Lecture Assignment 7

Taiki Yamashita

2024-04-25

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

5.4 Question 3

```r
# What does the any_of() function do? Why might it be helpful in conjunction with this vector?
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, any_of(vars))
```

```
## # A tibble: 336,776 x 5
##     year month   day dep_delay arr_delay
##    <int> <int> <int>     <dbl>     <dbl>
##  1  2013     1     1         2        11
##  2  2013     1     1         4        20
##  3  2013     1     1         2        33
##  4  2013     1     1        -1       -18
##  5  2013     1     1        -6       -25
##  6  2013     1     1        -4        12
##  7  2013     1     1        -5        19
##  8  2013     1     1        -3       -14
##  9  2013     1     1        -3        -8
## 10  2013     1     1        -2         8
## # i 336,766 more rows
```

The code selects from the 'flights' dataframe where the column names match any of the nmes in the 'vars' vector. It is helpful when you want to select columns dynamically based on a predefined set of column names.

Especially when you havee a large dataframe with many columns and you want to select only a subset of columns based on certain criteria, such as a list of variable names, this will be very useful.

5.5 Question 1

```
# Currently dep_time and sched_dep_time are convenient to look at, but hard to compute with a
# because they're not really continuous numbers. Convert them to a more convenient representation
# of number of minutes since midnight.

hours2mins <- function(x) {
  x %/% 100 * 60 + x %% 100
}

# with integer division
mutate(flights,
       dep_time = hours2mins(dep_time),
       sched_dep_time = hours2mins(sched_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>    <dbl>          <dbl>     <dbl>    <int>          <int>
##  1  2013     1     1      317            315         2      830            819
##  2  2013     1     1      333            329         4      850            830
##  3  2013     1     1      342            340         2      923            850
##  4  2013     1     1      344            345        -1     1004           1022
##  5  2013     1     1      354            360        -6      812            837
##  6  2013     1     1      354            358        -4      740            728
##  7  2013     1     1      355            360        -5      913            854
##  8  2013     1     1      357            360        -3      709            723
##  9  2013     1     1      357            360        -3      838            846
## 10  2013     1     1      358            360        -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>
```

Defines a function 'hours2mins()' to convert time values from the HHMM format to minutes since midnight, and then applies this function to transform 'dep_time' and 'sched_dep_time' variables in the 'flights' dataset using 'mutate()'. It simplifies time calculations by converting them into a continuous numerical representation.

5.5 Question 2

```
# Compare air_time with arr_time - dep_time. What do you expect to see?
# What do you see? What do you need to do to fix it?

flights_airtime <-
  mutate(flights,
    dep_time = (dep_time %/% 100 * 60 + dep_time %% 100) %% 1440,
```

```r
    arr_time = (arr_time %/% 100 * 60 + arr_time %% 100) %% 1440,
    air_time_diff = air_time - arr_time + dep_time
  )

nrow(filter(flights_airtime, air_time_diff != 0))
```

```
## [1] 327150
```

What I expect to see that air_time is the difference between the arrival and departure times. In other words, air_time = arr_time - dep_time. There should be no flights with non-zero values of arr_time_diff. But it turns out that there are many flights for which arr_time != arr_time - dep_time. To fix these time-zone issues, I would want to convert all the times to a date-time to handle overnight flights and from local time to a common time zone, most likely to UTC, to handle flights crossing time-zones.

5.6 Question 2

```r
# Come up with another approach that will give you the same output as not_cancelled %>%
# count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count())
not_cancelled <-
  flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  count(dest)
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # i 94 more rows
```

```r
# and

not_cancelled %>%
  count(tailnum, wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 N0EGMQ  239143
```

```
##   3 N10156  109664
##   4 N102UW   25722
##   5 N103US   24619
##   6 N104UW   24616
##   7 N10575  139903
##   8 N105UW   23618
##   9 N107US   21677
##  10 N108UW   32070
## # i 4,027 more rows
```

```r
# (without using count()).

# we can combine group_by() and summarise() verbs.

not_cancelled %>%
  group_by(dest) %>%
  summarise(n = n())
```

```
## # A tibble: 104 x 2
##    dest      n
##    <chr> <int>
##  1 ABQ     254
##  2 ACK     264
##  3 ALB     418
##  4 ANC       8
##  5 ATL   16837
##  6 AUS    2411
##  7 AVL     261
##  8 BDL     412
##  9 BGR     358
## 10 BHM     269
## # i 94 more rows
```

```r
# and

# similar to earlier, we can replicate count() by combining group_by() and summarise() verbs.
# this time, instead of using length(), we will use sum() with the weighting variable.
not_cancelled %>%
  group_by(tailnum) %>%
  summarise(n = sum(distance))
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##    <chr>    <dbl>
##  1 D942DN    3418
##  2 NOEGMQ  239143
##  3 N10156  109664
##  4 N102UW   25722
##  5 N103US   24619
##  6 N104UW   24616
##  7 N10575  139903
##  8 N105UW   23618
##  9 N107US   21677
```

```
## 10 N108UW   32070
## # i 4,027 more rows
```

5.7 Question 3

```
# What time of day should you fly if you want to avoid delays as much as possible?

flights %>%
  group_by(hour) %>%
  summarise(arr_delay = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(arr_delay)
```

```
## # A tibble: 20 x 2
##     hour arr_delay
##    <dbl>     <dbl>
##  1     7    -5.30
##  2     5    -4.80
##  3     6    -3.38
##  4     9    -1.45
##  5     8    -1.11
##  6    10     0.954
##  7    11     1.48
##  8    12     3.49
##  9    13     6.54
## 10    14     9.20
## 11    23    11.8
## 12    15    12.3
## 13    16    12.6
## 14    18    14.8
## 15    22    16.0
## 16    17    16.0
## 17    19    16.7
## 18    20    16.7
## 19    21    18.4
## 20     1   NaN
```

We can group the hour of the flight. The earlier the flight is scheduled, the lower its expected delay is. Morning flights have fewer previous flights that can delay them.