

# 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

### i Solution

```
?flights  
  
flights |>  
glimpse()  
  
Rows: 336,776  
Columns: 19  
$ year <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~  
$ month <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
$ day <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
$ dep_time <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, ~  
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, ~  
$ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -2, -2, -2, -2, -1~  
$ arr_time <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849, ~  
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851, ~  
$ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~  
$ carrier <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~  
$ flight <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~  
$ tailnum <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~  
$ origin <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA", ~  
$ dest <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD", ~  
$ air_time <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~  
$ distance <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~  
$ hour <dbl> 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~  
$ minute <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~  
$ time_hour <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~  
  
airports |>  
glimpse()  
  
Rows: 1,458  
Columns: 8  
$ faa <chr> "04G", "06A", "06C", "06N", "09J", "0A9", "0G6", "0P2", "~  
$ name <chr> "Lansdowne Airport", "Moton Field Municipal Airport", "Schaumbur~  
$ lat <dbl> 41.13047, 32.46057, 41.98934, 41.43191, 31.07447, 36.37122, 41.4~  
$ lon <dbl> -80.61958, -85.68003, -88.10124, -74.39156, -81.42778, -82.17342~  
$ alt <dbl> 1044, 264, 801, 523, 11, 1593, 730, 492, 1000, 108, 409, 875, 10~  
$ tz <dbl> -5, -6, -6, -5, -5, -5, -5, -8, -5, -6, -5, -5, -5, -5, ~  
$ dst <chr> "A", "A", "A", "A", "A", "A", "A", "U", "A", "A", "U", "A", "A", ~  
$ tzone <chr> "America/New_York", "America/Chicago", "America/Chicago", "Ameri~
```

## First Queries (the dplyr way)

Find all flights that

- Had an arrival delay of two or more hours

**i** Solution

```
flights |>
  filter(arr_delay >= 120) |>
  nrow()

[1] 10200

flights_lite |>
  filter(arr_delay >= 120) |>
  count() |>
  show_query()

<SQL>
SELECT COUNT(*) AS `n`
FROM (
  SELECT `nycflights13::flights`.*
  FROM `nycflights13::flights`
  WHERE (`arr_delay` >= 120.0)
) AS `q01`
```

**i** Solution

We can translate the dplyr pipeline into an SQL query.

```
flights_lite |>
  filter(arr_delay >= 120) |>
  show_query()

<SQL>
SELECT `nycflights13::flights`.*
FROM `nycflights13::flights`
WHERE (`arr_delay` >= 120.0)
```

We can even get some explanations

```
flights_lite |>
  filter(arr_delay >= 120) |>
  explain()
```

- Flew to Houston (IAH or HOU)

**i** Solution

```
flights |>
  filter(dest %in% c("HOU", "IAH")) |>
  nrow()

[1] 9313

flights |>
  filter(dest == "HOU" | dest == "IAH") |>
  count()

# A tibble: 1 x 1
      n
  <int>
1 9313
```

### i Solution

```
flights_lite |>
  filter(dest %in% c("HOU", "IAH")) |>
  show_query()

<SQL>
SELECT `nycflights13::flights`.*
FROM `nycflights13::flights`
WHERE (`dest` IN ('HOU', 'IAH'))
```

### i Solution

```
flights_lite |>
  filter(dest == "HOU" | dest == "IAH") |>
  show_query()

<SQL>
SELECT `nycflights13::flights`.*
FROM `nycflights13::flights`
WHERE (`dest` = 'HOU' OR `dest` = 'IAH')
```

- 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



## i Solution

We build on the tip above to extract the matching airlines codes.  
We may proceed using a *subquery*

```
flights |>
  filter(carrier %in% (
    airlines |>
      filter(stringr::str_detect(name, ("United|American|Delta"))) |>
      pluck("carrier")
  )
) |>
head(6)

# A tibble: 6 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>     <int>          <int>     <dbl>     <int>          <int>
1 2013     1     1      517            515       2     830          819
2 2013     1     1      533            529       4     850          830
3 2013     1     1      542            540       2     923          850
4 2013     1     1      554            600      -6     812          837
5 2013     1     1      554            558      -4     740          728
6 2013     1     1      558            600      -2     753          745
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>

flights_lite |>
  filter(carrier %in% c("AA", "DL", "UA")) |>
  show_query()

<SQL>
SELECT `nycflights13::flights`.*
FROM `nycflights13::flights`
WHERE (`carrier` IN ('AA', 'DL', 'UA'))

SELECT *
FROM `nycflights13::flights`
WHERE carrier IN (SELECT carrier
  FROM `nycflights13::airlines`
  WHERE "name" LIKE 'United%' OR
        "name" LIKE 'American%' OR
        "name" LIKE 'Delta%')
) ;
```

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

### i Solution

```
flights |>
  filter(month %in% c(7,8,9)) |>
  count()

# A tibble: 1 x 1
  n
  <int>
1 86326

A more ambitious (and sustainable) approach relies on the date/time manipulation function from lubridate

flights |>
  filter(lubridate::month(time_hour) %in% 7:9) |>
  head()

# A tibble: 6 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
1 2013     7     1       1           2029      212     236        2359
2 2013     7     1       2           2359       3     344        344
3 2013     7     1      29           2245      104     151        1
4 2013     7     1      43           2130      193     322        14
5 2013     7     1      44           2150      174     300        100
6 2013     7     1      46           2051      235     304        2358
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
# tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
# hour <dbl>, minute <dbl>, time_hour <dttm>
or even

my_locale <- 'en_US.UTF-8'

flights |>
  filter(lubridate::month(time_hour, label=T, abbr=F,
                           locale=my_locale) %in%
         c('juillet', 'August', 'September')) |>
  head()

# A tibble: 6 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
1 2013     8     1      12           2130      162     257        14
2 2013     8     1      12           2359       13     349        350
3 2013     8     1      22           2146      156     255        30
4 2013     8     1      26           2051      215     333        2358
5 2013     8     1      32           2359      33     420        344
6 2013     8     1      33           2231      122     412        226
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
# tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
# hour <dbl>, minute <dbl>, time_hour <dttm>
```

! 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

### i Solution

```
flights |>
  filter(arr_delay >= 120, dep_delay <= 0) |>
  count()

# A tibble: 1 x 1
      n
  <int>
1     29
```

- Were delayed by at least an hour, but made up over 30 minutes in flight

### i Solution

```
flights |>
  filter(dep_delay > 60,
        arr_delay < dep_delay -30) |>
  count()

# A tibble: 1 x 1
      n
  <int>
1 1819
```

- Departed between midnight and 6am (inclusive)

**i** Solution

```
flights |>
  filter(dep_time<=600, dep_time>0) |>
  head()

# A tibble: 6 x 19
# ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>

flights |>
  filter(lubridate::hour(time_hour)<=5 | (
    lubridate::hour(time_hour)==6 & minute(time_hour)==0)) |>
  head()

# A tibble: 6 x 19
# ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

! 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`?

## i Solution

```
flights |>
  filter(is.na(dep_time)) |>
  count(by=origin)

# A tibble: 3 x 2
  by      n
  <chr> <int>
1 EWR     3239
2 JFK     1863
3 LGA     3153

flights_lite |>
  filter(is.na(dep_time)) |>
  count(by=origin) |>
  show_query()

<SQL>
SELECT `by`, COUNT(*) AS `n`
FROM (
  SELECT `nycflights13::flights`.* , `origin` AS `by`
  FROM `nycflights13::flights`
  WHERE ((`dep_time` IS NULL))
) AS `q01`
GROUP BY `by`
```

Not far from a spontaneous answer! We could obtain the latter anyway.

```
flights_lite |>
  filter(is.na(dep_time)) |>
  group_by(origin) |>
  summarise(n=n()) |>
  show_query()

<SQL>
SELECT `origin`, COUNT(*) AS `n`
FROM (
  SELECT `nycflights13::flights`.*
  FROM `nycflights13::flights`
  WHERE ((`dep_time` IS NULL))
) AS `q01`
GROUP BY `origin`
```

Note that combining `dplyr` verbs and pipes (`|>` or `|>`) provides a much more readable and modular approach than vanilla SQL.

- Using `dplyr` and pipe, the order in which operations are executed is clear and obvious. In SQL, it is counterintuitive.
- 

In table `flights`, 8255 (`nrow(filter(flights, is.na(dep_time)))`) rows have a missing `dep_time` field.

- What other variables are missing?

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

**i Solution**

```
flights |>
  filter(is.na(dep_time)) |>
  summarise(across(everything(), ~ all(is.na(.)))) |>
  pivot_longer(cols = everything()) |>
  filter(value) |>
  select(name)

# A tibble: 5 x 1
  name
  <chr>
1 dep_time
2 dep_delay
3 arr_time
4 arr_delay
5 air_time

flights_lite |>
  filter(is.na(dep_time)) |>
  show_query()

<SQL>
SELECT `nycflights13::flights`.*
FROM `nycflights13::flights`
WHERE ((`dep_time` IS NULL))
```

- What might these rows with missing data represent?

All the information you can only get if the flight did take off.

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

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

**i Solution**

```
flights |>
  summarise(across(everything(), ~ sum(is.na(.)))) |>
  pivot_longer(cols = everything()) |>
  filter(value > 0) |>
  arrange(desc(value))

# A tibble: 6 x 2
  name      value
  <chr>    <int>
1 arr_delay 9430
2 air_time   9430
3 arr_time   8713
4 dep_time   8255
5 dep_delay  8255
6 tailnum    2512
```

**i Solution**

```
flights |>
  skimr::skim()
```

Table 2: Data summary

Name	flights
Number of rows	336776
Number of columns	19
Column type frequency:	
character	4
numeric	14
POSIXct	1
Group variables	None

**Variable type: character**

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
carrier	0	1.00	2	2	0	16	0
tailnum	2512	0.99	5	6	0	4043	0
origin	0	1.00	3	3	0	3	0
dest	0	1.00	3	3	0	105	0

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
year	0	1.00	2013.00	0.00	2013	2013	2013	2013	2013	2013
month	0	1.00	6.55	3.41	1	4	7	10	12	
day	0	1.00	15.71	8.77	1	8	16	23	31	
dep_time	8255	0.98	1349.11	488.28	1	907	1401	1744	2400	
sched_dep_time	0	1.00	1344.25	467.34	106	906	1359	1729	2359	
dep_delay	8255	0.98	12.64	40.21	-43	-5	-2	11	1301	
arr_time	8713	0.97	1502.05	533.26	1	1104	1535	1940	2400	
sched_arr_time	0	1.00	1536.38	497.46	1	1124	1556	1945	2359	
arr_delay	9430	0.97	6.90	44.63	-86	-17	-5	14	1272	
flight	0	1.00	1971.92	1632.47	1	553	1496	3465	8500	
air_time	9430	0.97	150.69	93.69	20	82	129	192	695	
distance	0	1.00	1039.91	733.23	17	502	872	1389	4983	
hour	0	1.00	13.18	4.66	1	9	13	17	23	
minute	0	1.00	26.23	19.30	0	8	29	44	59	

**Variable type: POSIXct**

skim_variable	missing	complete_rate	max	median	n_unique
time_hour	0	1	2013-01-01 05:00:00	2013-12-31 23:00:00	2013-07-03 10:00:00

## Arrange

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

### i Solution

- Sort flights to find the most delayed flights.

### i Solution

```
flights |>
  arrange(desc(dep_delay)) |>
  select(dep_delay, arr_delay, everything())  
  
# A tibble: 336,776 x 19  
  dep_delay arr_delay year month day dep_time sched_dep_time arr_time  
    <dbl>     <dbl> <int> <int> <int>      <int>          <int>     <int>  
1     1301      1272  2013     1     9       641            900     1242  
2     1137      1127  2013     6    15      1432           1935     1607  
3     1126      1109  2013     1    10      1121           1635     1239  
4     1014      1007  2013     9    20      1139           1845     1457  
5     1005       989  2013     7    22       845            1600     1044  
6      960       931  2013     4    10      1100           1900     1342  
7      911       915  2013     3    17      2321            810      135  
8      899       850  2013     6    27       959            1900     1236  
9      898       895  2013     7    22      2257            759      121  
10     896       878  2013    12     5       756            1700     1058  
# i 336,766 more rows  
# i 11 more variables: sched_arr_time <int>, carrier <chr>, flight <int>,  
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,  
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- Pick the ten most delayed flights (with finite `dep_delay`)

**i** Solution

```
flights |>
  slice_max(order_by=dep_delay, n = 10, na_rm = T)

# A tibble: 10 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>     <int>        <int>    <dbl>    <int>        <int>
1 2013     1     9      641          900 1301 1242        1530
2 2013     6    15     1432         1935 1137 1607        2120
3 2013     1    10     1121         1635 1126 1239        1810
4 2013     9    20     1139         1845 1014 1457        2210
5 2013     7    22      845          1600 1005 1044        1815
6 2013     4    10     1100          1900  960 1342        2211
7 2013     3    17     2321          810  911 135         1020
8 2013     6    27      959          1900  899 1236        2226
9 2013     7    22     2257          759  898 121         1026
10 2013    12     5      756          1700  896 1058        2020
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- Find the flights that left earliest.

**i** Solution

```
flights |>
  arrange(dep_time) |>
  select(dest, origin, dep_time)

# A tibble: 336,776 x 3
  dest   origin dep_time
  <chr>  <chr>    <int>
1 SYR    JFK       1
2 MDW    LGA       1
3 SJU    JFK       1
4 BQN    JFK       1
5 SJU    JFK       1
6 SJU    JFK       1
7 BQN    JFK       1
8 MDW    LGA       1
9 PWM    JFK       1
10 PSE   JFK       1
# i 336,766 more rows
```

- Sort flights to find the fastest (highest speed) flights.



## i Solution

```
flights |>
  filter(!is.na(air_time)) |>
  mutate(speed = distance/air_time) |>
  arrange(desc(speed)) |>
  select(speed, distance, air_time, origin, dest, everything())
```

# A tibble: 327,346 x 20

	speed	distance	air_time	origin	dest	year	month	day	dep_time	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<int>	<int>	<int>	<int>
1	11.7	762	65	LGA	ATL	2013	5	25	1709									
2	10.8	1008	93	EWR	MSP	2013	7	2	1558									
3	10.8	594	55	EWR	GSP	2013	5	13	2040									
4	10.7	748	70	EWR	BNA	2013	3	23	1914									
5	9.86	1035	105	LGA	PBI	2013	1	12	1559									
6	9.4	1598	170	JFK	SJU	2013	11	17	650									
7	9.29	1598	172	JFK	SJU	2013	2	21	2355									
8	9.27	1623	175	JFK	STT	2013	11	17	759									
9	9.24	1598	173	JFK	SJU	2013	11	16	2003									
10	9.24	1598	173	JFK	SJU	2013	11	16	2349									

# i 327,336 more rows

# i 11 more variables: sched\_dep\_time <int>, dep\_delay <dbl>, arr\_time <int>,  
# sched\_arr\_time <int>, arr\_delay <dbl>, carrier <chr>, flight <int>,  
# tailnum <chr>, hour <dbl>, minute <dbl>, time\_hour <dttm>

This is an overkill. We are sorting in order to perform maximum selection. Recall that sorting requires more comparisons than selection. The comparison between sorting and selecting with respect to data shuffling is even less favourable.

```
flights |>
  filter(!is.na(air_time)) |>
  mutate(speed = distance/air_time) |>
  slice_max(n=10, speed) |>
  select(speed, distance, air_time, origin, dest, everything())
```

# A tibble: 15 x 20

	speed	distance	air_time	origin	dest	year	month	day	dep_time	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<int>	<int>	<int>	<int>
1	11.7	762	65	LGA	ATL	2013	5	25	1709									
2	10.8	1008	93	EWR	MSP	2013	7	2	1558									
3	10.8	594	55	EWR	GSP	2013	5	13	2040									
4	10.7	748	70	EWR	BNA	2013	3	23	1914									
5	9.86	1035	105	LGA	PBI	2013	1	12	1559									
6	9.4	1598	170	JFK	SJU	2013	11	17	650									
7	9.29	1598	172	JFK	SJU	2013	2	21	2355									
8	9.27	1623	175	JFK	STT	2013	11	17	759									
9	9.24	1598	173	JFK	SJU	2013	11	16	2003									
10	9.24	1598	173	JFK	SJU	2013	11	16	2349									
11	9.24	1598	173	JFK	SJU	2013	11	17	851									
12	9.24	1598	173	JFK	SJU	2013	11	17	1926									
13	9.24	1598	173	JFK	SJU	2013	12	5	1858									
14	9.24	1598	173	JFK	SJU	2013	2	10	1658									
15	9.24	1598	173	JFK	SJU	2013	2	10	1958									

# i 11 more variables: sched\_dep\_time <int>, dep\_delay <dbl>, arr\_time <int>,  
# sched\_arr\_time <int>, arr\_delay <dbl>, carrier <chr>, flight <int>,  
# tailnum <chr>, hour <dbl>, minute <dbl>, time\_hour <dttm>

- 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](https://en.wikipedia.org/wiki/Haversine_formula)

### **i** Solution

```
flights |>
  arrange(desc(distance)) |>
  distinct(distance, dest, origin)

# A tibble: 226 x 3
  distance dest   origin
     <dbl> <chr> <chr>
1      4983 HNL    JFK
2      4963 HNL    EWR
3      3370 ANC    EWR
4      2586 SFO    JFK
5      2576 OAK    JFK
6      2569 SJC    JFK
7      2565 SFO    EWR
8      2521 SMF    JFK
9      2475 LAX    JFK
10     2465 LGB    JFK
# i 216 more rows
```

- Which travelled the shortest?

### i Solution

```
flights |>
  arrange(distance) |>
  select(distance, dest, origin, everything())

# A tibble: 336,776 x 19
  distance dest  origin  year month   day dep_time sched_dep_time dep_delay
  <dbl> <chr> <chr> <int> <int> <int>     <int>        <int>      <dbl>
1       17 LGA   EWR    2013     7     27       NA         106       NA
2       80 PHL   EWR    2013     1      3     2127        2129      -2
3       80 PHL   EWR    2013     1      4     1240        1200      40
4       80 PHL   EWR    2013     1      4     1829        1615     134
5       80 PHL   EWR    2013     1      4     2128        2129      -1
6       80 PHL   EWR    2013     1      5     1155        1200      -5
7       80 PHL   EWR    2013     1      6     2125        2129      -4
8       80 PHL   EWR    2013     1      7     2124        2129      -5
9       80 PHL   EWR    2013     1      8     2127        2130      -3
10      80 PHL   EWR    2013     1      9     2126        2129      -3
# i 336,766 more rows
# i 10 more variables: arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, air_time <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
Avoid an overkill ...

flights |>
  distinct(distance, origin, dest) |>
  slice_min(distance, n=10)

# A tibble: 11 x 3
  distance origin dest
  <dbl> <chr>  <chr>
1       17 EWR    LGA
2       80 EWR    PHL
3       94 JFK    PHL
4       96 LGA    PHL
5      116 EWR    BDL
6      143 EWR    ALB
7      160 EWR    PVD
8      169 EWR    BWI
9      173 JFK    MVY
10      184 JFK    BWI
11      184 LGA    BOS
```

### Projection

- Brainstorm as many ways as possible to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from `flights`.

**i** Solution

```
flights |>
  select(dep_time, dep_delay, arr_time, arr_delay) |>
  head()

# A tibble: 6 x 4
  dep_time dep_delay arr_time arr_delay
  <int>     <dbl>     <int>     <dbl>
1     517         2     830        11
2     533         4     850        20
3     542         2     923        33
4     544        -1    1004       -18
5     554        -6     812       -25
6     554        -4     740        12
```

**i** Solution

```
flights |>
  select(starts_with("dep"), starts_with("arr")) |>
  head()

# A tibble: 6 x 4
  dep_time dep_delay arr_time arr_delay
  <int>     <dbl>     <int>     <dbl>
1     517         2     830        11
2     533         4     850        20
3     542         2     923        33
4     544        -1    1004       -18
5     554        -6     812       -25
6     554        -4     740        12
```

**i** Solution

```
flights |>
  select(starts_with("dep_") | starts_with("arr")) |>
  head()

# A tibble: 6 x 4
  dep_time dep_delay arr_time arr_delay
  <int>     <dbl>     <int>     <dbl>
1     517         2     830        11
2     533         4     850        20
3     542         2     923        33
4     544        -1    1004       -18
5     554        -6     812       -25
6     554        -4     740        12
```

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

**i Solution**

It is included once in the result.

```
flights |>
  select(arr_time, starts_with("dep_") | starts_with("arr"), arr_time) |>
  head()

# A tibble: 6 x 4
  arr_time dep_time dep_delay arr_delay
  <int>     <int>     <dbl>     <dbl>
1     830      517       2       11
2     850      533       4       20
3     923      542       2       33
4    1004      544      -1      -18
5     812      554      -6      -25
6     740      554      -4       12
```

- 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")
```

**i** Solution

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")

flights |>
  filter(across(any_of(vars), is.numeric))

Warning: Using `across()` in `filter()` was deprecated in dplyr 1.0.8.
i Please use `if_any()` or `if_all()` instead.
# A tibble: 336,776 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>
1 2013     1     1      517            515        2     830          819
2 2013     1     1      533            529        4     850          830
3 2013     1     1      542            540        2     923          850
4 2013     1     1      544            545       -1    1004         1022
5 2013     1     1      554            600       -6     812          837
6 2013     1     1      554            558       -4     740          728
7 2013     1     1      555            600       -5     913          854
8 2013     1     1      557            600       -3     709          723
9 2013     1     1      557            600       -3     838          846
10 2013    1     1      558            600       -2     753          745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>

flights |>
  filter(if_any(vars, \((x) x<0)) |>
  relocate(ends_with('delay')) |>
  head()
```

Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.  
i Please use `all\_of()` or `any\_of()` instead.

```
# Was:
data %>% select(vars)
```

```
# Now:
data %>% select(all_of(vars))
```

See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.

```
# A tibble: 6 x 19
  dep_delay arr_delay year month   day dep_time sched_dep_time arr_time
  <dbl>     <dbl> <int> <int> <int>    <int>          <int>    <int>
1      -1     -18  2013     1     1      544            545        1004
2      -6     -25  2013     1     1      554            600        812
3      -4     -12  2013     1     1      554            558        740
4      -5     -19  2013     1     1      555            600        913
5      -3     -14  2013     1     1      557            600        709
6      -3     -8   2013     1     1      557            600        838
# i 11 more variables: sched_arr_time <int>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- 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> <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
```

- How do the select helpers deal with case by default?

### i Solution

By default, `select` helpers ignore case. This complies with a similar behavior in RDBMS

- How can you change that default?

### i Solution

```
select(flights,  
       contains("TIME", ignore.case=FALSE))  |>  
       head()  
  
# A tibble: 6 x 0
```

## 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.

**i** Solution

```
flights |>
  mutate(across(ends_with('dep_time') | ends_with('arr_time'),
               (\(x) 60L * (x %/% 100L) + (x %% 100L)),
               .names=".col")) |>
  select(ends_with('dep_time'), ends_with('arr_time'), everything())

# A tibble: 336,776 x 23
  dep_time sched_dep_time cor_dep_time cor_sched_dep_time arr_time
  <int>       <int>       <int>           <int>       <int>
1     517         515       317            315       830
2     533         529       333            329       850
3     542         540       342            340       923
4     544         545       344            345      1004
5     554         600       354            360       812
6     554         558       354            358       740
7     555         600       355            360       913
8     557         600       357            360       709
9     557         600       357            360       838
10    558         600       358            360       753
# i 336,766 more rows
# i 18 more variables: sched_arr_time <int>, cor_arr_time <int>,
#   cor_sched_arr_time <int>, year <int>, month <int>, day <int>,
#   dep_delay <dbl>, arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>

df <- flights |>
  mutate(across(ends_with('dep_time') | ends_with('arr_time'),
               (\(x) 60L * (x %/% 100L) + (x %% 100L)),
               .names=".col"))
```

- 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?

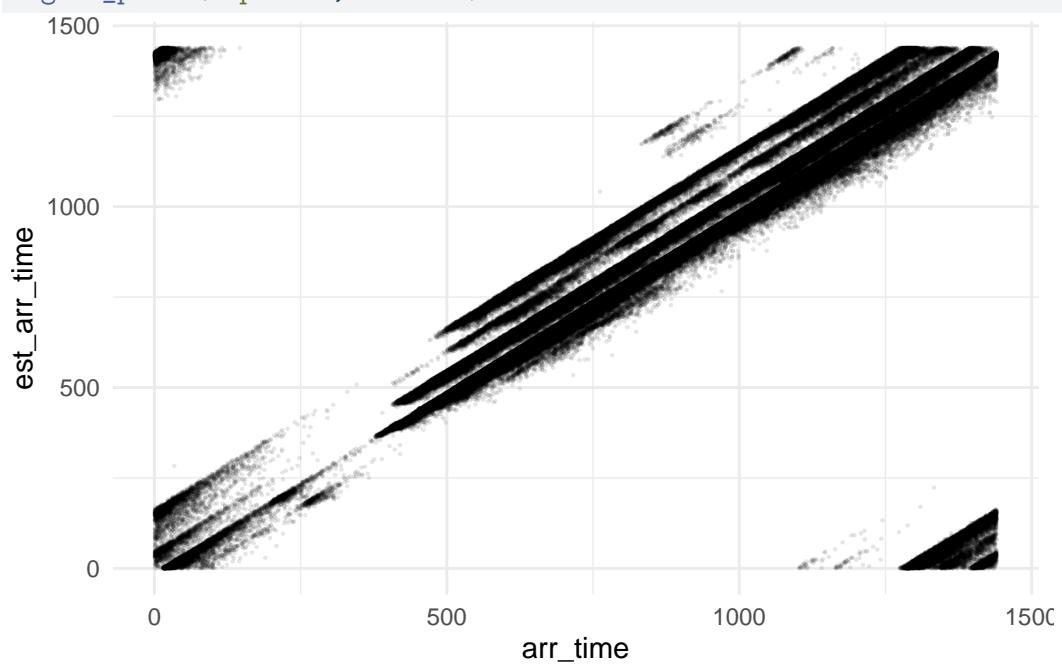
## i Solution

```
df |>
  mutate(est_arr_time = (dep_time + air_time) %% 1440) |>
  filter(abs(est_arr_time - arr_time)<1000) |>
  arrange(desc(abs(est_arr_time - arr_time))) |>
  select(dest, arr_time, est_arr_time, ends_with("time"), everything())

# A tibble: 314,414 x 20
  dest   arr_time est_arr_time dep_time sched_dep_time sched_arr_time air_time
  <chr>    <int>        <dbl>    <int>        <int>        <int>    <dbl>
1 HNL      881        1226     594        600        890     632
2 HNL      858        1203     594        600        910     609
3 HNL      865        1210     594        600        900     616
4 HNL     1084        1429     810        815       1097     619
5 HNL      855        1199     598        600        890     601
6 HNL     1090        1434     815        809       1093     619
7 HNL      842        1186     585        600        885     601
8 HNL     1062        1405     809        809       1093     596
9 HNL      889        1232     597        600        890     635
10 HNL     900        1243     597        600        930     646
# i 314,404 more rows
# i 13 more variables: year <int>, month <int>, day <int>, dep_delay <dbl>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

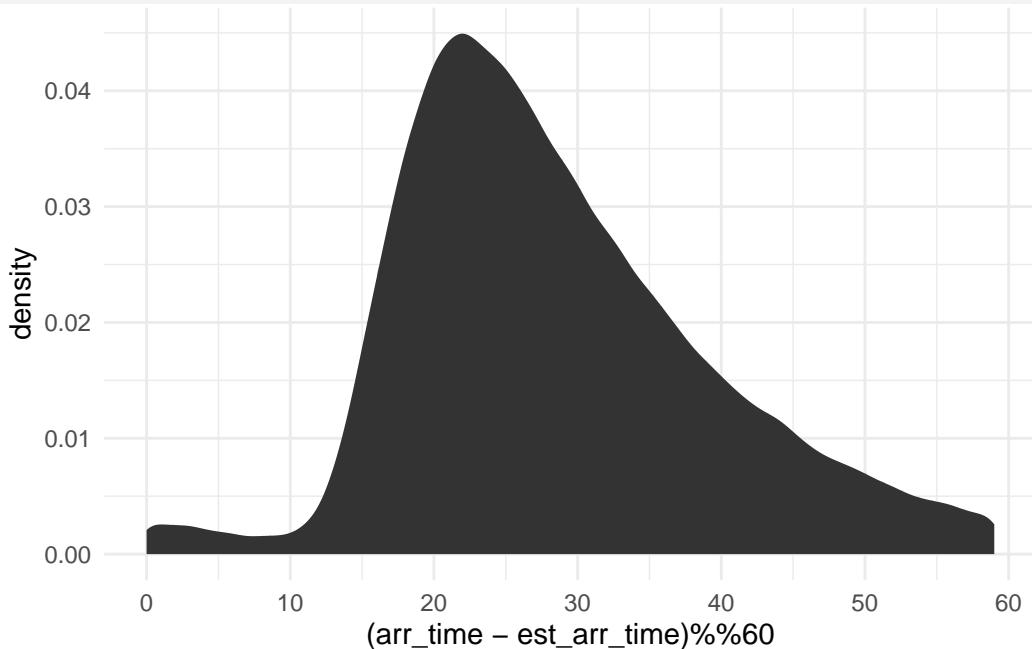
**i Solution**

```
df |>  
  mutate(est_arr_time = (dep_time + air_time) %% 1440) |>  
  filter(!is.na(arr_time), !is.na(est_arr_time)) |>  
  ggplot() +  
  aes(x = arr_time, y = est_arr_time) +  
  geom_point(alpha=.1, size=.1)
```



### i Solution

```
df |>
  mutate(est_arr_time = (dep_time + air_time) %% 1440) |>
  filter(!is.na(arr_time), !is.na(est_arr_time)) |>
  filter(abs(arr_time - est_arr_time) < 600) |>
  ggplot() +
  aes(x = (arr_time - est_arr_time) %% 60) +
  stat_density() +
  scale_x_continuous(breaks=seq(0, 60, 10))
```



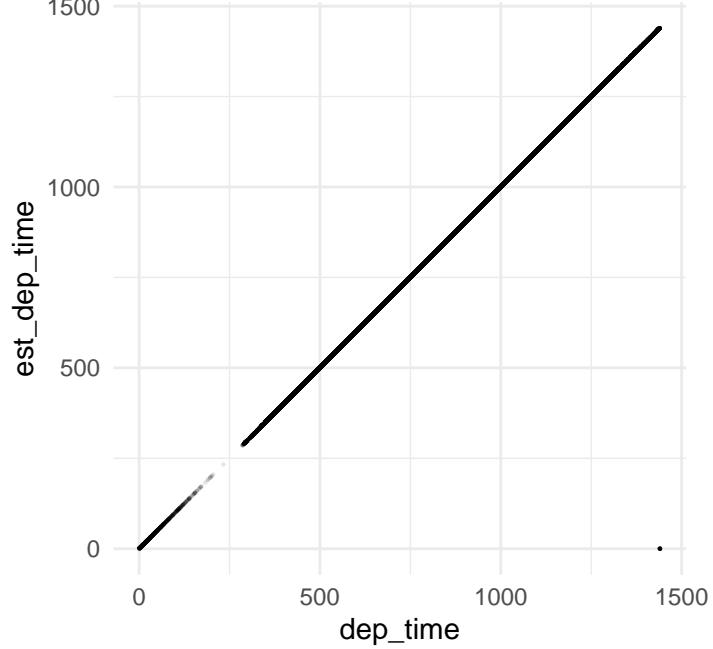
### i Solution

- Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?

We expect `dep_time == sched_dep_time + dep_delay %% 1440`

### i Solution

```
df |>
  filter(!is.na(dep_time)) |>
  mutate(est_dep_time = (sched_dep_time + dep_delay) %% 1440L) |>
  ggplot() +
  aes(x=dep_time, y=est_dep_time) +
  geom_point(alpha=.1, size=.1) +
  coord_fixed()
```



For most of the flights, the discrepancy between `dep_time` and the computed value is 0.  
For 29 flights it is positive. The discrepancy is then exactly equal to 24 hours.

## i Solution

```
df |>
  filter(!is.na(dep_time)) |>
  mutate(discr = dep_time - (sched_dep_time + dep_delay) %% 1440L) |>
  filter(discr > 0) |>
  select(dest, origin, carrier, distance, descr) |>
  left_join(airports, by=c("dest"="faa")) |>
  arrange(carrier, desc(distance), dest) |>
  select(dest, origin, carrier, name, descr)

# A tibble: 29 x 5
  dest   origin carrier name      descr
  <chr>  <chr>   <chr>  <chr>     <dbl>
1 MIA    LGA     AA     Miami Intl    1440
2 ORD    LGA     AA     Chicago Ohare Intl 1440
3 PSE    JFK     B6     <NA>       1440
4 PSE    JFK     B6     <NA>       1440
5 PSE    JFK     B6     <NA>       1440
6 PSE    JFK     B6     <NA>       1440
7 PSE    JFK     B6     <NA>       1440
8 PSE    JFK     B6     <NA>       1440
9 SJU    JFK     B6     <NA>       1440
10 SJU   JFK     B6     <NA>      1440
# i 19 more rows
```

- 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.

### i Solution

```
flights |>
  filter(dense_rank(-arr_delay) <=10) |>
  arrange(desc(arr_delay)) |>
  sample_n(10)

# A tibble: 10 x 19
# ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>

  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
1 2013     6    15      1432         1935    1137    1607        2120
2 2013     5     3      1133         2055     878    1250        2215
3 2013     9    20      1139         1845    1014    1457        2210
4 2013     4    10      1100         1900     960    1342        2211
5 2013     3    17      2321         810     911    135         1020
6 2013    12     5      756          1700     896    1058        2020
7 2013     1     9      641          900    1301    1242        1530
8 2013     1    10      1121         1635    1126    1239        1810
9 2013     7    22      2257         759     898    121         1026
10 2013     7    22      845          1600    1005    1044        1815
```

## 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.

## i Solution

```
flights_gp <- flights |>
  group_by(flight, carrier, dest, origin)

flights_gp |>
  summarize(q50 = quantile(arr_delay,.5, , na.rm=T),
            q90 = quantile(arr_delay,.9, , na.rm=T),
            .groups="drop") |>
  filter(q50<=-15, q90>=15) |>
  head()

# A tibble: 6 x 6
  flight carrier dest  origin    q50    q90
  <int> <chr>   <chr> <chr>  <dbl>  <dbl>
1      5 AS      SEA   EWR     -21   26.4
2      6 DL      SLC   JFK     -18   20
3     10 UA     IAH   LGA     -19   28.5
4     11 AS     SEA   EWR     -15   26.4
5     31 DL     SFO   JFK     -22   17.2
6     59 AA     SFO   JFK     -16   20.2
```

Another approach builds on experimental `reframe()`

```
quantile_df <- function(x, probs = c(0.5, 0.9)) {
  tibble(
    val = quantile(x, probs, na.rm = TRUE),
    quant = probs
  )
}

tmp <- flights_gp |>
  reframe(q= quantile_df(arr_delay)) |>
  unnest(cols=q)
```

`reframe()` generates a list-column (a column is tibble-valued). The result has to be *unnested*.

```
tmp |>
  pivot_wider(
    id_cols=c(flight, carrier, dest, origin),
    names_from= quant,
    values_from= val,
    names_prefix="q"
  ) |>
  head()

# A tibble: 6 x 6
  flight carrier dest  origin    q0.5    q0.9
  <int> <chr>   <chr> <chr>  <dbl>   <dbl>
1      1 AA      LAX   JFK     -13     19
2      1 B6      FLL   JFK      2     47.8
3      1 DL      SJU   JFK     -21    -6.40
4      1 UA      ORD   EWR     -26    -26
5      1 UA      PBI   EWR     -19   -14.2
6      1 WN      MDW   LGA      2.5   33.6
```

```
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`
```

**i** Solution

```
flights |>
  group_by(flight, carrier, dest, origin) |>
  summarise(q = quantile(arr_delay,.5, na.rm=T), m=mean(arr_delay, na.rm=TRUE)) |>
  filter((q < -15) | (q > 15)) |>
  arrange(desc(m))

`summarise()` has grouped output by 'flight', 'carrier', 'dest'. You can
override using the ` `.groups` argument.

# A tibble: 4,161 x 6
# Groups:   flight, carrier, dest [4,106]
  flight carrier dest  origin     q     m
  <int> <chr>   <chr> <chr> <dbl> <dbl>
1    372 UA      CLE   LGA     455   455
2    488 UA      DEN   LGA     359   359
3    521 WN      BNA   EWR     335   335
4   5017 EV      DTW   LGA     334   334
5    468 UA      MCO   EWR     323   323
6    390 DL      DEN   LGA     318   318
7    809 DL      SLC   EWR     299   299
8   3261 EV      CLE   EWR     292   292
9   3760 9E      RIC   JFK     288   288
10   1510 UA     IAH   EWR     283   283
# i 4,151 more rows
```

**i** Solution

```
flights |>
  group_by(jour=lubridate::wday(time_hour, label=T, abbr = F)) |>
  summarise(n_cancelled=sum(is.na(dep_time)), n_tot=n()) |>
  mutate(prop_cancelled = n_cancelled/n_tot)

# A tibble: 7 x 4
  jour      n_cancelled n_tot prop_cancelled
  <ord>          <int> <int>        <dbl>
1 dimanche      714  46357       0.0154
2 lundi         1220  50690       0.0241
3 mardi         1152  50422       0.0228
4 mercredi      1202  50060       0.0240
5 jeudi         1562  50219       0.0311
6 vendredi      1608  50308       0.0320
7 samedi         797  38720       0.0206

?lubridate::wday
```

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

**i** Solution

- Come up with another approach that will give you the same output as `not_cancelled |> count(dest)` and (without using `count()`).

**i** Solution

```
not_cancelled |>
  summarise(n_distinct(dest), n_distinct(tailnum))

# A tibble: 1 x 2
  `n_distinct(dest)` `n_distinct(tailnum)`
  <int>                <int>
1          104                 4037
```

- Our definition of cancelled flights (`is.na(dep_delay) | is.na(arr_delay)`) is slightly suboptimal. Why? Which is the most important column?

**i** Solution

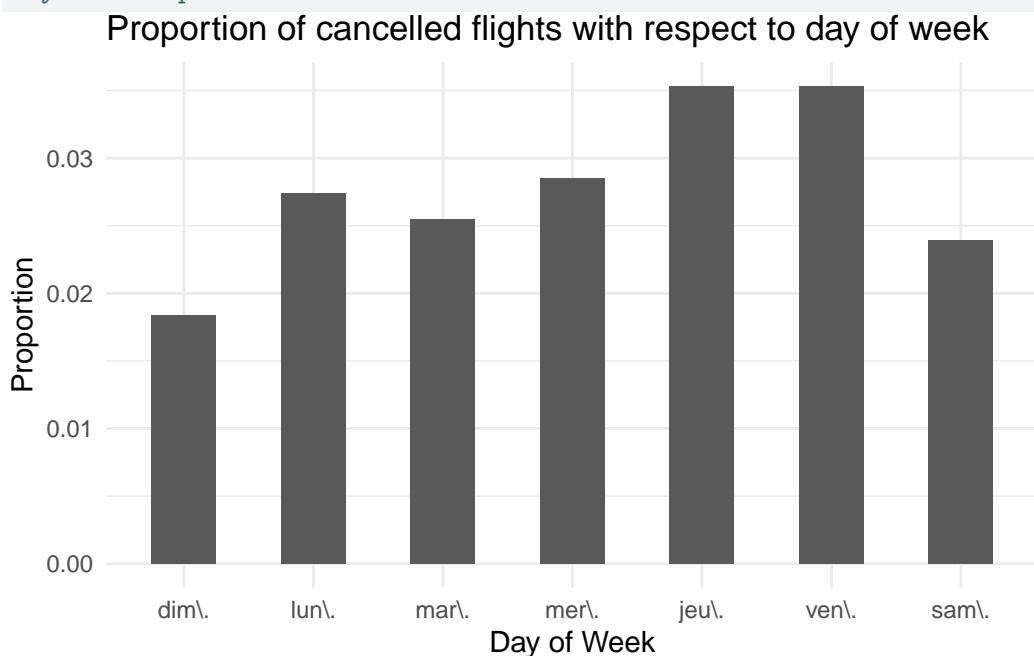
```
flights |>
  filter(xor(is.na(arr_delay), is.na(dep_delay))) |>
  group_by(is.na(arr_delay), is.na(dep_delay)) |>
  summarise(n())

`summarise()` has grouped output by 'is.na(arr_delay)'. You can override using
the ` `.groups` argument.
# A tibble: 1 x 3
# Groups:   is.na(arr_delay) [1]
  `is.na(arr_delay)` `is.na(dep_delay)` `n()`
  <lgl>                  <lgl>                <int>
1 TRUE                   FALSE                 1175
```

- Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

### i Solution

```
flights |>
  mutate(day_of_week=wday(time_hour, label=TRUE)) |>
  group_by(day_of_week) |>
  summarise(n_cancelled= sum(is.na(arr_delay)), n(), prop= n_cancelled/n()) |>
  ggplot() +
  aes(x=day_of_week, y=prop) +
  geom_col(width=.5) +
  labs(title="Proportion of cancelled flights with respect to day of week") +
  xlab("Day of Week") +
  ylab("Proportion")
```



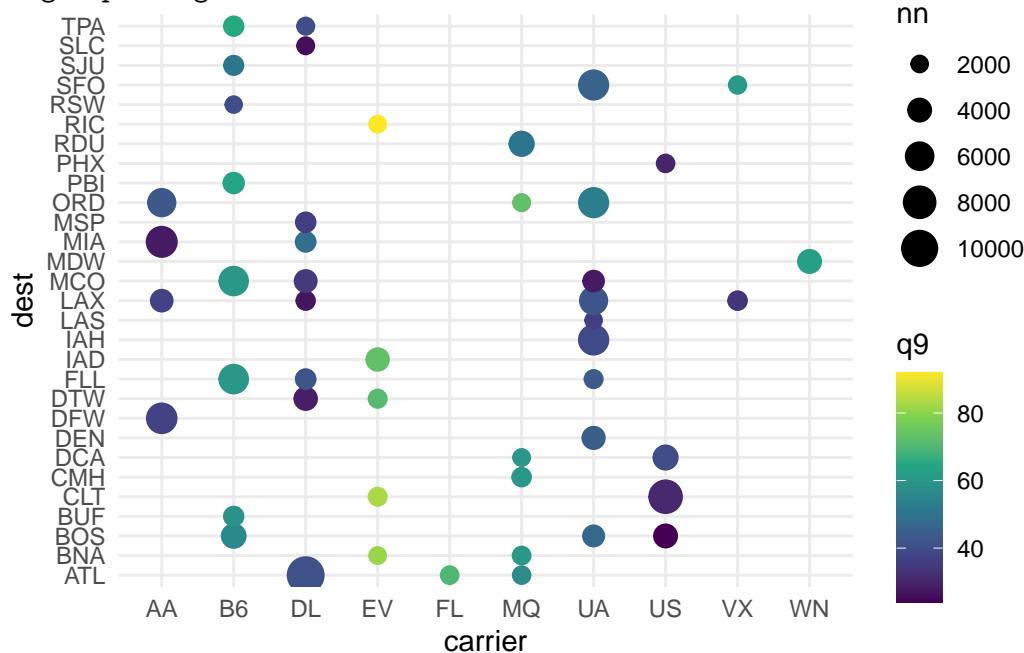
- Which carrier has the worst delays?

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())`)

### i Solution

```
flights |>
  filter(!is.na(arr_delay)) |>
  group_by(carrier, dest) |>
  summarise(nn=mean(arr_delay), q9=quantile(arr_delay, c(9)/10)), nn=n() |>
  ungroup() |>
  filter(min_rank(-nn) <50) |>
  ggplot() +
  aes(x=carrier, y=dest) +
  geom_point(aes(color=q9, size=nn)) +
  scale_color_viridis_c() +
  scale_size_area()

`summarise()` has grouped output by 'carrier'. You can override using the
`.groups` argument.
```



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

### i Solution

## Miscellanea

- Which carriers serve all destination airports (in the table) ?

**i** Solution

```
flights |>
  distinct(origin, dest) |>
  mutate(tot_dest = n_distinct(dest)) |>
  group_by(tot_dest, origin) |>
  summarise(n_dest=n(), .groups="drop") |>
  filter(n_dest==tot_dest)

# A tibble: 0 x 3
# i 3 variables: tot_dest <int>, origin <chr>, n_dest <int>
👉 Verb mutate() handles function n_distinct() as a window function with trivial
window.
```

- i** There is no need to perform grouping and aggregation to answer this question. Basic relational algebra is enough. This operation is called *division* in SQL language

```
R <- flights |>
  distinct(carrier, dest)

C <- R |> distinct(carrier)
D <- R |> distinct(dest)

setdiff(C, crossing(C, D) |>
  setdiff(R) |>
  distinct(carrier))

# A tibble: 0 x 1
# i 1 variable: carrier <chr>
```

Let  $\mathcal{R}(A, B)$  be the schema of some table  $R$ , then

$$R \div \pi_B(R) = \pi_A(R) \setminus (\pi_A(\pi_A(R) \times \pi_B(R)) \setminus R)$$

More generally if  $S$  has schema  $\mathcal{S}(B)$

$$R \div S = \pi_A(R) \setminus (\pi_A(\pi_A(R) \times \pi_B(S)) \setminus R)$$

- Refer back to the lists of useful mutate and filtering functions.
- Describe how each operation changes when you combine it with grouping.

**i** Solution

- Which plane (`tailnum`) has the worst on-time record amongst planes with at least ten flights?

**i** Solution

```
flights |>
  filter(!is.na(arr_delay)) |>
  group_by(tailnum) |>
  summarise(m = mean(arr_delay), n_flights=n() ) |>
  filter(n_flights > 10 ) |>
  filter(min_rank(-m) == 1) |>
  inner_join(distinct(flights, tailnum, carrier))  
  
Joining with `by = join_by(tailnum)`  
# A tibble: 1 x 4  
  tailnum      m n_flights carrier  
  <chr>    <dbl>     <int> <chr>  
1 N337AT    66.5        13   FL
```

- What time of day should you fly if you want to avoid delays as much as possible?

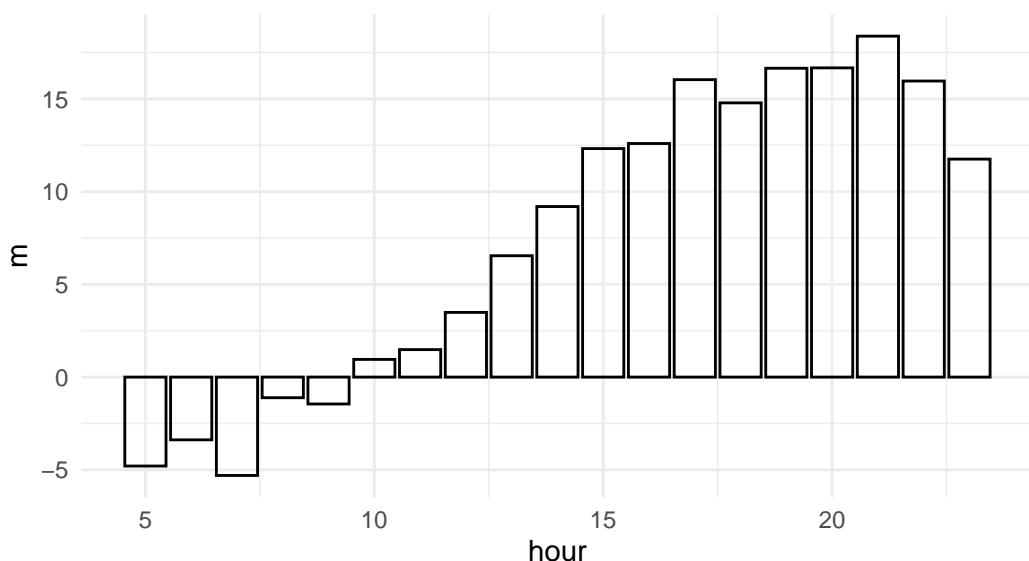
**i** Solution

Definitely before 9a.m.

```
flights |>
  filter(!is.na(arr_delay)) |>
  group_by(hour) |>
  summarise(m = mean(arr_delay, na.rm=T)) |>
  arrange(m) |>
  ggplot() +
  aes(x=hour, y=m) +
  geom_col(color="black", fill="white", alpha=.2) +
  labs(title="Mean arrival delay as a function of departure hour",
       subtitle= "nycflights13 data")
```

Mean arrival delay as a function of departure hour

nycflights13 data



- For each destination, compute the total minutes of delay.

### i Solution

```
flights |>
  filter(!is.na(arr_delay)) |>
  group_by(dest) |>
  summarise(m = sum(arr_delay, na.rm = T)) |>
  arrange(desc(m)) |>
  rename(faa=dest) |>
  dplyr::inner_join(airports)

Joining with `by = join_by(faa)`
# A tibble: 100 x 9
# ... with 9 variables:
#   faa     m name          lat    lon    alt    tz dst tzone
#   <chr> <dbl> <chr>       <dbl>  <dbl>  <dbl>  <dbl> <chr> <chr>
# 1 ATL    190260 Hartsfield Jackson Atlanta~ 33.6  -84.4  1026  -5 A  Amer~
# 2 CLT    100645 Charlotte Douglas Intl      35.2  -80.9   748  -5 A  Amer~
# 3 ORD    97352 Chicago Ohare Intl        42.0  -87.9   668  -6 A  Amer~
# 4 FLL    96153 Fort Lauderdale Hollywood ~ 26.1  -80.2    9  -5 A  Amer~
# 5 DCA    82609 Ronald Reagan Washington N~ 38.9  -77.0   15  -5 A  Amer~
# 6 RDU    78107 Raleigh Durham Intl      35.9  -78.8   435  -5 A  Amer~
# 7 MCO    76185 Orlando Intl           28.4  -81.3   96  -5 A  Amer~
# 8 IAD    74631 Washington Dulles Intl    38.9  -77.5   313  -5 A  Amer~
# 9 BNA    71867 Nashville Intl         36.1  -86.7   599  -6 A  Amer~
# 10 DEN   61700 Denver Intl            39.9  -105.  5431  -7 A  Amer~
# # i 90 more rows
```

- For each flight, compute the proportion of the total positive arrival delays for its destination.

### i Solution

Would be easy with ROLLUP (dest, flight)

```
WITH R AS (
  SELECT dest, flight, sum(arr_delay)
  FROM flights
  WHERE arr_delay >0
  GROUP BY ROLLUP(dest, flight)
)
SELECT r1.dest, r2.flight, r2.sum/r1.sum AS rap
FROM (SELECT dest, sum
      FROM R WHERE flight IS NULL) r1
      JOIN R r2
      ON (r1.dest=r2.dest)
WHERE r2.flight IS NOT NULL AND r2.sum/r1.sum >.1
ORDER BY r1.dest, r2.flight ;
```

With WINDOW it is even easier

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

## i Solution

```
flights |>
  dplyr::filter(arr_delay >0 ) |>
  group_by(dest, flight) |>
  summarise(total_delay_flight=sum(arr_delay)) |> # result is a grouped tibble
  mutate(total_delay_dest=sum(total_delay_flight),
         delay_ratio= total_delay_flight/total_delay_dest) |>
  filter(dest %in% c('LAX', 'ATL'), delay_ratio > .02)

`summarise()` has grouped output by 'dest'. You can override using the
`.groups` argument.
# A tibble: 20 x 5
# Groups: dest [2]
  dest   flight total_delay_flight total_delay_dest delay_ratio
  <chr>    <int>            <dbl>           <dbl>        <dbl>
1 ATL       348             7000          300299     0.0233
2 ATL       926             9908          300299     0.0330
3 ATL       947             8670          300299     0.0289
4 ATL      1147             6968          300299     0.0232
5 ATL      1499             6155          300299     0.0205
6 ATL      1942             6224          300299     0.0207
7 ATL      2042            11583          300299     0.0386
8 ATL      4705             6458          300299     0.0215
9 LAX       21              8378          203226     0.0412
10 LAX      119             4685          203226     0.0231
11 LAX      133             5644          203226     0.0278
12 LAX      169             6560          203226     0.0323
13 LAX      181             8228          203226     0.0405
14 LAX      185             5925          203226     0.0292
15 LAX      413             6271          203226     0.0309
16 LAX      415            10102          203226     0.0497
17 LAX      523             5055          203226     0.0249
18 LAX      535             6030          203226     0.0297
19 LAX      771             6963          203226     0.0343
20 LAX     2363             4273          203226     0.0210
```

### i Solution

We have used the fact that the output of `summarise(., total_delay_flight=sum(arr_delay))` is still a grouped dataframe/tibble (with grouping variable `dest`). At the following line, `mutate` performs aggregation with `sum()` in a groupwise manner, that is per group, and the result of the aggregation is appended to each row of the group, because we use `mutate` instead of `summarize`.

```
WITH R AS (
    SELECT dest,
           flight,
           SUM(arr_delay) AS total_delay_per_flight
      FROM flights
     WHERE arr_delay > 0
   GROUP BY dest, flight
)
SELECT dest, flight,
       total_delay_per_flight,
       SUM(total_delay_per_flight) OVER w_per_dest AS      total_delay_per_dest
  FROM R
WINDOW w_per_dest AS (PARTITION BY dest) ;
```

- 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** Solution

```
flights |>
  group_by(origin) |>
  arrange(year, month, day, sched_dep_time, .by_group=T) |> # .by_group -> see docu
  mutate(prev_delay=lag(dep_delay, n=1L), prev_origin=lag(origin, n=1L)) |>
  slice_head(n=6L) |> # inspect the result
  select(origin, prev_origin, dep_delay, prev_delay, everything())
```

# A tibble: 18 x 21

# Groups: origin [3]

	origin	prev_origin	dep_delay	prev_delay	year	month	day	dep_time	<chr>	<chr>	<dbl>	<dbl>	<int>	<int>	<int>	<int>
1	EWR	<NA>	2	NA	2013	1	1	517	1	EWR	2	NA	2013	1	1	517
2	EWR	EWR	-4	2	2013	1	1	554	2	EWR	-4	2	2013	1	1	554
3	EWR	EWR	-5	-4	2013	1	1	555	3	EWR	-5	-4	2013	1	1	555
4	EWR	EWR	-2	-5	2013	1	1	558	4	EWR	-2	-5	2013	1	1	558
5	EWR	EWR	-1	-2	2013	1	1	559	5	EWR	-1	-2	2013	1	1	559
6	EWR	EWR	1	-1	2013	1	1	601	6	EWR	1	-1	2013	1	1	601
7	JFK	<NA>	2	NA	2013	1	1	542	7	JFK	2	NA	2013	1	1	542
8	JFK	JFK	-1	2	2013	1	1	544	8	JFK	-1	2	2013	1	1	544
9	JFK	JFK	0	-1	2013	1	1	559	9	JFK	0	-1	2013	1	1	559
10	JFK	JFK	-3	0	2013	1	1	557	10	JFK	-3	0	2013	1	1	557
11	JFK	JFK	-2	-3	2013	1	1	558	11	JFK	-2	-3	2013	1	1	558
12	JFK	JFK	-2	-2	2013	1	1	558	12	JFK	-2	-2	2013	1	1	558
13	LGA	<NA>	4	NA	2013	1	1	533	13	LGA	4	NA	2013	1	1	533
14	LGA	LGA	-6	4	2013	1	1	554	14	LGA	-6	4	2013	1	1	554
15	LGA	LGA	-3	-6	2013	1	1	557	15	LGA	-3	-6	2013	1	1	557
16	LGA	LGA	-2	-3	2013	1	1	558	16	LGA	-2	-3	2013	1	1	558
17	LGA	LGA	-1	-2	2013	1	1	559	17	LGA	-1	-2	2013	1	1	559
18	LGA	LGA	0	-1	2013	1	1	600	18	LGA	0	-1	2013	1	1	600

# i 13 more variables: sched\_dep\_time <int>, arr\_time <int>,  
# sched\_arr\_time <int>, arr\_delay <dbl>, carrier <chr>, flight <int>,  
# tailnum <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
# minute <dbl>, time\_hour <dttm>

**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?

### i Solution

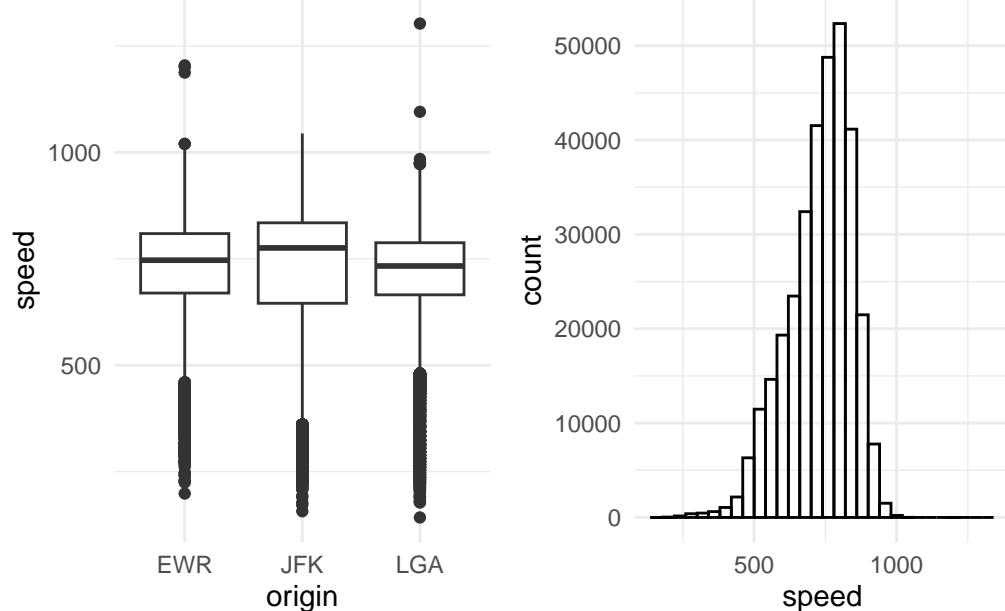
```
df <- flights |>
  filter(!is.na(air_time), air_time > 0) |>
  mutate(speed= (distance * 1.852)/ (air_time/60))

p1 <- df |>
  ggplot() +
  aes(x=origin, y=speed) +
  geom_boxplot()

p2 <- df |>
  ggplot() +
  aes(x=speed) +
  geom_histogram(color="black", fill="white", alpha=.2)

patchwork::wrap_plots(p1 + p2 )
```

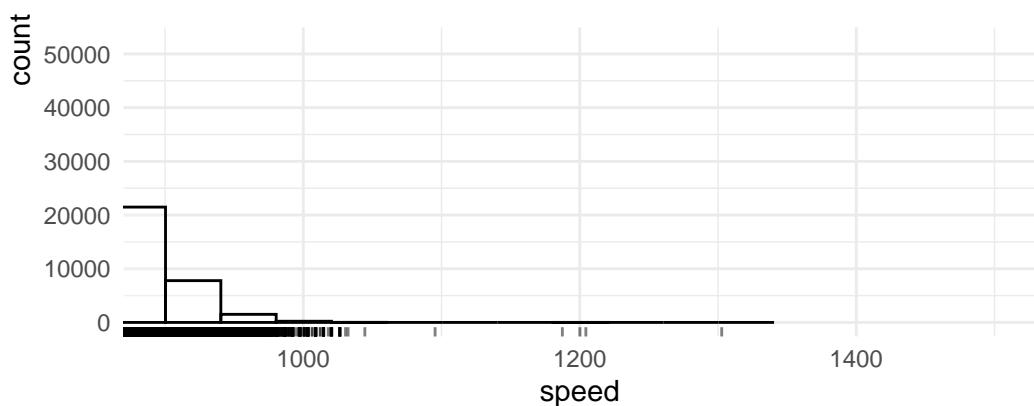
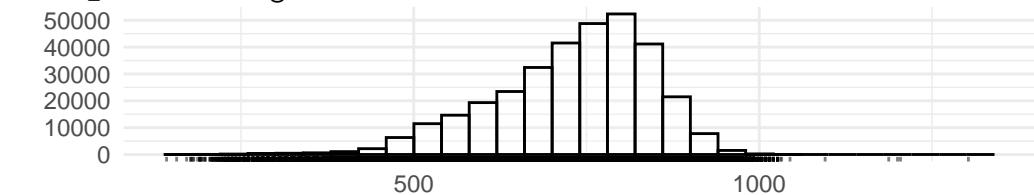
`stat\_bin()` using `bins = 30` . Pick better value with `binwidth` .



### i Solution

```
p2 + geom_rug(alpha=.5) +
  facet_zoom(xlim=c(900,1500))
```

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

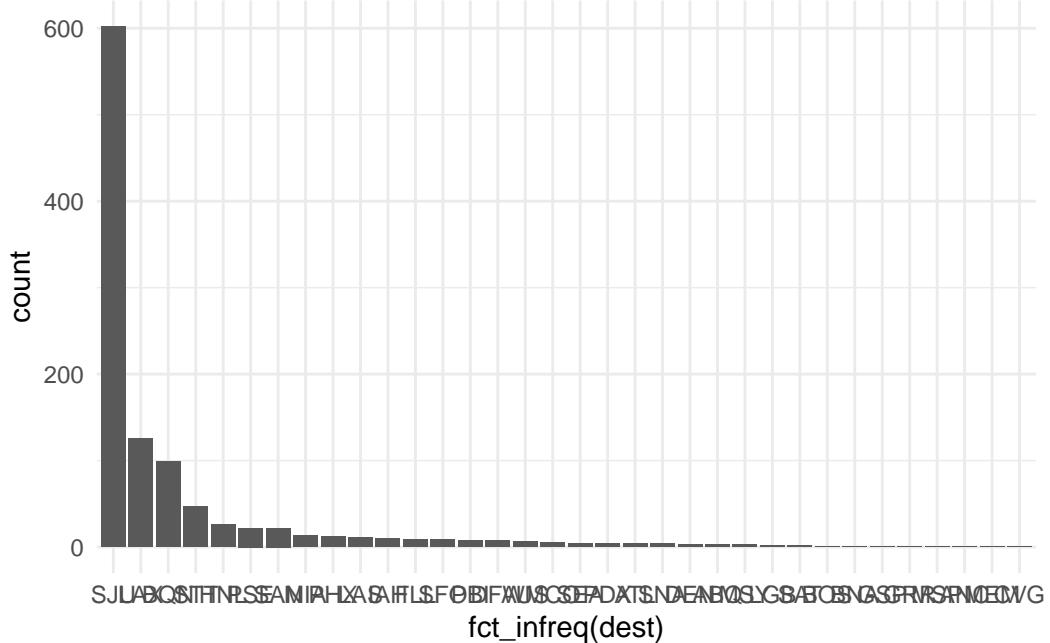


Consider all flights with average speed above 950km/h as suspicious.

Let us visualize destinations and origins of the speedy flights.

**i** Solution

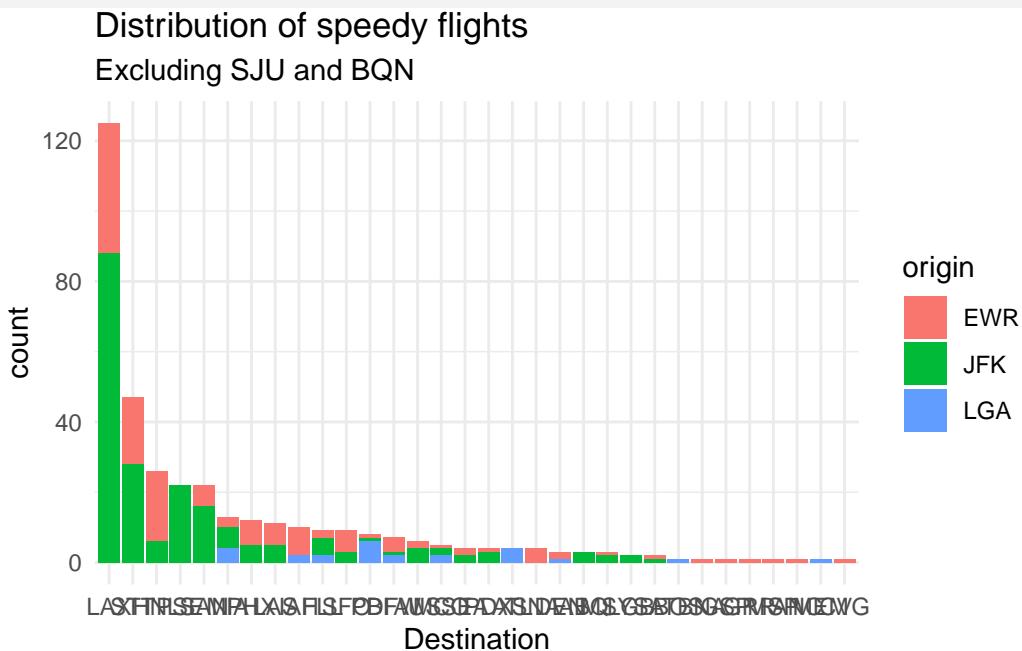
```
df |>  
  filter(speed>950) |>  
  mutate(dest=as_factor(dest)) |>  
  ggplot() +  
    aes(x=fct_infreq(dest)) +  
    geom_bar()
```



**i** SJU, BQN are located in Puerto Rico. LAX is Los Angeles airport. STT is located in Virgin Islands.

**i** Solution

```
df |>
  filter(speed>950, ! dest %in% c('SJU', 'BQN')) |>
  mutate(dest=as_factor(dest)) |>
  ggplot() +
    aes(x=fct_infreq(dest), fill=origin) +
    geom_bar() +
    labs(x="Destination",
         title="Distribution of speedy flights",
         subtitle="Excluding SJU and BQN")
```



- Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.

**i** Solution

- 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.

## i Solution

```
flights |>
  filter(!is.na(tailnum)) |>
  group_by(tailnum) |>
  arrange(year, month, day, sched_dep_time, .bygroup=T) |>
  mutate(rnk=row_number(), tot=n()) |>
  filter(is.na(arr_time) | arr_delay >=60) |>
  slice_head(n=1) |>
  mutate(rel_rnk = rnk/tot) |>
  select(tailnum, rnk, tot, rel_rnk, carrier) |>
  arrange(desc(tot), desc(rel_rnk)) |>
  ungroup()

# A tibble: 3,454 x 5
  tailnum    rnk    tot  rel_rnk carrier
  <chr>   <int> <dbl> <dbl> <chr>
1 N725MQ     17    575  0.0296  MQ
2 N722MQ     31    513  0.0604  MQ
3 N723MQ     24    507  0.0473  MQ
4 N711MQ     24    486  0.0494  MQ
5 N713MQ      4    483  0.00828 MQ
6 N258JB      8    427  0.0187  B6
7 N298JB     17    407  0.0418  B6
8 N353JB     34    404  0.0842  B6
9 N351JB     10    402  0.0249  B6
10 N735MQ     3    396  0.00758 MQ
# i 3,444 more rows
```

## 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>