Class_Exercises_ClassNotes_3

Tobias Boggess

2/14/2022

Section 4.2 Exercises

Exercise 1: Show the following

a) select(.data = flights, year, day)
Results:

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

```
##
## 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
```

```
library(ggplot2)
select(.data = flights, year, day)
```

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

My guess: Will return the data from columns year and day.

b) select(.data = flights, year:day)
Results:

```
select(.data = flights, year:day)
```

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

My Guess: This will return all columns from year to day.

c) select(.data = flights, -(year:day))
Results

```
select(.data = flights, -(year:day))
```

```
## # A tibble: 336,776 x 16
##    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
##       <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
##  1      517            515         2      830            819        11 UA
##  2      533            529         4      850            830        20 UA
##  3      542            540         2      923            850        33 AA
##  4      544            545        -1     1004           1022       -18 B6
##  5      554            600        -6      812            837       -25 DL
##  6      554            558        -4      740            728        12 UA
##  7      555            600        -5      913            854        19 B6
##  8      557            600        -3      709            723       -14 EV
##  9      557            600        -3      838            846        -8 B6
## 10      558            600        -2      753            745         8 AA
## # ... with 336,766 more rows, and 9 more variables: flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

My Guess: All columns will be shown except the ones from year to day.

Exercise 2: Guess and check.

Code:

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

a) select(.data = flights, starts_with("sched"))
Results:

```
select(.data = flights, starts_with("sched"))
```

```
## # A tibble: 336,776 x 2
##    sched_dep_time sched_arr_time
##             <int>          <int>
##  1            515            819
##  2            529            830
##  3            540            850
##  4            545           1022
##  5            600            837
##  6            558            728
##  7            600            854
##  8            600            723
##  9            600            846
## 10            600            745
## # ... with 336,766 more rows
```

My Guess: This will show columns that have a header matching "sched".

b) select(.data = flights, contains("arr"))
Results:

```
select(.data = flights, contains("arr"))
```

```
## # A tibble: 336,776 x 4
##    arr_time sched_arr_time arr_delay carrier
##       <int>          <int>     <dbl> <chr>
##  1      830            819        11 UA
##  2      850            830        20 UA
##  3      923            850        33 AA
##  4     1004           1022       -18 B6
##  5      812            837       -25 DL
##  6      740            728        12 UA
##  7      913            854        19 B6
##  8      709            723       -14 EV
##  9      838            846        -8 B6
## 10      753            745         8 AA
## # ... with 336,766 more rows
```

My Guess: This will show columns containing the string "arr".

c) select(.data = flights, starts_with("dep_"), starts_with("arr_"))
Results:

```
select(.data = flights, starts_with("dep_"), starts_with("arr_"))
```

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

My Guess: This will show all the columns that start with "dep_" or "arr_".

Section 4.4 Exercises

Exercise 3: Report R commands that use filter() and the logical operators ('&', '|', and '!') with the flights data to find all flights that:

- a) Had an arrival delay of two or more hours.
- b) Flew to Houston (IAH or HOU).
- c) Were operated by United, American, or Delta.
- d) Departed in summer (July, August, or September).
- e) Departed between midnight and 6:00 AM (inclusive).
- f) Were operated by United, departed in July, and had an arrival delay of two or more hours. Code:

```
filter(.data = flights, arr_delay >= 120)
```

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

```
filter(.data = flights, dest == "IAH" | dest == "HOU")
```

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

```
filter(.data = flights,
       carrier == "UA" |
         carrier == "AA" | carrier == "DL")
```

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

```
filter(.data = flights, month == 7 | month == 8 | month == 9)
```

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

```
filter(.data = flights, dep_time == 2400 | dep_time <= 600)
```

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

```
filter(.data = flights, carrier == "UA" &
         month == 7 & arr_delay >= 120)
```

```
## # A tibble: 233 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      900            800        60     1123            922
##  2  2013     7     1     1125            900       145     1247           1026
##  3  2013     7     1     1310           1057       133     1551           1338
##  4  2013     7     1     1411           1117       174     1651           1337
##  5  2013     7     1     1439           1155       164     1624           1359
##  6  2013     7     1     1505           1309       116     1813           1559
##  7  2013     7     1     1707           1448       139     1943           1742
##  8  2013     7     2     2005           1830        95     2327           2126
##  9  2013     7     3      931            659       152     1034            820
## 10  2013     7     3     1711           1300       251     1835           1425
## # ... with 223 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Section 4.5 Exercises

Exercise 4: Report R commands that use arrange() to sort the flights data to:

- a) Find the flights that had the shortest delays.
- b) Find the flights that had the longest delays.
- c) Find the flights that had the earliest departure times.
- d) Find the flights that had the latest departure times.
- e) Find the flights that traveled the shortest distance.
- f) Find the flights that traveled the longest distance.
  Code:

```
arrange(.data = flights, dep_delay)
```

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

```
arrange(.data = flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
arrange(.data = flights, dep_time)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1    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
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
arrange(.data = flights, desc(dep_time))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    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
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
arrange(.data = flights, distance)
```

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

```
arrange(.data = flights, desc(distance))
```

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

Exercise 5: Do the following.

Given Data Frame:

```
x <- data.frame(
  x1 = c(2, 1, NA, 8, 7, 5, 4),
  x2 = c("a", NA, "c", "d", "c", "a", "d"),
  stringsAsFactors = FALSE
)
x
```

```
##   x1   x2
## 1  2    a
## 2  1 <NA>
## 3 NA    c
## 4  8    d
## 5  7    c
## 6  5    a
## 7  4    d
```

a) Guess what the following will do.
Code:

```
arrange(.data = x, is.na(x1))
```

```
##   x1   x2
## 1  2    a
## 2  1 <NA>
## 3  8    d
## 4  7    c
## 5  5    a
## 6  4    d
## 7 NA    c
```

My Guess: This will arrange the results so na is at the bottom of the data frame.

b) Guess what the following will do.
Code:

```
arrange(.data = x, desc(is.na(x1)))
```

```
##   x1   x2
## 1 NA    c
## 2  2    a
## 3  1 <NA>
## 4  8    d
## 5  7    c
## 6  5    a
## 7  4    d
```

My Guess: This will arrange all the values that are na at the top based on the x1 column.

Section 4.6 Exercises

Exercise 6: Report an R command that uses mutate() or transmute(), with flights, to compute arr_time - dep_time, and compare it with air_time. Why are they different?

Code:

```
mutate(.data = flights, new_air_time = arr_time - dep_time)
```

```
## # 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     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 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>,
## #   new_air_time <int>
```

```
all(flights$air_time == flights$new_air_time)
```

```
## Warning: Unknown or uninitialised column: ‘new_air_time‘.
```

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

They are different and its because one uses hours and minutes and the other air_time variable is in minutes.

Exercise 7: Execute the code.

Code:

```
flights_small <- select(.data = flights,
                        year:day,
                        ends_with("delay"),
                        distance,
                        air_time)
mutate(
  .data = flights_small,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

```
## # A tibble: 336,776 x 10
##     year month   day dep_delay arr_delay distance air_time  gain hours
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227    -9 3.78
##  2  2013     1     1         4        20     1416      227   -16 3.78
##  3  2013     1     1         2        33     1089      160   -31 2.67
##  4  2013     1     1        -1       -18     1576      183    17 3.05
##  5  2013     1     1        -6       -25      762      116    19 1.93
##  6  2013     1     1        -4        12      719      150   -16 2.5
##  7  2013     1     1        -5        19     1065      158   -24 2.63
##  8  2013     1     1        -3       -14      229       53    11 0.883
##  9  2013     1     1        -3        -8      944      140     5 2.33
## 10  2013     1     1        -2         8      733      138   -10 2.3
## # ... with 336,766 more rows, and 1 more variable: gain_per_hour <dbl>
```

The gain_per_hour does seem to get computed.

Section 4.7 Exercises

Exercise 8: Use rename() to change the names of the variables in z to new_z1, new_z2, and new_z3. Report your R command(s).

Code:

```
z <- data.frame(
  z1 = c(5, 4, 3),
  z2 = c("a", "c", "b"),
  z3 = c(14, 22, 13)
)
new_z <- rename(
  .data = z,
  new_z1 = z1,
  new_z2 = z2,
  new_z3 = z3
)
z
```

```
##   z1 z2 z3
## 1  5  a 14
## 2  4  c 22
## 3  3  b 13
```

```
new_z
```

```
##   new_z1 new_z2 new_z3
## 1      5      a     14
## 2      4      c     22
## 3      3      b     13
```

Section 4.8 Exercises

Exercise 9: Create the not_cancelled data frame (using flights):

Code:

```
not_cancelled <-
  filter(.data=flights,
         !is.na(dep_delay),
         !is.na(arr_delay)
         )
```

a) Use summarize() with median() to find the median departure delay and the median arrival delay. Report the two values.
Code:

```
summarize(.data = not_cancelled,
          med_dep_delay = median(dep_delay),
          med_arr_delay = median(arr_delay))
```

```
## # A tibble: 1 x 2
##   med_dep_delay med_arr_delay
##           <dbl>         <dbl>
## 1            -2            -5
```

b) Use summarize() with max() to find the longest departure delay and the longest arrival delay.Report the two values.
Code:

```
summarize(.data = not_cancelled,
          long_dep_delay = max(dep_delay),
          long_arr_delay = max(arr_delay))
```

```
## # A tibble: 1 x 2
##   long_dep_delay long_arr_delay
##            <dbl>          <dbl>
## 1           1301           1272
```

c) Use summarize() with min() to find the shortest departure delay and the shortest arrival delay.Report the two values.
Code:

```
summarize(.data = not_cancelled,
          short_dep_delay = min(dep_delay),
          short_arr_delay = min(arr_delay))
```

```
## # A tibble: 1 x 2
##   short_dep_delay short_arr_delay
##             <dbl>           <dbl>
## 1             -43             -86
```

Exercise 10: Answer the following.

a) What does the following command do?
Code:

```
summarize(.data = not_cancelled,
          total_flights = n())
```

```
## # A tibble: 1 x 1
##   total_flights
##           <int>
## 1        327346
```

This shows the total number of flights in the flights data set.

b) What does the following command do?
Code:

```
summarize(.data = not_cancelled,
          hour_arr_delay_total = sum(arr_delay > 60))
```

```
## # A tibble: 1 x 1
##   hour_arr_delay_total
##                  <int>
## 1                27789
```

This will show the number of times the arrival delay was greater than an hour.

c) What does the following command do?
Code:

```
summarize(.data = not_cancelled,
          hour_arr_delay_proportion =
            sum(arr_delay > 60) / n()
          )
```

```
## # A tibble: 1 x 1
##   hour_arr_delay_proportion
##                       <dbl>
## 1                    0.0849
```

This will show the proportion of times a flight was delayed by more than an hour by the number of flights.

Section 4.9 Exercises

Exercise 11: Do the following.

a) Explain in words what the following commands do (recall that dest is the destination of the flight):
Code:

```
by_dest <- group_by(.data = flights, dest)
delay_by_dest <- summarize(.data = by_dest,
                           mean_arr_delay = mean(arr_delay, na.rm = TRUE))
delay_by_dest
```

```
## # A tibble: 105 x 2
##    dest  mean_arr_delay
##    <chr>          <dbl>
##  1 ABQ             4.38
##  2 ACK             4.85
##  3 ALB            14.4
##  4 ANC            -2.5
##  5 ATL            11.3
##  6 AUS             6.02
##  7 AVL             8.00
##  8 BDL             7.05
##  9 BGR             8.03
## 10 BHM            16.9
## # ... with 95 more rows
```

The above code calculates the mean arrival delay of every destination airport flown to from New York.

b) summarize() can summarize more than one variable at a time. Explain in words what the following commands do:
Code:

```
by_dest <- group_by(.data = flights, dest)
delay_dist_by_dest <- summarize(
  .data = by_dest,
  mean_dist = mean(distance, na.rm = TRUE),
  mean_arr_delay = mean(arr_delay, na.rm = TRUE)
)
delay_dist_by_dest
```

```
## # A tibble: 105 x 3
##    dest  mean_dist mean_arr_delay
##    <chr>     <dbl>          <dbl>
##  1 ABQ       1826            4.38
##  2 ACK        199            4.85
##  3 ALB        143           14.4
##  4 ANC       3370           -2.5
##  5 ATL        757.          11.3
##  6 AUS       1514.           6.02
##  7 AVL        584.           8.00
##  8 BDL        116            7.05
##  9 BGR        378            8.03
## 10 BHM        866.          16.9
## # ... with 95 more rows
```

Summarize can do more than one variable at a time. The output of them is the mean of the distance traveled between a New York airport and the destination airport. The second variable is the mean distribution of the arrival delay at each airport.

Exercise 12: Do the following.

Code:

```
resp <- c(23, 11, 14, 16, 19, 26, 24, 29, 31, 28, 34, 25)
trt <- c(rep("Ctrl", 4), rep("TrtA", 4), rep("TrtB", 4))
age <- c(33, 45, 30, 24, 22, 31, 39, 40, 29, 19, 27, 25)
gndr <- c("m", "m", "f", "f", "m", "f", "f", "m", "f", "m", "f", "m")
ExpData <- data.frame(
  TrtGrp = trt,
  SubjectGender = gndr,
  SubjectAge = age,
  Response = resp,
  stringsAsFactors = FALSE
)
```

a) Explain in words what the following commands do:
Code:

```
by_TrtGrp <- group_by(.data = ExpData, TrtGrp)
summarize(.data = by_TrtGrp, Count = n())
```

```
## # A tibble: 3 x 2
##   TrtGrp Count
##   <chr>  <int>
## 1 Ctrl       4
## 2 TrtA       4
## 3 TrtB       4
```

The command will show how many of each group there are.

b) Use group_by() and summarize() to compute the mean Response by TrtGrp. Report the three mean Response values.
Code:

```
TrtGroupBy <- group_by(.data = ExpData, TrtGrp)
summarize(.data = TrtGroupBy, meanResponse = mean(Response, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   TrtGrp meanResponse
##   <chr>         <dbl>
## 1 Ctrl           16
## 2 TrtA           24.5
## 3 TrtB           29.5
```

c) Now use group_by() and summarize() to compute the mean Response and mean SubjectAge by TrtGrp. Report the three mean Response values and the three mean SubjectAge values.
Code:

```
TrtGroupBy <- group_by(.data = ExpData, TrtGrp)
summarize(
  .data = TrtGroupBy,
  meanResponse =
    mean(Response, na.rm = TRUE),
  meanSubjectAge = mean(SubjectAge, na.rm = TRUE)
)
```

```
## # A tibble: 3 x 3
##   TrtGrp meanResponse meanSubjectAge
##   <chr>         <dbl>          <dbl>
## 1 Ctrl           16               33
## 2 TrtA           24.5             33
## 3 TrtB           29.5             25
```

Exercise 13: Do the following with the flights data set.

a) Use group_by(), summarize() with n(), and arrange() to determine which tailnum (i.e. which individual airplane) flew the most times. Report the tailnum value of the airplane.
Code:

```
tailNumGroup <- group_by(.data = flights, tailnum)
mostTailNum <- summarize(.data = tailNumGroup, Count = n())
arrange(.data = mostTailNum, desc(Count))
```

```
## # A tibble: 4,044 x 2
##    tailnum Count
##    <chr>   <int>
##  1 <NA>     2512
##  2 N725MQ    575
##  3 N722MQ    513
##  4 N723MQ    507
##  5 N711MQ    486
##  6 N713MQ    483
##  7 N258JB    427
##  8 N298JB    407
##  9 N353JB    404
## 10 N351JB    402
## # ... with 4,034 more rows
```

b) Use group_by(), summarize() with n(), and arrange() to determine which dest (i.e. which destination) was flown to the most times. Report the (abbreviated) destination name.
Code:

```
destGroup <- group_by(.data = flights, dest)
mostDestGroup <- summarize(.data = destGroup, Count = n())
arrange(.data = mostDestGroup, desc(Count))
```

```
## # A tibble: 105 x 2
##    dest  Count
##    <chr> <int>
##  1 ORD   17283
##  2 ATL   17215
##  3 LAX   16174
##  4 BOS   15508
##  5 MCO   14082
##  6 CLT   14064
##  7 SFO   13331
##  8 FLL   12055
##  9 MIA   11728
## 10 DCA    9705
## # ... with 95 more rows
```

Section 4.10 Exercises

Exercise 14: Do the following with the command below.

Code:

```
x <- c(2, 5, 4, 3, 7, 9)
```

a) What does the following command do?
Code:

```
x %>% mean()
```

```
## [1] 5
```

```
mean(x)
```

```
## [1] 5
```

This is a way to apply x to the mean function.

b) What does the following command do?
Code:

```
x %>% mean() %>% sqrt() %>% round(digits = 2)
```

```
## [1] 2.24
```

```
round(sqrt(mean(x)), digits = 2)
```

```
## [1] 2.24
```

This will pass x into the mean() function and from there the result will be passed to the square root function and the result of that will be passed to the round function.

c) Rewrite the following sequence of commands using the pipe operator.
Original Code:

```
mean_x <- mean(x)
sqrt_mean_x <- sqrt(mean_x)
round_sqrt_mean_x <- round(sqrt_mean_x, digits = 2)
round_sqrt_mean_x
```

```
## [1] 2.24
```

Modified Code:

```
x %>% mean() %>% sqrt() %>% round(digits = 2)
```

```
## [1] 2.24
```

d) Rewrite the following command using the pipe operator. Report your R command(s).
Original Code:

```
round(sqrt(mean(x)), digits = 2)
```

```
## [1] 2.24
```

Modified Code:

```
x %>% mean() %>% sqrt() %>% round(digits = 2)
```

```
## [1] 2.24
```

Exercise 15: Do the following.

a) Rewrite the following command using the pipe operator.
Command:

```
delay <- select(.data = flights, arr_delay)
```

Pipe Operator version:

```
flights %>% select(arr_delay)
```

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

b) Rewrite the following command using the pipe operator.
Original:

```
dest_delay <- select(.data = flights, dest, arr_delay)
```

Pipe Operator version:

```
flights %>% select(dest, arr_delay)
```

```
## # A tibble: 336,776 x 2
##    dest  arr_delay
##    <chr>     <dbl>
##  1 IAH          11
##  2 IAH          20
##  3 MIA          33
##  4 BQN         -18
##  5 ATL         -25
##  6 ORD          12
##  7 FLL          19
##  8 IAD         -14
##  9 MCO          -8
## 10 ORD           8
## # ... with 336,766 more rows
```

c) Rewrite the following pair of commands using the pipe operator.
Original:

```
dest_delay <- select(.data = flights, dest, arr_delay)
sea_den <- filter(.data = dest_delay,
                  dest == "SEA" | dest == "DEN")
sea_den
```

```
## # A tibble: 11,189 x 2
##    dest  arr_delay
##    <chr>     <dbl>
##  1 DEN          -6
##  2 SEA         -10
##  3 DEN           7
##  4 SEA           3
##  5 DEN          -4
##  6 DEN          33
##  7 DEN           7
##  8 DEN          32
##  9 SEA         -25
## 10 DEN           9
## # ... with 11,179 more rows
```

Pipe Operator version:

```
flights %>% select(dest, arr_delay) %>% filter(dest == "SEA" | dest == "DEN")
```

```
## # A tibble: 11,189 x 2
##    dest  arr_delay
##    <chr>     <dbl>
##  1 DEN          -6
##  2 SEA         -10
##  3 DEN           7
##  4 SEA           3
##  5 DEN          -4
##  6 DEN          33
##  7 DEN           7
##  8 DEN          32
##  9 SEA         -25
## 10 DEN           9
## # ... with 11,179 more rows
```

d) Rewrite the following sequence of commands using the pipe operator.
Original:

```
dest_delay <- select(.data = flights, dest, arr_delay)
sea_den <- filter(.data = dest_delay,
                  dest == "SEA" | dest == "DEN")
by_dest <- group_by(sea_den, dest)
delay_by_dest <- summarize(by_dest,
                           mean_arr_delay = mean(arr_delay, na.rm = TRUE))
delay_by_dest
```

```
## # A tibble: 2 x 2
##   dest  mean_arr_delay
##   <chr>          <dbl>
## 1 DEN             8.61
## 2 SEA            -1.10
```

Pipe Operator Version:

```
flights %>% select(dest, arr_delay) %>% filter(dest == "SEA" |
                                                 dest == "DEN") %>% group_by(dest) %>%     summarize(mea
```

```
## # A tibble: 2 x 2
##   dest  mean_arr_delay
##   <chr>          <dbl>
## 1 DEN             8.61
## 2 SEA            -1.10
```

Exercise 16: Rewrite the following command using the pipe operator.

Original Code:

```
den_delays <- summarize(
  filter(.data = flights,
         dest == "DEN"),
  mean_dep_delay = mean(dep_delay, na.rm = TRUE),
  mean_arr_delay = mean(arr_delay, na.rm = TRUE)
)
den_delays
```

```
## # A tibble: 1 x 2
##   mean_dep_delay mean_arr_delay
##            <dbl>          <dbl>
## 1           15.2           8.61
```

Revised Code:

```
flights %>% filter(dest == "DEN") %>% summarize(
  mean_dep_delay = mean(dep_delay, na.rm = TRUE),
  mean_arr_delay = mean(arr_delay, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 2
##   mean_dep_delay mean_arr_delay
##            <dbl>          <dbl>
## 1           15.2           8.61
```

Section 4.11 Exercises

Exercise 17: Do the following with the given data frames.

Data frames:

```
df1 <- data.frame(Respondent_ID = c(1001, 1002, 1003),
                  Q1_Response = c(55, 62, 39))
df1
```

```
##   Respondent_ID Q1_Response
## 1          1001          55
## 2          1002          62
## 3          1003          39
```

```
df2 <- data.frame(
  Respondent_ID = c(1002, 1003, 1004),
  Q2_Response = c("yes", "no", "yes")
)
df2
```

```
##   Respondent_ID Q2_Response
## 1          1002         yes
## 2          1003          no
## 3          1004         yes
```

a) Guess what the result of the following command will be, then check your answer and report the result.
Code:

```
inner_join(x = df1, y = df2, by = "Respondent_ID")
```

```
##   Respondent_ID Q1_Response Q2_Response
## 1          1002          62         yes
## 2          1003          39          no
```

This code will display the matching rows in both data frames based on the Respondent ID.

b) Guess what the result of the following command will be, then check your answer and report the result.
Code:

```
left_join(x = df1, y = df2, by = "Respondent_ID")
```

```
##   Respondent_ID Q1_Response Q2_Response
## 1          1001          55        <NA>
## 2          1002          62         yes
## 3          1003          39          no
```

This will show all the rows from the df1 data frame and include information about the Q2_response for al variables in df1.

c) Guess what the result of the following command will be, then check your answer and report the result.
Code:

```
full_join(x = df1, y = df2, by = "Respondent_ID")
```

```
##   Respondent_ID Q1_Response Q2_Response
## 1          1001          55        <NA>
## 2          1002          62         yes
## 3          1003          39          no
## 4          1004          NA         yes
```

This code will show all rows for both df1 and df2 even if df1 doesn't have anything for the column containing df2.

d) If we didn't specify by = "Respondent_ID", by default what key variable would each of the *_join() functions use to match rows?
Code:

```
full_join(x = df1, y = df2)
```

```
## Joining, by = "Respondent_ID"
```

```
##   Respondent_ID Q1_Response Q2_Response
## 1          1001          55        <NA>
## 2          1002          62         yes
## 3          1003          39          no
## 4          1004          NA         yes
```

The default is the first column of df1.

e) What would happen if Q1_Response and Q2_Response were both named Response in the two data frames, e.g.
Code:

```
#df1 <- rename(.data = df1, Response = Q1_Response)
#df2 <- rename(.data = df2, Response = Q2_Response)
#full_join(x = df1, y = df2)
```

This will result in an error "Joining, by = c("Respondent_ID", "Response") Error in `full_join()`: ! Can't join on `x$Response` x `y$Response` because of incompatible types. i `x$Response` is of type >. i `y$Response` is of type >. Backtrace: 1. dplyr::full_join(x = df1, y = df2) 2. dplyr:::full_join.data.frame(x = df1, y = df2) Error in full_join(x = df1, y = df2) : i `x$Response` is of type >. i `y$Response` is of type >.".

f) What would happen if, as in part e, Q1_Response and Q2_Response were both named Response, and we typed:
Code:

```
#df1 <- rename(.data = df1, Response = Q1_Response)
#df2 <- rename(.data = df2, Response = Q2_Response)
#inner_join(x = df1, y = df2)
```

This reports an error, "Error in `stop_subscript()`: ! Can't rename columns that don't exist. x Column `Q1_Response` doesn't exist. Backtrace: 1. dplyr::rename(.data = df1, Response = Q1_Response) 2. dplyr:::rename.data.frame(.data = df1, Response = Q1_Response) 3. tidyselect::eval_rename(expr(c(...)), .data) 4. tidyselect:::rename_impl(...) 5. tidyselect:::eval_select_impl(...) ... 20. tidyselect:::chr_as_locations(x, vars) 21. vctrs::vec_as_location(x, n = length(vars), names = vars) 22. vctrs `<fn>`() 23. vctrs:::stop_subscript_oob(...) 24. vctrs:::stop_subscript(...) Error in stop_subscript(class ="vctrs_error_subscript_oob", i = i, subscript_type = subscript_type, :".

Exercise 18: Do the following with the given data frames.

Data Frames:

```
df1 <- data.frame(Respondent_ID = c(1000, 1001, 1002, 1003, 1004, 1005, 1006),
Q1_Response = c(55, 62, 39, 45, 70, 77, 56))
df1
```

```
##   Respondent_ID Q1_Response
## 1          1000          55
## 2          1001          62
## 3          1002          39
## 4          1003          45
## 5          1004          70
## 6          1005          77
## 7          1006          56
```

```
df2 <- data.frame(Respondent_ID = c(1003, 1002, 1000, 1004, 1006, 1001, 1005),
Q2_Response = c(12, 17, 23, 24, 19, 30, 20))
df2
```

```
##   Respondent_ID Q2_Response
## 1          1003          12
## 2          1002          17
## 3          1000          23
## 4          1004          24
## 5          1006          19
## 6          1001          30
## 7          1005          20
```

a) What happens to the ordering of the rows of df2 when you combine it with df1 using:
Code:

```
inner_join(x = df1, y = df2, by = "Respondent_ID")
```

```
##   Respondent_ID Q1_Response Q2_Response
## 1          1000          55          23
## 2          1001          62          30
## 3          1002          39          17
## 4          1003          45          12
## 5          1004          70          24
## 6          1005          77          20
## 7          1006          56          19
```

The ordering will be based on the Respondent ID in df1.

b) How would the result differ if you swapped the roles of df1 and df2, e.g.:
Code:

```
inner_join(x = df2, y = df1, by = "Respondent_ID")
```

```
##   Respondent_ID Q2_Response Q1_Response
## 1          1003          12          45
## 2          1002          17          39
## 3          1000          23          55
## 4          1004          24          70
## 5          1006          19          56
## 6          1001          30          62
## 7          1005          20          77
```

The code changes so the Respondent ID will be based on df2 instead of df1.

Exercise 19: Do the following with the given data frames.

Data frames:

```
dfX <- data.frame(
  LastName = c("Smith", "Smith", "Jones", "Smith",
               "Olsen", "Taylor", "Olsen"),
  FirstName = c("John", "Kim", "John", "Marge", "Bill",
                "Bill", "Erin"),
  Gender = c("M", "F", "M", "F", "M", "M", "F"),
  ExamScore = c(75, 80, 64, 78, 90, 89, 79)
)
dfX
```

```
##   LastName FirstName Gender ExamScore
## 1    Smith      John      M        75
## 2    Smith       Kim      F        80
## 3    Jones      John      M        64
## 4    Smith     Marge      F        78
## 5    Olsen      Bill      M        90
## 6   Taylor      Bill      M        89
## 7    Olsen      Erin      F        79
```

```
dfY <- data.frame(
  LastName = c("Olsen", "Jones", "Taylor", "Smith",
               "Olsen", "Smith", "Smith"),
  FirstName = c("Bill", "John", "Bill", "Kim", "Erin",
                "John", "Marge"),
  Gender = c("M", "M", "M", "F", "F", "M", "F"),
  Grade = c("A", "D", "B", "B", "C", "C", "C")
)
dfY
```

```
##   LastName FirstName Gender Grade
## 1    Olsen      Bill      M     A
## 2    Jones      John      M     D
## 3   Taylor      Bill      M     B
## 4    Smith       Kim      F     B
## 5    Olsen      Erin      F     C
## 6    Smith      John      M     C
## 7    Smith     Marge      F     C
```

a) Write a command involving, say, full_join() that combines the two data frames by person.
Code:

```
full_join(x = dfX, y = dfY, by = c("LastName", "FirstName"))
```

```
##   LastName FirstName Gender.x ExamScore Gender.y Grade
## 1    Smith      John        M        75        M     C
## 2    Smith       Kim        F        80        F     B
## 3    Jones      John        M        64        M     D
## 4    Smith     Marge        F        78        F     C
## 5    Olsen      Bill        M        90        M     A
## 6   Taylor      Bill        M        89        M     B
## 7    Olsen      Erin        F        79        F     C
```

b) What happens with the third variable (Gender) when you only specify the other two (LastName and FirstName) as the key via the by argument?
Code:

```
full_join(x = dfX, y = dfY, by = c("LastName", "FirstName", "Gender"))
```

```
##   LastName FirstName Gender ExamScore Grade
## 1    Smith      John      M        75     C
## 2    Smith       Kim      F        80     B
## 3    Jones      John      M        64     D
## 4    Smith     Marge      F        78     C
## 5    Olsen      Bill      M        90     A
## 6   Taylor      Bill      M        89     B
## 7    Olsen      Erin      F        79     C
```

```
full_join(x = dfX, y = dfY)
```

```
## Joining, by = c("LastName", "FirstName", "Gender")
```

```
##   LastName FirstName Gender ExamScore Grade
## 1    Smith      John      M        75     C
## 2    Smith       Kim      F        80     B
## 3    Jones      John      M        64     D
## 4    Smith     Marge      F        78     C
## 5    Olsen      Bill      M        90     A
## 6   Taylor      Bill      M        89     B
## 7    Olsen      Erin      F        79     C
```

```
full_join(x = dfX, y = dfY, by = c("LastName", "FirstName"))
```

```
##   LastName FirstName Gender.x ExamScore Gender.y Grade
## 1    Smith      John        M        75        M     C
## 2    Smith       Kim        F        80        F     B
## 3    Jones      John        M        64        M     D
## 4    Smith     Marge        F        78        F     C
## 5    Olsen      Bill        M        90        M     A
## 6   Taylor      Bill        M        89        M     B
## 7    Olsen      Erin        F        79        F     C
```

c) What would happen if you tried to combine dfX and dfY only specifying LastName as the key variable?
Code:

```
full_join(x = dfX, y = dfY, by = "LastName")
```

```
##    LastName FirstName.x Gender.x ExamScore FirstName.y Gender.y Grade
## 1     Smith        John        M        75         Kim        F     B
## 2     Smith        John        M        75        John        M     C
## 3     Smith        John        M        75       Marge        F     C
## 4     Smith         Kim        F        80         Kim        F     B
## 5     Smith         Kim        F        80        John        M     C
## 6     Smith         Kim        F        80       Marge        F     C
## 7     Jones        John        M        64        John        M     D
## 8     Smith       Marge        F        78         Kim        F     B
## 9     Smith       Marge        F        78        John        M     C
## 10    Smith       Marge        F        78       Marge        F     C
## 11    Olsen        Bill        M        90        Bill        M     A
## 12    Olsen        Bill        M        90        Erin        F     C
## 13   Taylor        Bill        M        89        Bill        M     B
## 14    Olsen        Erin        F        79        Bill        M     A
## 15    Olsen        Erin        F        79        Erin        F     C
```