FIRST WORK

Alejandro Vergara Rincon 81190

2023-08-17

HOMEWORK

5.2.4 Exercises:

1

The first step is to load the nycflights13 and tidyverse libraries.

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

Then the flight data is assigned to a called variable, in this case the variable "exercise". This allows you to use the variable "exercise" to reference the data in the code.

```
exercise <- nycflights13::flights
```

A new dataset named Data_1 is created by using the filter() function, which is part of the dplyr package within tidyverse. This function allows you to filter the data according to a given condition.

```
Data_1 <- filter(exercise, arr_delay >= "2")
```

The condition used is arr_delay >= "2", which indicates that only flights where the arrival delay (arr_delay) is greater than or equal to 2 minutes are selected.

Table 1: In this table you can see flight 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 this case the kable function was used to display and limit the rows and columns of the table, and as can be seen in the last column "arr_delay" the delay time is => than 2.

2

Flew to Houston (`IAH` or `HOU`)

```
Data_2 <- filter(exercise, dest == "IAH" | dest == "HOU")
```

In this code, we use the OR operator (|) to filter out flights that are bound for Houston, either IAH or HOU. The condition dest == "IAH" | dest == "HOU" ensures that flights that meet either of these two options are included.

Table 2: In this table you can see flight and destination 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
2013	HOU	JFK	38
2013	IAH	LGA	0
2013	HOU	EWR	12
2013	IAH	EWR	19
2013	IAH	LGA	-9

5.3.1 Exercises:

1

```
Data_3 <- flights %>% arrange(desc(is.na(dep_time)))
```

is.na(dep_time) is used as a sorting criterion. is.na() is a function that evaluates whether each value in the dep_time column is a missing value (NA). When sorting according to this evaluation, flights with missing values in the output time (dep_time) will be placed at the top of the resulting data set.

Table 3: In this table you can see flight with missing values

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

year	dep_time	dest	origin	arr_delay
2013	NA	LAX	JFK	NA

2

```
Data_4 <- flights %>% arrange(desc(arr_delay))
```

In this line, another dataset called Data_4 is generated. arrange() is used to sort the flights based on the arr_delay column, which represents the delay in arrival. By adding desc(arr_delay), the instruction is given that flights with longer delays will be placed at the top of the list.

Table 4: In this table you can see the flights sorted by delays

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

3

```
Data_5 <- flights %>% mutate(speed = distance / air_time) %>% arrange(desc(speed))
```

Mutate() is used to add a new column named speed to the dataset. The speed column calculates the speed by dividing the distance by the flight time (air_time). Then, using arrange(), the flights are sorted based on the new speed column in descending order. The fastest flights will appear first in the list.

Table 5: In this table you can see the flights sorted by speed

year	distance	sched_arr_time	speed
2013	762	1937	11.723077
2013	1008	1719	10.838710
2013	594	2226	10.800000
2013	748	2043	10.685714
2013	1035	1917	9.857143
2013	1598	1150	9.400000
2013	1598	438	9.290698
2013	1623	1255	9.274286
2013	1598	36	9.236994
2013	1598	440	9.236994

4.1

```
Data_6 <- flights %>% arrange(desc(distance))
```

In this line, Data_6 is created. arrange() is used to arrange the flights by distance traveled in descending order. This shows the flights that traveled the longest distances first.

Table 6: In this table you can see which flights traveled the longest distances first

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

4.2

```
Data_7 <- flights %>% arrange(distance)
```

Finally, Data_7 is created. arrange() is used to sort the flights by distance traveled in ascending order. This places flights with shorter distances at the beginning of the data set.

Table 7: In this table you can see which flights traveled the shortest distances first

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

5.4.1 Exercises

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

Answer: Variable appears repeatedly in a result

2. 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 using a character vector consisting of column names. In relation to the provided line of code, this function could be really advantageous.

3. Does the result of running the following code surprise you? How do the select helpers deal whit case by default? How can y

```
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 chooses the columns that have "TIME" in their names and also contain the word "TIME" in their cells. The selected columns are renamed as "dep_time", "sched_dep_time", "arr_time", "sched_arr_time" and "air_time" is renamed as "hour_time".

5.5.2 Exercises

1

```
Data_8 <- 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 part, you create a new dataset called Data_8. You use mutate() to add two new columns: dep_time_mins and sched_dep_time_mins. These columns represent the departure time and the scheduled departure time converted to minutes since midnight. The calculation (dep_time %/% 100) * 60 + dep_time %% 100 converts the hour and minutes to total minutes.

Table 8: In this table you can see the information in minutes

year	dest	dep_time_mins	sched_dep_time_mins
2013	IAH	317	315
2013	IAH	333	329
2013	MIA	342	340
2013	BQN	344	345
2013	ATL	354	360

year	dest	dep_time_mins	sched_dep_time_mins
2013	ORD	354	358
2013	FLL	355	360
2013	IAD	357	360
2013	MCO	357	360
2013	ORD	358	360

2

```
Data_9 <- Data_8 %>% 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_time, arr_dep_time_diff)
```

In this section, you create Data_9. you use mutate() to calculate the difference between arrival and departure times converted into minutes (arr_time - dep_time_mins). Then, you use filter() to remove rows that have missing values in air_time or in the time difference between arrival and departure. Finally, select() is used to have only the two columns air_time (flight time) and arr_dep_time_diff (difference between arrival and departure time in minutes).

Table 9: In this table you can see flight information

air_time	arr_time	arr_dep_time_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

1. **Median Arrival Delay:** This involves determining the central value that best represents the typical delay experienced upon flight arrival. The median arrival delay is calculated for a specific group of flights.

2. **Proportion of Flights with Specific Delays:** This analysis focuses on identifying the percentage of flights that arrive either significantly early or late. Delays of 15 minutes, 30 minutes, and 2 hours are considered, providing insights into the distribution of delay occurrences within the flight group.

3. **Average Departure Delay:** This analysis calculates the average delay encountered before a flight's departure. It offers insights into the typical delay experienced before takeoff.

4. **Punctuality Percentage:** The punctuality percentage is determined by calculating the ratio of on-time flights (those with no arrival delay) to the total number of flights. Additionally, this percentage is contrasted with the percentage of flights that experience significant delays (2 hours late). This comparison highlights the disparity between punctual flights and those with substantial delays.

5. **Arrival Delay Distribution:** This analysis involves creating a histogram or density plot that visually presents the distribution of arrival delays across all flights. By visualizing the data, common delay ranges and outliers can be identified, aiding in understanding the overall delay patterns.

5.7.1 Exercises

```
Data_10 <- 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))

Data_10
```

```
## # 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 code snippet calculates the aircraft with the poorest punctuality using the "flights" dataset. Here's how it works step by step:

1. **Grouping by Aircraft:** The code groups the data by the unique tail numbers of aircraft (tailnum).
2. **Summarizing Data:** Within each group (each aircraft), the code calculates three key values:
   - **total_flights:** The total number of flights made by each aircraft, counted using the "n()" function.
   - **punctual_flights:** The count of flights that had punctual or early arrivals (arrival delay less than or equal to 0), calculated using the "sum()" function. The "na.rm = TRUE" argument ensures that missing values are ignored.
   - **punctuality_percentage:** The percentage of punctual flights in relation to total flights. It's calculated by dividing punctual_flights by total_flights and then multiplying by 100.
3. **Sorting and Filtering:** The summarized data is sorted in ascending order based on the punctuality_percentage, meaning the aircraft with the worst punctuality percentages are listed first. Then, the "filter()" function removes rows where punctuality_percentage is not available (NA).

The resulting dataset, named "worst_punctuality," presents the aircraft with the poorest punctuality based on the calculated punctuality percentages.

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