Assignment 6

Veronika Palotai

2019 10 22

Exercise 0: Manage your time. If you spend too much time on the assignment, it may be too long, and you may benefit from skipping an exercise. Since this lesson is lost on most people, exercise 0 requires you to skip one of the other exercises this week. That is, write down which exercise you want to skip and why ("saves the most of time because it is the hardest" or "it's the most useless as you are quite confident in material"). Do not work on it. Of course, if you skip any additional exercise due to lack of time, highlight this, but that will cost some part of the grade, whereas skipping an exercise as part of exercise 0 will not.

Answer: I chose to skip `Exercise 4` due to lack of time.

Exercise 1: Using the nycflights13 data. Note that it also contains a tibble called `airports` (as well as others). Use these two dataframes to find the answer to 3 of the following, and print them out in a separate chunk (i.e. the chunk should print the tibble, thus showing the first 10 lines of each):

Answer:

First of all, let's load the necessary libraries

```
library(tidyverse)
library(ggplot2)
library(nycflights13)
library(dplyr)
```

The number of flights (in the whole year) to each destination

```
flights %>%
  group_by(year,dest) %>%
  summarise(
    count = n()
  )
```

```
## # A tibble: 105 x 3
## # Groups:   year [1]
##     year dest  count
##    <int> <chr> <int>
##  1  2013 ABQ     254
##  2  2013 ACK     265
##  3  2013 ALB     439
##  4  2013 ANC       8
##  5  2013 ATL   17215
##  6  2013 AUS    2439
##  7  2013 AVL     275
##  8  2013 BDL     443
##  9  2013 BGR     375
## 10  2013 BHM     297
## # ... with 95 more rows
```

The number and list of distinct airports in the US

- The number of distinct airports in the US:

```
airports %>%
  summarise(
    nr_of_airports = n_distinct(name, na.rm = TRUE)
  )
```

```
## # A tibble: 1 x 1
##   nr_of_airports
##            <int>
## 1           1440
```

- The list of distinct airports in the US along with how many times they are featured in the `airports` tibble:

```
airports %>%
  group_by(name) %>%
  summarise(
    nr_of_times_featured = n_distinct(name, na.rm = TRUE)
  )
```

```
## # A tibble: 1,440 x 2
##    name                        nr_of_times_featured
##    <chr>                                      <int>
##  1 Aberdeen Regional Airport                      1
##  2 Abilene Rgnl                                   1
##  3 Abraham Lincoln Capital                        1
##  4 Acadiana Rgnl                                  1
##  5 Adak Airport                                   1
##  6 Adams Fld                                      1
##  7 Addison                                        1
##  8 Adirondack Regional Airport                    1
##  9 Akhiok Airport                                 1
## 10 Akiak Airport                                  1
## # ... with 1,430 more rows
```

The number and list of distinct airports that have at least one flight in the whole year from NYC

- The number of distinct airports that have at least one flight in the whole year from NYC

```
flights %>%
  filter(origin == 'JFK') %>%
  summarise(
    nr_of_times_featured = n_distinct(dest, na.rm = TRUE)
  )
```

```
## # A tibble: 1 x 1
##   nr_of_times_featured
##                  <int>
## 1                   70
```

- The list of distinct airports that have at least one flight in the whole year from NYC

```
flights %>%
  group_by(dest) %>%
  filter(origin == 'JFK') %>%
  summarise(
    nr_of_times_featured = n_distinct(dest, na.rm = TRUE)
  )
```

```
## # A tibble: 70 x 2
##    dest  nr_of_times_featured
##    <chr>                 <int>
##  1 ABQ                       1
##  2 ACK                       1
##  3 ATL                       1
##  4 AUS                       1
##  5 BHM                       1
##  6 BNA                       1
##  7 BOS                       1
##  8 BQN                       1
##  9 BTV                       1
## 10 BUF                       1
## # ... with 60 more rows
```

Exercise 2: Find all the rows with NA values in **the first two columns** for the following datasets:

- diamonds

```
diamonds %>%
  filter(is.na(carat) | is.na(cut))
```

```
## # A tibble: 0 x 10
## # ... with 10 variables: carat <dbl>, cut <ord>, color <ord>,
## #   clarity <ord>, depth <dbl>, table <dbl>, price <int>, x <dbl>,
## #   y <dbl>, z <dbl>
```

```
# seems like there are no such rows
```

- flights

```
flights %>%
  filter(is.na(year) | is.na(month))
```

```
## # A tibble: 0 x 19
## # ... with 19 variables: year <int>, month <int>, day <int>,
## #   dep_time <int>, sched_dep_time <int>, dep_delay <dbl>, arr_time <int>,
## #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# seems like there are no such rows
```

- mtcars

```
mtcars %>%
  filter(is.na(mpg) | is.na(cyl))
```

```
##  [1] mpg  cyl  disp hp   drat wt   qsec vs   am   gear carb
## <0 rows> (or 0-length row.names)
```

```
# seems like there are no such rows
```

The next exercise asks you to check *all* columns, but you don't want to do that with `filter()`. Why do you not want to do that with filter, especially for a dataset with hundreds of columns?

If there are lots of columns, we would need a very long expression with many repetitions of is.na() which is difficult to read.

Exercise 3: Look up `filter_all` and look at the examples at the end of the documentation. Use this (with some google-fu or discourse help) to find all the rows with NA values in *any* column for the following datasets:

Looking up `filter_all`:

```
?filter_all
```

- diamonds

```
filter_all(diamonds, any_vars(is.na(.)))
```

```
## # A tibble: 0 x 10
## # ... with 10 variables: carat <dbl>, cut <ord>, color <ord>,
## #   clarity <ord>, depth <dbl>, table <dbl>, price <int>, x <dbl>,
## #   y <dbl>, z <dbl>
```

```
# seems like there are no such rows
```

- flights

```
filter_all(flights, any_vars(is.na(.)))
```

```
## # A tibble: 9,430 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1     1525           1530        -5     1934
##  2  2013     1     1     1528           1459        29     2002
##  3  2013     1     1     1740           1745        -5     2158
##  4  2013     1     1     1807           1738        29     2251
##  5  2013     1     1     1939           1840        59       29
##  6  2013     1     1     1952           1930        22     2358
##  7  2013     1     1     2016           1930        46       NA
##  8  2013     1     1       NA           1630        NA       NA
##  9  2013     1     1       NA           1935        NA       NA
## 10  2013     1     1       NA           1500        NA       NA
## # ... with 9,420 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- mtcars

```
filter_all(mtcars, any_vars(is.na(.)))
```

```
##  [1] mpg  cyl  disp hp   drat wt   qsec vs   am   gear carb
## <0 rows> (or 0-length row.names)
```

```
# seems like there are no such rows
```

Thus, the output should be those rows that *do* contain NA values. Then look up `na.omit` (hat tip @kristof) and use that do achieve the same goal.

Looking up `na.omit`:

```
?na.omit
```

- diamonds

```
na.omit(diamonds, invert=TRUE)
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

- flights

```
na.omit(flights, invert = TRUE)
```

```
## # A tibble: 327,346 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 327,336 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

- mtcars

```
na.omit(mtcars, invert = TRUE)
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

The argument invert does not seem to work (if set to TRUE, only the rows with NA should remain). The reason for this according to comments of people on several forums is that in this version of R `na.omit()` does not have the argument invert anymore. As simply negating the function with '!' does not work either, I could not figure out how to keep only the rows with NA values using `na.omit()`.

Exercise 5: Come up with an exercise to help you – and others – learn `summarise` and `group_by` better. The more confused you are, the more you should simply try to come up with, or even copy, an example from somewhere and highlight what confuses you. Is it the order or arguments? Their role? If you are less confused, try to find a (non-obvious) use. Mention any resources used.

Answer: How much time did those planes (identified by tailnum) gain on average in air in a month which took off at least 30 minutes late but arrived earlier than scheduled time or on time?

```
flights %>%
  group_by(year, month, tailnum) %>%
  filter(dep_delay >= 30.0, arr_delay <= 0) %>%
  summarise(gain_time_mean = mean(dep_delay, na.rm = TRUE) - mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 519 x 4
## # Groups:   year, month [12]
##     year month tailnum gain_time_mean
##    <int> <int> <chr>            <dbl>
##  1  2013     1 N369NW              43
##  2  2013     1 N3730B              50
##  3  2013     1 N377DA              45
##  4  2013     1 N3BJAA              34
##  5  2013     1 N436UA              31
##  6  2013     1 N437UA              38
##  7  2013     1 N438UA              36
##  8  2013     1 N4WAAA              45
##  9  2013     1 N524JB              37
## 10  2013     1 N539UA              56
## # ... with 509 more rows
```

Exercise 6: Work through sections 11.1 and 11.2 (skip exercises).

Answer:

11.1 Prerequisites

```
library(tidyverse)
```

11.2 Getting started

Reading a csv

```
testdata3 <- read_csv("D:/Egyetem/CEU/Coding_1/R-Coding/lecture6/test-data3.csv")

(testdata3)
```

```
## # A tibble: 3,009 x 5
##    amound also_amound who_knows name  has_passed
##     <dbl>       <dbl>     <dbl> <chr> <lgl>
##  1      1           2         3 a     TRUE
##  2      4           5         6 b     TRUE
##  3      7           8         9 c     FALSE
##  4      1           2         3 a     TRUE
##  5      4           5         6 b     TRUE
##  6      7           8         9 c     FALSE
##  7      7           8         9 c     FALSE
##  8      7           8         9 c     FALSE
##  9      7           8         9 c     FALSE
## 10      7           8         9 c     FALSE
## # ... with 2,999 more rows
```

Supplying inline csv

```
read_csv("a,b,c
1,2,3
4,5,6")
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Skipping the first n lines

```
read_csv("First line of metadata goes here
  Second line of metadata goes here
  x,y,z
  1,2,3", skip = 2)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

Comments to skip

```
read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

When there are no column names

```
read_csv("1,2,3\n4,5,6", col_names = FALSE)
```

```
## # A tibble: 2 x 3
##       X1    X2    X3
##    <dbl> <dbl> <dbl>
## 1      1     2     3
## 2      4     5     6
```