HW_Lab02

Ryan Waterman

2025-01-16

Lab 2

Link for textbook

https://r4ds.hadley.nz/ (https://r4ds.hadley.nz/)

Library Imports

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

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
library(ggplot2)
```

(1) Section 3.2.5 Questions

Getting an initial look at the data:

```
head(flights, 25)
```

```
## # A tibble: 25 × 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 15 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>
```

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

- Had an arrival delay of two or more hours

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

```
## # A tibble: 10,200 × 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>
```

- Flew to Houston (IAH or HOU)

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

```
## # A tibble: 9,313 × 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>
```

- Were operated by United, American, or Delta

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

```
## # A tibble: 139,504 × 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>
```

- Departed in summer (July, August, and September)

```
flights|>
  filter(month == 7 | month == 8 | month == 9)
```

```
## # A tibble: 86,326 × 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>
```

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

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

```
## # A tibble: 29 × 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>
```

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

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

```
## # A tibble: 2,074 × 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
## # ℹ 2,064 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.

```
#Longest departure delays
longest_dep_delay_df <- flights |>
  arrange(desc(dep_delay))
head(longest_dep_delay_df, 10)
```

```
## # A tibble: 10 × 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 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>
```

```
#Earliest in the morning
earliest <- flights |>
  arrange(dep_time)
head(earliest, 10)
```

```
## # A tibble: 10 × 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    13        1           2249        72      108           2357
##  2  2013     1    31        1           2100       181      124           2225
##  3  2013    11    13        1           2359         2      442            440
##  4  2013    12    16        1           2359         2      447            437
##  5  2013    12    20        1           2359         2      430            440
##  6  2013    12    26        1           2359         2      437            440
##  7  2013    12    30        1           2359         2      441            437
##  8  2013     2    11        1           2100       181      111           2225
##  9  2013     2    24        1           2245        76      121           2354
## 10  2013     3     8        1           2355         6      431            440
## # 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.)

```
#Arrange the flights by the arrival time minus the departure time.
fastest <- flights |>
  arrange(arr_time-dep_time)
head(fastest, 10)
```

```
## # A tibble: 10 × 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    17     2400           2142       138       54           2259
##  2  2013    12     9     2400           2250        70       59           2356
##  3  2013     6    12     2338           2129       129       17           2235
##  4  2013    12    29     2332           2155        97       14           2300
##  5  2013    11     6     2335           2215        80       18           2317
##  6  2013     2    25     2347           2145       122       30           2239
##  7  2013     8    13     2351           2152       119       35           2258
##  8  2013    10    11     2342           2030       192       27           2205
##  9  2013     2    26     2356           2000       236       41           2104
## 10  2013     1    24     2342           2159       103       28           2300
## # 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>
```

- This seems like it could produce an incorrect result, unless R handles math functions on HHMM time formats implicitly. For example, an arrival time of 54 minus a departure time of 2400 would be -2346 if R just calls a subtraction on the integer, which may not be the fastest flight. The code block, below, verifies the output of the subtraction of dep_time from arr_time:

Source for understanding lists in R: https://stackoverflow.com/questions/2050790/how-to-correctly-use-lists (https://stackoverflow.com/questions/2050790/how-to-correctly-use-lists)

Source for appending to lists in R: https://stackoverflow.com/questions/26508519/how-to-add-elements-to-a-list-in-r-loop (https://stackoverflow.com/questions/26508519/how-to-add-elements-to-a-list-in-r-loop)

Newline print source: https://www.geeksforgeeks.org/r-program-to-print-a-new-line-in-string/ (https://www.geeksforgeeks.org/r-program-to-print-a-new-line-in-string/)

Source for defining functions: https://www.w3schools.com/r/r_functions.asp (https://www.w3schools.com/r/r_functions.asp)

String splitting resource: https://builtin.com/articles/strsplit (https://builtin.com/articles/strsplit)

String length source: https://stackoverflow.com/questions/11134812/how-to-find-the-length-of-a-string-in-r (https://stackoverflow.com/questions/11134812/how-to-find-the-length-of-a-string-in-r)

Converting departure time to string source: https://www.geeksforgeeks.org/convert-an-object-to-a-string-in-r-programming-tostring-function/ (https://www.geeksforgeeks.org/convert-an-object-to-a-string-in-r-programming-tostring-function/)

Source for adding list as column: https://www.geeksforgeeks.org/insert-list-as-dataframe-column-in-r/ (https://www.geeksforgeeks.org/insert-list-as-dataframe-column-in-r/)

Source for creating a date sequence: https://stackoverflow.com/questions/25677035/how-to-create-a-range-of-dates-in-r (https://stackoverflow.com/questions/25677035/how-to-create-a-range-of-dates-in-r)

```
# Note: I left the entire walk through of the process in this code block.
# This is in no way optimized, I just wanted to keep track of the process to come
# to my conclusion. Also, I am sure I made this far more complicated than I needed
# to, and I am willing to bet I misunderstood the question or there is a nice, clean
# function in the R standard library that does all of this for me. But, this was
# a fun exercise and I learned a lot doing it.

# Remove NA's and NULL's
non_na_flights <- flights |>
  filter(!is.na(dep_time) & !is.na(arr_time) & !is.null(dep_time) & !is.null(arr_time))

# Sort the departure time by descending
desc_dep_time <- non_na_flights |>
  arrange(desc(dep_time))

# Get the number of rows in desc_dep_time
#print(nrow(desc_dep_time))

# Loop through the data frame and perform the subtraction.
# Append the subtracted items to a new list
# Also, append the raw data to it's own list to verify the subtraction
flight_time <- list()
data <- list()
for (i in 1:nrow(desc_dep_time)) {
  flight_time[[i]] <- desc_dep_time$arr_time[i] - desc_dep_time$dep_time[i]
  data[[i]] <- c(desc_dep_time$arr_time[i],desc_dep_time$dep_time[i])
}

# The for loop, below, was used for debugging an error when correcting the flight
# time, below.
# for (i in which(is.na(flight_time))) {
#   print(paste(desc_dep_time$arr_time[i], desc_dep_time$dep_time[i],flight_time[[i]]))
# }

# Verify the output
print("Original data,  Calculated Flight Time")
```

```
## [1] "Original data,  Calculated Flight Time"
```

```
paste(data[c(1:100)], flight_time[c(1:100)])
```

```
##   [1] "c(327, 2400) -2073" "c(515, 2400) -1885" "c(427, 2400) -1973"
##   [4] "c(432, 2400) -1968" "c(59, 2400) -2341"  "c(432, 2400) -1968"
##   [7] "c(434, 2400) -1966" "c(302, 2400) -2098" "c(432, 2400) -1968"
##  [10] "c(443, 2400) -1957" "c(251, 2400) -2149" "c(324, 2400) -2076"
##  [13] "c(339, 2400) -2061" "c(339, 2400) -2061" "c(339, 2400) -2061"
##  [16] "c(347, 2400) -2053" "c(338, 2400) -2062" "c(339, 2400) -2061"
##  [19] "c(102, 2400) -2298" "c(107, 2400) -2293" "c(101, 2400) -2299"
##  [22] "c(225, 2400) -2175" "c(54, 2400) -2346"  "c(247, 2400) -2153"
##  [25] "c(411, 2400) -1989" "c(110, 2400) -2290" "c(354, 2400) -2046"
##  [28] "c(411, 2400) -1989" "c(203, 2400) -2197" "c(506, 2359) -1853"
##  [31] "c(429, 2359) -1930" "c(435, 2359) -1924" "c(439, 2359) -1920"
##  [34] "c(437, 2359) -1922" "c(500, 2359) -1859" "c(202, 2359) -2157"
##  [37] "c(420, 2359) -1939" "c(428, 2359) -1931" "c(417, 2359) -1942"
##  [40] "c(419, 2359) -1940" "c(759, 2359) -1600" "c(440, 2359) -1919"
##  [43] "c(136, 2359) -2223" "c(123, 2359) -2236" "c(502, 2359) -1857"
##  [46] "c(507, 2359) -1852" "c(425, 2359) -1934" "c(428, 2359) -1931"
##  [49] "c(443, 2359) -1916" "c(59, 2359) -2300"  "c(240, 2359) -2119"
##  [52] "c(347, 2359) -2012" "c(339, 2359) -2020" "c(325, 2359) -2034"
##  [55] "c(309, 2359) -2050" "c(354, 2359) -2005" "c(339, 2359) -2020"
##  [58] "c(353, 2359) -2006" "c(56, 2359) -2303"  "c(332, 2359) -2027"
##  [61] "c(355, 2359) -2004" "c(345, 2359) -2014" "c(201, 2359) -2158"
##  [64] "c(335, 2359) -2024" "c(344, 2359) -2015" "c(412, 2359) -1947"
##  [67] "c(200, 2359) -2159" "c(118, 2359) -2241" "c(239, 2359) -2120"
##  [70] "c(401, 2359) -1958" "c(338, 2359) -2021" "c(252, 2359) -2107"
##  [73] "c(113, 2359) -2246" "c(354, 2359) -2005" "c(212, 2359) -2147"
##  [76] "c(327, 2359) -2032" "c(351, 2359) -2008" "c(343, 2359) -2016"
##  [79] "c(342, 2359) -2017" "c(350, 2359) -2009" "c(333, 2359) -2026"
##  [82] "c(340, 2359) -2019" "c(345, 2359) -2014" "c(346, 2359) -2013"
##  [85] "c(429, 2358) -1929" "c(436, 2358) -1922" "c(233, 2358) -2125"
##  [88] "c(502, 2358) -1856" "c(142, 2358) -2216" "c(57, 2358) -2301"
##  [91] "c(352, 2358) -2006" "c(350, 2358) -2008" "c(423, 2358) -1935"
##  [94] "c(440, 2358) -1918" "c(425, 2358) -1933" "c(246, 2358) -2112"
##  [97] "c(447, 2358) -1911" "c(129, 2358) -2229" "c(441, 2358) -1917"
## [100] "c(506, 2358) -1852"
```

```
cat("\n\n")
```

```
# Correct the negative flight times. This can be done by adding the departure
# time to the result
corrected_flight_time <- list()
for (i in 1:nrow(desc_dep_time)) {
   if (flight_time[[i]] <= 0 & !is.na(flight_time[[i]])) {
     corrected_flight_time[[i]] <- (2400 - desc_dep_time$dep_time[i]) + (desc_dep_time$dep_time
[i] + flight_time[[i]])
     #print(paste(i,corrected_flight_time[[i]]))
   } else if (!is.na(flight_time[[i]])) {
     corrected_flight_time[[i]] <- flight_time[[i]]
   }
}

paste(corrected_flight_time[c(1:50)])
```

```
##  [1] "327" "515" "427" "432" "59"  "432" "434" "302" "432" "443" "251" "324"
## [13] "339" "339" "339" "347" "338" "339" "102" "107" "101" "225" "54"  "247"
## [25] "411" "110" "354" "411" "203" "547" "470" "476" "480" "478" "541" "243"
## [37] "461" "469" "458" "460" "800" "481" "177" "164" "543" "548" "466" "469"
## [49] "484" "100"
```

```
# Based on index 50 of the result, it is clear the math is incorrect because of
# the departure and arrival time formats (this should be 60, not 100).
# Time to fix it...
```

```
convert_time_to_minutes <- function(input_time) {
  # String splitting depends on the length of the string
  # Split the departure time string and store it as a total
  if (nchar(input_time) == 4){
    # Split the string:
    hour <- as.integer(paste0(unlist(strsplit(input_time, split = ""))[1:2], collapse = ""))
    minute <- as.integer(paste0(unlist(strsplit(input_time, split = ""))[3:4], collapse = ""))
  } else if (nchar(input_time) == 3) {
    # Split after the first character
    hour <- as.integer(paste0(unlist(strsplit(input_time, split = ""))[1], collapse = ""))
    minute <- as.integer(paste0(unlist(strsplit(input_time, split = ""))[2:3], collapse = ""))
  } else if (nchar(input_time) <= 2) {
    hour <- 0
    minute <- as.integer(input_time)
  }

  return (hour * 60 + minute)
}

# Correct the format of the arrival and departure times and store as the total
# number of minutes in the day
flight_time_in_minutes_list <- list()
flight_duration <- list()

for (i in 1:nrow(desc_dep_time)) {
  # Convert the departure time to a string:
  dep_time_minutes <- convert_time_to_minutes(toString(desc_dep_time$dep_time[i]))
  arr_time_minutes <- convert_time_to_minutes(toString(desc_dep_time$arr_time[i]))

  # Compute the flight duration
  flight_duration[[i]] <- arr_time_minutes - dep_time_minutes

  # correct the negative flight durations
  if (flight_duration[[i]] < 0){
    flight_time_in_minutes_list[[i]] <- ((60*24) - dep_time_minutes) + arr_time_minutes
  } else {
    flight_time_in_minutes_list[[i]] <- flight_duration[[i]]
  }

}
print("Corrected negative flight times in minutes:")
```

```
## [1] "Corrected negative flight times in minutes:"
```

```
paste(flight_time_in_minutes_list[c(1:100)])
```

```
##   [1] "207" "315" "267" "272" "59"  "272" "274" "182" "272" "283" "171" "204"
##  [13] "219" "219" "219" "227" "218" "219" "62"  "67"  "61"  "145" "54"  "167"
##  [25] "251" "70"  "234" "251" "123" "307" "270" "276" "280" "278" "301" "123"
##  [37] "261" "269" "258" "260" "480" "281" "97"  "84"  "303" "308" "266" "269"
##  [49] "284" "60"  "161" "228" "220" "206" "190" "235" "220" "234" "57"  "213"
##  [61] "236" "226" "122" "216" "225" "253" "121" "79"  "160" "242" "219" "173"
##  [73] "74"  "235" "133" "208" "232" "224" "223" "231" "214" "221" "226" "227"
##  [85] "271" "278" "155" "304" "104" "59"  "234" "232" "265" "282" "267" "168"
##  [97] "289" "91"  "283" "308"
```

```
cat("\n\n")
```

```
# Based on the output above, the math is now correct for negative flight times.

# Check the output for non-negative flight times
print("Corrected non-negative flight times in minutes:")
```

```
## [1] "Corrected non-negative flight times in minutes:"
```

```
paste(head(data[c(which(flight_duration > 0))],50), head(flight_time_in_minutes_list[c(which(fli
ght_duration > 0))],50))
```

```
##  [1] "c(2358, 2318) 40" "c(2357, 2313) 44" "c(2400, 2310) 50" "c(2358, 2307) 51"
##  [5] "c(2353, 2307) 46" "c(2359, 2307) 52" "c(2355, 2306) 49" "c(2357, 2306) 51"
##  [9] "c(2350, 2306) 44" "c(2357, 2305) 52" "c(2357, 2305) 52" "c(2356, 2305) 51"
## [13] "c(2359, 2305) 54" "c(2356, 2304) 52" "c(2354, 2304) 50" "c(2400, 2303) 57"
## [17] "c(2358, 2303) 55" "c(2354, 2303) 51" "c(2400, 2303) 57" "c(2357, 2303) 54"
## [21] "c(2358, 2303) 55" "c(2358, 2303) 55" "c(2342, 2302) 40" "c(2355, 2302) 53"
## [25] "c(2356, 2302) 54" "c(2358, 2302) 56" "c(2400, 2302) 58" "c(2400, 2301) 59"
## [29] "c(2357, 2301) 56" "c(2350, 2300) 50" "c(2359, 2300) 59" "c(2356, 2300) 56"
## [33] "c(2351, 2300) 51" "c(2400, 2300) 60" "c(2357, 2300) 57" "c(2357, 2259) 58"
## [37] "c(2356, 2259) 57" "c(2348, 2259) 49" "c(2400, 2259) 61" "c(2357, 2259) 58"
## [41] "c(2400, 2259) 61" "c(2359, 2259) 60" "c(2359, 2258) 61" "c(2351, 2258) 53"
## [45] "c(2350, 2258) 52" "c(2400, 2258) 62" "c(2355, 2258) 57" "c(2359, 2258) 61"
## [49] "c(2358, 2258) 60" "c(2359, 2258) 61"
```

```
cat("\n\n")
```

```
# This now seems to be outputting the data correctly... time to sort the original
# data set by the indices of corrected flight duration
```

```
# The easiest way to go about this is to add the list as a column to the data
# frame, then call the arrange() function on the column.

desc_dep_time <- add_column(desc_dep_time, flight_duration = unlist(flight_time_in_minutes_lis
t))
head(desc_dep_time, 100)
```

```
## # A tibble: 100 × 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    10    30     2400           2359         1      327            337
##  2  2013    11    27     2400           2359         1      515            445
##  3  2013    12     5     2400           2359         1      427            440
##  4  2013    12     9     2400           2359         1      432            440
##  5  2013    12     9     2400           2250        70       59           2356
##  6  2013    12    13     2400           2359         1      432            440
##  7  2013    12    19     2400           2359         1      434            440
##  8  2013    12    29     2400           1700       420      302           2025
##  9  2013     2     7     2400           2359         1      432            436
## 10  2013     2     7     2400           2359         1      443            444
## # ℹ 90 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>, flight_duration <dbl>
```

```
# Now sort desc_dep_time by flight duration
desc_dep_time <- desc_dep_time|>
  arrange(flight_duration)
head(desc_dep_time, 100)
```

```
## # A tibble: 100 × 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     8    14     1133           1148       -15     1206           1249
##  2  2013     1     5     1323           1325        -2     1358           1421
##  3  2013     2    22     1312           1316        -4     1347           1413
##  4  2013     9     3     1203           1153        10     1238           1250
##  5  2013     3    19     1455           1329        86     1531           1426
##  6  2013     8    14      722            730        -8      758            830
##  7  2013     8    18      722            729        -7      758            830
##  8  2013     9    24      718            725        -7      754            822
##  9  2013     3    18     1456           1329        87     1533           1426
## 10  2013     3    27     1418           1329        49     1455           1426
## # ℹ 90 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>, flight_duration <dbl>
```

```
tail(desc_dep_time, 100)
```

```
## # A tibble: 100 × 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     2    27      629            630        -1     1303           1140
##  2  2013     1    26      901            900         1     1536           1530
##  3  2013     7     3      636            640        -4     1311           1040
##  4  2013    12     6      932            930         2     1608           1527
##  5  2013    11     6      927            930        -3     1603           1530
##  6  2013     6    14       11           2359        12      647            350
##  7  2013     1    25     1755           1341       254       34           1935
##  8  2013     1     9     1340           1341        -1     2019           1935
##  9  2013    11    12      955           1000        -5     1634           1555
## 10  2013     9    25     2052           2045         7      332             53
## # ℹ 90 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>, flight_duration <dbl>
```

After all of that, I took another look at the book and realized I could have stepped through most of the process with the *mutate()* function… :(

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

```
# First, filter by year, add a YYYY-MM-DD column, and get distinct dates
filtered_by_year <- flights |>
  filter(year == 2013) |>
  mutate(date = as.Date(paste(year, month, day, sep="-"))) |>
  distinct(date)

# Generate the date range
dates <- seq(as.Date("2013-01-01"), by = "day", length.out = 365)

# Loop through filtered dates and see if any were missed
no_fly_count <- 0
fly_count <- 0
for (i in dates){
  if (length(which(filtered_by_year$date==i)) == 0){
    paste("There were zero flights on ",filtered_by_year)
    no_fly_count <- no_fly_count + 1
  } else {
    fly_count <- fly_count + 1
  }
}
cat("Days with flights: ", fly_count, "\tDays without flights: ", no_fly_count)
```

```
## Days with flights:  365  Days without flights:  0
```

- There was a flight on every day in 2013.

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

```
flight_distance <- flights |>
  arrange(distance) |>
  distinct(flight)


shortest <- head(flight_distance, 10)
longest <- tail(flight_distance, 10)

new_df <- as.data.frame(shortest$flight, longest$flight)
new_df
```

```
##      shortest$flight
## 106             1632
## 1700            3833
## 1196            4193
## 1546            4502
## 173             4645
## 167             4619
## 1865            3271
## 512             4616
## 1765            4457
## 1933            4088
```

```
cat("Farthest Flights:\tShortest Flights:\n")
```

```
## Farthest Flights:    Shortest Flights:
```

```
for (i in 1:10) {
  cat(longest$flight[[i]],"\t\t\t",shortest$flight[[i]],"\n")
}
```

```
## 106          1632
## 1700             3833
## 1196             4193
## 1546             4502
## 173          4645
## 167          4619
## 1865             3271
## 512          4616
## 1765             4457
## 1933             4088
```

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.

- Yes, it does matter (mostly for large datasets). The filter function will remove rows from the data frame based on a condition (meaning it has to *search* for the condition among all rows), where arrange sorts based on a criteria. Based on my existing knowledge of computer science and some additional information from this blog (https://www.quora.com/Is-sorting-faster-than-searching), searches can be performed more efficiently on a sorted data set, especially when the number of search targets are unknown.

(1) Section 3.3.5 Questions

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

```
head(flights,10)
```

```
## # A tibble: 10 × 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 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>
```

- I would expect the relationship to be: dep_delay = dep_time - sched_dep_time

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

```
flights
```

```
## # A tibble: 336,776 × 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 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>
```

```
selected1 <- flights |>
  select(dep_time, dep_delay, arr_time, arr_delay)
selected1
```

```
## # A tibble: 336,776 × 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
```

```
selected2 <- flights |>
  select(starts_with("arr"), starts_with("dep"))
selected2
```

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

```
char_vec <- c("dep_time", "dep_delay", "arr_time", "arr_delay")

selected3 <- flights |>
  select(all_of(char_vec))
selected3
```

```
## # A tibble: 336,776 × 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
```

```
selected4 <- flights |>
  select(any_of(char_vec))
selected4
```

```
## # A tibble: 336,776 × 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
```

```
selected5 <- flights |>
  select(!year:day, !sched_dep_time, !sched_arr_time, !carrier:time_hour)
```

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

```
selected <- flights |>
  select(dep_time, dep_time, dep_time, dep_time)
selected
```

```
## # A tibble: 336,776 × 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
```

- It seems like R internally removes duplicate variable specifications.

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

> variables <- c(“year”, “month”, “day”, “dep_delay”, “arr_delay”)

- Any of will select all variables in the data frame from the list, ignoring names that don't exist. This is helpful for large datasets because the vector could be generated programatically.

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"))

```
flights |> select(contains("TIME"))
```

```
## # A tibble: 336,776 × 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
```

- This seems like a reasonable result, many programming languages have case handling built in. For R, there is an "ignore.case" parameter in the "contains" function that is defaulted to TRUE. This can be disabled by passing it with the function, as seen below:

```
flights |> select(contains("TIME", ignore.case = FALSE))
```

```
## # A tibble: 336,776 × 0
```

-As expected, this returned an empty tibble, because all variables in this dataset that contain "time" are lowercase.

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

```
renamed <- flights |>
  rename(air_time_min = air_time ) |>
  relocate(air_time_min)
renamed
```

```
## # A tibble: 336,776 × 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)
```

- This does not work because tailnum is the only selected column. The data frame cannot be arranged by the arr_delay variable because it does not exist in the selected set. The error indicates the arr_delay object in the flights data frame cannot be found.

(1) Section 3.3.5 Questions

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()))

```
carriers <- flights |>
  group_by(carrier) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE)
  ) |>
  mutate(sum_of_avg_delay = avg_dep_delay + avg_arr_delay) |>
  arrange(sum_of_avg_delay)

tail(carriers,3)
```

```
## # A tibble: 3 × 4
##   carrier avg_dep_delay avg_arr_delay sum_of_avg_delay
##   <chr>           <dbl>         <dbl>            <dbl>
## 1 EV               20.0          15.8             35.8
## 2 FL               18.7          20.1             38.8
## 3 F9               20.2          21.9             42.1
```

- The worst average delays come from EV, FL, and F9 carriers.

Challenge

```
# As a means to better understand the group_by function, first group by each
# of the variables independently
challenge1 <- flights |>
  group_by(carrier) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE),
  )|>
  mutate(sum_of_avg_delay = avg_dep_delay + avg_arr_delay) |>
  arrange(desc(sum_of_avg_delay))
challenge1
```

```
## # A tibble: 16 × 4
##    carrier avg_dep_delay avg_arr_delay sum_of_avg_delay
##    <chr>           <dbl>         <dbl>            <dbl>
##  1 F9              20.2         21.9              42.1
##  2 FL              18.7         20.1              38.8
##  3 EV              20.0         15.8              35.8
##  4 YV              19.0         15.6              34.6
##  5 WN              17.7          9.65             27.4
##  6 OO              12.6         11.9              24.5
##  7 9E              16.7          7.38             24.1
##  8 B6              13.0          9.46             22.5
##  9 MQ              10.6         10.8              21.3
## 10 UA              12.1          3.56             15.7
## 11 VX              12.9          1.76             14.6
## 12 DL               9.26         1.64             10.9
## 13 AA               8.59         0.364             8.95
## 14 US               3.78         2.13              5.91
## 15 HA               4.90        -6.92             -2.01
## 16 AS               5.80        -9.93             -4.13
```

```
challenge2 <- flights |>
  group_by(origin) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE),
  )|>
  mutate(sum_of_avg_delay = avg_dep_delay + avg_arr_delay) |>
  arrange(desc(sum_of_avg_delay))
challenge2
```

```
## # A tibble: 3 × 4
##   origin avg_dep_delay avg_arr_delay sum_of_avg_delay
##   <chr>          <dbl>         <dbl>            <dbl>
## 1 EWR             15.1          9.11             24.2
## 2 JFK             12.1          5.55             17.7
## 3 LGA             10.3          5.78             16.1
```

```
challenge3 <- flights |>
  group_by(dest) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE),
  )|>
  mutate(sum_of_avg_delay = avg_dep_delay + avg_arr_delay) |>
  arrange(desc(sum_of_avg_delay))
challenge3
```

```
## # A tibble: 105 × 4
##    dest  avg_dep_delay avg_arr_delay sum_of_avg_delay
##    <chr>         <dbl>         <dbl>            <dbl>
##  1 CAE            35.6          41.8             77.3
##  2 TUL            34.9          33.7             68.6
##  3 OKC            30.6          30.6             61.2
##  4 JAC            26.5          28.1             54.6
##  5 TYS            28.5          24.1             52.6
##  6 BHM            29.7          16.9             46.6
##  7 DSM            26.2          19.0             45.2
##  8 MSN            23.6          20.2             43.8
##  9 RIC            23.6          20.1             43.8
## 10 CAK            20.8          19.7             40.5
## # ℹ 95 more rows
```

- Based on the output, the worst carriers are F9, FL, and EV, the worst origin airports are EWR, JFK, and LGA, and the worst destination airports are CAE, TUL, and OKC. Now, try grouping by all three variables simultaneously…

```
challenge <- flights |>
  group_by(carrier, origin, dest) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE),
    avg_arr_delay = mean(arr_delay, na.rm = TRUE),
  )|>
  mutate(sum_of_avg_delay = avg_dep_delay + avg_arr_delay) |>
  arrange(desc(sum_of_avg_delay))
```

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

```
challenge
```

```
## # A tibble: 439 × 6
## # Groups:   carrier, origin [35]
##    carrier origin dest  avg_dep_delay avg_arr_delay sum_of_avg_delay
##    <chr>   <chr>  <chr>         <dbl>         <dbl>             <dbl>
##  1 EV      JFK    ATL           124           128               252
##  2 9E      LGA    MSP            90           100               190
##  3 EV      LGA    XNA            71           119               190
##  4 UA      EWR    STL            77.5         110               188.
##  5 OO      LGA    ORD            67           107               174
##  6 EV      LGA    MSY            74            72.5             146.
##  7 OO      EWR    DTW            61            68.5             130.
##  8 UA      EWR    RDU            60            56               116
##  9 EV      LGA    ATL            30            63                93
## 10 EV      LGA    PBI            48.7          40.7              89.3
## # ℹ 429 more rows
```

- It appears that EV flies out of LGA and JFK very often in the grouped list. This is a good indication that the EV carrier may not be the issue, but the origin airports skew the delays positive.

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

```
flight_delays <- flights |>
  group_by(flight, dest) |>
  summarize(
    avg_dep_delay = mean(dep_delay, na.rm = TRUE)
  ) |>
 arrange(desc(avg_dep_delay))
```

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

```
flight_delays
```

```
## # A tibble: 11,467 × 3
## # Groups:   flight [3,844]
##    flight dest  avg_dep_delay
##     <int> <chr>         <dbl>
##  1    372 CLE             483
##  2    488 DEN             379
##  3   5017 DTW             372
##  4    521 BNA             346
##  5    468 MCO             334
##  6   3261 CLE             321
##  7    809 SLC             298
##  8   1510 IAH             278
##  9   1275 DEN             277
## 10   5478 GSO             242
## # ℹ 11,457 more rows
```

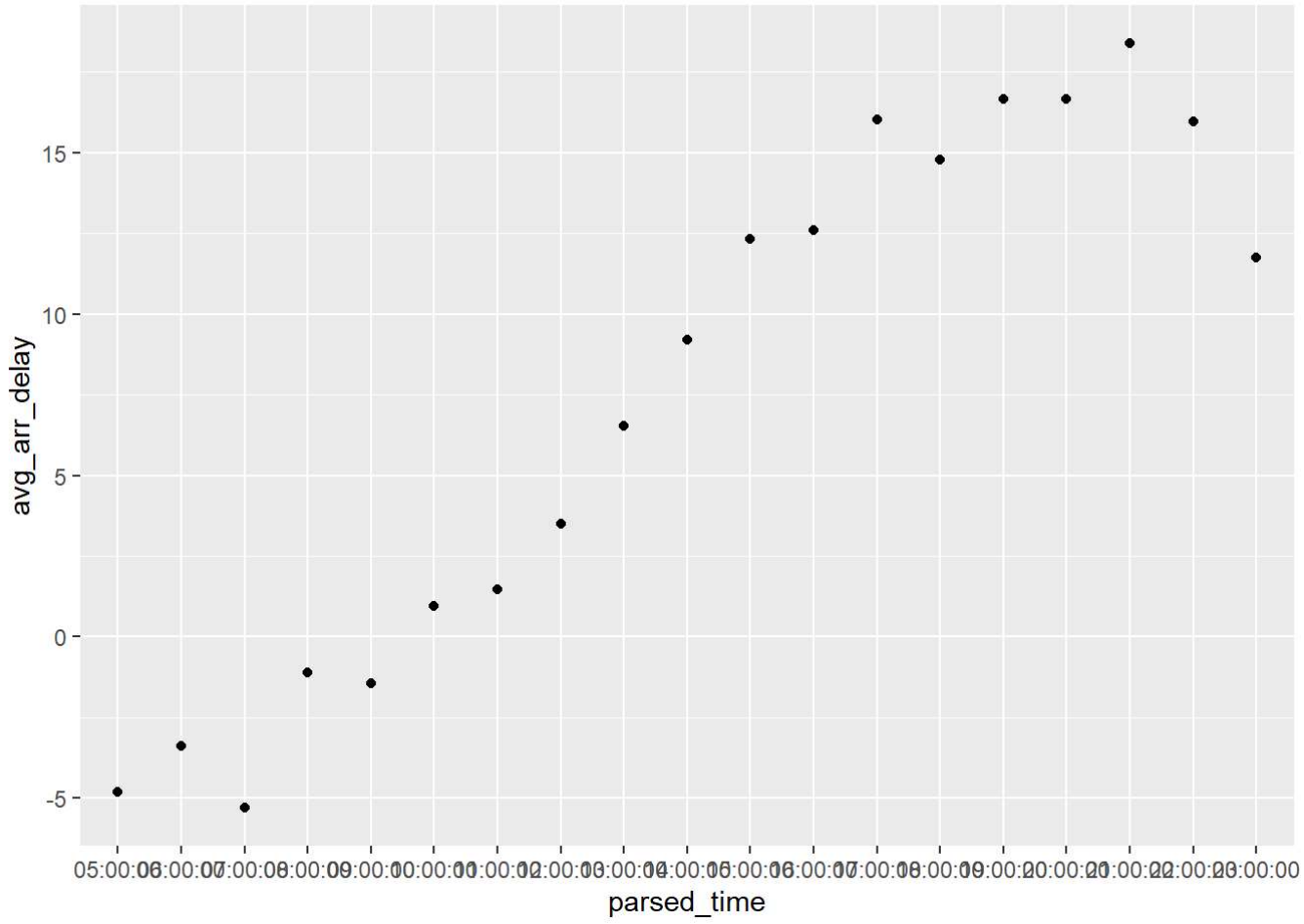
3. How do delays vary over the course of the day? Illustrate your answer with a plot.

Source for extracting the time from POSIXct data type: https://stackoverflow.com/questions/9839343/extracting-time-from-posixct (https://stackoverflow.com/questions/9839343/extracting-time-from-posixct)

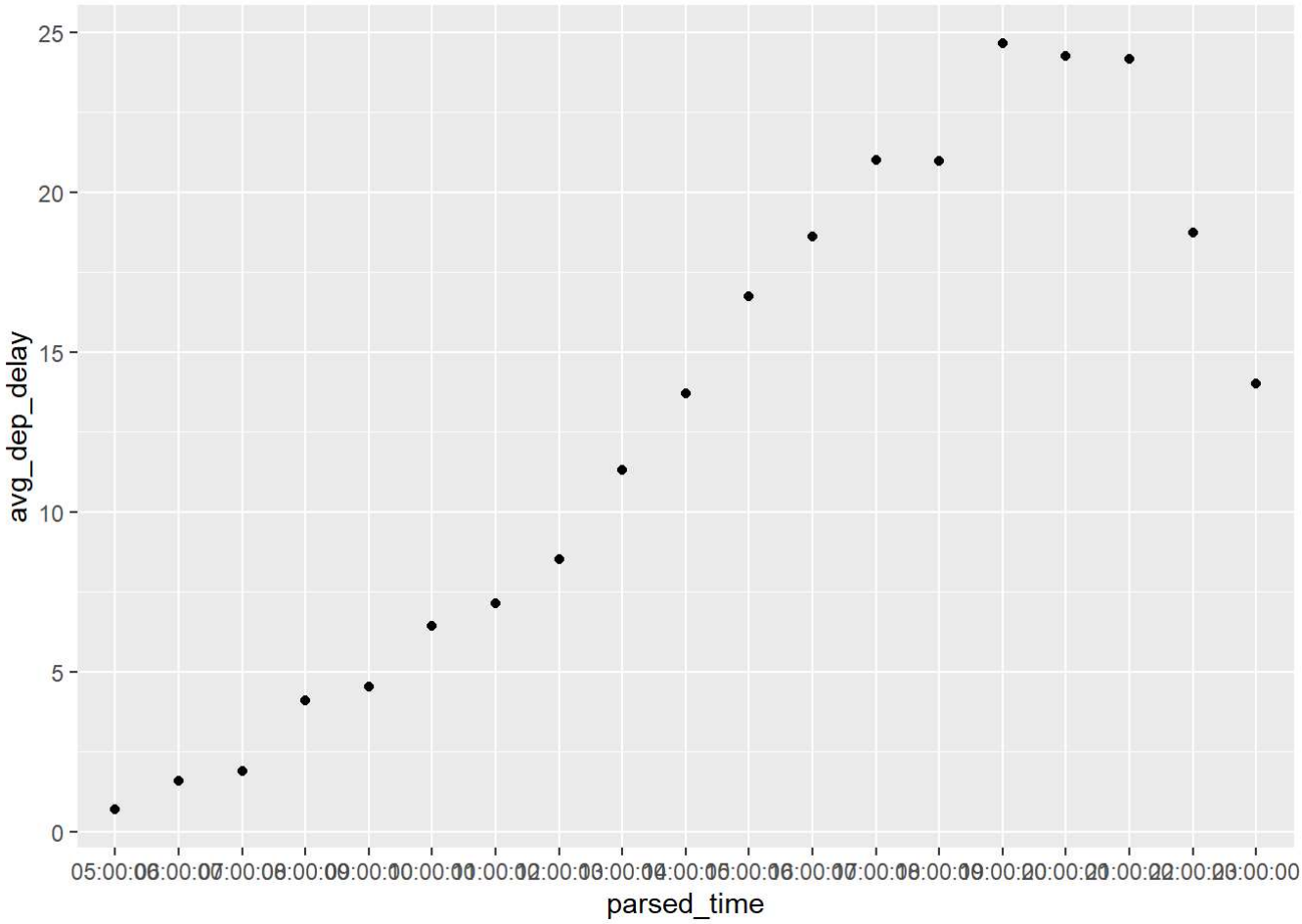
```
# Create a function to take in a year, day, and month and return a filtered,
# summarized dataframe
grouped_data <- function() {
  temp_df <- flights |>
    # Extracting the time from POSIXct data type
    mutate(
      parsed_time = strftime(time_hour, format="%H:%M:%S")
      ) |>
    # Group by time_hour, as this can be create the x axis over the course
    # of the day
    group_by(parsed_time) |>
    # Remove all NA values
    filter(!is.na(arr_delay) & !is.na(dep_delay)) |>
    # Then summarize the average arrival and departure delays across each timestamp
    summarize(
      avg_arr_delay = mean(arr_delay, na.rm=TRUE),
      avg_dep_delay = mean(dep_delay, na.rm=TRUE)
    )

  return (temp_df)
}

# Let the function, above, process the data for the sake of making this neat
df <- grouped_data()
df
```

```
## # A tibble: 19 × 3
##    parsed_time avg_arr_delay avg_dep_delay
##    <chr>               <dbl>         <dbl>
##  1 05:00:00           -4.80          0.689
##  2 06:00:00           -3.38          1.60
##  3 07:00:00           -5.30          1.91
##  4 08:00:00           -1.11          4.11
##  5 09:00:00           -1.45          4.54
##  6 10:00:00            0.954         6.45
##  7 11:00:00            1.48          7.15
##  8 12:00:00            3.49          8.52
##  9 13:00:00            6.54         11.3
## 10 14:00:00            9.20         13.7
## 11 15:00:00           12.3          16.8
## 12 16:00:00           12.6          18.6
## 13 17:00:00           16.0          21.0
## 14 18:00:00           14.8          21.0
## 15 19:00:00           16.7          24.7
## 16 20:00:00           16.7          24.2
## 17 21:00:00           18.4          24.2
## 18 22:00:00           16.0          18.7
## 19 23:00:00           11.8          14.0
```

```
# plot a line graph over the course of the day for arrival and departure delays
ggplot(
  data = df,
  mapping = aes(x=parsed_time, y=avg_arr_delay)
) + geom_point()
```




```
ggplot(
  data = df,
  mapping = aes(x=parsed_time, y=avg_dep_delay)
) + geom_point()
```



- The average delay for a given time increases over the course of the day, with the sharpest spike in delays occurring between 12:00 and 18:00.

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

```
# Test each slice function with n=1
flights |> slice_head(n = 1)
```

```
## # A tibble: 1 × 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
## # 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 |> slice_tail(n = 1)
```

```
## # A tibble: 1 × 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     9    30       NA            840        NA       NA           1020
## # 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 |> slice_min(arr_delay, n = 1)
```

```
## # A tibble: 1 × 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     1715           1729       -14     1944           2110
## # 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 |> slice_max(arr_delay, n = 1)
```

```
## # A tibble: 1 × 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
## # 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 |> slice_sample(n = 1)
```

```
## # A tibble: 1 × 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    11    15     1755           1759        -4     2059           2053
## # 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>
```

```
# Test each slice function with n=-1
flights |> slice_head(n = -1)
```

```
## # A tibble: 336,775 × 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 336,765 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 |> slice_tail(n = -1)
```

```
## # A tibble: 336,775 × 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      533            529         4      850            830
##  2  2013     1     1      542            540         2      923            850
##  3  2013     1     1      544            545        -1     1004           1022
##  4  2013     1     1      554            600        -6      812            837
##  5  2013     1     1      554            558        -4      740            728
##  6  2013     1     1      555            600        -5      913            854
##  7  2013     1     1      557            600        -3      709            723
##  8  2013     1     1      557            600        -3      838            846
##  9  2013     1     1      558            600        -2      753            745
## 10  2013     1     1      558            600        -2      849            851
## # ℹ 336,765 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 |> slice_min(arr_delay, n = -1)
```

```
## # A tibble: 336,776 × 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     1715           1729       -14     1944           2110
##  2  2013     5    20      719            735       -16      951           1110
##  3  2013     5     2     1947           1949        -2     2209           2324
##  4  2013     5     6     1826           1830        -4     2045           2200
##  5  2013     5     4     1816           1820        -4     2017           2131
##  6  2013     5     2     1926           1929        -3     2157           2310
##  7  2013     5     6     1753           1755        -2     2004           2115
##  8  2013     5     7     2054           2055        -1     2317             28
##  9  2013     5    13      657            700        -3      908           1019
## 10  2013     1     4     1026           1030        -4     1305           1415
## # 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>
```

```
flights |> slice_max(arr_delay, n = -1)
```

```
## # A tibble: 336,776 × 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
## # 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>
```

```
flights |> slice_sample(n = -1)
```

```
## # A tibble: 336,775 × 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    27     1122           1130        -8     1259           1327
##  2  2013     6     5     1218           1221        -3     1513           1537
##  3  2013    12     7     1459           1459         0     1610           1625
##  4  2013     8     2      602            608        -6      708            719
##  5  2013    10    27     1654           1659        -5     1946           2007
##  6  2013    11    11      828            825         3      919            935
##  7  2013     9    20     1012           1000        12     1124           1131
##  8  2013     2    19     1527           1500        27     1838           1825
##  9  2013     5    25     1056           1052         4     1342           1406
## 10  2013     6    21     1732           1734        -2     1949           1956
## # ℹ 336,765 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>
```

- The functions perform as follows with a negative input:
    a. slice_head returns every row except the last
    b. slice_tail returns every row except the first
    c. slice_min sorts from minimum to maximum
    d. slice_max sorts from maximum to minimum
    e. slice_sample returns the dataframe minus the randomly sliced sample

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

```
understanding_count <- flights |>
  group_by(time_hour) |>
  count(time_hour)
understanding_count
```

```
## # A tibble: 6,936 × 2
## # Groups:   time_hour [6,936]
##    time_hour               n
##    <dttm>              <int>
##  1 2013-01-01 05:00:00     6
##  2 2013-01-01 06:00:00    52
##  3 2013-01-01 07:00:00    49
##  4 2013-01-01 08:00:00    58
##  5 2013-01-01 09:00:00    56
##  6 2013-01-01 10:00:00    39
##  7 2013-01-01 11:00:00    37
##  8 2013-01-01 12:00:00    56
##  9 2013-01-01 13:00:00    54
## 10 2013-01-01 14:00:00    48
## # ℹ 6,926 more rows
```

```
understanding_count <- flights |>
  summarize(
    max_time_hour = max(time_hour)
    ) |>
  count(max_time_hour)
understanding_count
```

```
## # A tibble: 1 × 2
##   max_time_hour           n
##   <dttm>              <int>
## 1 2013-12-31 23:00:00     1
```

```
understanding_count <- flights |>
  count(time_hour) |>
  summarize(
    avg_n = mean(n)
    )
understanding_count
```

```
## # A tibble: 1 × 1
##   avg_n
##   <dbl>
## 1  48.6
```

- Count returns the number of instances of a given variable in the modified data frame. It functions similarly to group_by, but it also returns the instances as a new row. The last *understanding_count* data frame first counts time_hour, then finds the average number of instances across all times.

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 × 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
```

EXPECTED OUTPUT:

x	y	z
1	a	K
2	b	K
3	a	L
4	a	L
5	b	K

```
df1 <- df |>
  group_by(y)
df1
```

```
## # A tibble: 5 × 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
```

```
group_by does not modify the tibble, it just creates groups based on the unique values in the gr
ouped column.
```

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 × 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
```

EXPECTED OUTPUT:

x	y	z
1	a	K
3	a	L
4	a	L
2	b	K
5	b	K

```
df2 <- df |>
  arrange(y)
df2
```

```
## # A tibble: 5 × 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
```

```
arrange sorts the input column in ascending order, which, unlike group_by, actively modifies the row order in the data frame. I made a guess about the implementation of arrange, assuming that it would loop row-wise until it found a non *a* value, store the index, then continue until an *a* was found and flip the rows by index. It seems that was the case based on the resulting row order.
```

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 × 2
##   y     mean_x
##   <chr>  <dbl>
## 1 a       2.67
## 2 b       3.5
```

EXPECTED OUTPUT:

y	mean_x
a	2.67
b	3.5

```
df3 <- df |>
  group_by(y) |>
  summarize(mean_x = mean(x))
df3
```

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

```
Summarizing on a variable will collapse the output to the grouped range. In this case, x is aver
aged on each of the y groups, therefore the calculation is: mean_x(a) = (1+3+4)/3 = 2.667 and me
an_x(b) = (2+5)/2 = 3.5. The pipeline feeds the output of the df.group_by(y) function into the s
ummarize function. Programmatically, this would look like summarize(df.group_by(y), mean_x = mea
n(x)). The pipeline recursively wraps the output of the prior function into the next function in
the pipeline.
```

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.

```
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 × 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
```

EXPECTED OUTPUT:

y	z	mean_x
a	K	1.0
a	L	3.5
b	k	3.5

```
df4 <- df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

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

```
df4
```

```
## # A tibble: 3 × 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
```

```
- The pipeline groups the input dataframe by y values, then z values. The y group has two instan
ces of *a* because there are both (a,K) and (a,L) pairs of (y,z) groups for y=a, whereas there a
re only (b, K) pairs of y=b groups for (y,z). The mean function behaves as follows for each grou
p: mean((y,z)_(a,K)) = 1/1 = 1, mean((y,z)_(a,L)) = (3+4)/2 = 3.5, mean((y,z)_(b,K)) = (2+5)/2 =
3.5. The message is in regard to the implementation of the summarize(group_by(var1,var2,var3,et
c.)) function. By default, this function drops the last group, but this can be mitigated using t
he .group argument.
```

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)?

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

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

EXPECTED OUTPUT:

y	z	mean_x
a	K	1.0
a	L	3.5
b	k	3.5

```
df5 <- df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x), .groups = "drop")
df5
```

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

```
The pipeline, in this case, does the exact same set of operations as the last question. The .gro
ups = "drop" argument drops the groups, leaving just the resulting tibble in the output, not a t
ibble with a set of three groups ((a,K),(a,L),(b,K)).
```

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?

```
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 × 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
```

EXPECTED OUTPUT:

y	z	mean_x
a	K	1.0
a	L	3.5
b	k	3.5

```
df6 <- df |>
  group_by(y, z) |>
  summarize(mean_x = mean(x))
```

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

```
df6
```

```
## # A tibble: 3 × 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
```

```
This pipeline does the same process as the pipeline in the prior question, but it retains the gr
ouped dataframe.
```

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

```
## # A tibble: 5 × 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
```

EXPECTED OUTPUT:

x	y	z	mean_x
1	a	K	1.0
2	b	K	3.5
3	a	L	3.5
4	a	L	3.5
5	b	K	3.5

```
df7 <- df |>
  group_by(y, z) |>
  mutate(mean_x = mean(x))
df7
```

```
## # A tibble: 5 × 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
```

```
This pipeline does not compress the output into a 3x3 tibble like the prior pipeline. Instead, t
he entire data frame is retained, as well as the y and z groups, and additionally the calculated
mean column is added for each row. The grouped outputs remain the same, and they are distributed
across the remaining rows of the tibble. In the summarized tibbles, these rows are removed from
the data frame as they are nested into the (y,z) group. This is the key difference from the prev
ious pipeline.
```

(2) Exercise 3.1 from Yakir

Three sequences of data were saved in 3 R objects named “x1”, “x2” and “x3”, respectively. The application of the function “summary” to each of these objects is presented below:

```
summary(x1) Min. 1st Qu. Median Mean 3rd Qu. Max. 0.000 2.498 3.218 3.081 3.840 4.871
```

```
summary(x2) Min. 1st Qu. Median Mean 3rd Qu. Max. 0.0001083 0.5772000 1.5070000 1.8420000 2.9050000 4.9880000
```

```
summary(x3) Min. 1st Qu. Median Mean 3rd Qu. Max. 2.200 3.391 4.020 4.077 4.690 6.414
```

In Figure 3.3 one may find the histograms of these three data sequences, given in a random order. In Figure 3.4 one may find the box plots of the same data, given in yet a different order.

1. Match the summary result with the appropriate histogram and the appropriate box plot.

Dataset	Histogram	Boxplot
x1	1	2
x2	2	3
x3	3	1

2. Is the value 0.000 in the sequence “x1” an outlier?

- Yes, 0.000 in x1 appears to be an outlier as it is on the lower bound of the normal distribution and there is only a single data point for that value. Also, in the accompanying Boxplot 2, the point 0 is indicated on the plot as an open circle, identifying it as an outlier.

3. Is the value 6.414 in the sequence “x3” an outlier?

- No, this is not an outlier, although it appear to be on the upper end of the expected range. This value is within the variance, identified by the box plot.

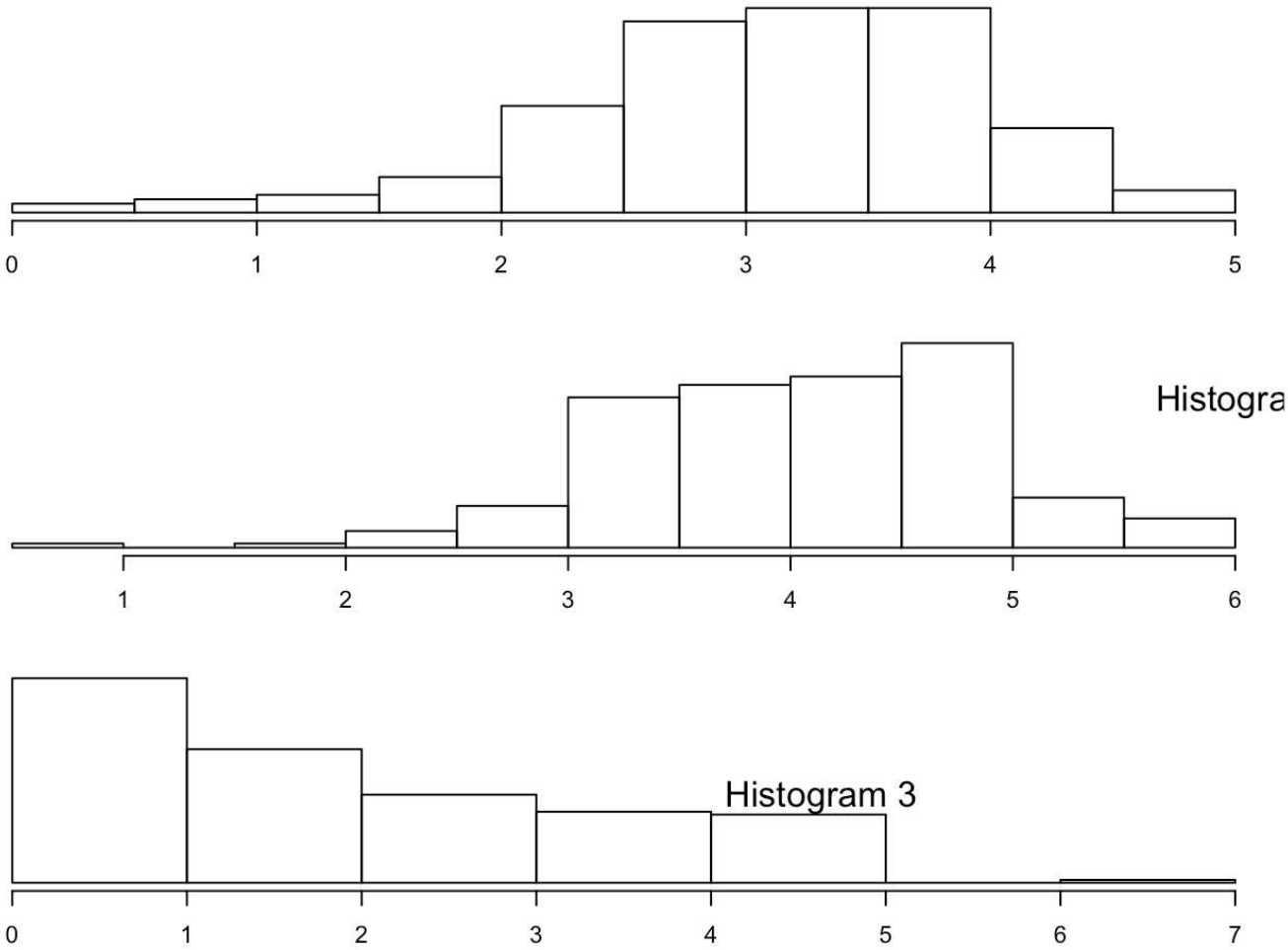


FIGURE 3.3: Three Histograms

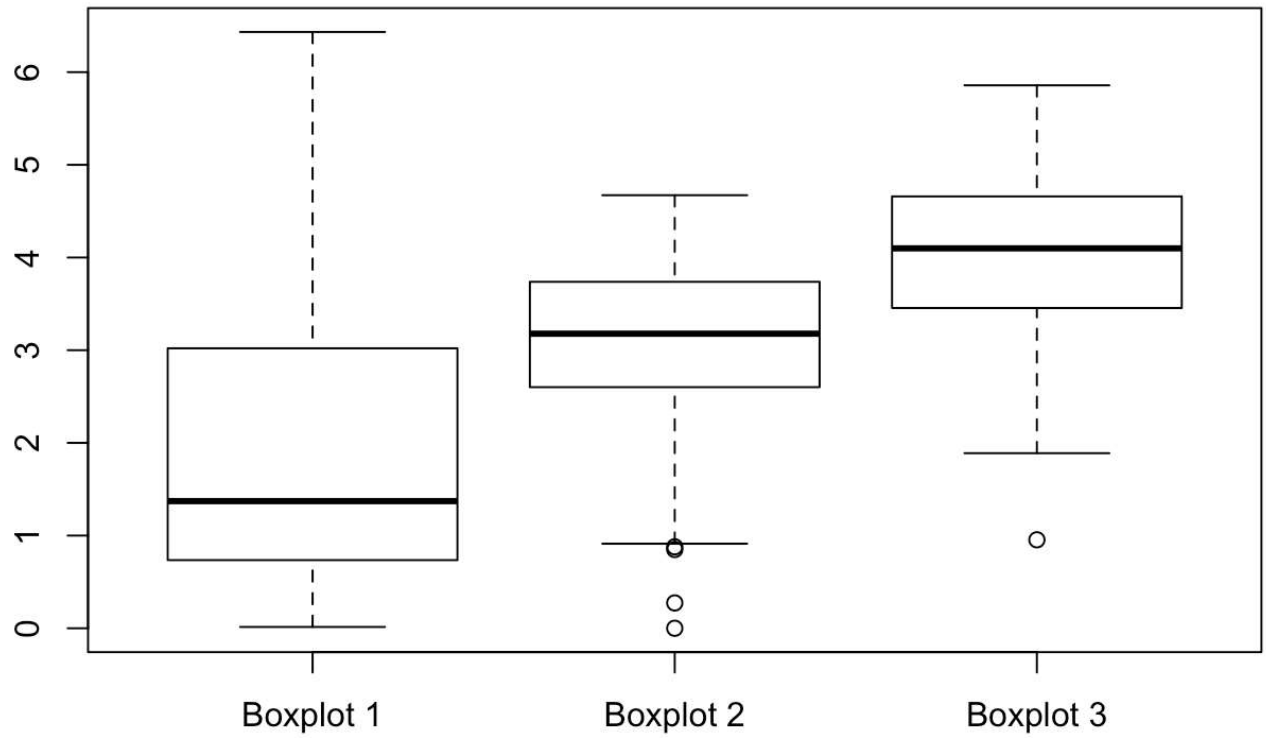


FIGURE 3.4: Three Box Plots

(3) Load the carprice, csv dataset into R, and use dplyr and ggplot to:

- Load carprice dataset:

```
library(readr)
```

```
carprice <- read_csv("carprice.csv")
```

```
## New names:
## Rows: 48 Columns: 10
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter: "," chr
## (1): Type dbl (9): ...1, Min.Price, Price, Max.Price, Range.Price, RoughRange,
## gpm100,...
## ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`
```

a. Use the basic df commands to find out what the variables in this data are, and how many rows and columns there are

```
variable.names(carprice)
```

```
##  [1] "...1"         "Type"         "Min.Price"    "Price"        "Max.Price"
##  [6] "Range.Price" "RoughRange"  "gpm100"       "MPG.city"     "MPG.highway"
```

```
nrow(carprice)
```

```
## [1] 48
```

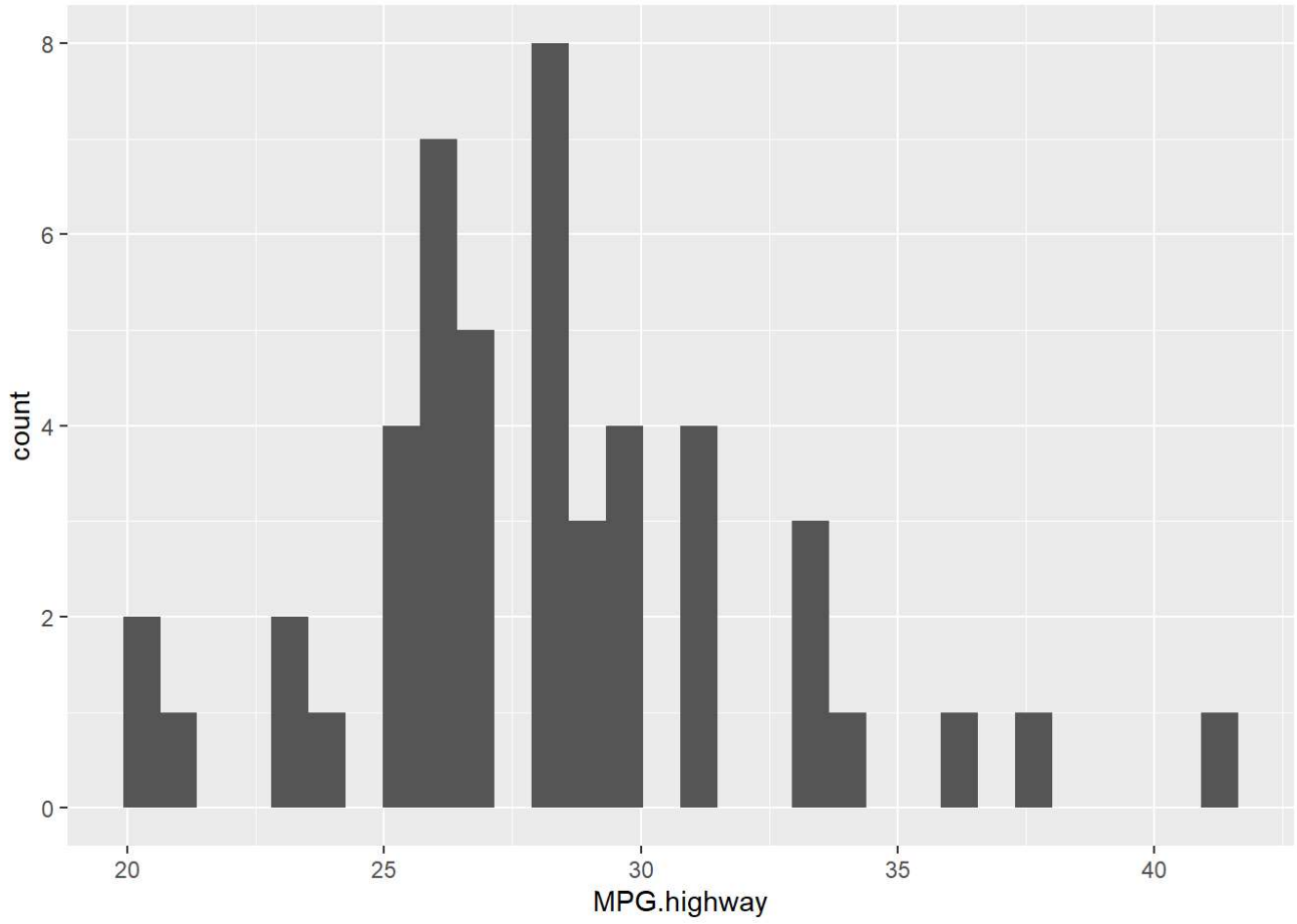
```
ncol(carprice)
```

```
## [1] 10
```

b. Show a histogram of mpg.highway and Price, compute mean, median, standard deviation, skew and kurtosis for these variables, explain how differences in these values relate to visual differences in the histograms

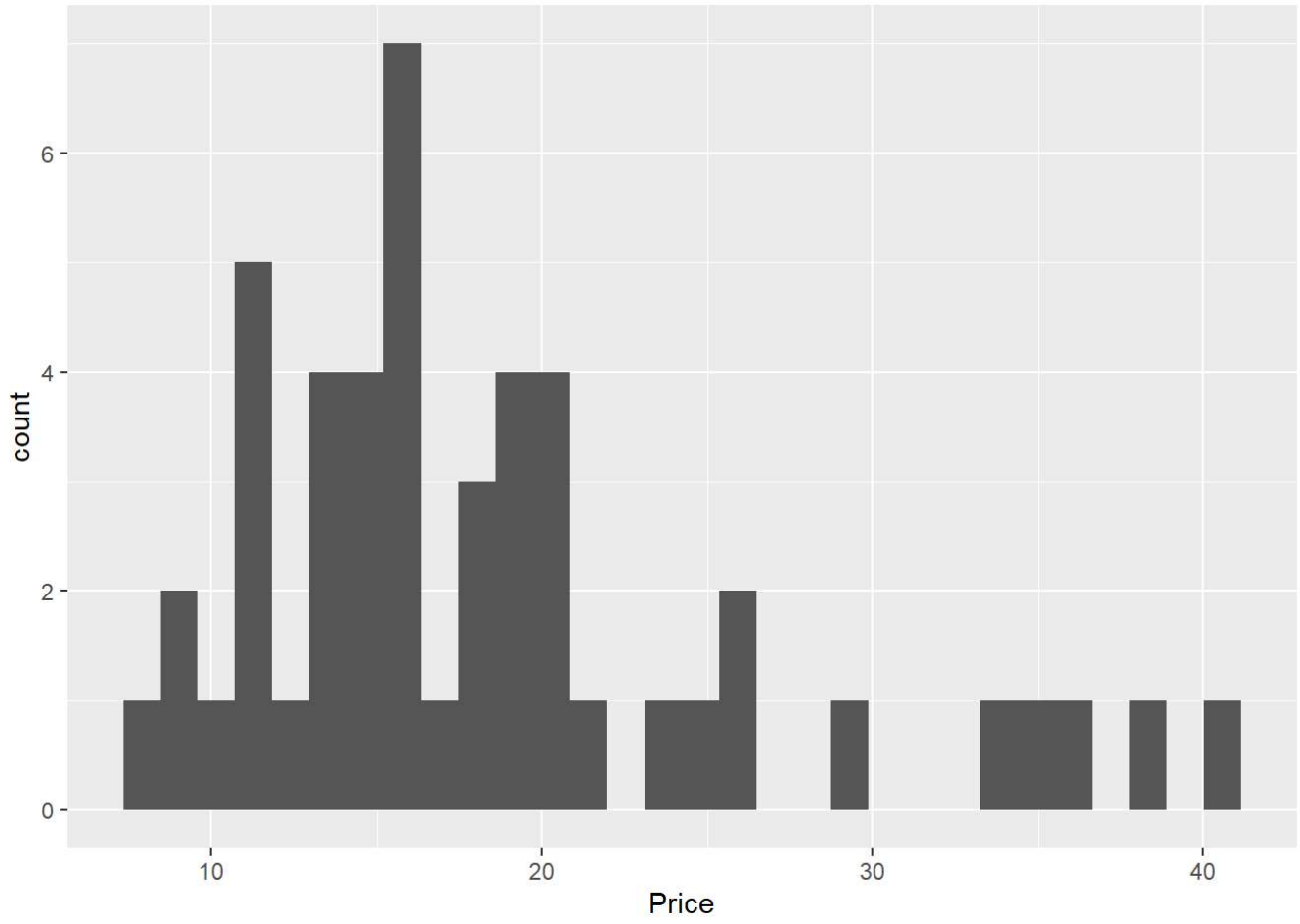
```
ggplot(
  data = carprice,
  mapping = aes(x = MPG.highway)
) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(
  data = carprice,
  mapping = aes(x = Price)
) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Skew and kurtosis calculation sources: https://www.geeksforgeeks.org/skewness-and-kurtosis-in-r-programming/ (https://www.geeksforgeeks.org/skewness-and-kurtosis-in-r-programming/)
https://www.sciencedirect.com/topics/mathematics/kurtosis (https://www.sciencedirect.com/topics/mathematics/kurtosis)

Source to understand kurtosis: https://ecampusontario.pressbooks.pub/econ/chapter/4-2-kurtosis/# (https://ecampusontario.pressbooks.pub/econ/chapter/4-2-kurtosis/#):~:text=Kurtosis%20is%203%20for%20the,E%20K%20=%20K%20%E2%88%92%203%20.

```
n <- length(carprice$MPG.highway)
computed_mpg_hwy <- carprice |>
  select(MPG.highway) |>
  summarize(
    computed_mean = mean(MPG.highway),
    computed_median = median(MPG.highway),
    computed_std_dev = sd(MPG.highway),
    computed_skew = (n * sum((MPG.highway - computed_mean)^3)) / ((n - 1) * (n - 2) * computed_s
td_dev^3),
    computed_kurtosis = (n * sum((MPG.highway - computed_mean)^4)) / ((n - 1) * (n - 2) * comput
ed_std_dev^4)
  )
computed_mpg_hwy
```

```
## # A tibble: 1 × 5
##   computed_mean computed_median computed_std_dev computed_skew computed_kurtosis
##           <dbl>           <dbl>            <dbl>         <dbl>              <dbl>
## 1          28.1              28             4.15         0.695               4.29
```

```
n <- length(carprice$Price)
computed_price <- carprice |>
  select(Price) |>
  summarize(
    computed_mean = mean(Price),
    computed_median = median(Price),
    computed_std_dev = sd(Price),
    computed_skew = (n * sum((Price - computed_mean)^3)) / ((n - 1) * (n - 2) * computed_std_dev
^3),
    computed_kurtosis = (n * sum((Price - computed_mean)^4)) / ((n - 1) * (n - 2) * computed_std
_dev^4)
  )
computed_price
```

```
## # A tibble: 1 × 5
##   computed_mean computed_median computed_std_dev computed_skew computed_kurtosis
##           <dbl>           <dbl>            <dbl>         <dbl>              <dbl>
## 1          18.6            16.3             7.82          1.23               3.92
```

Verification of my math using the moments package:

```
library(moments)
```

```
skew_mpg <- skewness(carprice$MPG.highway)
kurtosis_mpg <- kurtosis(carprice$MPG.highway)
cat("MPG Highway Skew\t", skew_mpg, "\tMPG Highway Kurtosis\t", kurtosis_mpg, "\n")
```

```
## MPG Highway Skew  0.6727661  MPG Highway Kurtosis     4.195906
```

```
skew_price <- skewness(carprice$Price)
kurtosis_price <- kurtosis(carprice$Price)
cat("MPG Price Skew\t\t", skew_price, "\tMPG Price Kurtosis\t", kurtosis_price)
```

```
## MPG Price Skew        1.190775   MPG Price Kurtosis   3.840122
```

There is a slight difference between my output and the output from the moments package. I suspect this may be a result of the summarization. I'll try setting up the pipeline with the moments package functions to see if that corrects the issue.

```
n <- length(carprice$MPG.highway)
computed_mpg_hwy <- carprice |>
  select(MPG.highway) |>
  summarize(
    computed_mean = mean(MPG.highway),
    computed_median = median(MPG.highway),
    computed_std_dev = sd(MPG.highway),
    computed_skew = skewness(MPG.highway),
    computed_kurtosis = kurtosis(MPG.highway)
  )
computed_mpg_hwy
```

```
## # A tibble: 1 × 5
##   computed_mean computed_median computed_std_dev computed_skew computed_kurtosis
##           <dbl>           <dbl>            <dbl>         <dbl>              <dbl>
## 1          28.1              28             4.15         0.673               4.20
```

```
n <- length(carprice$Price)
computed_price <- carprice |>
  select(Price) |>
  summarize(
    computed_mean = mean(Price),
    computed_median = median(Price),
    computed_std_dev = sd(Price),
    computed_skew = skewness(Price),
    computed_kurtosis = kurtosis(Price)
  )
computed_price
```

```
## # A tibble: 1 × 5
##   computed_mean computed_median computed_std_dev computed_skew computed_kurtosis
##           <dbl>           <dbl>            <dbl>         <dbl>              <dbl>
## 1          18.6            16.3             7.82          1.19               3.84
```

Hmmmmm, still didn't fix it. I am going to blame this on rounding throughout the pipeline steps 😳. Anywho, onto the rest of the question.

Mean: The differences in mean between the two histograms affects where the "center" of the distribution falls on the x-axis. A higher mean, like that of the MPG.highway column, would place the center of the normal distribution at a higher x value.

Median: The median's relationship to the mean, i.e. how close the two values are, gives a cursory indication of how widely distributed the data set is. For example, the MPG.highway mean and median are almost equivalent, and this dataset appear to be quite normal. For Price, the median is about 12% lower than the mean, which indicates there are some large values that are skewing the mean high, which is evident in the histogram.

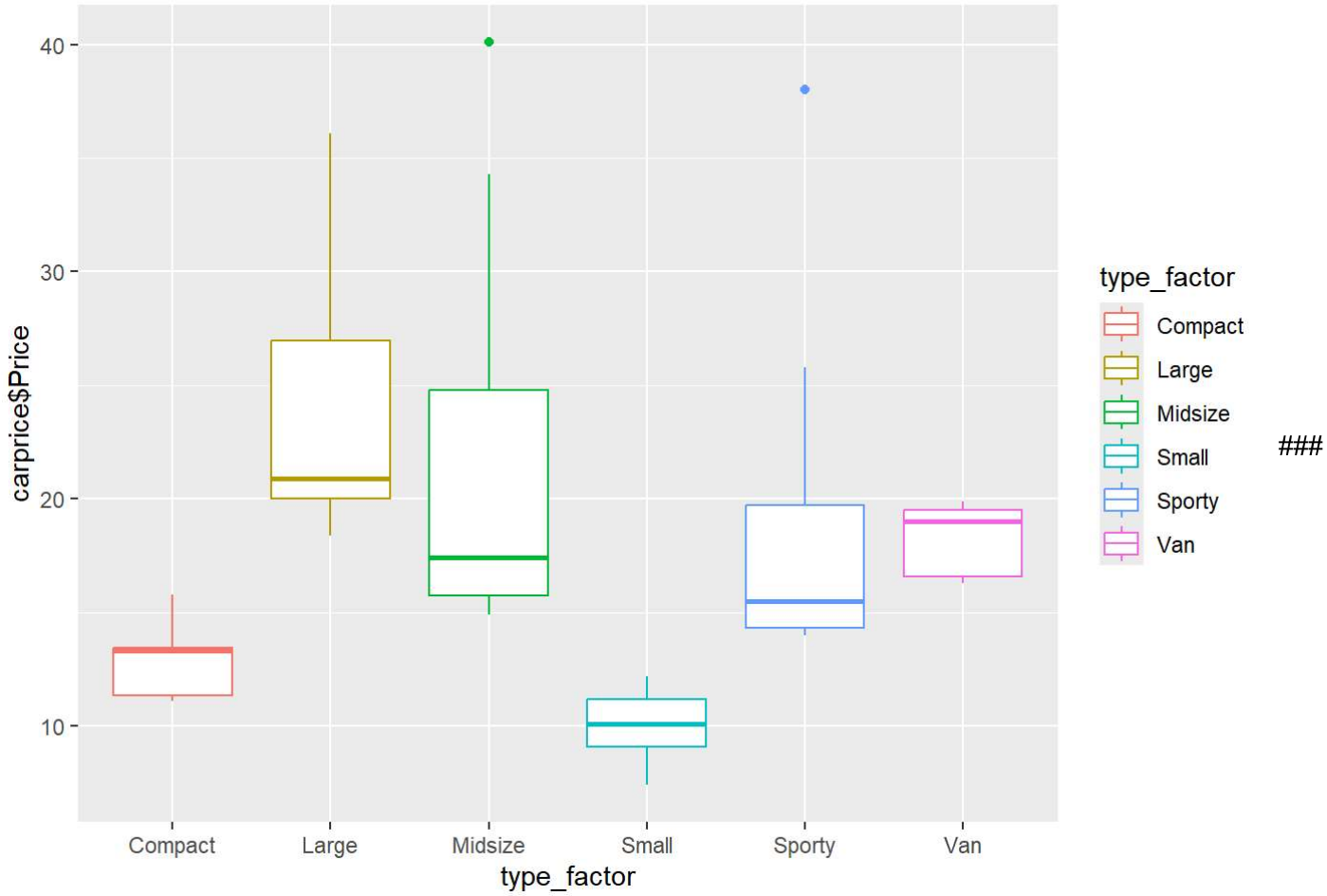
Standard Deviation: Standard deviation describes the spread of the dataset relative to the mean. As expected, the Price standard deviation is higher than the MPG.highway standard deviation, as the MPG.highway dataset is more evenly distributed about the mean.

Skewness: Skewness is a measure of the tail length/thickness relative to the mean. A positive skew indicates a tail to the right of the normal distribution, which is prominently seen in the Price dataset (skewness=1.19). The MPG.highway dataset also has a positive skew, but it is closer to 0, which would be a symmetrical distribution.

Kurtosis: Kurtosis describes how "peakey" the distribution is. Perfectly normal distributions have a kurtosis of K=3. Excess kurtosis (EK) is defined as EK = K-3, which can be used to quickly identify the shape of the distribution. Positive EK is a narrower distribution, EK = 0 is a perfect normal distribution, and negative EK is a wide distribution. Based on the EK values of MPG.highway and Price, 1.19 and 0.84, respectively, both datasets are narrower than normal, MPG.highway more than Price.

c. Create a boxplot that shows Price (not min or max price) as a function of Type (make sure Type is set to be a factor and use ggplot)

```
type_factor <- as.factor(carprice$Type)
ggplot(
  data = carprice,
  mapping = aes(x=type_factor, y=carprice$Price, color = type_factor)
) + geom_boxplot()
```



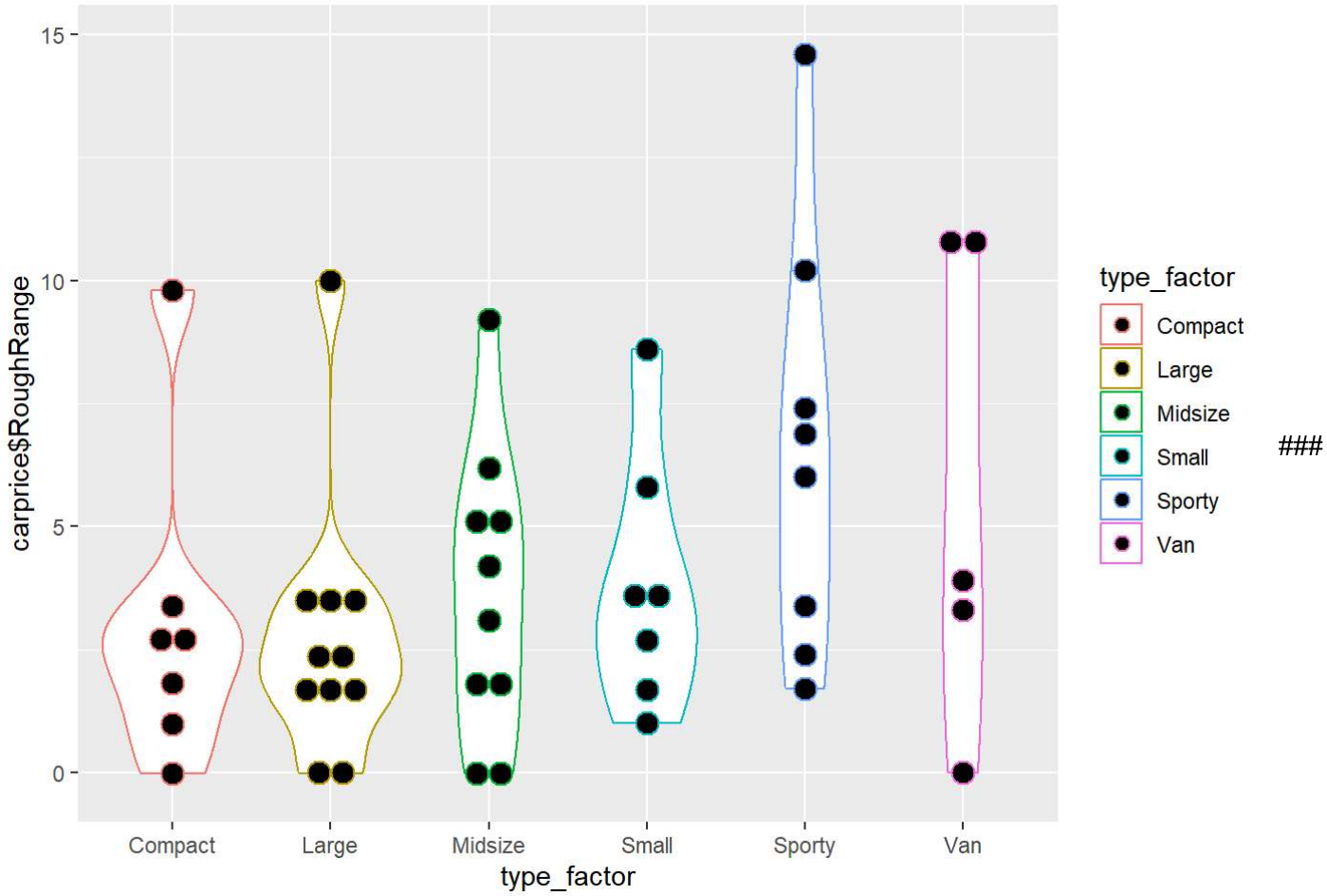
###

d. Create a violin plot that shows RoughRange of all cars. Add the dot plot on top of it

Source for dotplot: https://www.sthda.com/english/wiki/ggplot2-dot-plot-quick-start-guide-r-software-and-data-visualization (https://www.sthda.com/english/wiki/ggplot2-dot-plot-quick-start-guide-r-software-and-data-visualization)

```
type_factor <- as.factor(carprice$Type)
ggplot(
  data = carprice,
  mapping = aes(x=type_factor, y=carprice$RoughRange, color = type_factor)
) + geom_violin() + geom_dotplot(binaxis='y', stackdir='center')
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with
## `binwidth`.
```



###

e. Create a table that shows the mean Price, mean MPG.Highway and mean Rough.range by car type

```
table <- carprice |>
  group_by(Type) |>
  summarize(
    mean_price = mean(Price),
    mean_MPG_highway = mean(MPG.highway),
    mean_rough_range = mean(RoughRange)
  )
table
```

```
## # A tibble: 6 × 4
##   Type    mean_price mean_MPG_highway mean_rough_range
##   <chr>        <dbl>            <dbl>            <dbl>
## 1 Compact       12.8             30.6             3.06
## 2 Large         24.3             26.7             2.74
## 3 Midsize       21.8             27.9             3.64
## 4 Small         10.0             33.9             3.85
## 5 Sporty        19.4             27.5             6.57
## 6 Van           18.3             21.4             5.76
```

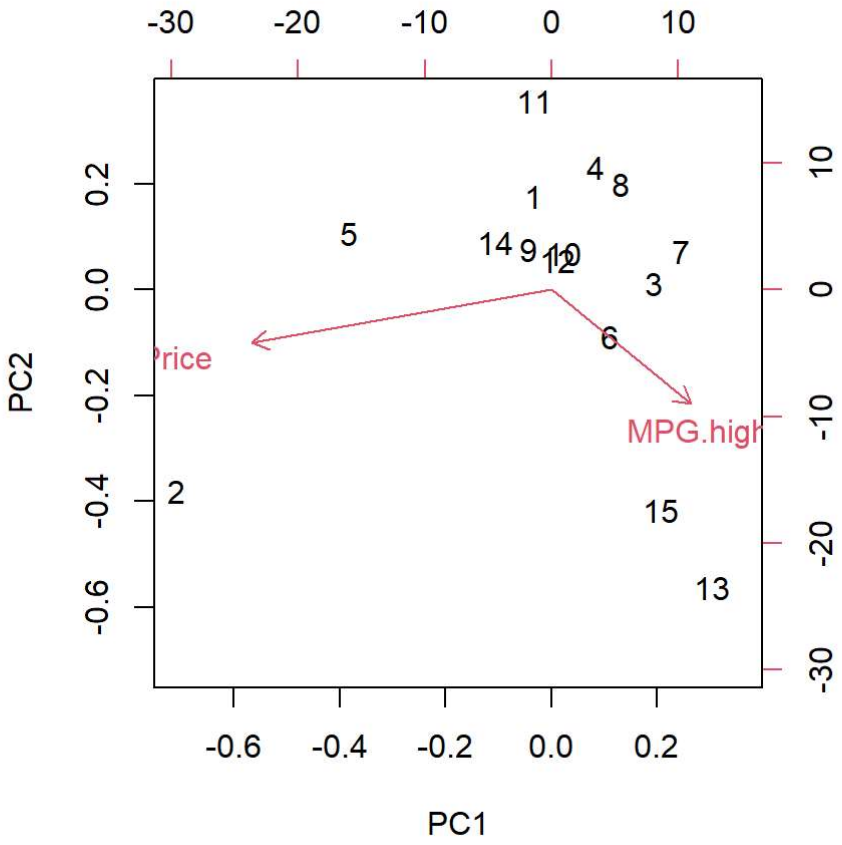
f. Plot a bi-plot (x-y graph) of Price vs MPG.highway for all cars that are listed as “small” or “sporty”

Bi-plot Source: https://www.geeksforgeeks.org/how-to-create-a-biplot-in-r/# (https://www.geeksforgeeks.org/how-to-create-a-biplot-in-r/#)

```
# Filter by type
type_filtered <- carprice |>
  mutate(type_as_factor = as.factor(Type)) |>
  filter(type_as_factor == "Small" | type_as_factor == "Sporty") |>
  select(Price, MPG.highway)

type_filtered_pca_result <- prcomp(type_filtered)
type_filtered_pca_result
```

```
## Standard deviations (1, .., p=2):
## [1] 8.430594 3.215460
##
## Rotation (n x k) = (2 x 2):
##                    PC1        PC2
## Price       -0.9068812 -0.4213863
## MPG.highway  0.4213863 -0.9068812
```

```
# Create bi-plot
biplot(type_filtered_pca_result)
```



###

g. Plot a bi-plot of Price vs MPG.highway, color coding by Type

```
library(ggfortify)
```

```
# Filter by type
pca_analysis <- carprice |>
  mutate(type_as_factor = as.factor(Type)) |>
  filter(type_as_factor == "Small" | type_as_factor == "Sporty") |>
  select(Price, MPG.highway)

pca_result <- prcomp(pca_analysis)
pca_result
```

```
## Standard deviations (1, .., p=2):
## [1] 8.430594 3.215460
##
## Rotation (n x k) = (2 x 2):
##                    PC1        PC2
## Price       -0.9068812 -0.4213863
## MPG.highway  0.4213863 -0.9068812
```

```
type_filtered <- carprice |>
  mutate(type_as_factor = as.factor(Type)) |>
  filter(type_as_factor == "Small" | type_as_factor == "Sporty") |>
  select(type_as_factor, Price, MPG.highway)

# Create a biplot using ggplot2 and ggfortify
autoplot(pca_result, data = type_filtered, colour = 'type_as_factor',
         loadings = TRUE, loadings.label = TRUE, loadings.label.size = 3)
```