FirstActivity

Jennyfer Chala

2023-08-26

This R language practice is performed, where you will know aspects of the data to enter, such as flight, punctuality, times and delay. Using the package **nycflights13** and **tidyverse** in R

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

The flight function is loaded into the package **nycflights13** and flight data set with **flights_data**

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

Exercises

5.2.4 Exercises: Items 1 and 2

To resolve, use the **filter()** function and select the rows **flights_data** with the value of the column **arr_delay** which is greater than or equal to 2. when stored on **filtered_de flights**.

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

A table is drawn by **Knitr** taking into account the tail number of the aircraft and the time delay in landing

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

Now in the 2 exercise, using **filter()** to select rows where the value in the dest column is equal to “HOU”. The result is stored in **filtered_hou_flights**.

```
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.

We purchase flights from the dataset and pass them through the operator % >%, allowing sequence operations. Where the function **arrange()** sorts the descending rows according to the column in **s. na(dep_time)**. The column is logical and becomes TRUE if **dep_time** (output time) is absent and FALSE if it is not and 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

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

Flights **dataset** are purchased by passing through the operator % >% by **arrange()** we sort the rows according to the column **arr_delay**. which means that flights with longer delays come first.

```
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

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

The data is taken **flights** by passing through the operator **%>%**. For the function **mutate()** and create a new column called **speed**. Calculating the speed and dividing the column **distance** between the column **air_time**. To give us the speed of each flight. And then the **arrange()** function is used to sort the rows in descending order according to the **speed** column. That flights with the highest speed will appear first.

```
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

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

The data is taken **flights** by passing through the operator **%>%**.The **arrange()** function is then used by sorting the rows in descending order based on the column **distance**. Making flights with longer distances will appear first.

```
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

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

The data is taken **flights** by passing through the operator **%>%**.The function **arrange()** is used by sorting the rows in ascending order according to the column **distance**. Making flights with shorter 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

Add the function **select()** to include the name of a variable several times, appearing in the resulting output

```
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
```

The column **dep_time** is included twice for the output and with the function **any_of()** select columns from a dataframe based on a character vector of the column names. It works when you have a column name vector and you want only to get those columns that match any of the names in the 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
```

```
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
```

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)
```

The **flights** data set is taken and the **%>%** operator is used to process it.

Using the **mutate()** function, new columns are added to the data. In this case, two additional columns are being created: **dep_time_mins** and **sched_dep_time_mins**. For both columns, calculations are performed to

convert time values (stored in HHMM format) to minutes after midnight. For this, the **%/%** operation is used to obtain the hours and **%** to obtain the minutes.

After executing this code snippet, the **flights_modified** dataset will contain the original columns, together with the two new columns, **dep_time_mins** and **sched_dep_time_mins**, representing the output times (programmed and actual) in minutes counted from 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)
```

The data **flights_modified** above is acquired and passed through **%>%**. Where the function **mutate()** is used Creating a new column called **arr_dep_time_diff**. Calculating the difference between arrival time (**arr_time**) and departure time in minutes from midnight (**dep_time_mins**).

Then **filter()** is used to remove rows where values are missing in columns **air_time** or **arr_dep_time_diff**. and at the end **select()** * is used to choose only columns **air_time** and **arr_dep_time_diff**.

```
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

- Calculate the median delay experienced on arrival by the set of flights. This gives us a central value that reflects the typical delay at the time of arrival.
- Determine the percentage of flights arriving with an advance or delay of 15 minutes, 30 minutes or even 2 hours compared to your scheduled schedule. This helps us understand how various delay scenarios are distributed within the flight group.
- Calculate the average delay before departure for the set of flights. This provides information about the average delay before a flight takes off.
- Calculate the percentage of flights that are on time (without delay on arrival) and compare it with the percentage of flights that experience significant delays of 2 hours or more. This gives us an idea of how often flights are on time compared to those with substantial delays.
- Create a density chart that illustrates how arrival delays are distributed across all flights. This visual representation helps us identify common delay ranges and detect outliers.

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

This question seeks to determine which of the two factors, either the delay in arrival or the delay in departure, has the greatest impact on the overall flight experience. This depends on various elements at play. A delay in arrival affects passenger schedules, flight connections and ground transportation plans. Both aspects are crucial, delay and experience but their importance may vary according to 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:

Data is acquired **`flights`** * and can be used with **`%>%`** for their respective operations. Data are collected by aircraft tail number (**`tailnum`**). Subsequent calculations will be performed separately. Then the function **`summary()`** Calculate the summary statistics of each aircraft group. What goes in the function **`summary()`**:

1. **`total_flights`** is calculated with the function **`n()`**, giving the total number of flights for each aircraft.
2. **`punctual_flights`** is calculated with the function **`sum()`**. Counting the number of flights on which the arrival delay (**`arr_delay`**) is less than or equal to 0 indicating the arrivals on time or in advance. The code **`na.rm = TRUE`** determines the missing values in the column **`arr_delay`**.
3. **`punctuality_percentage`** is defined as the relationship between point flights and total flights, multiplied by 100 to obtain the total or necessary percentage.

After **`summary()`**, use **`arrange()`** sorting aircraft according to their punctuality percentages in ascending order. To use **`filter()`** * and remove rows where **`punctuality_percentage`** is not available (***`NA`**) Where the resulting data group is called **`wor_punctuality`**, containing aircraft tail numbers, the total number of flights,

the number of punctual flights and the corresponding percentage of punctuality. Ending in the sample of data **worst_punctuality** where the tail numbers of aircraft are seen with the lowest punctuality percentages.

```
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