Classwork1

Elizabeth Daniela Monsalve Forero

2023-08-18

Loading the libraries

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

In this step, the nycflights13 library is loaded, which contains flight-related datasets and the tidyverse library, which is a collection of related packages that facilitate data management and analysis.

5.2.4 Exercises: items 1, and 2

```
flights_d <- nycflights13::flights

Delay2 <- filter(flights_d,arr_delay >="120")

F_HOU <- filter(flights_d,dest =="HOU")
```

In the first line of code it loads the flights dataset from the nycflights13 package into the flights_d variable. This creates a data frame containing information about flights.

Subsequently, the filter() function of the dplyr package is used to filter the flights_d data frame. The function arr_delay >= "120" filters out flights that have an arrival delay of at least 2 hours (120 minutes) and is stored in a variable called Delay2.

```
library(knitr)
kable(Delay2[1:10,c(11,12,9)],caption= "DELAYED FLIGHTS", align = "c")
```

Table 1: DELAYED FLIGHTS

flight	tailnum	arr_delay
1714	N24211	20
1141	N619AA	33
507	N516JB	19
301	N3ALAA	8
194	N29129	7
707	N3DUAA	31
4401	N730MQ	16
3768	N9EAMQ	32

flight	tailnum	arr_delay
303	N532UA	14
135	N635JB	4

Again, the filter() function is used to filter the flights_d data frame. An object called F_HOU is created which will contain the flights with destination (dest) to the airport with the three-letter code "HOU" (Houston).

```
library(knitr)
kable(F_HOU[1:10,c(11,12,14)],caption= "DESTINATION HOUSTON", align = "c")
```

Table 2: DESTINATION HOUSTON

flight	tailnum	dest
625	N239JB	HOU
2596	N773SA	HOU
1066	N778SW	HOU
629	N192JB	HOU
20	N485WN	HOU
625	N192JB	HOU
3294	N950WN	HOU
2529	N901WN	HOU
629	N273JB	HOU
20	N7740A	HOU

5.3.1 Excercises: all items

1

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

The original flights dataset is taken and the operator is used and the arrange(desc(desc(is.na(dep_time))) function is used, which sorts the data frame in descending order (desc()) based on whether the output time (dep_time) is missing or not (is.na(dep_time)). This places rows with absent dep_time values at the beginning of the data frame, while rows with present values are placed after.

```
library(knitr)
kable(missing_flights_sorted[1:10,c(11,12,4)],caption= "MISSING FLIGHTS STORTED", align = "c")
```

Table 3: MISSING FLIGHTS STORTED

flight	tailnum	dep_time
4308	N18120	NA
791	N3EHAA	NA
1925	N3EVAA	NA
125	N618JB	NA

flight	tailnum	dep_time
4352	N10575	NA
4406	N13949	NA
4434	N10575	NA
4935	N759EV	NA
3849	N13550	NA
133	NA	NA

2

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

The arrange(desc(arr_delay)) function is used, which sorts the data frame based on the arrival delay (arr_delay) in descending order. This places the flights with the longest delays at the beginning.

```
library(knitr)
kable(most_delayed[1:10,c(11,12,9)],caption= "MOST DELAYED FLIGHT", align = "c")
```

Table 4: MOST DELAYED FLIGHT

flight	tailnum	arr_delay
51	N384HA	1272
3535	N504MQ	1127
3695	N517MQ	1109
177	N338AA	1007
3075	N665MQ	989
2391	N959DL	931
2119	N927DA	915
2047	N6716C	895
172	N5DMAA	878
3744	N523MQ	875

3

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

In this line, first, use mutate() to add a new column named speed representing the speed of the flight (distance / time in the air). Then, you use the arrange(desc(speed)) function to sort the data frame based on the speed column in descending order. This places the fastest flights at the top.

```
library(knitr)
kable(fastest_flights[1:10,c(11,12,20)],caption= "FASTEST FLIGHT", align = "c")
```

Table 5: FASTEST FLIGHT

flight	tailnum	speed
1499	N666DN	11.723077
4667	N17196	10.838710
4292	N14568	10.800000
3805	N12567	10.685714
1902	N956DL	9.857143
315	N3768	9.400000
707	N779JB	9.290698
936	N5FFAA	9.274286
347	N3773D	9.236994
1503	N571JB	9.236994

4

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

Using the arrange(desc(distance)) function, the data frame is sorted based on the distance of the flight in descending order. This places the longest flights at the top.

```
library(knitr)
kable(farthest_flights[1:10,c(11,12,16)],caption= "FARTHEST FLIGHT", align = "c")
```

Table 6: FARTHEST FLIGHT

flight	tailnum	distance
51	N380HA	4983
51	N380HA	4983
51	N380HA	4983
51	N384HA	4983
51	N381HA	4983
51	N385HA	4983
51	N385HA	4983
51	N389HA	4983
51	N384HA	4983
51	N388HA	4983

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

With the arrange(distance) function, the data frame is sorted according to the distance of the flight in ascending order. This places the shortest flights at the beginning.

```
library(knitr)
kable(shortest_flights[1:10,c(11,12,16)],caption= "SHORTEST FLIGHT", align = "c")
```

Table 7: SHORTEST FLIGHT

flight	tailnum	distance
1632	NA	17
3833	N13989	80
4193	N14972	80
4502	N15983	80
4645	N27962	80
4193	N14902	80
4619	N22909	80
4619	N33182	80
4619	N11194	80
4619	N17560	80

5.4.1 Exercises: items 2, 3, and 4

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

```
v2<- select (flights, year, month, day, dep_time, dep_time)
library(knitr)
kable(v2[1:10,],caption= "DUPLICATED VARIABLE")
```

Table 8: DUPLICATED VARIABLE

year	month	day	dep_time
2013	1	1	517
2013	1	1	533
2013	1	1	542
2013	1	1	544
2013	1	1	554
2013	1	1	554
2013	1	1	555
2013	1	1	557
2013	1	1	557
2013	1	1	558

After generating the table we confirm that the software just skips over the repeated variable, since departure time was only displayed once.

3a What does the any_of() function do?

It allows to select variables from character vectors, like that of all_of. What any_of does is look at variables contained in a character vector without checking for missing variables.

3b Why might it be helpful in conjunction with this vector?

```r
variables <- c("year", "month", "day", "dep_delay", "arr_delay")
vars<-select(flights, any_of(variables))
library(knitr)
kable(vars[1:10,],caption= "VARIABLES")
```

Table 9: VARIABLES

year	month	day	dep_delay	arr_delay
2013	1	1	2	11
2013	1	1	4	20
2013	1	1	2	33
2013	1	1	-1	-18
2013	1	1	-6	-25
2013	1	1	-4	12
2013	1	1	-5	19
2013	1	1	-3	-14
2013	1	1	-3	-8
2013	1	1	-2	8

The any_of() function is useful because it avoids the need to have to constantly retype variable names each time they are applied. By creating a vector that is able to rationalize the variables in an easy and consistent manner

4a Does the result of running the following code surprise you?

```r
time <- select(flights, contains("TIME"))
library(knitr)
kable(time[1:10,])
```

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

Contains("TIME") is a function that is used as an argument inside select to specify that all columns containing the string "TIME" in their names should be selected, thus generating a table composed of variables that share a common characteristic in a simple and fast way.

4b How do the select helpers deal with case by default?

The contains() function can be used to ensure that all the variables belonging to the same category are taken automatically, quickly, and avoiding human errors that may not take any variable into account in a study.

4c How can you change that default?

```
default<-select(flights, contains("TIME", ignore.case = FALSE))
library(knitr)
kable(default[1:10,])
```

In this case, contains("TIME", ignore.case = FALSE) is an argument within the function that specifies that all columns whose names contain the character string "TIME" in its exact form should be selected, without ignoring case differences.

5.5.2 Exercises: items 1, and 2

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.

```
minutes_since_midnight<-transmute(flights,deptime = dep_time/60, schedeptime=sched_dep_time/60)
library(knitr)
kable(minutes_since_midnight[1:10,])
```

deptime	schedeptime
8.616667	8.583333
8.883333	8.816667
9.033333	9.000000
9.066667	9.083333
9.233333	10.000000
9.233333	9.300000
9.250000	10.000000
9.283333	10.000000
9.283333	10.000000
9.300000	10.000000

Transmute is a function used to create new columns or modify existing columns in a dataframe. In this case, two new columns are being created:

deptime = dep_time/60: A column called deptime is created containing the values of the dep_time column (representing the departure time in HHMM format), divided by 60 to convert them to minutes since midnight. This will give you the number of minutes elapsed from midnight to the departure time.

schedeptime = sched_dep_time/60: A column called schedeptime is created containing the values of the sched_dep_time column (representing the scheduled departure time in HHMM format), also divided by 60 to convert them to minutes since midnight. This will give you the number of minutes elapsed from midnight to the scheduled departure time.

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?

```
function1<-flights %>%
mutate(dep_time = (dep_time %/% 100) * 60 + (dep_time %% 100),
         sched_dep_time = (sched_dep_time %/% 100) * 60 + (sched_dep_time %% 100),
         arr_time = (arr_time %/% 100) * 60 + (arr_time %% 100),
         sched_arr_time = (sched_arr_time %/% 100) * 60 + (sched_arr_time %% 100)) %>%
transmute((arr_time - dep_time) %% (60*24) - air_time)
```

The mutate function is used to create or modify columns in the dataframe, in this case the formula is used:

¨variable" = (¨variable" %/% 100) * 60 + (¨variable" %% 100): in which an operation is performed to convert the variable time in HHMM format, resulting in 3 new variables: actual departure time (dep_time),scheduled departure time (sched_dep_time), actual arrival time (arr_time), scheduled arrival time (sched_arr_time).

After the above transformation using mutate, the transmute function is chained. It calculates the difference in minutes between the actual arrival time (arr_time) and the actual departure time (dep_time). Then, the operation modulo (arr_time - dep_time) %% (60*24) is applied to obtain the difference in minutes considering a cycle of 24 hours. Finally, the flight time (air_time) is subtracted from this difference.

```
library(knitr)
kable(function1[1:10,])
```

(arr_time - dep_time)%%(60 * 24) - air_time
-34
-30
61
77
22
-44
40
19
21
-23

5.6.7 Exercises: item 1

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

A flight is 15 minutes early 50% of the time, and 15 minutes late 50% of the time.

A flight is always 10 minutes late.

A flight is 30 minutes early 50% of the time, and 30 minutes late 50% of the time.

99% of the time a flight is on time. 1% of the time it's 2 hours late.

```
fifteen_early <- flights %>%
  filter(arr_delay == -15, na.rm = TRUE)
library(knitr)
kable(fifteen_early[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
715	N651JB	-15
1177	N765US	-15
987	N496UA	-15
5710	N835AS	-15
1972	N319NB	-15
1726	N75425	-15
1047	N643DL	-15
217	N592JB	-15
91	N523JB	-15
927	N432UA	-15

```r
fifteen_late <- flights %>%
  filter(arr_delay == 15, na.rm = TRUE)
library(knitr)
kable(fifteen_late[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
905	N274JB	15
783	N810UA	15
3895	N487WN	15
4180	N13955	15
4935	N678CA	15
1231	N926DL	15
702	N484UA	15
675	N804JB	15
1491	N540UW	15
507	N630JB	15

```r
ten_always <- flights %>%
  filter(arr_delay == 10, na.rm = TRUE)
library(knitr)
kable(ten_always[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
4599	N518MQ	10
850	N978AT	10
59	N336AA	10
219	N273JB	10
675	N654AW	10
120	N713TW	10
743	N426AA	10
4135	N21537	10
1053	N203JB	10
3985	N606MQ	10

```r
thirty_early <- flights %>%
  filter(arr_delay == -30, na.rm = TRUE)
```

```r
library(knitr)
kable(thirty_early[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
1843	N955DL	-30
1519	N24715	-30
143	N534JB	-30
643	N804JB	-30
23	N855VA	-30
642	N557UA	-30
3728	N512MQ	-30
27	N836VA	-30
3830	N820AY	-30
677	N583JB	-30

```r
thirty_late <- flights %>%
  filter(arr_delay == 30, na.rm = TRUE)
library(knitr)
kable(thirty_late[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
1111	N37456	30
766	N957WN	30
4564	N856MQ	30
763	N3CFAA	30
1668	N33262	30
3369	N919XJ	30
3899	N8976E	30
47	N807JB	30
227	N229JB	30
1623	N14102	30

```r
percentage_on_time <- flights %>%
  filter(arr_delay == 0, na.rm = TRUE)
library(knitr)
kable(percentage_on_time[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
27	N535UW	0
269	N308DE	0
1847	N956DL	0
1171	N376NW	0
1757	N545AA	0
4404	N828MQ	0
1280	N26210	0
615	N306JB	0
5675	N15572	0

flight	tailnum	arr_delay
537	N563JB	0

```
two_hours  <- flights %>%
  filter(arr_delay > 120, na.rm = TRUE)
library(knitr)
kable(two_hours[1:10,c(11,12,9)], align = "c")
```

flight	tailnum	arr_delay
4576	N531MQ	137
3944	N942MQ	851
856	N534UA	123
1086	N76502	145
4497	N17984	127
525	N231JB	125
4181	N21197	136
5712	N826AS	123
4092	N16911	123
4622	N504MQ	138

fifteen_early <- flights %>% filter(arr_delay == -15, na.rm = TRUE): In this block, the observations of the flights dataset are filtered to create a new dataset called fifteen_early. Only the observations where the arrival delay (arr_delay) is equal to -15 are selected, and na.rm = TRUE is used to exclude the observations with values that do not apply.

Similarly, the other data subsets are created:

fifteen_late: contains the flights with arrival delay equal to 15. ten_always: Contains the flights with arrival delay equal to 10. thirty_early: Contains flights with arrival delay equal to -30. thirty_late: Contains flights with arrival delay equal to 30. percentage_on_time: Contains flights with arrival delay equal to 0 (i.e. flights that arrived on time). two_hours: Contains flights with arrival delay greater than 120 (i.e. flights with a delay of more than 2 hours). ## 5.6.8

```
    worst_punctuality <- 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))
library(knitr)
kable(worst_punctuality[1:10,])
```

tailnum	total_flights	punctual_flights	punctuality_percentage
N121DE	2	0	0
N136DL	1	0	0
N143DA	1	0	0

tailnum	total_flights	punctual_flights	punctuality_percentage
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

worst_punctuality <- flights: Start taking the original flights dataset.

group_by(tailnum): Groups the observations by the tail number of the aircraft (tailnum).

summarize(…): For each group of flights (aircraft) the following calculations are performed:

total_flights: Calculates the total number of flights associated with the aircraft. punctual_flights: Calculates the number of flights that were punctual or arrived on time (defined as those with an arrival delay equal to or less than 0). punctuality_percentage: Calculates the percentage of on-time flights relative to the total number of flights on the aircraft, multiplied by 100 to get a percentage. arrange(punctuality_percentage): After calculating the metrics the results are sorted in ascending order by punctuality percentage. This means that the planes with the worst punctuality will be at the top.