

Tables manipulation II

M1 MIDS/MFA/LOGOS

[Université Paris Cité](#)

Année 2024

[Course Homepage](#)

[Moodle](#)



! Objectives

Setup

We will use the following packages. If needed, we install them.

```
old_theme <- theme_set(theme_minimal())
```

Check [nycflights13](#) for any explanation concerning the tables and their columns.

Data loading

```
flights <- nycflights13::flights
weather <- nycflights13::weather
airports <- nycflights13::airports
airlines <- nycflights13::airlines
planes <- nycflights13::planes
```

```
con <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
flights_lite <- copy_to(con, nycflights13::flights)
airports_lite <- copy_to(con, nycflights13::airports)
planes_lite <- copy_to(con, nycflights13::planes)
weather_lite <- copy_to(con, nycflights13::weather)
airlines_lite <- copy_to(con, nycflights13::airlines)
```

```
flights_lite |>
  select(contains("delay")) |>
  show_query()
```

<SQL>

```
SELECT `dep_delay`, `arr_delay`
FROM `nycflights13::flights`
```

View data in spreadsheet style.

```
View(flights)
```

Ask for help about table `flights`

First Queries (the dplyr way)

Find all flights that

- Had an arrival delay of two or more hours
- Flew to Houston (IAH or HOU)
- Were operated by United, American, or Delta

💡 Package `stringr` could be useful.

```
airlines |>
  filter(stringr::str_starts(name, "United") |
         stringr::str_starts(name, "American") |
         stringr::str_starts(name, "Delta"))

# A tibble: 3 x 2
  carrier name
  <chr>   <chr>
1 AA      American Airlines Inc.
2 DL      Delta Air Lines Inc.
3 UA      United Air Lines Inc.

airlines |>
  filter(stringr::str_detect(name, ("United|American|Delta"))) |>
  pull(carrier)

[1] "AA" "DL" "UA"

#| eval: false
airlines_lite |>
  filter(stringr::str_starts(name, "United") |
         stringr::str_starts(name, "American") |
         stringr::str_starts(name, "Delta")) |>
  show_query()

SELECT *
FROM `nycflights13::airlines`
WHERE "name" LIKE 'United%' OR
      "name" LIKE 'American%' OR
      "name" LIKE 'Delta%' ;

stringr is part of tidyverse
```

- Departed in summer (July, August, and September)

! When manipulating temporal information (date, time, duration), keep an eye on [what lubridate offers](#). The API closely parallels what RDMS and Python offer.

- Arrived more than two hours late, but didn't leave late
- Were delayed by at least an hour, but made up over 30 minutes in flight
- Departed between midnight and 6am (inclusive)

! Read [filter\(\) in R for Data Science 1st Ed](#)
Read [Chapter Transform in R for Data Science 2nd Ed](#)

Missing data

- How many flights per `origin` have a missing `dep_time`?
- What other variables are missing?

! The introduction to `tidyselect` is a must read.

- What might these rows with missing data represent?

```
not_cancelled <- flights |>
  filter(!is.na(dep_time))
```

- More questions: for each column in `flight` report the number of missing values.

Arrange

- How could you use `arrange()` to sort all missing values to the start? (Hint: use `is.na()`).
- Sort flights to find the most delayed flights.
- Pick the ten most delayed flights (with finite `dep_delay`)
- Find the flights that left earliest.
- Sort flights to find the fastest (highest speed) flights.
- Which flights travelled the farthest?

i The database provides all we need with columns `distance` and `air_time`. Otherwise, with the positions of airports from table `airports`, we should be able to compute distances using :

‘Haversine’ formula.

https://en.wikipedia.org/wiki/Haversine_formula

- Which travelled the shortest?

Projection

- Brainstorm as many ways as possible to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from `flights`.
- What happens if you include the name of a variable multiple times in a `select()` call?
- 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")
```

- Does the result of running the following code surprise you?

```
select(
  flights,
  contains("TIME", ignore.case = TRUE)) |>
  head()
```

```
# A tibble: 6 x 6
```

```
  dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
    <int>         <int>    <int>         <int>      <dbl> <dtm>
```

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

- How do the select helpers deal with case by default?
- How can you change that default?

Mutations

- 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.
- 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?
- Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?
- Find the 10 most delayed flights using a ranking function. How do you want to handle ties?

Carefully read the documentation for `min_rank()`.

Windowed rank functions.

Aggregations

- 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 10% 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.

```
flights |>
  group_by(dest) |>
  summarise(n_cancelled = sum(is.na(dep_time)))
```

```
# A tibble: 105 x 2
  dest   n_cancelled
  <chr>         <int>
1 ABQ             0
2 ACK             0
3 ALB            20
4 ANC             0
5 ATL           317
6 AUS            21
7 AVL            12
8 BDL            31
9 BGR            15
10 BHM           25
# i 95 more rows
```

```
flights_lite |>
  group_by(dest) |>
  summarise(n_cancelled = sum(is.na(dep_time))) |>
  show_query()
```

Warning: Missing values are always removed in SQL aggregation functions.

Use ``na.rm = TRUE`` to silence this warning

This warning is displayed once every 8 hours.

<SQL>

```
SELECT `dest`, SUM(`dep_time` IS NULL) AS `n_cancelled`
FROM `nycflights13::flights`
GROUP BY `dest`
```

- Which is more important: arrival delay or departure delay?
- Come up with another approach that will give you the same output as `not_cancelled |> count(dest)` and `(without using count())`.
- Our definition of cancelled flights `(is.na(dep_delay) | is.na(arr_delay))` is slightly suboptimal. Why? Which is the most important column?
- Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?
- Which carrier has the worst delays?

Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about `flights |> group_by(carrier, dest) |> summarise(n())`)

- What does the `sort` argument to `count()` do. When might you use it?

Miscellanea

- Which `carriers` serve all destination airports (in the table) ?
- Refer back to the lists of useful mutate and filtering functions.
- Describe how each operation changes when you combine it with grouping.
- Which plane (`tailnum`) has the worst on-time record amongst planes with at least ten flights?
- What time of day should you fly if you want to avoid delays as much as possible?
- For each destination, compute the total minutes of delay.
- For each flight, compute the proportion of the total positive arrival delays for its destination.

Using `dplyr`, it is easy. See [A second look at group_by](#)

- Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using `lag()`, explore how the delay of a flight is related to the delay of the immediately preceding flight.

i `lag()` is an example of *window* function. If we were using SQL, we would define a WINDOW using an expression like

```
WINDOW w As (PARTITION BY origin ORDER BY year, month, day, sched_dep_time)
```

Something still needs fixing here: some flights never took off (`is.na(dep_time)`). Should they be sided out? assigned an infinite departure delay?

- Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time of a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

Consider all flights with average speed above 950km/h as suspicious.

Let us visualize destinations and origins of the speedy flights.

- Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.
- For each plane, count the number of flights before the first delay greater than 1 hour.

i Assume a plane is characterized by `tailnum`. Some flights have no `tailnum`. We ignore them.

References

- [Data transformation cheatsheet](#)
- [R4Data Science Tidy](#)
- [Benchmarking](#)
- [dplyr and vctrs](#)
- [Posts on dplyr](#)
- [Window functions on dplyr](#)

<https://www.youtube.com/watch?v=Ue08LVuk790>