chapter-two-exercise

Anyanwu Chinedu

Chapter two exercise

Package(s) & dataset(s) used in this exercise includes:

- tidyverse
- nycflights13 data set

Importing the libary and data set

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

`nycflights13 dataset` is a data frame containing all the 336,776 flights that departed from the New York City in 2013.

Filter Row with filter()

Question 1. Find all the flights that:

a. Had an interval delay of two or more hours
b. Flew to Houston (IAH or HOU)
c. Were operated by United, America, or Delta
d. Departed in summer(July, August, and September)
e. Arrived more than two hours late,but didn't leave late
f. Were delayed by at least an hour, but made up over 30 minutes in flight
g. Departed between midnight and 6 a.m

Answer(a):

```
filter(nycflights13::flights, arr_delay >= 2)
```

```
## # A tibble: 127,929 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            558        -4      740            728
##  5  2013     1     1      555            600        -5      913            854
##  6  2013     1     1      558            600        -2      753            745
##  7  2013     1     1      558            600        -2      924            917
##  8  2013     1     1      559            600        -1      941            910
##  9  2013     1     1      600            600         0      837            825
## 10  2013     1     1      602            605        -3      821            805
## # i 127,919 more rows
## # i 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>
```

Answer(b):

```
filter(nycflights13::flights, dest == 'IAH'| dest == '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
## # i 9,303 more rows
## # i 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>
```

Answer(c):

```
filter(nycflights13::flights, carrier %in% c('AA','DL','UA'))
```

```
## # 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
## # i 139,494 more rows
## # i 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>
```

Answer(d):

```
filter(nycflights13::flights, month %in% c(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
## # i 86,316 more rows
## # i 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>
```

Answer(e):

```
filter(flights, arr_delay > 2 & dep_delay <= 0)
```

```
## # A tibble: 34,583 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      554            558        -4      740            728
##  2  2013     1     1      555            600        -5      913            854
##  3  2013     1     1      558            600        -2      753            745
##  4  2013     1     1      558            600        -2      924            917
##  5  2013     1     1      559            600        -1      941            910
##  6  2013     1     1      600            600         0      837            825
##  7  2013     1     1      602            605        -3      821            805
##  8  2013     1     1      622            630        -8     1017           1014
##  9  2013     1     1      624            630        -6      909            840
## 10  2013     1     1      624            630        -6      840            830
## # i 34,573 more rows
## # i 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>
```

Answer(f):

```
filter(flights, dep_delay >= 1 & air_time > 30)
```

```
## # A tibble: 127,205 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      601            600         1      844            850
##  5  2013     1     1      608            600         8      807            735
##  6  2013     1     1      611            600        11      945            931
##  7  2013     1     1      613            610         3      925            921
##  8  2013     1     1      623            610        13      920            915
##  9  2013     1     1      632            608        24      740            728
## 10  2013     1     1      644            636         8      931            940
## # i 127,195 more rows
## # i 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>
```

Answer(g):

```
filter(flights, sched_dep_time %in% c(000:600))
```

```
## # A tibble: 8,970 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
## # i 8,960 more rows
## # i 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>
```

Note: The answer above might not be reliable. Though an execution of ?flights reveals that `sched_dep_time` has the format `HHMM` or `HMM`, but an attempt to print only scheduled departure times for midnight returned empty record. So my use of 000 here to represent midnight time is an assumption which favors the 24hr clock format

```
flights$sched_dep_time[flights$sched_dep_time == 000]
```

```
## integer(0)
```

Question 2. Using the between() dplyr tool to simplify the problems above

The between() tool is used to detect where values fall in a specified range

Answer(d):

```
filter(flights, 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
## # i 86,316 more rows
## # i 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>
```

Answer(g):

```
filter(flights, between(sched_dep_time, 000, 600))
```

```
## # A tibble: 8,970 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
## # i 8,960 more rows
## # i 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>
```

Question 3. How many flights have a missing dep_time?. What other variables are missing? What might these rows represent?

```
count(filter(flights, is.na( dep_time)))
```

Answer: To find the number of flights with missing departure time:

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1  8255
```

Answer: To find other missing rows

```
filter(flights, 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
## # i 8,245 more rows
## # i 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>
```

Other missing variables include `dep_delay` and `arr_time` which represent missing observations

Arrange Rows with arrange()

Question 1: How could you used arrange() to sort all missing values to the start?(Hint: use is.na())

Answer: The arguments to the arrange() method must start with checks for unavailable data as shown below

```
df <- tibble(x=c(2:8, NA))
arrange(df, -is.na(x))
```

```
## # A tibble: 8 x 1
##       x
##   <int>
## 1    NA
## 2     2
## 3     3
## 4     4
## 5     5
## 6     6
## 7     7
## 8     8
```

Question 2: Sort `flights` to find the most delayed flights. Find the flights that left earliest

Answer(most delayed flights appear topmost):

```
arrange(flights, desc(dep_delay))
```

```
## # 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     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 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>
```

Answer(flights that left erliest appear topmost):

```
arrange(flights, -desc(dep_delay))
```

```
## # 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
## # i 336,766 more rows
## # i 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>
```

Question: sort `flights` to find the fastest flights

Answer:(ensuring distances are accounted for, the fastest flights appear topmost):

```
arrange(flights,desc(distance), -desc(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     7      959           1000        -1     1401           1500
##  2  2013     6     6     1044           1000        44     1441           1435
##  3  2013     9    29      957           1000        -3     1405           1445
##  4  2013     6     7      952           1000        -8     1354           1435
##  5  2013     6     8      951           1000        -9     1352           1435
##  6  2013     9     6      955           1000        -5     1359           1445
##  7  2013     2    26     1000            900        60     1513           1540
##  8  2013     5     6      956           1000        -4     1358           1500
##  9  2013     9    28      955           1000        -5     1412           1445
## 10  2013     7     3      957           1000        -3     1410           1430
## # i 336,766 more rows
## # i 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>
```

Question 3: Which flights traveled the longest distance? Which traveled the shortest?

Answer(Longest distance):

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

```
## # 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
## # i 336,766 more rows
## # i 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>
```

Answer(Shortest distance):

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

```
## # 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
## # i 336,766 more rows
## # i 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>
```

Select Columnss with the select()

Question 1: Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time and arr_delay from the flights

```
select(flights, starts_with('dep') & ends_with('time'))
```

using the `starts_with` and `ends_with` with the `&` operator

```
## # A tibble: 336,776 x 1
##    dep_time
##       <int>
##  1      517
##  2      533
##  3      542
##  4      544
##  5      554
##  6      554
##  7      555
##  8      557
##  9      557
## 10      558
## # i 336,766 more rows
```

```
select(flights, starts_with('dep') & ends_with('delay'))
```

```
## # A tibble: 336,776 x 1
##    dep_delay
##        <dbl>
##  1         2
##  2         4
##  3         2
##  4        -1
##  5        -6
##  6        -4
```

```
##  7        -5
##  8        -3
##  9        -3
## 10        -2
## # i 336,766 more rows
```

```
select(flights, starts_with('arr') & ends_with('time'))
```

```
## # A tibble: 336,776 x 1
##    arr_time
##       <int>
##  1      830
##  2      850
##  3      923
##  4     1004
##  5      812
##  6      740
##  7      913
##  8      709
##  9      838
## 10      753
## # i 336,766 more rows
```

```
select(flights, starts_with('arr') & ends_with('delay'))
```

```
## # A tibble: 336,776 x 1
##    arr_delay
##        <dbl>
##  1        11
##  2        20
##  3        33
##  4       -18
##  5       -25
##  6        12
##  7        19
##  8       -14
##  9        -8
## 10         8
## # i 336,766 more rows
```

```
select(flights, contains('dep_time'))
```

Using the contains function

```
## # A tibble: 336,776 x 2
##    dep_time sched_dep_time
##       <int>          <int>
##  1      517            515
##  2      533            529
##  3      542            540
##  4      544            545
##  5      554            600
##  6      554            558
##  7      555            600
##  8      557            600
```

```
##  9      557            600
## 10      558            600
## # i 336,766 more rows
```

```
select(flights, contains('dep_delay'))
```

Using the contains function

```
## # A tibble: 336,776 x 1
##    dep_delay
##        <dbl>
##  1         2
##  2         4
##  3         2
##  4        -1
##  5        -6
##  6        -4
##  7        -5
##  8        -3
##  9        -3
## 10        -2
## # i 336,766 more rows
```

```
select(flights, contains('arr_time'))
```

Using the contains function

```
## # A tibble: 336,776 x 2
##    arr_time sched_arr_time
##       <int>          <int>
##  1      830            819
##  2      850            830
##  3      923            850
##  4     1004           1022
##  5      812            837
##  6      740            728
##  7      913            854
##  8      709            723
##  9      838            846
## 10      753            745
## # i 336,766 more rows
```

```
select(flights, contains('arr_delay'))
```

Using the contains function

```
## # A tibble: 336,776 x 1
##    arr_delay
##        <dbl>
##  1        11
##  2        20
##  3        33
##  4       -18
```

```
##  5       -25
##  6        12
##  7        19
##  8       -14
##  9        -8
## 10         8
## # i 336,766 more rows
```

Alternatively, subsetting can also be applied

```
tibble(flights$dep_time)
```

```
## # A tibble: 336,776 x 1
##    `flights$dep_time`
##                 <int>
##  1                517
##  2                533
##  3                542
##  4                544
##  5                554
##  6                554
##  7                555
##  8                557
##  9                557
## 10                558
## # i 336,766 more rows
```

```
tibble(flights$dep_delay)
```

```
## # A tibble: 336,776 x 1
##    `flights$dep_delay`
##                  <dbl>
##  1                   2
##  2                   4
##  3                   2
##  4                  -1
##  5                  -6
##  6                  -4
##  7                  -5
##  8                  -3
##  9                  -3
## 10                  -2
## # i 336,766 more rows
```

```
tibble(flights$arr_time)
```

```
## # A tibble: 336,776 x 1
##    `flights$arr_time`
##                 <int>
##  1                830
##  2                850
##  3                923
##  4               1004
##  5                812
##  6                740
##  7                913
##  8                709
```

```
##  9                838
## 10                753
## # i 336,766 more rows
```

```
tibble(flights$arr_delay)
```

```
## # A tibble: 336,776 x 1
##    `flights$arr_delay`
##                  <dbl>
##  1                  11
##  2                  20
##  3                  33
##  4                 -18
##  5                 -25
##  6                  12
##  7                  19
##  8                 -14
##  9                  -8
## 10                   8
## # i 336,766 more rows
```

Question 2:What happens if you include the name of a variable multiple times in a select() call?

Answer: The `select()` function silently ignores variable repetitions as shown below

```
select(flights, arr_time, arr_time, arr_time, arr_time)
```

```
## # A tibble: 336,776 x 1
##    arr_time
##       <int>
##  1      830
##  2      850
##  3      923
##  4     1004
##  5      812
##  6      740
##  7      913
##  8      709
##  9      838
## 10      753
## # i 336,766 more rows
```

Question 3: What does the one_of() function do? Why might it helpful in conjunction with this vector

Answer: The `one_of()` `superseded` method when used in the `select()` method selects only the variables present in the data frame while ignoring the missing variables. If non of the arguments to the `one_of()` is present in the data frame, an error message such as `unknown columns:` would be thrown.

Question 4: Does the result of running the following code surprise you? HOW do the select helpers deal with case by default? How can you changed that default?

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

Answer: All the helper methods of the `select()`,including the `contains()` method by default has the `ignore.case` flag set to `TRUE`. To alter this default setting we set `ignore.case = FALSE`

Useful Creation Functions ***Question 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 representations of numbers of minutes since midnight***

Answer: First let's get a peek at some `sched_dep_time` and `dep_time` as contained in the data set

```
df <- print(data.frame(dep_time = flights$dep_time[1:43],
sched_dep_time = flights$sched_dep_time[1:43]))
```

```
##    dep_time sched_dep_time
## 1       517            515
## 2       533            529
## 3       542            540
## 4       544            545
## 5       554            600
## 6       554            558
## 7       555            600
## 8       557            600
## 9       557            600
## 10      558            600
## 11      558            600
## 12      558            600
## 13      558            600
## 14      558            600
## 15      559            600
## 16      559            559
## 17      559            600
## 18      600            600
## 19      600            600
## 20      601            600
## 21      602            610
## 22      602            605
## 23      606            610
## 24      606            610
## 25      607            607
## 26      608            600
## 27      611            600
## 28      613            610
## 29      615            615
## 30      615            615
## 31      622            630
## 32      623            610
## 33      623            627
## 34      624            630
## 35      624            630
## 36      627            630
## 37      628            630
## 38      628            630
## 39      629            630
## 40      629            630
## 41      629            630
## 42      632            608
## 43      635            635
```

From the data frame returned above, the formats for the times as same `-HMM` or `HHMM`. So we can use the `%/%`

integer division and the %% modulo division to extract the hours and minutes aspects of the time

```
raw_dep_time <- flights$dep_time # For convenience sake
raw_sched_dep_time <- flights$sched_dep_time # For convenience sake

 transmute(flights,
           dep_time_min =
           (raw_dep_time %/% 100)*60+raw_dep_time %% 100,
           sche_dep_time_min =
           (raw_sched_dep_time %/% 100)*60 + raw_sched_dep_time %% 100)
```

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

Question 3: Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three number to relate?

Answer: The `dep_time = sched_dep_time+dep_delay` for flights with delayed times and `dep_time = sched_dep_time - dep_delay` for flights that occured before their actual scheduled time. For instance, taking a peek on the first observation in that data set to check `dep_time, sched_dep_time` and `dep_delay` we have:

```
v <- data.frame(x = flights$dep_time, y = flights$sched_dep_time, z = flights$dep_delay)
```

Now using logical,we can verify that `dep_time = sched_dep_time+dep_delay` for flights with delayed departure times

```
#filter all flights with delayed departure
delayed_departure <- filter(flights, dep_delay > 0)

#drop all NA
delayed_departure <- drop_na(delayed_departure)
#check if dep_time == sched_dep_time+dep_delay
as_tibble(delayed_departure$dep_time == delayed_departure$sched_dep_time + delayed_departure$dep_delay)
```

```
## # A tibble: 127,745 x 1
##    value
##    <lgl>
##  1 TRUE
##  2 TRUE
##  3 TRUE
##  4 TRUE
##  5 TRUE
##  6 TRUE
##  7 TRUE
```

```
##  8 TRUE
##  9 TRUE
## 10 TRUE
## # i 127,735 more rows
```

Now for flights that occured before their actual scheduled time

```
#  filter flights with pre scheduled departure time
pre_sched <- filter(flights, flights$dep_delay < 0)

# drop flights with missing records
pre_sched <- drop_na(pre_sched)

# convert the schedule departure time to minutes
sched_dep_min <- (pre_sched$sched_dep_time %/% 100)*60 + pre_sched$sched_dep_time %% 100

# convert departure time to minutes
dep_time_min <- (pre_sched$dep_time %/% 100)*60 + pre_sched$dep_time %% 100

# check to confirm the return values confirmed the assertion
as_tibble(dep_time_min == sched_dep_min+pre_sched$dep_delay)
```

```
## # A tibble: 183,135 x 1
##    value
##    <lgl>
##  1 TRUE
##  2 TRUE
##  3 TRUE
##  4 TRUE
##  5 TRUE
##  6 TRUE
##  7 TRUE
##  8 TRUE
##  9 TRUE
## 10 TRUE
## # i 183,125 more rows
```

Note: A common logic could have been written to for the two scenarios,but for simplicity, they were split apart

Question 4: Find the 10 most delayed flights using a ranking function. How would you want to handle this? Carefully read the documentation for min_rank()

Answer: The steps to achieve this can be broken into the following:

step 1: Make data frame from `flights` with additional variable `delayed_rank` which is the rank per flights based on its `dep_delay`

step 2: Arrange the data frame in a descending order of `delayed_rank`

step 3: Use the `slice_head()` method to retrieve flight from first row upto the tenth row inclusive

```
# step 1: asings maximum ranks to most delayed flights
df <- mutate(flights, delayed_rank = min_rank(dep_delay), .after = dep_delay)

# step 2:
df <- arrange(df, desc(delayed_rank))
```

```
# step 3:
slice_head(df, n = 10) #or slice(df, 10)
```

```
## # A tibble: 10 x 20
##     year month   day dep_time sched_dep_time dep_delay delayed_rank arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>        <int>    <int>
##  1  2013     1     9      641            900      1301       328521     1242
##  2  2013     6    15     1432           1935      1137       328520     1607
##  3  2013     1    10     1121           1635      1126       328519     1239
##  4  2013     9    20     1139           1845      1014       328518     1457
##  5  2013     7    22      845           1600      1005       328517     1044
##  6  2013     4    10     1100           1900       960       328516     1342
##  7  2013     3    17     2321            810       911       328515      135
##  8  2013     6    27      959           1900       899       328514     1236
##  9  2013     7    22     2257            759       898       328513      121
## 10  2013    12     5      756           1700       896       328512     1058
## # i 12 more variables: sched_arr_time <int>, 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>
```

Alternatively

```
df <- drop_na(flights) # drop flights with missing records so as not to affect our ranking

# As in step 1 above, but assigns minimum ranks to most delayed flights
df <- mutate(df, delayed_rank = min_rank(desc(dep_delay)), .after = dep_delay)

# arrange the data frame in descending order of delayed_rank
df <- arrange(df, desc(delayed_rank))



# use the slice method to retrieve the 10 most delayed flights
slice(df, n():10) # from the last record up to the tenth record up
```

```
## # A tibble: 327,337 x 20
##     year month   day dep_time sched_dep_time dep_delay delayed_rank arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>        <int>    <int>
##  1  2013     1     9      641            900      1301            1     1242
##  2  2013     6    15     1432           1935      1137            2     1607
##  3  2013     1    10     1121           1635      1126            3     1239
##  4  2013     9    20     1139           1845      1014            4     1457
##  5  2013     7    22      845           1600      1005            5     1044
##  6  2013     4    10     1100           1900       960            6     1342
##  7  2013     3    17     2321            810       911            7      135
##  8  2013     6    27      959           1900       899            8     1236
##  9  2013     7    22     2257            759       898            9      121
## 10  2013    12     5      756           1700       896           10     1058
## # i 327,337 more rows
## # i 12 more variables: sched_arr_time <int>, 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>
```

Question 5: What does `1:3+1:10` return? Why?

Answer: The operation basically makes a column-wise addition

```
# 1:3 returns the vector 1, 2, 3
# 1:10 returns the vector 1,2,3,...10
# 1:3+1:10 performs the operation 1(from 1:3)+1(from 1:10), 2+2 etc.
# Recursive additions are carried out until values in 1:10 are exhausted.
# The operation gives warning if the number of records in
# one vector is not an integer factor of the other


1:3+1:10
```

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

Question 6: What trigonometric functions does R provide?

Anwer: R provide trigonoetric function like `sin(x)`, `cos(x)`, `tan(x)`, `sinh(x)`, `cosh(x)`, `tanh(x)`, etc

Useful Summary Functions

Question 1: Brainstorm at least five different ways to assess the typical characteristics of a group of flights. Consider the following scenarios:

- A flight is 15 minutes early 50% of the the time, and 15 minutes late 50% of the time.
- A flight is always 10 minutes late.
- A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time
- 99% of the time a flight is on time. 1% of the time it's 2 hours late.

Answer:

```
# Scenario: A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.

#Step 1: filter out cancelled flights
#step 2: group the data set by their tailnum variable
#step 3: count the grouped data set, get 50% of the counts
#step 4: filter records whose flights are minutes and and 15 minutes late.
not_cancelled <- flights %>% filter(!is.na(arr_delay), !is.na(dep_delay))
#Brainstorm early departure
not_cancelled <- not_cancelled %>% group_by(tailnum) %>% summarise(count=n(), fifty_percent_of_time = co
not_cancelled
```

```
## # A tibble: 2 x 4
##   tailnum count fifty_percent_of_time sum_of_delays
##   <chr>   <int>                 <dbl>         <int>
## 1 N17627      2                     1             1
## 2 N583AS      2                     1             1
```

```
# Scenario: a flight is always 10 minutes late
# step 1: group the data set by tailnum
# filter the grouped data set to only display whose arrival delay is 10 mins

flights %>% filter(!is.na(arr_delay), !is.na(dep_delay)) %>%
  group_by(tailnum) %>% filter(all(arr_delay == 10)) %>% summarise(total = n())
```

```
## # A tibble: 1 x 2
##   tailnum total
##   <chr>   <int>
## 1 N801AW      1
```

```
# Scenario: a flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.
# step 1: group flights by their tail number
# step 2: filter flights with departure delay of 30 minutes 50% of time & early departure of 30 minutes

flights %>% filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  group_by(tailnum) %>% summarize(fifty_percent = 0.5*n(), early_arr = sum(arr_delay == -30), late_arr =
  filter(early_arr == fifty_percent, late_arr == fifty_percent)
```

```
## # A tibble: 0 x 4
## # i 4 variables: tailnum <chr>, fifty_percent <dbl>, early_arr <int>,
## #   late_arr <int>
```

```
# Scenario: 99% of the time, a flight is on time.
# step 1: group flights by tailnum variable
# step 2: get 99% of the count of each group
# step 3: get the count of all the time a each group of flight is on time
# step 4: filter out counts that are zero
flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  group_by(tailnum) %>%
  summarise(percent = round(0.99*n(), digits = 0), ontime = sum(arr_delay == 0)) %>%
  filter(ontime != 0) %>%
  filter(ontime == percent)
```

```
## # A tibble: 2 x 3
##   tailnum percent ontime
##   <chr>     <dbl>  <int>
## 1 N7BVAA        1      1
## 2 N956DN        1      1
```

```
# Scenario: 1% of the time, a flight is 2hrs late.
# step 1: group flights by tailnum variable
# step 2: get 99% of the count of each group
# step 3: get the count of all the time each group of flight is 2 hrs late
# step 4: filter out counts that are zero
flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  group_by(tailnum) %>%
  summarise(percent = round(0.01*n(), digits = 0), late = sum(arr_delay == 120)) %>%
  filter(late != 0) %>%
  filter(late == percent)
```

```
## # A tibble: 44 x 3
##    tailnum percent  late
##    <chr>     <dbl> <int>
##  1 N11164        1     1
##  2 N11199        1     1
##  3 N11206        1     1
##  4 N14125        1     1
##  5 N14148        1     1
##  6 N14168        1     1
##  7 N14171        1     1
##  8 N14179        1     1
##  9 N14974        2     2
## 10 N17146        1     1
```

```
## # i 34 more rows
```

Which is more important: arrival delay or departure delay? Answer: Arrival delay is important than departure delay for the followig reasons:

- There is fresh charge incurred in accommodation arrangement.
- There is the risk of missing subsequent flights already booked

Question 2: Come up with another approach that will give you the same result as as `not_cancelled %>% count(dest)` and `not_cancelled %>% count(tailnum, wt = distance)` (without using count())

```
# alternative approach to writing not_cancelled %>% count(dest)

not_cancel <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))


not_cancel %>% count(dest)
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # i 94 more rows
```

```
# alternatively;
not_cancel %>%
  group_by(dest) %>% summarise(n = n())
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # i 94 more rows
```

```
# alternative approach to writing not_cancelled %>% count(tail_num, wt = distance)
```

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

not_cancel %>% count(tailnum, wt=distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  239143
##  3 N10156  109664
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   24616
##  7 N10575  139903
##  8 N105UW   23618
##  9 N107US   21677
## 10 N108UW   32070
## # i 4,027 more rows
```

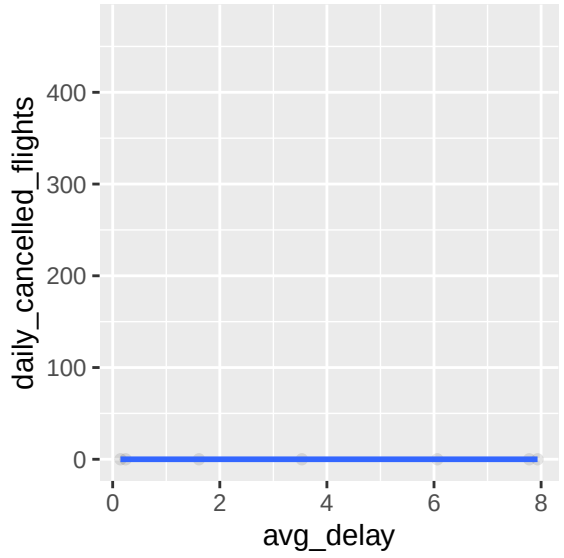
```
# alternatively;
not_cancel %>% group_by(tailnum) %>%
  summarise(n = sum(distance))
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  239143
##  3 N10156  109664
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   24616
##  7 N10575  139903
##  8 N105UW   23618
##  9 N107US   21677
## 10 N108UW   32070
## # i 4,027 more rows
```

Question 4: Look at the cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

Answer:

```
flights %>% group_by(year, month, day) %>%
  summarise(avg_delay = mean(dep_delay), daily_cancelled_flights = sum(is.na(dep_delay)
                                & is.na(arr_delay))) %>%
  ggplot(mapping = aes(x = avg_delay, y = daily_cancelled_flights))+
  geom_point(alpha = 1/10)+
  geom_smooth(se = FALSE)
```



Observation: it can be seen from the steady graph pattern that all delays occured at no flight cancellation.

Question 5: Which carrier has the worst delays? Challenge: can you disentangle the effect of bad airport versus bad carrier? (Hint: think about flights %>% grouped_by(carrier, dest)) %>% summarize(n())

Answer:

```
#step 1: filter out cancelled flights
#step 2: group the data set by their carrier & filter only delayed times (negative delays)
#step 3: get a count of the delays by their group
#step 4: add rank to the grouped data set to by their delay in descending order
#step 5: get the minimum ranked rows

flights %>% filter(!is.na(arr_delay), !is.na(dep_delay)) %>%
  group_by(carrier) %>%
  filter(dep_delay < 0) %>%
  summarise(delays = sum(dep_delay)) %>%
  mutate(rank = min_rank(delays)) %>%
  filter(rank %in% min(rank))
```

```
## # A tibble: 1 x 3
##   carrier  delays  rank
##   <chr>     <dbl> <int>
## 1 B6      -147794     1
```

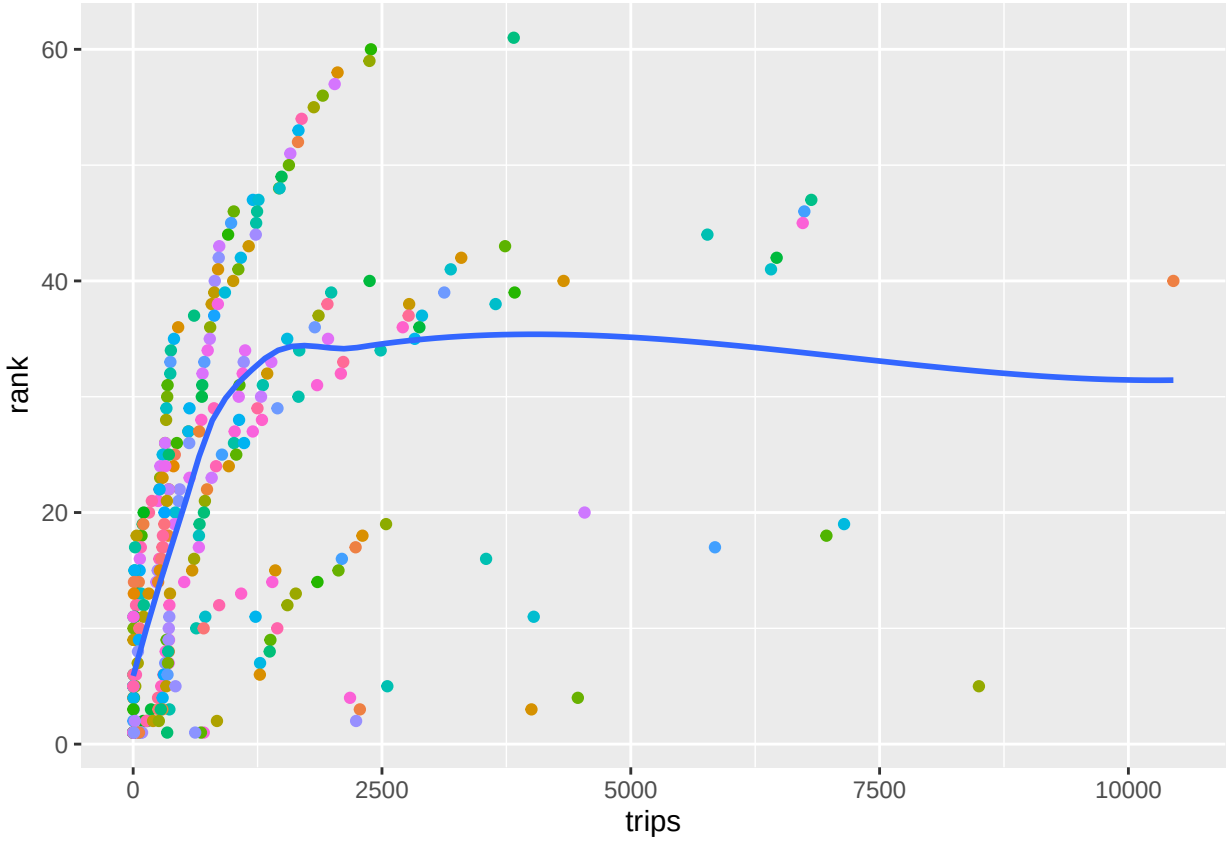
```
# scenario: effect of bad airports vs bad carriers
# Task: Get the number of trips per carrier per destination
# rank the group of carrier, destination by their number of trips
# the lower the ranking the worse is the destination and carrier pair
df <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  group_by(carrier, dest) %>%
  summarise(trips = n()) %>%
```

```
  mutate(rank = min_rank(trips)) %>%
  arrange(desc(rank), .by_group = TRUE)

df
```

```
## # A tibble: 312 x 4
## # Groups:   carrier [16]
##    carrier dest  trips  rank
##    <chr>   <chr> <int> <int>
##  1 9E      CVG    1466    48
##  2 9E      MSP    1203    47
##  3 9E      DCA    1011    46
##  4 9E      ORD     984    45
##  5 9E      DTW     954    44
##  6 9E      RDU     865    43
##  7 9E      PHL     860    42
##  8 9E      BOS     853    41
##  9 9E      PIT     820    40
## 10 9E      BWI     815    39
## # i 302 more rows
```

```
df %>% ggplot(mapping = aes(x = trips, y = rank))+
  geom_point(aes(color = dest, position = "jitter"), show.legend = FALSE)+
  geom_smooth(se = FALSE)
```



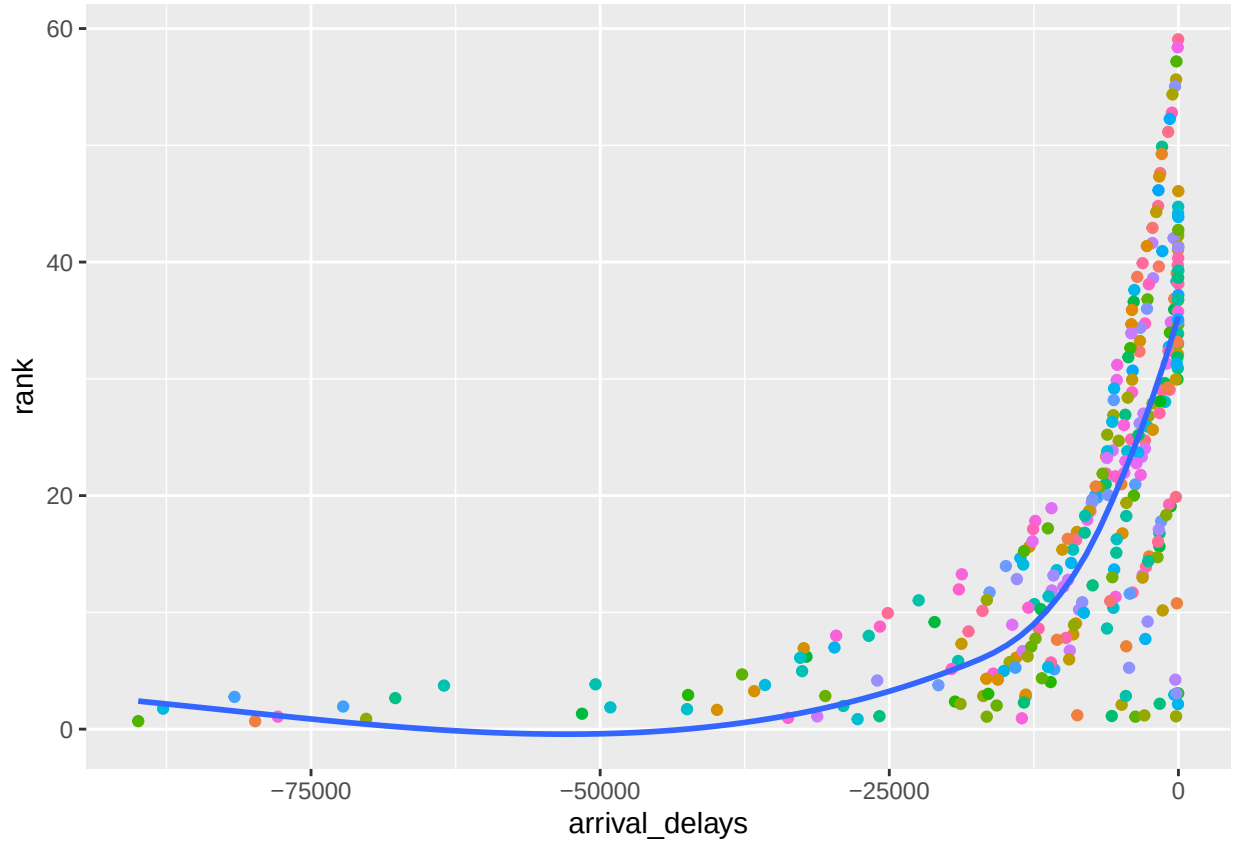

```
# scenario: effect of bad airports vs bad carriers
# Task: get the worst destinations with their corresponding carriers
```

```r
# step 1: filter flights that were not cancelled
# step 2: filter arrival that were delayed (negative delays)
# step 3: group the flights carrier and destinations
# step 4: aggregate the delays
# step 5: rank the flight by their aggregated delays
# step 6: arrange the flight by their ranking to show worst destinations and their corresponding carrie

df <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay)) %>%
  filter(arr_delay < 0) %>%
  group_by(carrier, dest) %>%
  summarise(arrival_delays = sum(arr_delay)) %>%
  mutate(rank = min_rank(arrival_delays)) %>%
  arrange(desc(rank), .by_group = TRUE)
df
```

```
## # A tibble: 301 x 4
## # Groups:   carrier [16]
##    carrier dest  arrival_delays  rank
##    <chr>   <chr>          <dbl> <int>
##  1 9E      BTV               -5    46
##  2 9E      LEX              -22    45
##  3 9E      MEM              -31    44
##  4 9E      AUS              -32    42
##  5 9E      CMH              -32    42
##  6 9E      CAE              -37    41
##  7 9E      TPA              -55    40
##  8 9E      AVL             -140    39
##  9 9E      MHT             -165    38
## 10 9E      ATL             -358    37
## # i 291 more rows
```

```r
df %>% ggplot(mapping = aes(x = arrival_delays, y = rank))+
  geom_point( aes(color = dest), position = "jitter", show.legend = FALSE)+
  geom_smooth(se = FALSE)
```

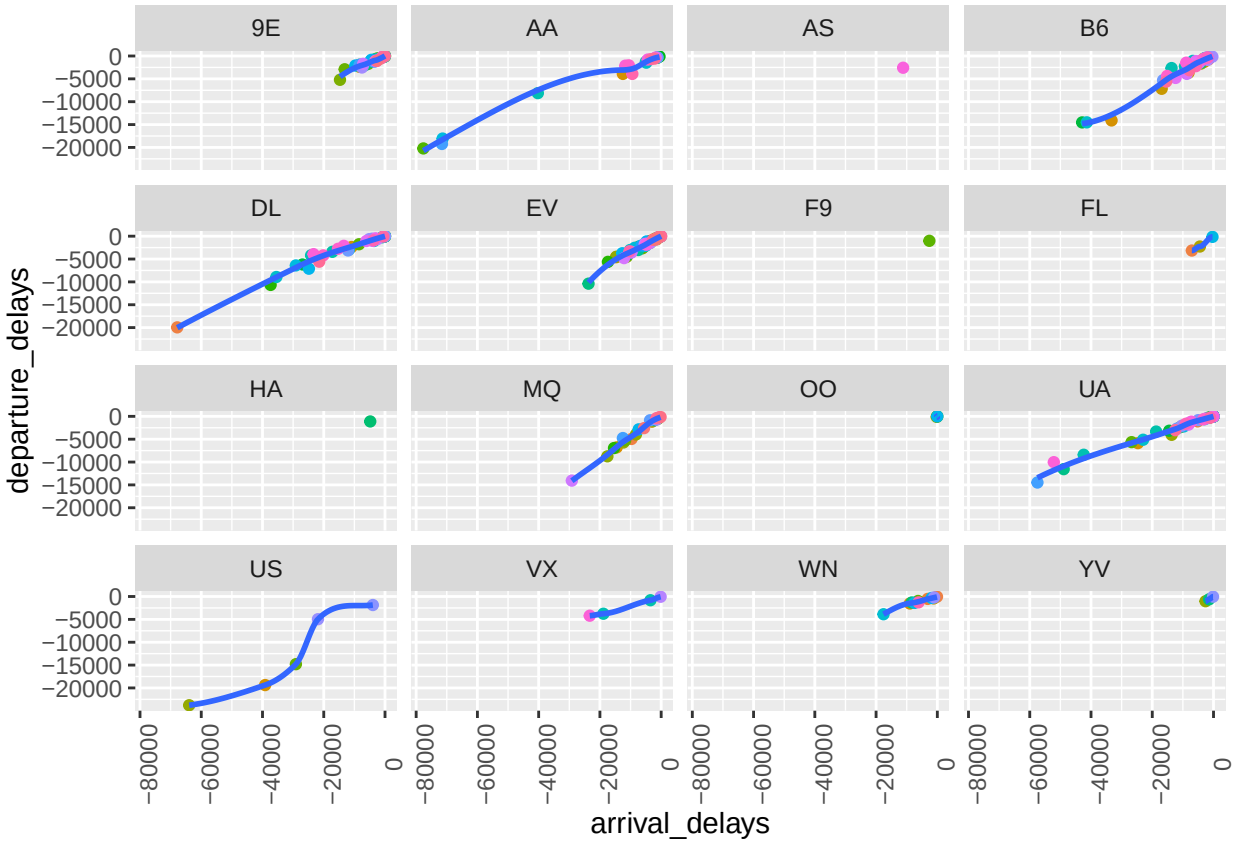


Observation: From the first the following obserations could be made:

- The more the number of trips to a particular destination (a factor of the carrier due to less departure delays), the higher the rank
- The less the number of arrival delays(a factor of the destination), the higher the rank

To further understand the relationship between carrier and destination, we observe departure delays of carriers and the respective arrival delays of their destinations

```
flights %>%
  filter(!is.na(arr_delay), !is.na(dep_delay), arr_delay < 0, dep_delay < 0) %>%
  group_by(carrier, dest) %>%
  summarise(departure_delays = sum(dep_delay), arrival_delays = sum(arr_delay)) %>%
  ggplot(mapping = aes(x = arrival_delays, y = departure_delays))+
  geom_point(aes(color = dest), show.legend = FALSE)+
  theme(axis.text.x = element_text(angle = 90))+
  geom_smooth(se = FALSE)+
  facet_wrap(~carrier)
```


9E
AA
AS
B6
DL
EV
F9
FL
HA
MQ
OO
UA
US
VX
WN
YV
departure_delays
arrival_delays
0
−5000
−10000
−15000
−20000
−80000
−60000
−40000
−20000
0