Electiva Activity

Dana Gabriela Salazar Toro

2023-08-29

Introduction

This report presents an analysis of flight data using the ***nycflights13*** package and ***tidyverse*** in R. Various aspects of the flight data, such as delays, times, and punctuality, are explored.

Load ne cessary libraries

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

In this section, we are loading the libraries needed to perform data analysis. nycflights13 contains the flight dataset and tidyverse is a set of packages that facilitate data manipulation and visualization.

Load the flights dataset

Here, we are loading the flight data set into the **flights_data** object, using the flights function of the **nycflights13** package.

```
f_d <- nycflights13::flights
```

5.2.4 Exercises: Items 1 and 2

This code block solves item 1 of the exercise. We are using the **filter()** function of the tidyverse package to select only the **flights_data** rows where the value in the **arr_delay** (delay on arrival) column is greater than or equal to 2. The result is stored in the **filtered_delayed_flights** object.

```
myDf1 <- filter(f_d, arr_delay >= 2)
```

In this code block:

We made use of the **Knitr** function in order to plot the table taking into account the tail number of the aircraft and the landing delay time:

```
library(knitr)
kable(f_d[1:10,c(12,9)],caption = "ARRIVE DELAY", align = "c")
```

Table 1: ARRIVE DELAY

tailnum	arr_delay
N14228	11
N24211	20
N619AA	33

tailnum	arr_delay
N804JB	-18
N668DN	-25
N39463	12
N516JB	19
N829AS	-14
N593JB	-8
N3ALAA	8

This code block solves item 2 of the exercise. Similarly, we are using the **filter()** function to select the rows where the value in the dest column is equal to "HOU". The result is stored in the **filtered_hou_flights** object.

```
myDf2 <- filter(f_d,dest =="HOU")
```

```
library(knitr)
kable(myDf2[1:10,c(13,14)],caption = "HOUSTON DESTINY", align = "c")
```

Table 2: HOUSTON DESTINY

origin	dest
JFK	HOU
EWR	HOU
EWR	HOU
JFK	HOU
EWR	HOU
JFK	HOU
EWR	HOU
EWR	HOU
JFK	HOU
EWR	HOU

5.3.1 Exercises: All Items

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

In this code block, we are performing the following actions:

1. We take the data set flights and pass it through the %>% operator, which allows us to chain operations.
2. We use the **arrange()** function to sort the rows in descending order based on the i**s.na(dep_time)** column. This column is logical and returns TRUE if **dep_time** (departure time) is absent and FALSE if it is not. This means that we are sorting the flights so that those with missing values in **dep_time** appear first.

```
library(knitr)
kable(sorted_flights_missing_first[1:10,c(7,12)],caption = "MISSING DATA FIRST", align = "c")
```

Table 3: MISSING DATA FIRST

arr_time	tailnum
NA	N18120
NA	N3EHAA
NA	N3EVAA
NA	N618JB
NA	N10575
NA	N13949
NA	N10575
NA	N759EV
NA	N13550
NA	NA

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

In this code block:

1. We take the flights **dataset** and pass it through the **%>%** operator.
2. We use the **arrange()** function to sort the rows in descending order based on the **arr_delay** column. This means that flights with longer arrival delays will appear first.

```r
library(knitr)
kable(most_delayed_flights[1:10,c(6,9,12)],caption = "MOST DELAYED FLIGHTS", align = "c")
```

Table 4: MOST DELAYED FLIGHTS

dep_delay	arr_delay	tailnum
1301	1272	N384HA
1137	1127	N504MQ
1126	1109	N517MQ
1014	1007	N338AA
1005	989	N665MQ
960	931	N959DL
911	915	N927DA
898	895	N6716C
896	878	N5DMAA
878	875	N523MQ

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

In this code:

1. We take the **`flights`** dataset and pass it through the **`%>%`** operator.

2. We use the **`mutate()`** function to create a new column called **`speed`**. We calculate the speed by dividing the **`distance`** column by the **`air_time`** column. This will give us the speed of each flight.

3. After creating the **`speed`** column, we use the **`arrange()`** function to sort the rows in descending order based on the **`speed`** column. This means that flights with the highest speed will appear first.

```r
library(knitr)
kable(fastest_flights_desc[1:10,c(12,20)],caption = "FASTEST FLIGHTS", align = "c")
```

Table 5: FASTEST FLIGHTS

tailnum	speed
N666DN	11.723077
N17196	10.838710
N14568	10.800000
N12567	10.685714
N956DL	9.857143
N3768	9.400000
N779JB	9.290698
N5FFAA	9.274286
N3773D	9.236994
N571JB	9.236994

```r
farthest_flights <- flights %>%
  arrange(desc(distance))
```

In this code:

1. We take the **`flights`** dataset and pass it through the **`%>%`** operator.
2. We use the **`arrange()`** function to sort the rows in descending order based on the **`distance`** column. This means that flights with the longest distances will appear first.

```r
library(knitr)
kable(farthest_flights[1:10,c(12,15)],caption = "FARTHEST FLIGHTS", align = "c")
```

Table 6: FARTHEST FLIGHTS

tailnum	air_time
N380HA	659
N380HA	638
N380HA	616
N384HA	639
N381HA	635
N385HA	611
N385HA	612
N389HA	645
N384HA	640
N388HA	633

```r
closest_flights <- flights %>%
  arrange(distance)
```

In this code:

1. We take the **`flights`** dataset and pass it through the **`%>%`** operator.
2. We use the **`arrange()`** function to sort the rows in ascending order based on the **`distance`** column. This means that flights with the shortest distances will appear first.

```
library(knitr)
kable(closest_flights[1:10,c(12,15)],caption = "CLOSEST FLIGHTS", align = "c")
```

Table 7: CLOSEST FLIGHTS

tailnum	air_time
NA	NA
N13989	30
N14972	30
N15983	28
N27962	32
N14902	29
N22909	22
N33182	25
N11194	30
N17560	27

5.4.1 Exercises: Items 2, 3, and 4

R's **select()** function, if you include the name of a variable multiple times, it will appear multiple times in the resulting output. For example:

```
select(flights, dep_time, dep_time)
```

```
## # A tibble: 336,776 x 1
##    dep_time
##       <int>
##  1      517
##  2      533
##  3      542
##  4      544
##  5      554
##  6      554
##  7      555
##  8      557
##  9      557
## 10      558
## # i 336,766 more rows
```

This would include the **dep_time** column twice in the resulting output.

The **any_of()** function is used to select columns from a dataframe based on a character vector of column names. It's helpful when you have a vector of column names and want to select only those columns that match any of the names in the vector. For instance:

```
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
```

This code selects columns with names "year," "month," "day," "dep_delay," and "arr_delay" from the **flights** dataframe.

The **contains()** helper function selects columns that contain a specified string in their names. By default, it's case-insensitive. For example:

```
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
```

This code selects columns whose names contain the string "TIME," such as "dep_time" and "arr_time."

5.5.2 Exercises: Items 1 and 2

```
flights_modified <- 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)
```

In this code block:

1. We take the **flights** dataset and pass it through the **%>%** operator.
2. We use the **mutate()** function to add new columns to the data. We're creating two new columns: **dep_time_mins** and **sched_dep_time_mins**.
3. For each of these columns, we're performing calculations to convert time values (stored in HHMM format) into minutes since midnight. We use the **%/%** operation to get the hours and **%** to get the minutes.

After executing this code block, the **flights_modified** dataset will contain the original columns along with the new **dep_time_mins** and **sched_dep_time_mins** columns, representing the departure times (scheduled and actual) in minutes since midnight.

```
library(knitr)
kable(flights_modified[1:10,c(12,4,5,20,21)],caption = "SCHEDULED DEPARTURE TIME", align = "c")
```

Table 8: SCHEDULED DEPARTURE TIME

tailnum	dep_time	sched_dep_time	dep_time_mins	sched_dep_time_mins
N14228	517	515	317	315
N24211	533	529	333	329
N619AA	542	540	342	340
N804JB	544	545	344	345
N668DN	554	600	354	360
N39463	554	558	354	358
N516JB	555	600	355	360
N829AS	557	600	357	360
N593JB	557	600	357	360
N3ALAA	558	600	358	360

```
comparison_result <- flights_modified %>%
  mutate(arr_dep_time_diff = arr_time - dep_time_mins) %>%
  filter(!is.na(air_time) & !is.na(arr_dep_time_diff)) %>%
  select(air_time, arr_dep_time_diff)
```

In this second code block:

1. We take the **`flights_modified`** dataset (the result of the previous block) and pass it through the **`%>%`** operator.
2. We use the **`mutate()`** function to create a new column called **`arr_dep_time_diff`**. We're calculating the difference between the arrival time (**`arr_time`**) and the departure time in minutes since midnight (**`dep_time_mins`**).
3. Next, we use **`filter()`** to remove rows where there are missing values in the **`air_time`** or **`arr_dep_time_diff`** columns.
4. Finally, we use **`select()`** to choose only the **`air_time`** and **`arr_dep_time_diff`** columns.

```
library(knitr)
kable(comparison_result[1:10,c(1,2)],caption = "COMPARISION OF ARRIVES AND DEPARTURES", align = "c")
```

Table 9: COMPARISION OF ARRIVES AND DEPARTURES

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

5.6.7 Exercises: item 1

1. **Median Arrival Delay**: Calculate the median arrival delay for the group of flights. This provides a central value that represents the typical delay experienced upon arrival.

2. **Proportion of Flights with Specific Delays**: Determine the percentage of flights that arrive either 15 minutes early or 15 minutes late, 30 minutes early or 30 minutes late, or 2 hours late. This helps understand the distribution of different delay scenarios within the group.
3. **Average Departure Delay**: Calculate the average departure delay for the group of flights. This gives insight into the average delay that occurs before a flight takes off.
4. **Punctuality Percentage**: Calculate the percentage of flights that are punctual (no arrival delay) and compare it to the percentage of flights that are extremely delayed (2 hours late). This provides an understanding of how often flights are on time versus significantly delayed.
5. **Arrival Delay Distribution**: Create a histogram or density plot showing the distribution of arrival delays across all flights. This visual representation helps identify common delay ranges and outliers.

Question: What's More Important - Arrival Delay or Departure Delay?

This question addresses whether arrival delay or departure delay has a greater impact on the overall flight experience. It depends on various factors. Arrival delay affects passengers' schedules, connecting flights, and ground transportation plans. Departure delay can impact scheduling and cause inconvenience, but it might be easier to manage in some cases compared to arrival delays that can ripple through subsequent plans.

Both aspects are important, but the significance may vary depending on passengers' priorities and the nature of their travel plans.

5.7.1 Exercises: item 2

```
## # 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
```

In this code:

1. We start with the `flights` dataset and use the `%>%` operator to chain subsequent operations.
2. We group the data by aircraft tail number (`tailnum`). This means that subsequent calculations will be performed separately for each aircraft.
3. We use the `summarize()` function to calculate summary statistics for each group (aircraft). Within the `summarize()` function:
   - `total_flights` is calculated using the `n()` function, which gives the total number of flights for each aircraft.
   - `punctual_flights` is calculated using the `sum()` function. It counts the number of flights where the arrival delay (`arr_delay`) is less than or equal to 0 (indicating on-time or early arrivals). The `na.rm = TRUE` argument handles missing values in the `arr_delay` column.
   - `punctuality_percentage` is calculated as the ratio of punctual flights to total flights, multiplied by 100 to get the percentage.

4. After the **summarize()** operation, we use **arrange()** to sort the groups (aircraft) based on their punctuality percentages in ascending order. This means aircraft with the lowest punctuality percentages will appear first.
5. We use **filter()** to remove rows where the **punctuality_percentage** is not available (**NA**).
6. The resulting dataset is assigned to **worst_punctuality**, and it contains aircraft tail numbers, the total number of flights, the number of punctual flights, and the corresponding punctuality percentages.
7. Finally, we display the **worst_punctuality** dataset to see the tail numbers of aircraft with the lowest punctuality percentages.

```r
library(knitr)
kable(worst_punctuality[1:10,c(1,2,3,4)],caption = "WORST PUNCTUALITY TOP", align = "c")
```

Table 10: WORST PUNCTUALITY TOP

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