
## 10  2013     1     2       NA           1545        NA       NA           1910
## # ... with 8,245 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Podemos ver que os voos com dep_time vazio apresentam outras colunas vazias, como dep_delay; arr_time; arr_delay; e air_time

Isso muito provavelmente indica voos que foram cancelados

5.2.4

Why is NA ˆ 0 not missing? Why is NA | TRUE not missing? Why is FALSE & NA not missing? Can you figure out the general rule? (NA * 0 is a tricky counterexample!)

```
# help(`^`)
# help(`|`)
```

Como podemos ver no texto de ajuda "1 ˆ y and y ˆ 0 are 1, always.", dessa forma o operador nem passa pela etapa de avaliar o NA, simplesmente retornando o resultado.

Similarmente, "NA is a valid logical object. Where a component of x or y is NA, the result will be NA if the outcome is ambiguous. In other words NA & TRUE evaluates to NA, but NA & FALSE evaluates to FALSE. See the examples below."

Logo como sempre (x | TRUE) retornaria TRUE e (x & FALSE) retornaria FALSE independentemente dos valores de x, logo retornam-se os valores lógicos.

Isso só ocorre quando o computador está explicitamente tomando a decisão de não avaliar a expressão como um todo, devido à um de seus lados. não existe tão decisão para NA * 0, por exemplo, logo o resultado esperado é NA.

```
NA * 0
```

```
## [1] NA
```

5.3 arrange()

5.3.1

How could you use arrange() to sort all missing values to the start? (Hint: use is.na().)

```
flights %>% arrange(
  desc(
    is.na(dep_time)
  ))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1       NA           1630        NA       NA           1815
##  2  2013     1     1       NA           1935        NA       NA           2240
##  3  2013     1     1       NA           1500        NA       NA           1825
##  4  2013     1     1       NA            600        NA       NA            901
##  5  2013     1     2       NA           1540        NA       NA           1747
##  6  2013     1     2       NA           1620        NA       NA           1746
##  7  2013     1     2       NA           1355        NA       NA           1459
##  8  2013     1     2       NA           1420        NA       NA           1644
##  9  2013     1     2       NA           1321        NA       NA           1536
## 10  2013     1     2       NA           1545        NA       NA           1910
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.3.2

Sort flights to find the most delayed flights. Find the flights that left earliest.

```
flights %>% arrange(
  desc(
    (dep_delay + arr_delay)
  ))  # maior atraso somado entre saída e chegada
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     7    22     2257            759       898      121           1026
##  9  2013    12     5      756           1700       896     1058           2020
## 10  2013     5     3     1133           2055       878     1250           2215
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights %>% arrange(
  dep_delay
  )   # menor atraso de saída
```

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

5.3.3

Sort flights to find the fastest (highest speed) flights.

```r
flights %>% arrange(
  desc(
    distance / air_time
  )
)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     5    25     1709           1700         9     1923           1937
##  2  2013     7     2     1558           1513        45     1745           1719
##  3  2013     5    13     2040           2025        15     2225           2226
##  4  2013     3    23     1914           1910         4     2045           2043
##  5  2013     1    12     1559           1600        -1     1849           1917
##  6  2013    11    17      650            655        -5     1059           1150
##  7  2013     2    21     2355           2358        -3      412            438
##  8  2013    11    17      759            800        -1     1212           1255
##  9  2013    11    16     2003           1925        38       17             36
## 10  2013    11    16     2349           2359       -10      402            440
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.3.4

Which flights traveled the longest? Which traveled the shortest?

```
flights %>% arrange(
  desc(
    distance
  ))    # voos mais longos
```

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

```
flights %>% arrange(
  distance
  )    # voos mais curtos
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     7    27       NA            106        NA       NA            245
##  2  2013     1     3     2127           2129        -2     2222           2224
##  3  2013     1     4     1240           1200        40     1333           1306
##  4  2013     1     4     1829           1615       134     1937           1721
##  5  2013     1     4     2128           2129        -1     2218           2224
##  6  2013     1     5     1155           1200        -5     1241           1306
##  7  2013     1     6     2125           2129        -4     2224           2224
##  8  2013     1     7     2124           2129        -5     2212           2224
##  9  2013     1     8     2127           2130        -3     2304           2225
## 10  2013     1     9     2126           2129        -3     2217           2224
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5.4 select()

5.4.1

Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time, and arr_delay from flights.

Por mais que uma partida de regex golf sempre tenha seu valor nostalgico a função starts_with() resolve o problema com mais simplicidade

```
flights %>% select(
  starts_with("dep_"
  ) |
  starts_with("arr_"
  )
)
```

```
## # 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
```

5.4.2

What happens if you include the name of a variable multiple times in a select() call?

```
flights %>% select(
  dep_time, dep_time, dep_time, arr_time, dep_time
)
```

```
## # A tibble: 336,776 x 2
##    dep_time arr_time
##       <int>    <int>
##  1      517      830
##  2      533      850
##  3      542      923
##  4      544     1004
##  5      554      812
##  6      554      740
##  7      555      913
##  8      557      709
##  9      557      838
## 10      558      753
## # ... with 336,766 more rows
```

Somente uma copia dessa coluna chega ao resultado final

5.4.3

What does the one_of() function do? Why might it be helpful in conjunction with this vector?

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

```
# ?tidyselect::one_of
```

Como podemos ver o próprio tidyverse sugere o uso das mais precisas all_of() ou any_off(), que servem para - em conjunto com o comando select() selecionar variáveis com nomes dentro de listas. o comando all_of retorna erro se algum dos nomes da lista não for encontrado como nome de coluna enquanto o any_off ignora as colunas que não forem encontradas. No caso ambos devem retornar o mesmo dataframe, visto que todas as colunas da lista existem.

```
flights %>% select(
  any_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
```

5.4.4

Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that 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> <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
```

```
# ?select
```

Isso não surpreende, mas poderia ser interessante esperar que o helper “contains()” considerasse caixa baiza ou alta. Para mudar seu comportamento para que passe a considerar isso basta usar o argumento ignore.case = FALSE

```
flights %>% select(
  contains(
    "TIME", ignore.case = F
  )
)
```

```
## # A tibble: 336,776 x 0
```

5.5 mutate()

5.5.1

Currently dep_time and sched_dep_time are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

```
(flights2 <- flights %>% mutate(
  dep_time_mins = (dep_time %/% 100) * 60 + dep_time %% 100,
  sched_dep_time_mins = (sched_dep_time %/% 100) * 60 + sched_dep_time %% 100
))
```

```
## # A tibble: 336,776 x 21
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 13 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   dep_time_mins <dbl>, sched_dep_time_mins <dbl>
```

5.5.2

Compare air_time with arr_time - dep_time. What do you expect to see? What do you need to do to fix it?

Primeiro vamos dar o mesmo tratamento que demos para os horários de saída para os horários de chegada

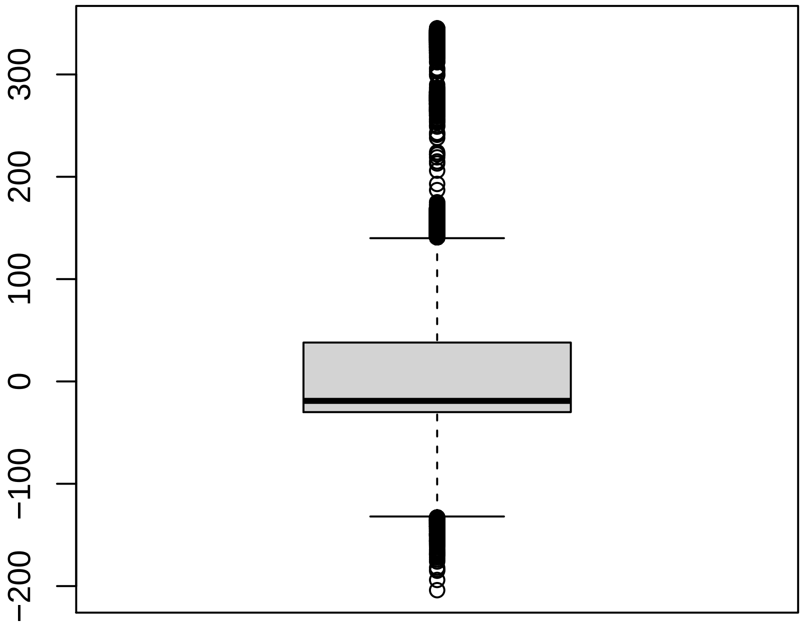
```
flights2 %<>% mutate(
  sched_arr_time_mins = (sched_arr_time %/% 100) * 60 + sched_arr_time %% 100,
  arr_time_mins = (arr_time %/% 100) * 60 + arr_time %% 100
)
```

Com isso vamos criar uma nova coluna, que compara o horário efetivo de saída com o de chegada, tomando alguns cuidados, especialmente com voos que saem em uma data e chegam em outra. Supoe-se que nenhum voo voa por mais de 24h nesse caso (e uma rápida olhada para a coluna air_time confirma isso)

```
flights2 %<>% mutate(
  arr_dep_time_diff = ifelse(
    arr_time_mins >= dep_time_mins, # se o voo chegou no dia seguinte a conta arr_time - dep_time daria
    arr_time_mins - dep_time_mins, # caso não hajam problemas
    arr_time_mins - dep_time_mins + 24*60) # adicionando o numero de minutos em um dia caso o voo tenha
  )
```

Agora vamos comparar a diferença entre horário de saída com a coluna air_time

```
flights2 %>%
  na.omit() %>%
  mutate(
  comparacao =
    air_time - arr_dep_time_diff
) %>%
  select(comparacao) %>%
  boxplot()
```



```
flights2 %>%
  na.omit() %>%
  mutate(
  comparacao =
    air_time - arr_dep_time_diff
) %>%
  select(comparacao) %>%
  summary()
```

```
##    comparacao
##  Min.   :-204.00
##  1st Qu.: -30.00
##  Median : -19.00
##  Mean   :  13.38
##  3rd Qu.:  38.00
##  Max.   : 345.00
```

Esperaria-se que essa comparação fosse sempre igual a 0, visto que se espera que o tempo de voo se iguale à diferença de hora de saída e hora de chegada, o problema é que neste caso estamos ignorando o tempo de taxi e que existam imperfeições de registro. (poderiamos ter questoes de fuso horario, mas ?flights nos informa que todos os horarios estão na timezone de NYC)

5.5.3

Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three numbers to be related?

```
flights2 %>% mutate(
  comparison = ifelse(
    dep_time_mins >= (sched_dep_time_mins + dep_delay),
    dep_time_mins - (sched_dep_time_mins + dep_delay),
    dep_time_mins - (sched_dep_time_mins + dep_delay) + 24*60
    )) %>%
  select(comparison) %>%
  summary()
```

```
##    comparison
##  Min.   :0
##  1st Qu.:0
##  Median :0
##  Mean   :0
##  3rd Qu.:0
##  Max.   :0
##  NA's   :8255
```

Esperaria-se que essa comparação retornasse valor nulo, o que confirmamos com o comando summary()

5.5.4

Find the 10 most delayed flights using a ranking function. How do you want to handle ties? Carefully read the documentation for min_rank().

```
flights %>%
  mutate(
  rank_delay = min_rank( desc(
      arr_delay + dep_delay
))) %>%
  arrange(rank_delay) %>%
  filter(rank_delay <= 10)
```

```
## # A tibble: 10 x 20
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     7    22     2257            759       898      121           1026
##  9  2013    12     5      756           1700       896     1058           2020
## 10  2013     5     3     1133           2055       878     1250           2215
## # ... with 12 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>, rank_delay <int>
```

```
# ?rank
```

Por sorte não temos empates usando essa formula, mas se tivessemos por usar a forma min_rank teriamos que empates teriam o mesmo rank, este sendo o minimo entre eles.

5.5.5

What does 1:3 + 1:10 return? Why?

```
1:3 + 1:10
```

```
## Warning in 1:3 + 1:10: longer object length is not a multiple of shorter object
## length
```

```
##  [1]  2  4  6  5  7  9  8 10 12 11
```

a lógica é 1+1 2+2 3+3 e então, como a lista menor acaba temos 1+4 2+5 3+6 1+7 2+8 3+9 1+10 E um aviso informando que a lista maior não tem uma quantidade de itens multipla da quantidade de itens na lista menor.

5.5.6

What trigonometric functions does R provide?

```
# ?sin
```

O texto de ajuda das funções trigonometricas lista as seguintes: "These functions give the obvious trigonometric functions. They respectively compute the cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent, and the two-argument arc-tangent.

5.6 summarise()

Para essa seção em alguns exercícios utilizaremos uma tabela com os voos que foram cancelados.

```r
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
```

5.6.1

Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights. Consider the following scenarios:

a A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.

b A flight is always 10 minutes late.

c A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.

d 99% of the time a flight is on time. 1% of the time it's 2 hours late.

e Which is more important: arrival delay or departure delay?

5.6.2

Come up with another approach that will give you the same output as not_cancelled %>% count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count()).

```r
identical(
  not_cancelled %>% count(dest),
  not_cancelled %>%
    group_by(dest) %>%
    summarise(n = n(), .groups = 'drop')
)
```

```
## [1] TRUE
```

```r
identical(
  not_cancelled %>% count(tailnum, wt = distance),
  not_cancelled %>%
    group_by(tailnum) %>%
    summarise(n = sum(distance), .groups = 'drop')
)
```

```
## [1] TRUE
```

5.6.3

Our definition of cancelled flights (is.na(dep_delay) | is.na(arr_delay)) is slightly suboptimal. Why? Which is the most important column?

Primeiro, vamos olhar para a coluna com a maior quantidade de NAs

```
flights %>% filter(is.na(arr_delay))
```

```
## # A tibble: 9,430 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1     1525           1530        -5     1934           1805
##  2  2013     1     1     1528           1459        29     2002           1647
##  3  2013     1     1     1740           1745        -5     2158           2020
##  4  2013     1     1     1807           1738        29     2251           2103
##  5  2013     1     1     1939           1840        59       29           2151
##  6  2013     1     1     1952           1930        22     2358           2207
##  7  2013     1     1     2016           1930        46       NA           2220
##  8  2013     1     1       NA           1630        NA       NA           1815
##  9  2013     1     1       NA           1935        NA       NA           2240
## 10  2013     1     1       NA           1500        NA       NA           1825
## # ... with 9,420 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Encontramos alguns voos com horário de saída, horário de chegada, mas nem tempo de voo nem atraso de chegada.

É difícil saber como considerar se estes voos foram ou não cancelados, mas me parece que o mais correto seria considerar um voo cancelado como aquele que não decolou, e portanto, tem dep_time == NA

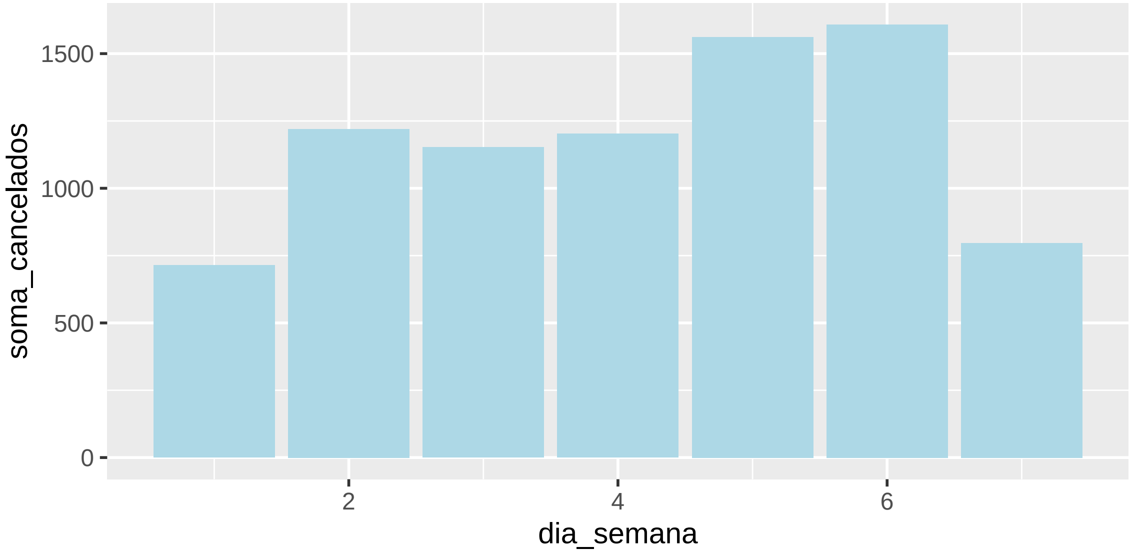
5.6.4

Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

É de se imaginar que se existe algum padrão de cancelamentos ou atrasos estes se dariam em datas especiais ou em dias da semana mais comuns. Para isso usando a coluna time_hour e o pacote lubridate podemos criar colunas para ter mais insights sobre esses padrões.

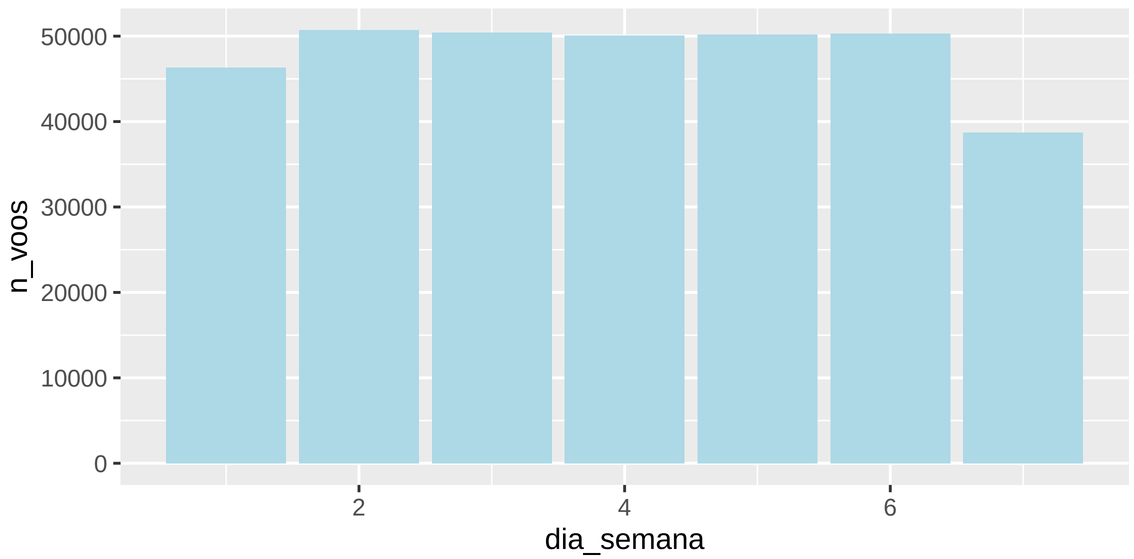
```r
flights3 <- flights %>%
  mutate(dia_semana = lubridate::wday(time_hour))

flights3 %>%
  group_by(dia_semana) %>%
  summarise(soma_cancelados = sum(is.na(dep_time)), .groups = "drop") %>%
  ggplot(mapping = aes(x = dia_semana, y = soma_cancelados)) +
  geom_col(fill = "lightblue") # voos cancelados por dia da semana
```



Primeiro podemos ver que muito menos voos foram cancelados durante o fim de semana, mas será que existe uma correlação com o número total de voos nesses dias?

```r
flights3 %>%
  group_by(dia_semana) %>%
  summarise(n_voos = n(), .groups = "drop") %>%
  ggplot(mapping = aes(x = dia_semana, y = n_voos)) +
  geom_col(fill = "lightblue") # voos totais por dia da semana
```



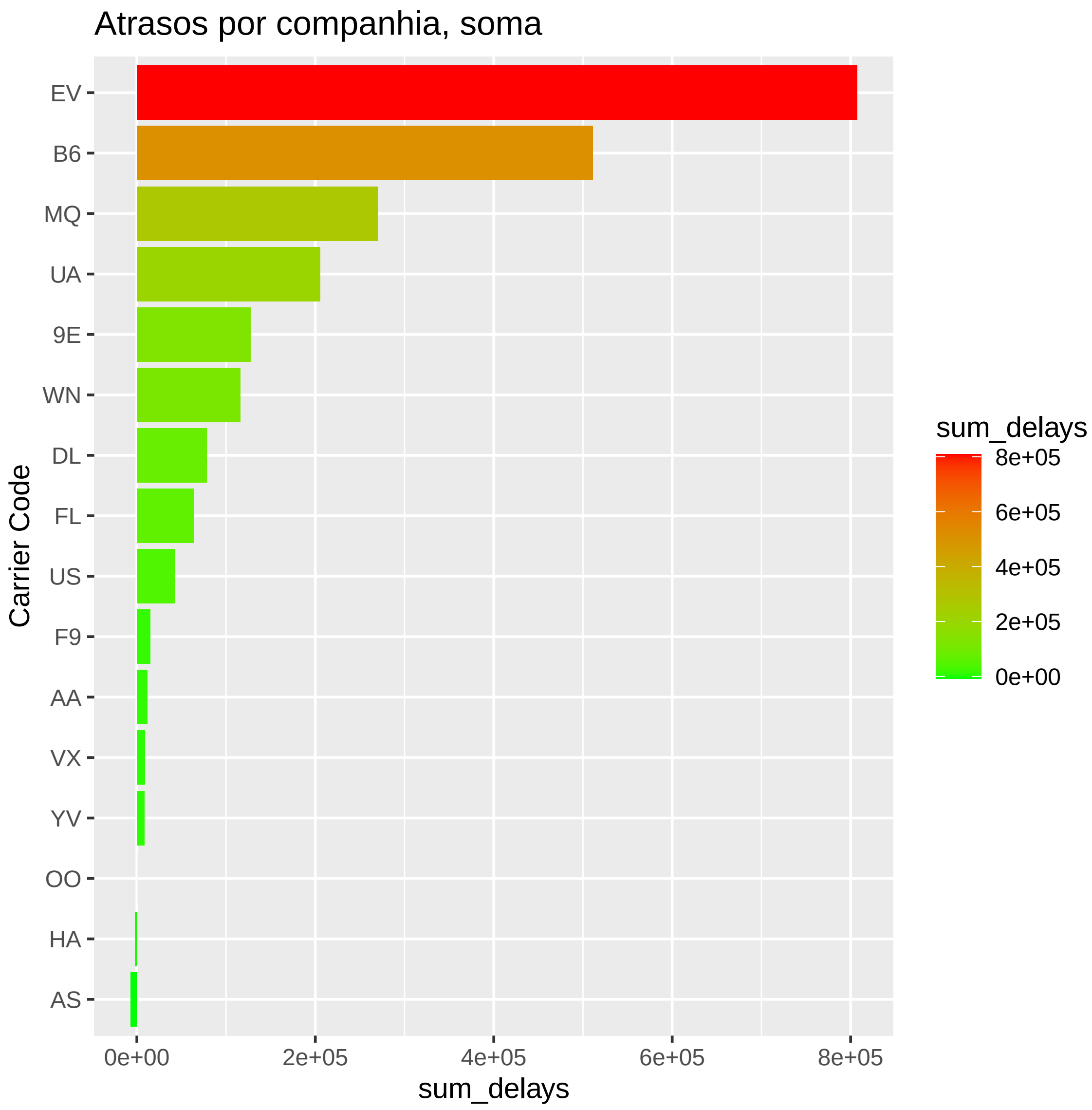
Por mais que realmente existam menos voos marcados durante fins de semana parece que muitos mais são cancelados durante a semana do que em fins de semana.

5.6.5

Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about flights %>% group_by(carrier, dest) %>% summarise(n()))

Para essa Análise me parece o mais razoável analizar somente o arr_delay, visto que por mais chato que seja ficar aguardando no aeroporto no final o que mais importa é se o avião chega ao seu destino final no horário, para os passageiros.

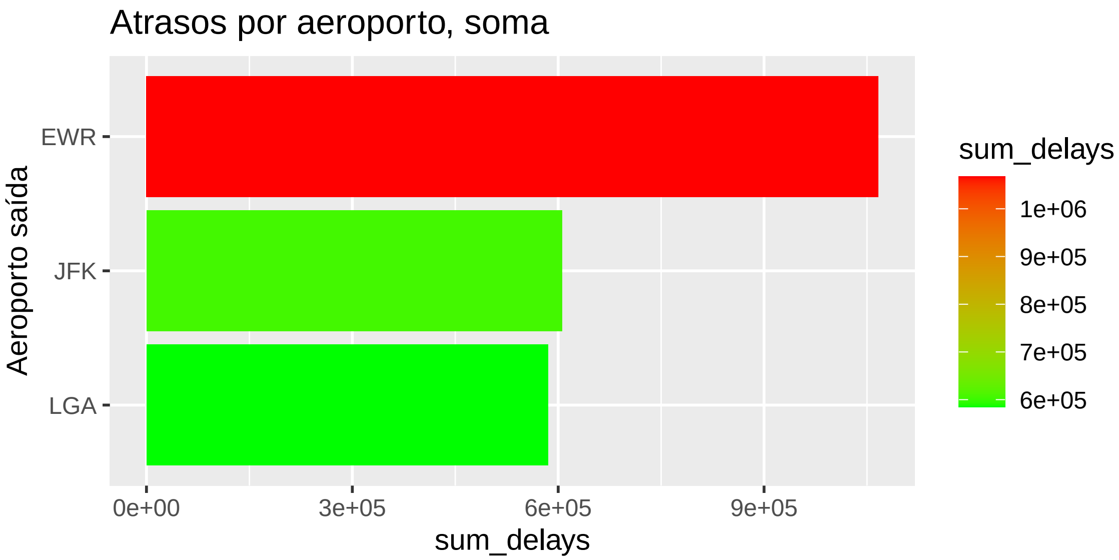
```
not_cancelled %>%
  group_by(carrier) %>%
  summarise(sum_delays = sum(arr_delay), .groups = 'drop') %>%
  ggplot(mapping = aes(x = reorder(carrier, X = sum_delays), y = sum_delays, fill = sum_delays)) +
  geom_col() +
  coord_flip() +
  scale_fill_gradient(low="green", high="red") +
  xlab("Carrier Code") +
  ggtitle("Atrasos por companhia, soma")
```



Por mais bonito que o gráfico possa ficar o mais justo seria considerar o atraso médio, não o total de atrasos no período (ou ainda vale pensar qual a medida estatística mais adequada, por exemplo uma medida mais robusta poderia ser mais interessante para o passageiro médio).

Feitas essas considerações seguimos com o exercício.

```r
not_cancelled %>%
  group_by(origin) %>%
  summarise(sum_delays = sum(arr_delay), .groups = 'drop') %>%
  ggplot(mapping = aes(x = reorder(origin, X = sum_delays), y = sum_delays, fill = sum_delays)) +
  geom_col() +
  coord_flip() +
  scale_fill_gradient(low="green", high="red") +
  xlab("Aeroporto saída") +
  ggtitle("Atrasos por aeroporto, soma")
```



É bastante evidente que as companhias que mais voam de EWR inevitavelmente incorrerão em mais atrasos na média.

Poderiamos tentar um score que ponderasse o atraso médio dos aeroportos na avaliação das companhias aéreas, mas ainda assim é possível argumentar que as responsáveis pelos atrasos nos aeroportos sejam as companhias que lá mais operam.

5.6.6

What does the sort argument to count() do. When might you use it?

```
# ?count
```

O argumento sort, se == TRUE fará com que na saída da função count() os maiores grupos estarão no topo, ordenados.

5.7 grouped mutates (and filters)

5.7.1

Refer back to the lists of useful mutate and filtering functions. Describe how each operation changes when you combine it with grouping.

5.7.2

Which plane (tailnum) has the worst on-time record?

5.7.3

What time of day should you fly if you want to avoid delays as much as possible?

5.7.4

For each destination, compute the total minutes of delay. For each flight, compute the proportion of the total delay for its destination.

5.7.5

Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using lag(), explore how the delay of a flight is related to the delay of the immediately preceding flight.

5.7.6

Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time of a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

5.7.7

Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.

5.7.8

For each plane, count the number of flights before the first delay of greater than 1 hour.