HOMEWORK 1

Jennyfer Iglandini Torres Piraquive

2023-08-18

Activity development

1. Download the libraries, "tidyverse" which promotes the concept of "sorted data", "nycflights13" which provides us with flight data to analyze and library(knitr) it allows you to display data tables in an orderly and eye-pleasing manner.

library(nycflights13)

library(tidyverse)

library(knitr)

2. DEVELOPMENT OF CLASSWORK

- 5.2.4 Exercises: items 1: Use the "filter" function to search for flights with an arrival delay of two hours or more.

```
library(nycflights13)

library(tidyverse)

exer  <- nycflights13::flights

my_DF1  <- filter(exer, arr_delay >="2")
```

The table we get after running the code corresponds to "my_DF1":

```
library(knitr)
kable(my_DF1[1:10, c(1, 14, 13, 9)],
      caption = "In this table you can see my_DF1 information",
      align = "c")
```

Table 1: In this table you can see my_DF1 information

year	dest	origin	arr_delay
2013	IAH	LGA	20
2013	MIA	JFK	33
2013	ORD	LGA	8
2013	LAX	JFK	7

year	dest	origin	arr_delay
2013	DFW	LGA	31
2013	ORD	EWR	32
2013	RSW	JFK	4
2013	PHX	EWR	3
2013	MIA	LGA	5
2013	MSP	EWR	29

- 5.2.4 Exercises: items 2: I flew to Houston (IAH or HOU), I choose the IAH airport and filter it with the "filter" function.

```
exer  <- nycflights13::flights

my_DF2  <- filter(exer, dest =="IAH")
```

The table we get after running the code corresponds to "my_DF2":

```
library(knitr)
kable(my_DF2[1:10, c(1, 13, 14)],
      caption = "In this table you can see my_DF2 information",
      align = "c")
```

Table 2: In this table you can see my_DF2 information

year	origin	dest
2013	EWR	IAH
2013	LGA	IAH
2013	LGA	IAH
2013	LGA	IAH
2013	EWR	IAH
2013	EWR	IAH
2013	LGA	IAH
2013	EWR	IAH
2013	LGA	IAH
2013	EWR	IAH

- 5.3.1 Exercises: items 1: How could you use `arrange()` to sort all missing values to the start? (Hint: use `is.na()`). As the help indicates, it´s used to check if the elements being parsed are missing values or NA.

```
my_DF3 <- flights %>% arrange(desc(is.na(dep_time)))
```

The table we get after running the code corresponds to "my_DF3":

```
library(knitr)
kable(my_DF3[1:10, c(1, 4, 6,7,9,15)],
      caption = "In this table you can see my_DF3 information",
      align = "c")
```

Table 3: In this table you can see my_DF3 information

year	dep_time	dep_delay	arr_time	arr_delay	air_time
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA
2013	NA	NA	NA	NA	NA

- 5.3.1 Exercises: items 2: Sort the flights to find the most delayed flights. Find the flights that left the earliest. For the development of this item we used the data from the colimna (arr_delay), these were ordered from longest to shortest delay.

```
my_DF4 <- flights %>% arrange(desc(arr_delay))
```

The table we get after running the code corresponds to "my_DF4":

```
library(knitr)
kable(my_DF4[1:10, c(1,9)],
      caption = "In this table you can see my_DF4 information",
      align = "c")
```

Table 4: In this table you can see my_DF4 information

year	arr_delay
2013	1272
2013	1127
2013	1109
2013	1007
2013	989
2013	931
2013	915
2013	895
2013	878
2013	875

- 5.3.1 Exercises: items 3: Sort the flights to find the fastest ones (higher speed).

  At this point we use the "mutate" function that helps us to add a new column called "speed", where the data will be assigned between "distance / air_time", once the data is obtained, we arrange them from highest to lowest with the help of the "arrange" and "desc" function.

```
my_DF5 <- flights %>% mutate(speed = distance / air_time)%>% arrange((desc(speed)))
```

The table we get after running the code corresponds to "my_DF5":

```
library(knitr)
kable(my_DF5[1:10, c(14,15,16,20)],
      caption = "In this table you can see my_DF5 information",
      align = "c")
```

Table 5: In this table you can see my_DF5 information

dest	air_time	distance	speed
ATL	65	762	11.723077
MSP	93	1008	10.838710
GSP	55	594	10.800000
BNA	70	748	10.685714
PBI	105	1035	9.857143
SJU	170	1598	9.400000
SJU	172	1598	9.290698
STT	175	1623	9.274286
SJU	173	1598	9.236994
SJU	173	1598	9.236994

- 5.3.1 Exercises: items 4: Which are the farthest and the shortest flights?

  For both cases use the "arrange" function to sort the data and use the "distance" column provided by the library(nycflights13), in the case of the farthest flights sort the data from farthest to shortest with the "desc" function and for the shortest flights sort them from shortest to longest.

```
my_DF6 <- flights %>% arrange(desc(distance))

my_DF7 <- flights %>% arrange(distance)
```

The table of the farthest flight is obtained after executing the code corresponding to "mi_DF6":

```
library(knitr)
kable(my_DF6[1:10, c(13,14,16)],
      caption = "In this table you can see my_DF6 information",
      align = "c")
```

Table 6: In this table you can see my_DF6 information

origin	dest	distance
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983
JFK	HNL	4983

The table of the shortest flight we obtain after executing the code corresponding to "mi_DF7":

```
library(knitr)
kable(my_DF7[1:10, c(13,14,16)],
      caption = "In this table you can see my_DF7 information",
      align = "c")
```

Table 7: In this table you can see my_DF7 information

origin	dest	distance
EWR	LGA	17
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80
EWR	PHL	80

- 5.4.1 Exercises: items 2: What happens if you include the name of a variable multiple times in a select() call?

  Answer: Variable appears repeatedly in a result

- 5.4.1 Exercises: items 3: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")
```

  Answer: The any_of() function is used to select columns from a data frame based on a character vector of column names, and, yes, it could be useful for that line of code.

- 5.4.1 Exercises: items 4: Does the result of running the following code surprise you? How do the select helpers deal whit case by default? How can you change that default?

```
select(flights, contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##    dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##       <int>          <int>    <int>          <int>    <dbl> <dttm>
##  1      517            515      830            819      227 2013-01-01 05:00:00
##  2      533            529      850            830      227 2013-01-01 05:00:00
##  3      542            540      923            850      160 2013-01-01 05:00:00
##  4      544            545     1004           1022      183 2013-01-01 05:00:00
##  5      554            600      812            837      116 2013-01-01 06:00:00
##  6      554            558      740            728      150 2013-01-01 05:00:00
##  7      555            600      913            854      158 2013-01-01 06:00:00
##  8      557            600      709            723       53 2013-01-01 06:00:00
##  9      557            600      838            846      140 2013-01-01 06:00:00
## 10      558            600      753            745      138 2013-01-01 06:00:00
## # i 336,766 more rows
```

Answer: This code selects the columns whose names contain the string "TIME" and which contain the text "TIME" in their cells as "dep_time", "sched_dep_time", "arr_time", "sched_arr_time", "air_time time_hour".

- 5.5.2 Exercises: items 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 mutate() function is employed to introduce two additional columns: dep_time_mins and sched_dep_time_mins. These columns indicate the departure time and scheduled departure time, both translated into minutes starting from midnight. The computation (dep_time %/% 100) * 60 + dep_time %% 100 transforms the given hour and minutes into a total count of minutes.

```
my_DF8 <- flights %>% mutate( dep_time_mins = (dep_time %/% 100) * 60 + dep_time %% 100,
          sched_dep_time_mins = (sched_dep_time %/% 100) * 60 + sched_dep_time %% 100)
```

The table we get after running the code corresponds to "my_DF8":

```
library(knitr)
kable(my_DF8[1:10, c(20,21)],
      caption = "In this table you can see my_DF8 information",
      align = "c")
```

Table 8: In this table you can see my_DF8 information

dep_time_mins	sched_dep_time_mins
317	315
333	329
342	340
344	345
354	360
354	358
355	360
357	360
357	360
358	360

- 5.5.2 Exercises: items 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?

Using the mutate() function, the time difference between arrival and departure is calculated in minutes (arr_time - dep_time_mins). Then, the filter() function is used to remove rows with missing values in air_time or the time difference. Finally, the select() function is applied to keep only two columns: air_time (flight duration) and arr_dep_time_diff (time gap between arrival and departure in minutes).

```
my_DF9 <- my_DF8%>% mutate(a_dep_t_diff = arr_time - dep_time_mins) %>%
  filter(!is.na(air_time) & !is.na(a_dep_t_diff)) %>% select(air_time, arr_time,a_dep_t_diff)
```

The table we get after running the code corresponds to "my_DF9":

```r
library(knitr)
kable(my_DF9[1:10, c(1,2,3)],
      caption = "In this table you can see my_DF9 information",
      align = "c")
```

Table 9: In this table you can see my_DF9 information

air_time	arr_time	a_dep_t_diff
227	830	513
227	850	517
160	923	581
183	1004	660
116	812	458
150	740	386
158	913	558
53	709	352
140	838	481
138	753	395

- 5.6.7 Exercises: item 1: Brainstorm at least 5 different ways to assess the typical characteristics of a group of fligh.

This summary outlines different analyses related to flight delays:

1. **Median Arrival Delay**: Finding the central value representing the typical delay experienced upon arrival by calculating the median arrival delay for a group of flights.
2. **Average Departure Delay**: Calculating the average delay before departure for a group of flights. This offers insight into the typical delay experienced before takeoff.
3. **Punctuality Percentage**: Calculating the percentage of flights that are punctual (no arrival delay) and comparing it to the percentage of flights significantly delayed (2 hours late). This contrasts on-time flights with extremely delayed ones.
4. **Proportion of Flights with Specific Delays:** Determining the percentage of flights arriving either significantly early or late (15 minutes, 30 minutes, or 2 hours). This provides insight into the distribution of delay scenarios within the group.
5. **Arrival Delay Distribution:** Creating a histogram or density plot showcasing the distribution of arrival delays across all flights. This visualization identifies common delay ranges and outliers.

- 5.7.1 Exercises: item 2: Which plane (`tailnum`) has the worst on-time record?

The dataset is categorized by aircraft tail number (tailnum). By employing the summarize() function, we compute summary statistics for every group (aircraft). Inside the summarize() function: "**total_flights**" is calculated through the n() function, yielding the aggregate count of flights for each aircraft. "**punctual_flights**" is derived using the sum() function, counting flights in which the arrival delay (arr_delay) is less than or equal to 0 (indicating punctual or early arrivals). "**punctuality_percentage**" is computed as the proportion of punctual flights to total flights, then multiplied by 100 to express it as a percentage.

Subsequent to the summarize() operation, we employ arrange() to arrange the groups (aircraft) based on their punctuality percentages in ascending sequence. This results in the aircraft with the lowest punctuality

percentages being presented first. filter() is utilized to eliminate rows where the punctuality_percentage is unavailable (NA). The resultant dataset is assigned to the variable my_DF10.

```
my_DF10 <- flights %>%
  group_by(tailnum) %>%
  summarize( total_flights = n(),punctual_flights = sum(arr_delay
  <= 0, na.rm = TRUE),punctuality_percentage =
  (punctual_flights / total_flights) * 100) %>%
  arrange(punctuality_percentage) %>%
  filter(!is.na (punctuality_percentage))
my_DF10
```

```
## # A tibble: 4,044 x 4
##    tailnum total_flights punctual_flights punctuality_percentage
##    <chr>           <int>            <int>                  <dbl>
##  1 N121DE              2                0                      0
##  2 N136DL              1                0                      0
##  3 N143DA              1                0                      0
##  4 N17627              2                0                      0
##  5 N240AT              5                0                      0
##  6 N26906              1                0                      0
##  7 N295AT              4                0                      0
##  8 N302AS              1                0                      0
##  9 N303AS              1                0                      0
## 10 N32626              1                0                      0
## # i 4,034 more rows
```

The table we get after running the code corresponds to "my_DF10":

```
library(knitr)
kable(my_DF10[1:10, c(1,2,3,4)],
caption = "In this table you can see my_DF10 information",
align = "c")
```

Table 10: In this table you can see my_DF10 information

tailnum	total_flights	punctual_flights	punctuality_percentage
N121DE	2	0	0
N136DL	1	0	0
N143DA	1	0	0
N17627	2	0	0
N240AT	5	0	0
N26906	1	0	0
N295AT	4	0	0
N302AS	1	0	0
N303AS	1	0	0
N32626	1	0	0