Tidyverse Classwork

Juan David Ortiz Cortés

2023-08-18

5.2.4 Exercises

To start with this workshop, the first thing that was done was to include the necessary libraries for its development. The libraries that were used were:

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

The dataset used is for flights that departed from New York City in 2013.

Table 1: Small view of nycflights13 dataset.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	555	600	-5	913	854	19	B6
2013	1	1	557	600	-3	709	723	-14	EV
2013	1	1	557	600	-3	838	846	-8	B6
2013	1	1	558	600	-2	753	745	8	AA

Item 1 - Find all flights that:

1. Had an arrival delay of two or more hours: Since the `arr_delay` variable is measured in minutes, flights with an arrival delay of 120 minutes or more are searched for; this is done through the `filter` function.

```
filter1 <- filter(flights, arr_delay >= 120)
```

Table 2: Small view of Filter1.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	811	630	101	1047	830	137	MQ
2013	1	1	848	1835	853	1001	1950	851	MQ
2013	1	1	957	733	144	1056	853	123	UA
2013	1	1	1114	900	134	1447	1222	145	UA
2013	1	1	1505	1310	115	1638	1431	127	EV
2013	1	1	1525	1340	105	1831	1626	125	B6
2013	1	1	1549	1445	64	1912	1656	136	EV
2013	1	1	1558	1359	119	1718	1515	123	EV

2. Flew to Houston (`IAH` or `HOU`): The flights that flew to Houston are those flights where the destination (dest) is either `IAH` or `HOU`; this is done through the `filter` function.

```
filter2 <- filter(flights, dest %in% c("IAH", "HOU"))
```

Table 3: Small view of Filter2.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	623	627	-4	933	932	1	UA
2013	1	1	728	732	-4	1041	1038	3	UA
2013	1	1	739	739	0	1104	1038	26	UA
2013	1	1	908	908	0	1228	1219	9	UA
2013	1	1	1028	1026	2	1350	1339	11	UA
2013	1	1	1044	1045	-1	1352	1351	1	UA

3. Were operated by United, American, or Delta: In the `nycflights13` dataset, the column carrier indicates the airline, but it uses two-character carrier codes.

```
airlines <- airlines
```

Table 4: Small view of Airlines.

carrier	name
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air
OO	SkyWest Airlines Inc.
UA	United Air Lines Inc.
US	US Airways Inc.
VX	Virgin America
WN	Southwest Airlines Co.
YV	Mesa Airlines Inc.

So, the carrier code for Delta is `DL`, for American is `AA`, and for United is `UA`. Using these carriers codes, it is check whether carrier is one of those; this is done through the `filter` function.

```
filter3 <- filter(flights, carrier %in% c("AA", "DL", "UA"))
```

Table 5: Small view of Filter3.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	558	600	-2	753	745	8	AA
2013	1	1	558	600	-2	924	917	7	UA
2013	1	1	558	600	-2	923	937	-14	UA

4. Departed in summer (July, August, and September): The variable `month` has the month, and it is numeric. So, the summer flights are those that departed in months 7 (July), 8 (August), and 9 (September); this is done through the `filter` function.

```
filter4 <- filter(flights, month >= 7, month <= 9)
```

Table 6: Small view of Filter4.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	7	1	1	2029	212	236	2359	157	B6
2013	7	1	2	2359	3	344	344	0	B6
2013	7	1	29	2245	104	151	1	110	B6
2013	7	1	43	2130	193	322	14	188	B6
2013	7	1	44	2150	174	300	100	120	AA
2013	7	1	46	2051	235	304	2358	186	B6
2013	7	1	48	2001	287	308	2305	243	VX
2013	7	1	58	2155	183	335	43	172	B6

5. Arrived more than two hours late, but didn't leave late: Flights that arrived more than two hours late, but didn't leave late will have an arrival delay of more than 120 minutes (`arr_delay > 120`) and a non-positive departure delay (`dep_delay <= 0`); this is done through the `filter` function.

```
filter5 <- filter(flights, arr_delay > 120, dep_delay <= 0)
```

Table 7: Small view of Filter5.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	27	1419	1420	-1	1754	1550	124	MQ
2013	10	7	1350	1350	0	1736	1526	130	EV
2013	10	7	1357	1359	-2	1858	1654	124	AA
2013	10	16	657	700	-3	1258	1056	122	B6
2013	11	1	658	700	-2	1329	1015	194	VX
2013	3	18	1844	1847	-3	39	2219	140	UA
2013	4	17	1635	1640	-5	2049	1845	124	MQ
2013	4	18	558	600	-2	1149	850	179	AA

6. Were delayed by at least an hour, but made up over 30 minutes in flight: If a flight was delayed by at least an hour, then `dep_delay >= 60`. If the flight didn't make up any time in the air, then its arrival would be delayed by the same amount as its departure, meaning `dep_delay == arr_delay`, or alternatively, `dep_delay - arr_delay == 0`. If it makes up over 30 minutes in the air, then the arrival delay must be at least 30 minutes less than the departure delay, which is stated as `dep_delay - arr_delay > 30`; this is done through the `filter` function.

```
filter6 <- filter(flights, dep_delay >= 60, dep_delay - arr_delay > 30)
```

Table 8: Small view of Filter6.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	2205	1720	285	46	2040	246	AA
2013	1	1	2326	2130	116	131	18	73	B6
2013	1	3	1503	1221	162	1803	1555	128	UA
2013	1	3	1839	1700	99	2056	1950	66	AA
2013	1	3	1850	1745	65	2148	2120	28	AA
2013	1	3	1941	1759	102	2246	2139	67	UA
2013	1	3	1950	1845	65	2228	2227	1	B6
2013	1	3	2015	1915	60	2135	2111	24	9E

7. Departed between midnight and 6am (inclusive): In `dep_time`, midnight is represented by `2400`, not `0`. Therefore, this means cannot simply check that `dep_time < 600`, because also have to consider the special case of midnight; this is done through the `filter` function.

```
filter7 <- filter(flights, dep_time <= 600 | dep_time == 2400)
```

Table 9: Small view of Filter7.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	555	600	-5	913	854	19	B6
2013	1	1	557	600	-3	709	723	-14	EV

Item 2 - Another useful dplyr filtering helper is `between()`. What does it do? Can you use it to simplify the code needed to answer the previous challenges?:

Yes, because the expression `between(x, left, right)` is equivalent to `x >= left & x <= right`. So, of the answers in the previous exercise, it could simplify the statement of departed in summer `(month >= 7 & month <= 9)` using the `between()` function; this is done through the `filter` function.

```
filter8 <- filter(flights, between(month, 7, 9))
```

Table 10: Small view of Filter8.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	7	1	1	2029	212	236	2359	157	B6
2013	7	1	2	2359	3	344	344	0	B6
2013	7	1	29	2245	104	151	1	110	B6
2013	7	1	43	2130	193	322	14	188	B6
2013	7	1	44	2150	174	300	100	120	AA
2013	7	1	46	2051	235	304	2358	186	B6

5.3.1 Exercises

Item 1 - How could you use `arrange()` to sort all missing values to the start? (Hint: use `is.na()`):

The `arrange()` function puts NA values last. So, the flights will first be sorted by `desc(is.na(dep_time))`. Since `desc(is.na(dep_time))` is either `TRUE` when `dep_time` is missing, or `FALSE`, when it is not, the rows with missing values of `dep_time` will come first, since `TRUE > FALSE`; this is done through the `arrange` function.

```
arrange1 <- arrange(flights, desc(is.na(dep_time)), dep_time)
```

Table 11: Small view of Arrange1.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	1	NA	1630	NA	NA	1815	NA	EV
2013	1	1	NA	1935	NA	NA	2240	NA	AA
2013	1	1	NA	1500	NA	NA	1825	NA	AA
2013	1	1	NA	600	NA	NA	901	NA	B6
2013	1	2	NA	1540	NA	NA	1747	NA	EV
2013	1	2	NA	1620	NA	NA	1746	NA	EV
2013	1	2	NA	1355	NA	NA	1459	NA	EV
2013	1	2	NA	1420	NA	NA	1644	NA	EV

Item 2 - Sort `flights` to find the most delayed flights. Find the flights that left earliest:

The most delayed flights are found by sorting the table by departure delay, `dep_delay`, in descending order; this is done through the `arrange` function.

```
arrange2 <- arrange(flights, desc(dep_delay))
```

Table 12: Small view of Arrange2.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	9	641	900	1301	1242	1530	1272	HA
2013	6	15	1432	1935	1137	1607	2120	1127	MQ
2013	1	10	1121	1635	1126	1239	1810	1109	MQ
2013	9	20	1139	1845	1014	1457	2210	1007	AA
2013	7	22	845	1600	1005	1044	1815	989	MQ
2013	4	10	1100	1900	960	1342	2211	931	DL
2013	3	17	2321	810	911	135	1020	915	DL
2013	6	27	959	1900	899	1236	2226	850	DL

Then, the most delayed flight was HA 51, JFK to HNL, which was scheduled to leave on January 09, 2013 09:00, but the departure was delayed 1,301 minutes, which is 21 hours, 41 minutes.

Similarly, the earliest departing flight can be found by sorting `dep_delay` in ascending order; this is done through the `arrange` function.

```
arrange3 <- arrange(flights, dep_delay)
```

Table 13: Small view of Arrange3.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	12	7	2040	2123	-43	40	2352	48	B6
2013	2	3	2022	2055	-33	2240	2338	-58	DL
2013	11	10	1408	1440	-32	1549	1559	-10	EV
2013	1	11	1900	1930	-30	2233	2243	-10	DL
2013	1	29	1703	1730	-27	1947	1957	-10	F9
2013	8	9	729	755	-26	1002	955	7	MQ
2013	10	23	1907	1932	-25	2143	2143	0	EV
2013	3	30	2030	2055	-25	2213	2250	-37	MQ

So, the flight that left earliest was flight B6 97 (JFK to DEN) scheduled to depart on December 07, 2013 at 21:23 departed 43 minutes early.

Item 3 - Sort `flights` to find the fastest (highest speed) flights:

"Fastest flight" can be interpreted in two ways. The first as "the flight with the shortest flight time" (arrange4) and the second as "the flight with the highest average forward speed" (arrange5); this is done through the `arrange` function.

```
arrange4 <- head(arrange(flights, air_time))
arrange5 <- head(arrange(flights, desc(distance / air_time)))
```

Table 14: Small view of Arrange4.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	16	1355	1315	40	1442	1411	31	EV
2013	4	13	537	527	10	622	628	-6	EV
2013	12	6	922	851	31	1021	954	27	EV
2013	2	3	2153	2129	24	2247	2224	23	EV
2013	2	5	1303	1315	-12	1342	1411	-29	EV
2013	2	12	2123	2130	-7	2211	2225	-14	EV

Table 15: Small view of Arrange5.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	5	25	1709	1700	9	1923	1937	-14	DL
2013	7	2	1558	1513	45	1745	1719	26	EV
2013	5	13	2040	2025	15	2225	2226	-1	EV
2013	3	23	1914	1910	4	2045	2043	2	EV
2013	1	12	1559	1600	-1	1849	1917	-28	DL
2013	11	17	650	655	-5	1059	1150	-51	DL

Item 4 - Which flights travelled the farthest? Which travelled the shortest?:

The terms "farthest" and "shortest" could refer to the time of the flight instead of the distance. The farthest and shortest flights by can be found by sorting by the air_time column. The farthest flights by airtime are the following; this is done through the `arrange` function.

```
arrange6 <- arrange(flights, desc(air_time))
```

Table 16: Small view of Arrange6.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	3	17	1337	1335	2	1937	1836	61	UA
2013	2	6	853	900	-7	1542	1540	2	HA
2013	3	15	1001	1000	1	1551	1530	21	HA
2013	3	17	1006	1000	6	1607	1530	37	HA
2013	3	16	1001	1000	1	1544	1530	14	HA
2013	2	5	900	900	0	1555	1540	15	HA
2013	11	12	936	930	6	1630	1530	60	UA
2013	3	14	958	1000	-2	1542	1530	12	HA

On the other hand, the shortest flights by airtime are the following; this is done through the `arrange` function.

```
arrange7 <- arrange(flights, air_time)
```

Table 17: Small view of Arrange7.

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
2013	1	16	1355	1315	40	1442	1411	31	EV
2013	4	13	537	527	10	622	628	-6	EV
2013	12	6	922	851	31	1021	954	27	EV
2013	2	3	2153	2129	24	2247	2224	23	EV
2013	2	5	1303	1315	-12	1342	1411	-29	EV
2013	2	12	2123	2130	-7	2211	2225	-14	EV
2013	3	2	1450	1500	-10	1547	1608	-21	US
2013	3	8	2026	1935	51	2131	2056	35	9E

5.4.1 Exercises

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

The `select()` call ignores the duplication. So, any duplicated variables are only included once, in the first location they appear; this is done through the `select` function.

```
select1 <- select(flights, year, month, day, year, year)
```

Table 18: Small view of Select1.

year	month	day
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1
2013	1	1

Item 3 - 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")
```

The `any_of()` function selects variables with a character vector rather than unquoted variable name arguments. This function is useful because it is easier to programmatically generate character vectors with variable names than to generate unquoted variable names, which are easier to type; this is done through the `select` function.

```
select2 <- select(flights, any_of(vars))
```

Table 19: Small view of Select2.

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

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?:

```
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
```

It's surprising since, the default behavior for `contains()` is to ignore case. This is important because users searching for variable names probably have a better sense of the letters in the variable than their capitalization. On the other hand, some of database engines have case insensitive column names, so making functions that match variable names case insensitive by default will make the behavior of `select()` consistent regardless of whether the table is stored as an R data frame or in a database. Finally, to change the behavior it is necessary to add the argument `ignore.case = FALSE`; this is done through the `select` function.

```
select3 <- select(flights, contains("TIME", ignore.case = FALSE))
```

Table: Small view of Select3.

As can be seen, Table does not appear.

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:

It could define a function `time2mins()`, which converts a vector of times in from the format used in flights to minutes since midnight.

```
time2mins <- function(x) {(x %/% 100 * 60 + x %% 100) %% 1440}
```

Using the function, the previous code simplifies to the following; this is done through the `mutate` function.

```
mutate1 <- mutate(flights, dep_time_mins = time2mins(dep_time),
                  sched_dep_time_mins = time2mins(sched_dep_time))
```

Additionally, to show only the relevant columns do the following; this is done through the `select` function.

```
select4 <- select(mutate1, dep_time, dep_time_mins, sched_dep_time, sched_dep_time_mins)
```

Table 20: Small view of Select4.

dep_time	dep_time_mins	sched_dep_time	sched_dep_time_mins
517	317	515	315
533	333	529	329
542	342	540	340
544	344	545	345
554	354	600	360
554	354	558	358
555	355	600	360
557	357	600	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?:

I expect that `air_time` is the difference between the arrival `(arr_time)` and departure times `(dep_time)`. In other words, `air_time = arr_time - dep_time`. What I can see is that `air_time` is not equal to `arr_time - dep_time`, because: 1. The flight passes midnight, so `arr_time < dep_time`. In these cases, the difference in airtime should be by 24 hours (1,440 minutes) and 2. The flight crosses time zones, and the total `air time` will be off by hours (multiples of 60). All flights in flights departed from New York City and are domestic flights in the US. This means that flights will all be to the same or more westerly time zones. Given the time-zones in the US, the differences due to time-zone should be 60 minutes (Central) 120 minutes (Mountain), 180 minutes (Pacific), 240 minutes (Alaska), or 300 minutes (Hawaii). Finally, To fix these time-zone issues, I would want to convert all the times to a date-time to handle overnight flights, and from local time to a common time zone, most likely UTC, to handle flights crossing time-zones, leaning on that, the relationship between `air_time`, `arr_time`, and `dep_time` is `air_time <= arr_time - dep_time`, supposing that the time zones of `arr_time` and `dep_time` are in the same time zone.

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.***

Which is more important: arrival delay or departure delay?:

- What this question gets at is a fundamental question of data analysis: the cost function. As analysts, the reason it is interested in flight delay because it is costly to passengers. But it is worth thinking carefully about how it is costly and use that information in ranking and measuring these scenarios.
- In many scenarios, arrival delay is more important. In most cases, being arriving late is more costly to the passenger since it could disrupt the next stages of their travel, such as connecting flights or scheduled meetings.
- If a departure is delayed without affecting the arrival time, this delay will not have those affects plans nor does it affect the total time spent traveling.
- This delay could be beneficial, if less time is spent in the cramped confines of the airplane itself, or a negative, if that delayed time is still spent in the cramped confines of the airplane on the runway.
- Variation in arrival time is worse than consistency. If a flight is always 30 minutes late and that delay is known, then it is as if the arrival time is that delayed time. The traveler could easily plan for this. But higher variation in flight times makes it harder to plan.

5.7.1 Exercises

Item 2 - Which plane (`tailnum`) has the worst on-time record?:

For a more exact result, it is decided to choose the plane with the worst time record that made at least 20 flights. Through the average number of minutes late, it obtain the following.

```
groupedmutate <- flights %>% filter(!is.na(arr_delay)) %>%
group_by(tailnum) %>% summarise(arr_delay = mean(arr_delay), n = n())%>%
filter(n >= 20) %>% filter(min_rank(desc(arr_delay)) == 1)
```

Table 21: Small view of Groupedmutate.

tailnum	arr_delay	n
N203FR	59.12195	41