Data transformation exercises

Maria Morales

2023-08-18

Introduction

This document will give a short explanation of the development of data transformation exercises implementing the `dplyr` and `nycflights13` packages found in `tidyverse`.

Some of the `dplyr` package functions used to solve the exercises are :

- `filter()`: This function allows you to filter rows according to a condition.
- `arrange()`: This function is used to order the rows of a data frame, by default the rows are arranged in ascending order.
- `select()`: This function selects the columns of a data frame.
- `mutate()`: This function allows you to create new variables in the data frame.
- `Summarise()`:This function is similar to the `mutate` function, however, it does not add new columns but creates a new data frame.

Package installation

For the development of the exercises it is necessary to install the packages `tidyverse`, `dplyr` and `nycflights13`, to do this go to the top bar by clicking on the **Tools** option, then click on **install packages** and proceed to install the packages.

Data transformation

When starting the programming to load the packages or libraries use the **library** function, in addition the documentation `nycflights13::flights` should be used, which contains data on 336 776 flights departing from New York City in 2013 provided by the U.S. Bureau of Transportation Statistics.

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- 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
```

```
flights_o <- nycflights13::flights
```

Exercises

Exercises with the `filter()` function 5.2.4 Exercises: item 1, and 2

1. Find all flights that had an arrival delay of two or more hours.

For this exercise you must take into account the variable `arr_delay`, and the time expressed in 120 minutes.

```
EX_1 <- filter(flights_o, arr_delay >= 120)
```

2. Find all flights that flew to Houston (`IAH` or`HOU`) For this exercise you must take into account `dest`.

```
Ex_2 <- filter(flights_o, dest  == "IAH" | dest == "HOU" )
```

Exercises with the `arrange()` function 5.3.1 Exercises: all items

1. How could I use `arrange()` to sort all the missing values at the beginning? use `is.na()`.

In this case the `is.na()` function indicates which elements are missing.

```
Ex_3 <- arrange(flights_o, desc(is.na(flights_o)))
```

2. Order `flights` to find the most delayed flights. Find the flights that departed earlier.

In this case the `dep_delay` departure delays are taken into account.

```
Ex_4 <- arrange(flights_o, desc(dep_delay))
```

3. Order `flights` to find the fastest flights (highest speed).

Taking into account that the dataset does not have the speed of the flights, a new variable containing this information must be created, using the variables `distance` and `air_time`.

```
Ex_5 <- select(flights_o,distance, air_time )%>%
  mutate(speed= distance/air_time)%>%
  select(speed)%>%
  arrange(desc(speed))
```

4. Which flights traveled the farthest? Which one traveled the least?

In this case the `distance` variable is implemented and in case there is a tie in the information a tie-breaker will be made with the `air_time` variables.

```
Ex_6 <- arrange(flights_o, desc(distance),desc(air_time))
```

Exercises with the `select ()` function 5.4.1 Exercises: item 2, 3 and 4

1. What happens if you include the name of a variable several times in a `select()`?

In this case when including the name of a variable several times the program selects that column only once, for this exercise the variable `air_time` was included twice.

```
Ex_7 <- select(flights_o, air_time, distance, hour, minute, air_time)
```

2. What does the `any_of()` function do, and why might it be useful in conjunction with this vector?

The `any_of()` function selects variables by matching patterns in their names and is useful since the variable names match in the character vector, thus selecting these specific columns.

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")

Ex_8 <-  select(flights_o, any_of(vars))
```

3. Are you surprised by the result of running the following code? How do the select helpers treat the case by default? How can that default be changed?

For this case, the `contains` helper helps to select all the columns whose variables have in their name the word **time**, the default value is changed where **TIME** example **delay** can be used.

```
Ex_9 <- select(flights, contains("TIME"))
```

Exercises with the `mutate ()` function 5.5.2 Exercises: item 1 and 2

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

The format of the data requires separating the hours from the minutes by dividing the integers by 100 and then multiplying the result by 60. In addition, the minutes are multiplied by 100 to eliminate the decimal of the hours, then they must be added and multiplied by the same operation equivalent to the midnight minutes.

```
Ex_10 <- select(flights_o,dep_time, sched_dep_time)%>%
      mutate (flights_dep_time= dep_time %/% 100 * 60 + dep_time %% 100) %% 1440 %>%
      mutate (fligths_s_t= sched_dep_time %/% 100 * 60 + sched_dep_time %% 100) %% 1440
```

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?

As in the previous exercise, the times must be transformed so that the data can be processed. Knowing the time in the air will be equal to the subtraction of the arrival time and the departure time.

```
Ex_11 <- select(flights_o, air_time, arr_time, dep_time)%>%
      mutate(arr_1= arr_time %/% 100 * 60 + arr_time %% 100) %% 1440%>%
      mutate(dep_1= dep_time %/% 100 * 60 + dep_time %% 100) %% 1440%>%
      mutate(air_time_diff = air_time - arr_1 + dep_1)
```

Exercises 5.6.7 Exercises: item 1

Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights. Consider the following scenarios:

A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.

A flight is always 10 minutes late.

A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.

99% of the time a flight is on time. 1% of the time it's 2 hours late.

Which is more important: arrival delay or departure delay?

Answer: Delay in arrival is more important than delay in departure, since the former may have consequences for the passenger, while delay in departure, although it may occur, in some cases does not necessarily entail consequences.

Exercises Grouped mutates (and filters) 5.7.1 Exercises: item 2

1. Which plane `(tailnum)` has the worst on-time record?

The variables arrival delay, arrival time, and grouping tailnum must be taken into account, then just leave the planes that had at least 20 flights.

```
Ex_12 <- filter(flights_o,!is.na(tailnum), is.na(arr_time) |!is.na(arr_delay) )%>%
      mutate(on_time= !is.na(arr_time) & (arr_delay <= 0))%>%
      group_by(tailnum) %>%
```

```
    summarise(on_time = mean(on_time), n = n()) %>%
    filter(n >= 20) %>%
    filter(min_rank(on_time) == 1)
```