CLASSWORK 1

Juan Sebastian Gonzalez Castillo

2023-08-28

ACTIVITY DEVELOPMENT

1. Download the required libraries like "tidyverse" and "nycflights13" which provides a flight data to analyze, also is necessary download de library "knitr" to display data tables in an attractive and orderly way.
2. CLASSWORK

5.2.4 Exercises:

Item 1: Use the function "filter" for find all flights that had an arrival delay of two or more hours

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
exer<-nycflights13::flights
DELAYS <- filter(exer,arr_delay >="2")
```

The table 1 shows some flights that had an arrival delay

```
library(knitr)
kable(DELAYS[1:10, c(1, 14, 13, 9)],
      caption = "Delays info",
      align = "c")
```

Table 1: Delays info

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

Item 2: Using the function “filter” find all flights that flew to IAH or HOU, for this exercise I selected HOU

```
exer<-nycflights13::flights
HOUSTON<-filter(exer,dest=="HOU")
```

The table 2 shows some flights that flew to HOU

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

Table 2: Destionation info

year	dest	origin
2013	HOU	JFK
2013	HOU	EWR
2013	HOU	EWR
2013	HOU	JFK
2013	HOU	EWR
2013	HOU	JFK
2013	HOU	EWR
2013	HOU	EWR
2013	HOU	JFK
2013	HOU	EWR

5.3.1 Exercises:

Item 1: How could you use `arrange()` to sort all missing values to the start? (Hint: use `is.na()`). That function is used to check the flights with no info.

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

In the table 3 we can see the flights with no information

```
library(knitr)
kable(MISSING[1:10, c(1, 4, 6, 7, 9, 15)],
      caption = "Flights with no Info",
      align = "c")
```

Table 3: Flights with no Info

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

Item 2: Sort `flights` to find the most delayed flights. Find the flights that left earliest. in this case we use de function "desc" to sort the information from most to least

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

In the table 4 we can see the most delayed flights sort from most to least delayed

```
library(knitr)
kable(MORE_DELAYED[1:10, c(1, 2, 3, 9)],
      caption = "Most delayed flights",
      align = "c")
```

Table 4: Most delayed flights

year	month	day	arr_delay
2013	1	9	1272
2013	6	15	1127
2013	1	10	1109
2013	9	20	1007
2013	7	22	989
2013	4	10	931
2013	3	17	915
2013	7	22	895
2013	12	5	878
2013	5	3	875

Item 3: Sort `flights` to find the fastest (highest speed) flights. In this case I associate speed with the flights with less time in the air so I sort the flights from slower to faster.

```
FASTER <- flights %>% arrange(air_time)
```

In the table 5 we can see the fastest flights

```
library(knitr)
kable(FASTER[1:10, c(1, 2, 3, 15)],
      caption = "Fastest flights" ,
      align = "c" )
```

Table 5: Fastest flights

year	month	day	air_time
2013	1	16	20
2013	4	13	20
2013	12	6	21
2013	2	3	21
2013	2	5	21
2013	2	12	21
2013	3	2	21
2013	3	8	21
2013	3	18	21
2013	3	19	21

Item 4: Which flights travelled the farthest? Which travelled the shortest? For this case I organize the table in two ways, first from farthest to shortest and then from shortest to farthest

```
AWAY<-flights %>% arrange(desc(distance))
NEARBY<-flights %>% arrange(distance)
```

The table 6 shows the farthest flights

```
library(knitr)
kable(AWAY[1:10, c(1, 2, 3, 16)],
      caption = "Farthest flights" ,
      align = "c" )
```

Table 6: Farthest flights

year	month	day	distance
2013	1	1	4983
2013	1	2	4983
2013	1	3	4983
2013	1	4	4983
2013	1	5	4983
2013	1	6	4983
2013	1	7	4983
2013	1	8	4983
2013	1	9	4983
2013	1	10	4983

year	month	day	distance

The table 7 shows the shortest flights

```r
library(knitr)
kable(NEARBY[1:10, c(1, 2, 3, 16)],
      caption = "Shortest flights" ,
      align = "c" )
```

Table 7: Shortest flights

year	month	day	distance
2013	7	27	17
2013	1	3	80
2013	1	4	80
2013	1	4	80
2013	1	4	80
2013	1	5	80
2013	1	6	80
2013	1	7	80
2013	1	8	80
2013	1	9	80

5.4.1 Exercises

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

Ans// Multiple copies of that variable will be included in the resulting data frame. In other words, the variable will be duplicated in the output.

Item 3: What does the `any_of()` function do? Why might it be helpful in conjunction with this vector?

Ans// n cases where you have a variable that contains the column names you want to select, using ‘any_ofany_of() can make your code more adaptable and maintainable, since you can easily modify the ‘any_ofany_of()-type variable to make your code more adaptable and maintainable.

Item 4: Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

Ans// This code selects the columns whose names contain the string “TIME” and which contain the text “TIME” in their cells

```r
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

Item 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 utilized to add a pair of new columns: dep_time_mins and sched_dep_time_mins. These columns represent the departure time and scheduled departure time, respectively, in terms of minutes elapsed since midnight. The calculation (dep_time %/% 100) * 60 + dep_time %% 100 converts the provided hours and minutes into a cumulative minute count.

```
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)
```

Table 8 show the two new columns created above

```
library(knitr)
kable(MODIFIED[1:10, c(1, 2, 3, 20, 21)],
      caption = "Time Modified" ,
      align = "c" )
```

Table 8: Time Modified

year	month	day	dep_time_mins	sched_dep_time_mins
2013	1	1	317	315
2013	1	1	333	329
2013	1	1	342	340
2013	1	1	344	345
2013	1	1	354	360
2013	1	1	354	358
2013	1	1	355	360
2013	1	1	357	360
2013	1	1	357	360
2013	1	1	358	360

Item 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?

```
COMPARISON <- 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)
```

Table 9 shows the comparison between the two variables created above

```r
library(knitr)
kable(COMPARISON[1:10, c(1, 2)],
      caption = "Comparison",
      align = "c" )
```

Table 9: Comparison

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: Brainstorm at least 5 different ways to assess the typical delay characteristics of a group of flights.

Ans// Certainly, here are seven different ways to assess the typical delay characteristics of the mentioned flight scenarios:

Average Delay: Calculate the average delay across all flights. For each scenario, this would provide an overview of the typical delay experienced by the flights.

Median Delay: Calculate the median delay for each scenario. This would give a middle point in the distribution of delays and is less influenced by outliers

Longest Delay: Identify the longest delay experienced in each scenario. This can highlight the extreme cases that might have significant impacts.

Impact of Extreme Delays: Calculate the weighted average of delays, considering the impact of extreme delays. For instance, the weighted average might consider not just the duration of the delay but also the frequency of occurrence.

Frequency of On-Time Flights: Calculate the percentage of flights that are on time (not delayed). This can help understand how often delays occur in each scenario.