Data Transformation

Gabriel Montes

2025-07-17

Exercise 3.2.5

1. In a single pipeline for each condition, find all flights that meet the condition:

```
library(nycflights13)
library(dplyr)
```

Had an arrival delay of two or more hours

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
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
## # i 10,190 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>
```

```
flights |>
  filter(dest == "IAH" | dest == "HOU")
```

Flew to Houston (IAH or 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>
```

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

Were operated by United, American, or Delta

```
## # 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>
```

```r
flights |>
  filter(month %in% c(7, 8, 9))
```

Departed in summer (July, August, and September)

```
## # 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>
```

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

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

```
## # 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
## # i 19 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>
```

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

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

```
## # A tibble: 1,844 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     2205           1720       285       46           2040
##  2  2013     1     1     2326           2130       116      131             18
##  3  2013     1     3     1503           1221       162     1803           1555
##  4  2013     1     3     1839           1700        99     2056           1950
##  5  2013     1     3     1850           1745        65     2148           2120
##  6  2013     1     3     1941           1759       102     2246           2139
##  7  2013     1     3     1950           1845        65     2228           2227
##  8  2013     1     3     2015           1915        60     2135           2111
##  9  2013     1     3     2257           2000       177       45           2224
## 10  2013     1     4     1917           1700       137     2135           1950
## # i 1,834 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>
```

2. Sort flights to find the flights with the longest departure delays. Find the flights that left earliest in the morning.

```
flights |>
  arrange(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>
```

3. Sort flights to find the fastest flights. (Hint: Try including a math calculation inside of your function.)

```
flights |>
  mutate(speed = distance / (air_time / 60)) |>
  arrange(desc(speed))
```

```
## # A tibble: 336,776 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     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
## # i 336,766 more rows
## # i 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>, speed <dbl>
```

4. Was there a flight on every day of 2013?

```r
flights |>
  count(year, month, day) |>
  nrow() == 365
```

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

5. Which flights traveled the farthest distance? Which traveled the least distance?

```r
flights |>
  arrange(desc(distance))
```

Farthest flight

```
## # 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>
```

```r
flights |>
  arrange(distance)
```

Shortest flight

```
## # 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>
```

6. Does it matter what order you used filter() and arrange() if you're using both? Why/why not? Think about the results and how much work the functions would have to do.

so if you use filter() first it filters the dataset thus making it smaller, if you use arrange() first, you are sorting the whole dataset, so order matters

Exercise 3.3.5

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

```
flights |>
  select(dep_time, sched_dep_time, dep_delay) |>
  mutate(
    dep_time_mins = (dep_time %/% 100) * 60 + (dep_time %% 100),
    sched_dep_time_mins = (sched_dep_time %/% 100) * 60 + (sched_dep_time %% 100),
    calculated_delay = dep_time_mins - sched_dep_time_mins
  )
```

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

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

```
flights |> select(dep_time, dep_delay, arr_time, arr_delay)
```

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

```
flights |> select(ends_with("time"), ends_with("delay"))
```

```
## # A tibble: 336,776 x 7
##    dep_time sched_dep_time arr_time sched_arr_time air_time dep_delay arr_delay
##       <int>          <int>    <int>          <int>    <dbl>     <dbl>     <dbl>
##  1      517            515      830            819      227         2        11
##  2      533            529      850            830      227         4        20
##  3      542            540      923            850      160         2        33
##  4      544            545     1004           1022      183        -1       -18
##  5      554            600      812            837      116        -6       -25
##  6      554            558      740            728      150        -4        12
##  7      555            600      913            854      158        -5        19
##  8      557            600      709            723       53        -3       -14
##  9      557            600      838            846      140        -3        -8
## 10      558            600      753            745      138        -2         8
## # i 336,766 more rows
```

3. What happens if you specify the name of the same variable multiple times in a select() call?

```
flights |> select(dep_time, dep_time, dep_time)
```

```
## # 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
```

they are ignored

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

```
vars <- c("year", "month", "day", "not_a_column")
flights |> select(any_of(vars))
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

it is safer to use than all_of

5. Does the result of running the following code surprise you? How do the select helpers deal with upper and lower case by default? How can you change that default?

```
flights |> select(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
## # i 336,766 more rows
```

it is case sensitive

```
flights |> select(contains("time", ignore.case = TRUE))
```

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

to make it non sensitive

6. Rename air_time to air_time_min to indicate units of measurement and move it to the beginning of the data frame.

```
flights |>
  rename(air_time_min = air_time) |>
  relocate(air_time_min)
```

```
## # A tibble: 336,776 x 19
##    air_time_min  year month   day dep_time sched_dep_time dep_delay arr_time
##           <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1          227  2013     1     1      517            515         2      830
##  2          227  2013     1     1      533            529         4      850
##  3          160  2013     1     1      542            540         2      923
##  4          183  2013     1     1      544            545        -1     1004
##  5          116  2013     1     1      554            600        -6      812
##  6          150  2013     1     1      554            558        -4      740
##  7          158  2013     1     1      555            600        -5      913
##  8           53  2013     1     1      557            600        -3      709
##  9          140  2013     1     1      557            600        -3      838
## 10          138  2013     1     1      558            600        -2      753
## # i 336,766 more rows
## # i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

7. Why doesn't the following work, and what does the error mean?

flights |> select(tailnum) |> arrange(arr_delay)

to fix it, because the arr_delay is removed, we need to include it in select

```r
flights |>
  select(tailnum, arr_delay) |>
  arrange(arr_delay)
```

```
## # A tibble: 336,776 x 2
##    tailnum arr_delay
##    <chr>       <dbl>
##  1 N843VA        -86
##  2 N840VA        -79
##  3 N851UA        -75
##  4 N3KCAA        -75
##  5 N551AS        -74
##  6 N24212        -73
##  7 N3760C        -71
##  8 N806UA        -71
##  9 N805JB        -71
## 10 N855VA        -70
## # i 336,766 more rows
```

Exercise 3.5.7

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

```r
library(nycflights13)
library(dplyr)

flights |>
  group_by(carrier) |>
  summarize(avg_delay = mean(dep_delay, na.rm = TRUE)) |>
  arrange(desc(avg_delay)) |>
  head()
```

```
## # A tibble: 6 x 2
##   carrier avg_delay
##   <chr>       <dbl>
## 1 F9           20.2
## 2 EV           20.0
## 3 YV           19.0
## 4 FL           18.7
## 5 WN           17.7
## 6 9E           16.7
```

to untangle

```
flights |>
  group_by(carrier, dest) |>
  summarize(avg_delay = mean(dep_delay, na.rm = TRUE), n = n()) |>
  filter(n > 50) |>
  arrange(desc(avg_delay))
```

```
## `summarise()` has grouped output by 'carrier'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 254 x 4
## # Groups:   carrier [15]
##    carrier dest  avg_delay     n
##    <chr>   <chr>     <dbl> <int>
##  1 EV      TYS        41.8   323
##  2 EV      CAE        36.7   113
##  3 EV      TUL        34.9   315
##  4 WN      MSY        33.4   298
##  5 EV      OKC        30.6   346
##  6 EV      BHM        29.7   297
##  7 EV      DSM        28.8   478
##  8 EV      TVC        27.5    68
##  9 9E      CLE        25.6   349
## 10 EV      PWM        25.3   813
## # i 244 more rows
```

2. Find the flights that are most delayed upon departure from each destination.

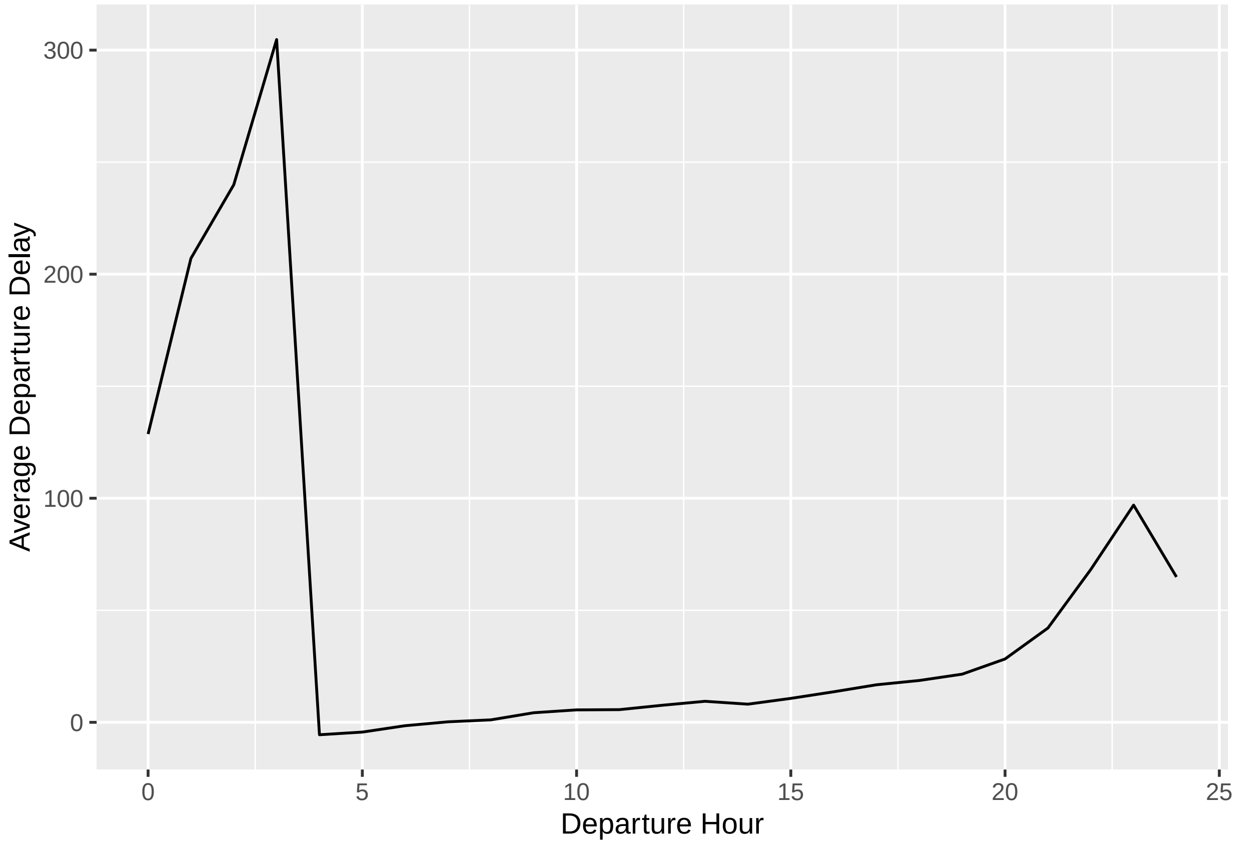
```
flights |>
  group_by(dest) |>
  slice_max(dep_delay, n = 1, with_ties = FALSE) |>
  select(dest, carrier, flight, dep_delay)
```

```
## # A tibble: 105 x 4
## # Groups:   dest [105]
##    dest  carrier flight dep_delay
##    <chr> <chr>    <int>     <dbl>
##  1 ABQ   B6          65       142
##  2 ACK   B6        1491       219
##  3 ALB   EV        4309       323
##  4 ANC   UA         887        75
##  5 ATL   DL        2047       898
##  6 AUS   UA         503       351
##  7 AVL   EV        4519       222
##  8 BDL   EV        4103       252
##  9 BGR   EV        5309       248
## 10 BHM   EV        5038       325
## # i 95 more rows
```

3. How do delays vary over the course of the day? Illustrate your answer with a plot.

```
library(ggplot2)
flights |>
  mutate(hour = dep_time %/% 100) |>
  group_by(hour) |>
  summarize(avg_delay = mean(dep_delay, na.rm = TRUE),
            count = n()) |>
  ggplot(aes(x = hour, y = avg_delay)) +
    geom_line() +
    labs(x = "Departure Hour", y = "Average Departure Delay")
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```



4. What happens if you supply a negative n to slice_min() and friends?

they will be ignored as they use absolute values

5. Explain what count() does in terms of the dplyr verbs you just learned. What does the sort argument to count() do?

count() is like grouping and then sorting is sorting, so you get the most common characteristics

6. Suppose we have the following tiny data frame:

```
df <- tibble(
  x = 1:5,
  y = c("a", "b", "a", "a", "b"),
  z = c("K", "K", "L", "L", "K")
)
```

a. Write down what you think the output will look like, then check if you were correct, and describe what group_by() does.

```
df |>
  group_by(y)
```

```
## # A tibble: 5 x 3
## # Groups:   y [2]
##       x y     z
##   <int> <chr> <chr>
## 1     1 a     K
## 2     2 b     K
## 3     3 a     L
## 4     4 a     L
## 5     5 b     K
```

groups it by y

b. Write down what you think the output will look like, then check if you were correct, and describe what arrange() does. Also, comment on how it's different from the group_by() in part (a).

```
df |>
  arrange(y)
```

```
## # A tibble: 5 x 3
##       x y     z
##   <int> <chr> <chr>
## 1     1 a     K
## 2     3 a     L
## 3     4 a     L
## 4     2 b     K
## 5     5 b     K
```

this will sort it alphabetically based on y

c. Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does.

```
df |>
  group_by(y) |>
  summarize(mean_x = mean(x))
```

```
## # A tibble: 2 x 2
##   y     mean_x
##   <chr>  <dbl>
## 1 a       2.67
## 2 b        3.5
```

will summarize one row per group

d. Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does. Then, comment on what the message says.

```r
df |> group_by(y, z) |> summarize(mean_x = mean(x))
```

```
## `summarise()` has grouped output by 'y'. You can override using the `.groups`
## argument.
```

```
## # A tibble: 3 x 3
## # Groups:   y [2]
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

one row for each combination of y and z

e. Write down what you think the output will look like, then check if you were correct, and describe what the pipeline does. How is the output different from the one in part (d)?

```r
df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x), .groups = "drop")
```

```
## # A tibble: 3 x 3
##   y     z     mean_x
##   <chr> <chr>  <dbl>
## 1 a     K        1
## 2 a     L        3.5
## 3 b     K        3.5
```

dropping groups prevents retention of grouping

f. Write down what you think the outputs will look like, then check if you were correct, and describe what each pipeline does. How are the outputs of the two pipelines different?

```r
df |>
  group_by(y, z) |>
  mutate(mean_x = mean(x))
```

```
## # A tibble: 5 x 4
## # Groups:   y, z [3]
##       x y     z     mean_x
##   <int> <chr> <chr>  <dbl>
## 1     1 a     K        1
## 2     2 b     K        3.5
## 3     3 a     L        3.5
## 4     4 a     L        3.5
## 5     5 b     K        3.5
```

mutate() adds a column with repeated summary stats, while summarize() condenses groups to single rows