Data Science Module 3 Exercises

Brady Lamson

2/16/2022

4: Data Wrangling

4.2:

Extracting Columns with select()

Exercise 1

a) This will pull only the years and days from the flights data set.

b) This pulls all the columns between year and day, inclusive.

c) This pulls all columns *except* year and day.

Exercise 2:

a) This will pull all column names starting with "sched", of which there are 2 columns in this data set. (sched_dep_time and sched_arr_time)

b) Same as (a) but with columns starting with "arr" (arr_time and arr_delay).

c) Pulls all columns starting either with "dep_" or "arr_".

4.4: Filtering Rows with filter()

Exercise 3:

```
# A
dplyr::filter(flights, arr_delay >= 120)
```

```
# B
flights %>%
    dplyr::filter(dest %in% c("IAH", "HOU"))
```

```
# C
flights %>%
    dplyr::filter(carrier %in% c("UA", "AA", "DL"))
```

```
# D
flights %>%
    dplyr::filter(month %in% 7:9)
```

```
# E
flights %>%
    dplyr::filter(dep_time %in% 0:600)
```

```
# F
flights %>%
    dplyr::filter(
        carrier == "UA",
        month == 7,
        arr_delay >= 120
    )
```

4.5: Arranging Rows with arrange()

Exercise 4:

```
# A
flights %>%
    dplyr::arrange(dep_delay, arr_delay)
```

```
# B
flights %>%
    dplyr::arrange(
        dplyr::desc(dep_delay),
        dplyr::desc(arr_delay)
    )
```

```
# C
flights %>%
    dplyr::arrange(dep_time)
```

```
# D
flights %>%
    dplyr::arrange(dplyr::desc(dep_time))
```

```
# E
flights %>%
    dplyr::arrange(distance)
```

```
# F
flights %>%
    dplyr::arrange(dplyr::desc(distance))
```

Exercise 5:

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

The following code will sort the data with NAs at the top of the *first* column.

```
x %>%
    dplyr::arrange(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
```

The following code will sort the data with NAs at the bottom of the *first* column.

```
x %>%
    dplyr::arrange(
        dplyr::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
```

4.6: Create New Variables (Columns) with mutate()

Exercise 6:

```
flights %>%
    dplyr::mutate(travel_time = arr_time - dep_time)
```

The arrival and departure time use totally different units than air time. The former are units of time, so 600 is 6:00am, whereas with air time 600 would be 600 minutes. They aren't equivalent.

Exercise 7:

```
flights_small <- select(
    .data = flights,
    year:day,
    ends_with("delay"),
    distance,
    air_time)
```

```
dplyr::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 variable gain_per_hour does get computed!

4.7: Renaming Variables (Columns) with rename()

Exercise 8:

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

```
z %>%
    dplyr::rename(
        new_z1 = z1,
        new_z2 = z2,
        new_z3 = z3
    )
```

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

4.8: Summarize Data with summarize()

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

Exercise 9:

```
#A

not_cancelled %>%
    dplyr::summarise(
        median_dep_delay = median(dep_delay),
        median_arr_delay = median(arr_delay)
    )
```

```
## # A tibble: 1 x 2
##   median_dep_delay median_arr_delay
##              <dbl>            <dbl>
## 1               -2               -5
```

```
# B
not_cancelled %>%
    dplyr::summarise(
        max_dep_delay = max(dep_delay),
        max_arr_delay = max(arr_delay)
    )
```

```
## # A tibble: 1 x 2
##   max_dep_delay max_arr_delay
##           <dbl>         <dbl>
## 1          1301          1272
```

```
not_cancelled %>%
    dplyr::summarise(
        shortest_dep_delay = min(dep_delay),
        shortest_arr_delay = min(arr_delay)
    )
```

```
## # A tibble: 1 x 2
##   shortest_dep_delay shortest_arr_delay
##                <dbl>              <dbl>
## 1                -43                -86
```

Exercise 10:

a) The following code will give you the total count of rows in the data set.

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

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

b) This counts the *number* of flights that have a delay of an hour or more.

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

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

c) This returns the ratio of flights with a delay of an hour or more compared to the total number of flights in the data set.

```
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
```

4.9: Applying summarize() to Groups using group_by()

Exercise 11:

```
# A

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 gives a breakdown of the average delay by the location of travel.

```
# B
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)
)
```

Exercise 12:

```
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

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
```

This command presents the total count of each treatment group. So we have 4 observations in the control group, 4 in treatment A and 4 in treatment B.

```
# B

dplyr::summarize(.data = by_TrtGrp, mean = base::mean(Response))
```

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

```
# C

dplyr::summarise(
    by_TrtGrp,
    mean_response = base::mean(Response),
    mean_subject_age = base::mean(SubjectAge)
)
```

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

Exercise 13:

```
# A

plane_count <- flights %>%
    dplyr::group_by(tailnum) %>%
    dplyr::summarise(count = n()) %>%
    dplyr::arrange(dplyr::desc(count)) %>%
    stats::na.omit()

glue::glue("
    The tail number with the most flights is {plane_count[1,1]}.
")
```

```
## The tail number with the most flights is N725MQ.
```

```
# B

dest_count <- flights %>%
    dplyr::group_by(dest) %>%
    dplyr::summarise(count = n()) %>%
    dplyr::arrange(dplyr::desc(count)) %>%
    stats::na.omit()

glue::glue("
    The destination visited the most times was {dest_count[1,1]}.
")
```

```
## The destination visited the most times was ORD.
```

4.10: Chaining Together Actions Using the Pipe Operator %>%

Exercise 14:

```
# A

x <- c(2, 5, 4, 3, 7, 9)
# x %>% mean()
```

The above command takes the mean of the vector x. It's equivalent to mean(x)

```
# B

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

Take the mean of x, then the square root, and then round that output to 2 digits.

c) Rewrite the command below.

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

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

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

d) Rewrite the command below.

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

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

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

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

Exercise 15

Rewrite the given commands using pipes.

```
# A

flights %>%
    dplyr::select(arr_delay) %>%
    head()
```

```
## # A tibble: 6 x 1
##   arr_delay
##       <dbl>
## 1        11
## 2        20
## 3        33
## 4       -18
## 5       -25
## 6        12
```

```
# B

flights %>%
    dplyr::select(dest, arr_delay) %>%
    head()
```

```
## # A tibble: 6 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
```

```
# C

flights %>%
    dplyr::select(dest, arr_delay) %>%
    dplyr::filter(dest %in% c("SEA", "DEN")) %>%
    head()
```

```
## # A tibble: 6 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
```

```
# D

flights %>%
    dplyr::select(dest, arr_delay) %>%
    dplyr::filter(dest %in% c("SEA", "DEN")) %>%
    dplyr::group_by(dest) %>%
    dplyr::summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE)) %>%
    head()
```

```
## # 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 pipes.

```
flights %>%
    dplyr::filter(dest == "DEN") %>%
    dplyr::summarise(
        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
```

4.11: Combining Multiple Data Frames

Exercise 17:

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

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

Guess the results of the following commands.

a) This will return a data set with *only* ID 1002 and 1003, with Q1 and Q2 responses for each.

```
# A

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

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

b) This will give Q2 responses to 1002 and 1003, but 1001's Q2 response will be NA.

```
# B

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
```

c) This will give a data set with all IDs in either data set. The Q1 and Q2 responses will either be filled in if they exist or left NA if they do not.

```
# C

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
```

d) I'm guessing it'd default to Respondent ID, because that column is in both data frames. So this would be just like (c).

```
# D

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
```

e) This would create a single Response column for the joined data frame, this one with both numbered Q1 responses AND yes/no responses from Q2. This seems like a bad idea.

update: It actually returned an error. This makes sense in retrospect as the values in a column (a vector) must all be the same type.

```
# E

df1 <- rename(.data = df1, Response = Q1_Response)
df2 <- rename(.data = df2, Response = Q2_Response)

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

f) This would either give the same error as last time or do nothing.

update: My guess was correct, it gave the same error as in part (e).

```
# F

# inner_join(x = df1, y = df2)
```

Exercise 18:

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

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

```
# A

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
```

```
# B

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
```

a) Thankfully it's smart enough to pick the values that match, so the responses are allocated accordingly.

b) This shows how the merged data frames are ordered, and that's by the data frame assigned to x.

Exercise 19

```
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))

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"))
```

```
# A

full_join(dfX, 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
```

```
# B

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
```

This creates two separate gender columns, one from the x data frame and one from the y data frame.

```r
# C

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
```

This creates an abomination of a data frame, having separate first name and gender columns! It essentially renders the data useless.