First Activity

Ricardo Figueroa

2023-08-18

library(nycflights13). library(tidyverse). library(knitr).

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

```
P  <- nycflights13::flights
P1  <- filter(P, arr_delay >="2")
kable(P1[1:10, c(1, 14, 13, 9)],
      caption = "In this table you could see P1 information",
      align = "c")
```

Table 1: In this table you could see P1 information

year	dest	origin	arr_delay
2013	IAH	LGA	20
2013	MIA	JFK	33
2013	ORD	LGA	8
2013	LAX	JFK	7
2013	DFW	LGA	31
2013	ORD	EWR	32
2013	RSW	JFK	4
2013	PHX	EWR	3
2013	MIA	LGA	5
2013	MSP	EWR	29

In the first point we made use of the filter function in order to obtain the data of the flights that had 2 or more hours of delay in arrival, for this we took the column arrive_delay and indicated that the data greater (>) 2 were displayed in the table above.

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.

```
P2 <- nycflights13::flights
P2.1  <- filter(P2, dest =="IAH")
```

```
kable(P2.1[1:10, c(1, 14, 13, 9)],
      caption = "In this table you could see P2.1 information",
      align = "c")
```

Table 2: In this table you could see P2.1 information

year	dest	origin	arr_delay
2013	IAH	EWR	11
2013	IAH	LGA	20
2013	IAH	LGA	1
2013	IAH	LGA	3
2013	IAH	EWR	26
2013	IAH	EWR	9
2013	IAH	LGA	11
2013	IAH	EWR	1
2013	IAH	LGA	145
2013	IAH	EWR	-2

In the previous point we wanted to show the data of the flights that landed in Houston "IAH or HOU". To develop this point we used the sign == which will allow us to take the data from the column with the word 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.

```
P3 <- flights %>% arrange(desc(is.na(dep_time)))
```

```
kable(P3[1:10, c(1, 14, 13, 9)],
      caption = "In this table you could see P3 information",
      align = "c")
```

Table 3: In this table you could see P3 information

year	dest	origin	arr_delay
2013	RDU	EWR	NA
2013	DFW	LGA	NA
2013	MIA	LGA	NA
2013	FLL	JFK	NA
2013	CVG	EWR	NA
2013	PIT	EWR	NA
2013	MHT	EWR	NA
2013	ATL	EWR	NA
2013	IND	EWR	NA
2013	LAX	JFK	NA

For the previous exercise we used the arrange function to first organize or locate the data that was empty in the dep_time column.

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.

```r
P4 <- flights %>% arrange(desc(arr_delay))
```

```r
kable(P4[1:10, c(1, 14, 13, 9)],
      caption = "In this table you can see P4 information",
      align = "c")
```

Table 4: In this table you can see P4 information

year	dest	origin	arr_delay
2013	HNL	JFK	1272
2013	CMH	JFK	1127
2013	ORD	EWR	1109
2013	SFO	JFK	1007
2013	CVG	JFK	989
2013	TPA	JFK	931
2013	MSP	LGA	915
2013	ATL	LGA	895
2013	MIA	EWR	878
2013	ORD	EWR	875

In the previous exercise, the arrange function was used to organize the data of the flights that had the most delays in arriving at their destination. Taking into account the parameter desc. This will allow us to have the organization from the longest to the shortest.

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

```r
P5 <- flights %>% mutate(speed = distance / air_time)%>% arrange((desc(speed)))
```

```r
kable(P5[1:10, c(1, 14, 13, 11, 20)],
      caption = "In this table you can see P5 information",
      align = "c")
```

Table 5: In this table you can see P5 information

year	dest	origin	flight	speed
2013	ATL	LGA	1499	11.723077
2013	MSP	EWR	4667	10.838710
2013	GSP	EWR	4292	10.800000
2013	BNA	EWR	3805	10.685714
2013	PBI	LGA	1902	9.857143
2013	SJU	JFK	315	9.400000
2013	SJU	JFK	707	9.290698
2013	STT	JFK	936	9.274286
2013	SJU	JFK	347	9.236994
2013	SJU	JFK	1503	9.236994

For the previous exercise we made use of the mutate function in order to add a new column that will contain the speed data of each flight, for this, we took the distance and the air time that the plane had, we applied the formula V = D/T, in this way we could determine which is the speed of each flight.

Finally, the data was organized in such a way as to visualize first the flights that had the highest speed.

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

```
P6 <- flights %>% arrange(desc(distance))
P7 <- flights %>% arrange(distance)
```

```
kable(P6[1:10, c(1, 14, 13, 16)],
      caption = "In this table you can see P6 information",
      align = "c")
```

Table 6: In this table you can see P6 information

year	dest	origin	distance
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983
2013	HNL	JFK	4983

```
kable(P7[1:10, c(1, 14, 13, 16)],
      caption = "In this table you can see P7 information",
      align = "c")
```

Table 7: In this table you can see P7 information

year	dest	origin	distance
2013	LGA	EWR	17
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80
2013	PHL	EWR	80

In the previous point, we made use of the arrange function in order to organize the data of the distance column, taking into account the "desc" particle that allows us to organize the data from highest to lowest, identifying the flights that took the longest distance.

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

Answer: The variable is consistently present multiple times within an outcome

5.4.1Exercises: 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 employed to choose specific columns from a data frame using a character vector of column names. This can be valuable for the mentioned 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?

```
print(kable(select(P[1:20,], contains("TIME")),
                caption = "In this table you can see TIMES that contain",
                align = "c"))
```

```
##
##
## Table: In this table you can see TIMES that contain
##
## | dep_time | sched_dep_time | arr_time | sched_arr_time | air_time |      time_hour      |
## |:--------:|:--------------:|:--------:|:--------------:|:--------:|:-------------------:|
## |   517    |      515       |   830    |      819       |   227    | 2013-01-01 05:00:00 |
## |   533    |      529       |   850    |      830       |   227    | 2013-01-01 05:00:00 |
## |   542    |      540       |   923    |      850       |   160    | 2013-01-01 05:00:00 |
## |   544    |      545       |   1004   |      1022      |   183    | 2013-01-01 05:00:00 |
## |   554    |      600       |   812    |      837       |   116    | 2013-01-01 06:00:00 |
## |   554    |      558       |   740    |      728       |   150    | 2013-01-01 05:00:00 |
## |   555    |      600       |   913    |      854       |   158    | 2013-01-01 06:00:00 |
## |   557    |      600       |   709    |      723       |    53    | 2013-01-01 06:00:00 |
## |   557    |      600       |   838    |      846       |   140    | 2013-01-01 06:00:00 |
## |   558    |      600       |   753    |      745       |   138    | 2013-01-01 06:00:00 |
## |   558    |      600       |   849    |      851       |   149    | 2013-01-01 06:00:00 |
## |   558    |      600       |   853    |      856       |   158    | 2013-01-01 06:00:00 |
## |   558    |      600       |   924    |      917       |   345    | 2013-01-01 06:00:00 |
## |   558    |      600       |   923    |      937       |   361    | 2013-01-01 06:00:00 |
## |   559    |      600       |   941    |      910       |   257    | 2013-01-01 06:00:00 |
## |   559    |      559       |   702    |      706       |    44    | 2013-01-01 05:00:00 |
## |   559    |      600       |   854    |      902       |   337    | 2013-01-01 06:00:00 |
## |   600    |      600       |   851    |      858       |   152    | 2013-01-01 06:00:00 |
## |   600    |      600       |   837    |      825       |   134    | 2013-01-01 06:00:00 |
## |   601    |      600       |   844    |      850       |   147    | 2013-01-01 06:00:00 |
```

Answer:

1. The dataset is grouped by aircraft tail number (tailnum), allowing subsequent calculations to be done separately for each aircraft.
2. The **`summarize()`** function is used to calculate summary statistics for each aircraft. Inside the **`summarize()`** function:
    - **`total_flights`** is determined using the **`n()`** function, which counts the total number of flights for each aircraft.

- **punctual_flights** is calculated using the **sum()** function, counting flights where the arrival delay (arr_delay) is less than or equal to 0 (on-time or early arrivals). The argument **na.rm = TRUE** handles missing values in the arr_delay column.
- **punctuality_percentage** is computed as the ratio of punctual flights to total flights, multiplied by 100 to obtain the percentage.

3. After summarization, the groups (aircraft) are sorted using the **arrange()** function based on their punctuality percentages in ascending order. This means aircraft with the lowest punctuality percentages will be listed first.

4. The **filter()** function is employed to remove rows where the punctuality_percentage is not available (NA).

5. The resulting dataset is stored in the variable **worst_punctuality**, which contains information such as aircraft tail numbers, total flights, punctual flights, and corresponding punctuality percentages.

6. Finally, the **worst_punctuality** dataset is displayed, showing the tail numbers of aircraft with the lowest punctuality percentages.

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.

```
P8 <- flights %>% mutate( dep_time_mins = (dep_time %/% 100) * 60 + dep_time %% 100, sched_dep_time_min
```

```
kable(P8[1:10, c(1, 14, 13, 9)],
      caption = "In this table you can see P8 information",
      align = "c")
```

Table 8: In this table you can see P8 information

year	dest	origin	arr_delay
2013	IAH	EWR	11
2013	IAH	LGA	20
2013	MIA	JFK	33
2013	BQN	JFK	-18
2013	ATL	LGA	-25
2013	ORD	EWR	12
2013	FLL	EWR	19
2013	IAD	LGA	-14
2013	MCO	JFK	-8
2013	ORD	LGA	8

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

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?

```r
P9 <- P8 %>% mutate(arr_dep_time_diff = arr_time - dep_time_mins) %>% filter(!is.na(air_time) & !is.na(a
```

```r
kable(P9[1:10, c(1, 2)],
      caption = "In this table you can see P9 information",
      align = "c")
```

Table 9: In this table you can see P9 information

air_time	arr_dep_time_diff
227	513
227	517
160	581
183	660
116	458
150	386
158	558
53	352
140	481
138	395

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)

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

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. **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.
3. **Average Departure Delay:** Calculating the average delay before departure for a group of flights. This offers insight into the typical delay experienced before takeoff.
4. **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.
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.

```
P10 <- 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))

kable(P10[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

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 P10