DSS5201 Data Visualization
Week 4

Yuting Huang

NUS DSDS

2024-09-01

Recap: Getting data from the web

Data on **Covid-19 infections in Singapore** is available here:

https://data.gov.sg/datasets/d_11e68bba3b3c76733475a72d09759eeb/view

```
import pandas as pd
import matplotlib.pyplot as plt
import requests

# URL for the data
base_url = "https://data.gov.sg/api/action/datastore_search"
url = base_url + "?resource_id=d_11e68bba3b3c76733475a72d09759eeb"
# Query for the data
response = requests.get(url)
results = response.json()
```

results


```
{'help': 'https://data.gov.sg/api/3/action/help_show?name=datastore_search',
 'success': True,
 'result': {'resource_id': 'd_11e68bba3b3c76733475a72d09759eeb',
  'fields': [{'type': 'numeric', 'id': 'epi_year'},
   {'type': 'text', 'id': 'epi_week'},
   {'type': 'numeric', 'id': 'est_count'},
   {'type': 'int4', 'id': '_id'}],
  'records': [{'_id': 1,
    'epi_year': '2023',
    'epi_week': '2023-09',
    'est_count': '4426'},
   {'_id': 2, 'epi_year': '2023', 'epi_week': '2023-10', 'est_count': '10352'},
   {'_id': 3, 'epi_year': '2023', 'epi_week': '2023-11', 'est_count': '10464'},
   {'_id': 4, 'epi_year': '2023', 'epi_week': '2023-12', 'est_count': '14467'},
   {'_id': 5, 'epi_year': '2023', 'epi_week': '2023-13', 'est_count': '28410'},
   {'_id': 6, 'epi_year': '2023', 'epi_week': '2023-14', 'est_count': '16018'},
   {'_id': 7, 'epi_year': '2023', 'epi_week': '2023-15', 'est_count': '26072'},
   {'_id': 8, 'epi_year': '2023', 'epi_week': '2023-16', 'est_count': '27818'},
   {'_id': 9, 'epi_year': '2023', 'epi_week': '2023-17', 'est_count': '23157'},
   {'_id': 10,
    'epi_year': '2023',
    'epi_week': '2023-18',
    'est_count': '22476'},
   {'_id': 11,
    'epi_year': '2023',
...
    'epi_week': '2024-08',
    'est_count': '2450'}],
```

Get data as a data frame

- After examining the structure of the query response, we find that data on infections are in `result -> records`.
- Save it in an object named `df`.

```
df = pd.DataFrame(results["result"]["records"])
df.head(6)
```

```
   _id epi_year epi_week est_count
0    1     2023  2023-09      4426
1    2     2023  2023-10     10352
2    3     2023  2023-11     10464
3    4     2023  2023-12     14467
4    5     2023  2023-13     28410
5    6     2023  2023-14     16018
```

```
df.dtypes
```

```
_id            int64
epi_year      object
epi_week      object
est_count     object
dtype: object
```

```
df["est_count"] = pd.to_numeric(df["est_count"])
df.dtypes
```

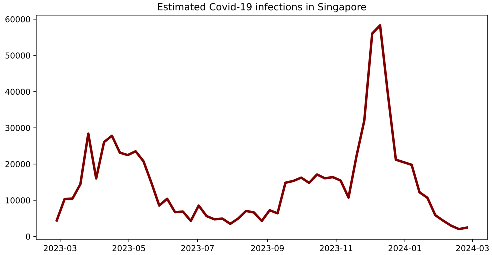
```
_id            int64
epi_year      object
epi_week      object
est_count      int64
dtype: object
```

```python
# Convert epi week into datetime (sunday of the week)
df["date_sundays"] = pd.to_datetime(df["epi_week"] + "-0",
                                    format = "%Y-%U-%w")
df.head(3)
```

```
   _id epi_year epi_week  est_count date_sundays
0    1     2023  2023-09       4426   2023-02-26
1    2     2023  2023-10      10352   2023-03-05
2    3     2023  2023-11      10464   2023-03-12
```

- `to_datetime()` provides flexible formats that makes conversions easier.
- Read the documentation here for more options.

```python
# Set figure size as needed
plt.figure(figsize = (10, 5))
# Plot
plt.plot(df["date_sundays"], df["est_count"],
         color = "maroon", linewidth = 3)
plt.title("Estimated Covid-19 infections in Singapore")
# Show the plot
plt.show()
```



Lastly, we can export the data frame to our working directory.

And export the plot we just created.

```
# Export data as CSV
df.to_csv("../data/wk3_infections.csv", index = False)
```

```
# Export a plot as PNG; do this before plt.show()
plt.figure(figsize = (10, 5))
plt.plot(df["date_sundays"], df["est_count"],
         color = "maroon", linewidth = 3)
plt.title("Estimated Covid-19 infections in Singapore")
plt.savefig("wk3_infections.png", dpi = 300)
```

Data manipulation

"Data janitor work"

It is extremely rare that the data you obtain will be in precisely the right format for the analysis that you wish to do. Very often, we need to do some or all of the following:

- Create some new variables
- Create data summaries
- Rename variables
- Reorder observations to make data easier to work with
- ...

You will learn how to do all these today, using a data set on flights that departed New York City in 2013.

Let's first import the necessary libraries into our environment.

```
import numpy as np
import pandas as pd
from nycflights13 import flights
flights.head()
```

```
   year  month  day  dep_time  ...  distance  hour  minute                 tim
0  2013      1    1     517.0  ...      1400     5      15  2013-01-01T10:
1  2013      1    1     533.0  ...      1416     5      29  2013-01-01T10:
2  2013      1    1     542.0  ...      1089     5      40  2013-01-01T10:
3  2013      1    1     544.0  ...      1576     5      45  2013-01-01T10:
4  2013      1    1     554.0  ...       762     6       0  2013-01-01T11:

[5 rows x 19 columns]
```

19 variables on flights to and from different airports in the New York City during 2013.

Variables	**Description**
year, month, day	Date of departure
dep_time, arr_time	Actual departure and arrival times
dep_delay, arr_delay	Actual departure and arrival delays
sched_dep_time, sched_arr_time	Scheduled departure and arrival times
hour, minute	Hours and minutes of scheduled departure
time_hour	Dates and hours of scheduled departure
carrier	2-letter carrier abbreviation
tailnum	Plane tail number
flight	Flight number
origin, dest	Origin and destination airports
air_time	Amount of time spent in the air
distance	Distance flown

```
flights.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 336776 entries, 0 to 336775
Data columns (total 19 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   year            336776 non-null  int64
 1   month           336776 non-null  int64
 2   day             336776 non-null  int64
 3   dep_time        328521 non-null  float64
 4   sched_dep_time  336776 non-null  int64
 5   dep_delay       328521 non-null  float64
 6   arr_time        328063 non-null  float64
 7   sched_arr_time  336776 non-null  int64
 8   arr_delay       327346 non-null  float64
 9   carrier         336776 non-null  object
 10  flight          336776 non-null  int64
```

Recap: Some of the most common data types we are likely to encounter:

- The data types are important – determines what kinds of operations we can perform on the column.

Dtype	**Type of data**
float64	real numbers
category	categories
datetime64	date times
int64	integers
bool	True of False
string	text
object	mixed types

We would like to work with the `time_hour` variable that contains date and time information of a flight.

```
flights["time_hour"].head()
```

```
0     2013-01-01T10:00:00Z
1     2013-01-01T10:00:00Z
2     2013-01-01T10:00:00Z
3     2013-01-01T10:00:00Z
4     2013-01-01T11:00:00Z
Name: time_hour, dtype: object
```

Pandas makes it easy to convert it into a datetime64 object.

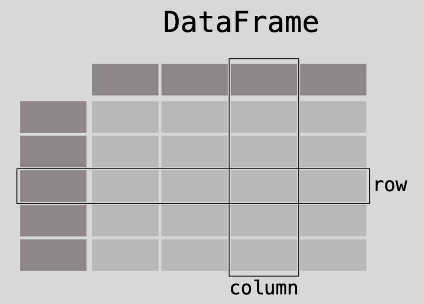
```
flights["time_hour"] = pd.to_datetime(
    flights["time_hour"], format = "%Y-%m-%dT%H:%M:%SZ")
flights["time_hour"].head()
```

```
0    2013-01-01 10:00:00
1    2013-01-01 10:00:00
2    2013-01-01 10:00:00
3    2013-01-01 10:00:00
4    2013-01-01 11:00:00
Name: time_hour, dtype: datetime64[ns]
```

- Read the documentation here for more options.

The most basic Pandas object is **DataFrame**.

- A 2-dimensional data structure that can store data of different types.
- Makes up of **rows**, **columns**, and two contextual information (row index and column names).



Row manipulations in data frames

We will walk through the following row manipulations in data frames:

1. Accessing a particular set of rows.
2. Filtering rows.
3. Re-arranging rows.

- Let's examine all flights on September 2nd from JFK.

```
# Select a subset of flights data
df = flights.query("month == 9 and day == 2 and origin == 'JFK'")
df.head()
```

```
        year  month  day  dep_time  ...  distance  hour  minute
309920  2013      9    2       8.0  ...       301    22      55 2013-09-03
309923  2013      9    2      15.0  ...      1598    23      59 2013-09-03
309930  2013      9    2     537.0  ...      1089     5      45 2013-09-02
309932  2013      9    2     542.0  ...      1576     5      45 2013-09-02
309943  2013      9    2     557.0  ...       944     6       0 2013-09-02

[5 rows x 19 columns]
```

What happened to the indices?

```
# Reset index (back to zero)
df = df.reset_index(drop = True)
# Also, subset a few columns
df = df[["month", "day", "dep_time", "sched_dep_time",
         "dep_delay", "carrier", "origin", "dest"]]
df.head()
```

	month	day	dep_time	sched_dep_time	dep_delay	carrier	origin	dest
0	9	2	8.0	2255	73.0	B6	JFK	BUF
1	9	2	15.0	2359	16.0	B6	JFK	SJU
2	9	2	537.0	545	-8.0	AA	JFK	MIA
3	9	2	542.0	545	-3.0	B6	JFK	BQN
4	9	2	557.0	600	-3.0	B6	JFK	MCO

To select rows based on a list of values, use `.isin()`.

```
df[df["carrier"].isin(["UA", "AA"])]
```

	month	day	dep_time	sched_dep_time	dep_delay	carrier	origin	dest
2	9	2	537.0	545	-8.0	AA	JFK	MIA
16	9	2	648.0	655	-7.0	AA	JFK	LAS
24	9	2	704.0	710	-6.0	AA	JFK	MIA
25	9	2	713.0	710	3.0	AA	JFK	MCO
31	9	2	737.0	740	-3.0	AA	JFK	SFO
40	9	2	759.0	800	-1.0	AA	JFK	LAX
46	9	2	817.0	820	-3.0	AA	JFK	SJU
49	9	2	823.0	825	-2.0	AA	JFK	BOS
52	9	2	828.0	830	-2.0	UA	JFK	LAX
54	9	2	831.0	825	6.0	UA	JFK	SFO
65	9	2	908.0	915	-7.0	AA	JFK	MIA
68	9	2	912.0	910	2.0	AA	JFK	LAX
77	9	2	948.0	955	-7.0	UA	JFK	SFO
88	9	2	1028.0	1030	-2.0	AA	JFK	LAX

Re-arranging rows

To re-order rows according to values in a particular column, use `.sort_values()`.

- By default, rows will be sorted in ascending order.

```
# Re-arrange rows based on values in "dep_time" (ascending order)
df.sort_values("dep_time", ascending = True)
```

```
     month  day  dep_time  sched_dep_time  dep_delay carrier origin dest
0        9    2       8.0            2255       73.0      B6    JFK  BUF
1        9    2      15.0            2359       16.0      B6    JFK  SJU
2        9    2     537.0             545       -8.0      AA    JFK  MIA
3        9    2     542.0             545       -3.0      B6    JFK  BQN
4        9    2     557.0             600       -3.0      B6    JFK  MCO
..     ...  ...       ...             ...        ...     ...    ...  ...
303      9    2       NaN            2035        NaN      9E    JFK  IAD
304      9    2       NaN            1835        NaN      9E    JFK  DFW
305      9    2       NaN            1935        NaN      9E    JFK  JAX
```

```python
# Re-arrange rows based on values in "dep_time" (descending order)
df.sort_values("dep_time", ascending = False)
```

```
     month  day  dep_time  sched_dep_time  dep_delay carrier origin dest
291      9    2    2400.0            2359        1.0      B6    JFK  BQN
290      9    2    2358.0            2359       -1.0      B6    JFK  PSE
289      9    2    2353.0            2145      128.0      B6    JFK  LAS
288      9    2    2353.0            2245       68.0      B6    JFK  BTV
287      9    2    2353.0            2150      123.0      B6    JFK  FLL
..     ...  ...       ...             ...        ...     ...    ...  ...
303      9    2       NaN            2035        NaN      9E    JFK  IAD
304      9    2       NaN            1835        NaN      9E    JFK  DFW
305      9    2       NaN            1935        NaN      9E    JFK  JAX
306      9    2       NaN            1645        NaN      MQ    JFK  ORF
307      9    2       NaN            1940        NaN      MQ    JFK  RDU

[308 rows x 8 columns]
```

Re-arranging rows

- If more than one column names are supplied, each additional column will be used to break ties in the values of the preceding column.

```
df.sort_values(["dep_time", "sched_dep_time"], ascending = [True, False])
```

```
     month  day  dep_time  sched_dep_time  dep_delay carrier origin dest
0        9    2       8.0            2255       73.0      B6    JFK  BUF
1        9    2      15.0            2359       16.0      B6    JFK  SJU
2        9    2     537.0             545       -8.0      AA    JFK  MIA
3        9    2     542.0             545       -3.0      B6    JFK  BQN
4        9    2     557.0             600       -3.0      B6    JFK  MCO
..     ...  ...       ...             ...        ...     ...    ...  ...
294      9    2       NaN            1455        NaN      MQ    JFK  CLE
298      9    2       NaN            1450        NaN      9E    JFK  DCA
296      9    2       NaN            1446        NaN      9E    JFK  BUF
302      9    2       NaN            1435        NaN      9E    JFK  BWI
292      9    2       NaN             600        NaN      EV    JFK  IAD
```

Your turn: New York Flights data

Now let's work on the `flights` data frame.

1. In the `flights` data frame, find
   - The number of flights with an an arrival delay of at least two hours.
   - The number of flights that flew to Houston (either "IAH" or "HOU").
   - The number of flights that departed in summer (July, August, and September).
   - The number of flights that arrive more than two hours late, but did not depart late.
2. Find the flight with longest departure delay. Which airport did this flight originated from?
3. Which flight traveled traveled the greatest distance? Find the origin and destination of this flight.

Column manipulations in data frames

Now we will learn about the common column manipulation in data frames.

1. Creating new columns.
2. Accessing columns.
3. Renaming columns.
4. Re-ordering columns.

We will continue to work on the `df` data frame.

```
df.head(3)
```

```
   month  day  dep_time  sched_dep_time  dep_delay carrier origin dest
0      9    2       8.0            2255       73.0      B6    JFK  BUF
1      9    2      15.0            2359       16.0      B6    JFK  SJU
2      9    2     537.0             545       -8.0      AA    JFK  MIA
```

We can create new columns either using new information or from existing columns.

- This can be done by passing a value or a list of values.
- We can also create a column that from existing columns.

```
df["sched_hour"] = df["sched_dep_time"]//100
df
```

```
     month  day  dep_time  sched_dep_time  ...  carrier origin dest sched_
0        9    2       8.0            2255  ...       B6    JFK  BUF
1        9    2      15.0            2359  ...       B6    JFK  SJU
2        9    2     537.0             545  ...       AA    JFK  MIA
3        9    2     542.0             545  ...       B6    JFK  BQN
4        9    2     557.0             600  ...       B6    JFK  MCO
..     ...  ...       ...             ...  ...      ...    ...  ...
303      9    2       NaN            2035  ...       9E    JFK  IAD
304      9    2       NaN            1835  ...       9E    JFK  DFW
```

- We can use `.select()` to classify flight departure status based on `dep_delay`.

```
df["dep_status"] = np.select(
    [df["dep_delay"] > 0, df["dep_delay"] < 0, df["dep_delay"] == 0],
    ["delayed", "early", "on time"],
    default = "unknown"
)
df[["dep_delay", "dep_status"]]
```

```
     dep_delay dep_status
0         73.0    delayed
1         16.0    delayed
2         -8.0      early
3         -3.0      early
4         -3.0      early
..         ...        ...
303        NaN    unknown
304        NaN    unknown
305        NaN    unknown
```

- After that, count the occurrences of each status.
- On September 13, 2013, there were 127 flights that departed on time or early from JFK.

```
df["dep_status"].value_counts()
```

```
dep_status
delayed    165
early      118
unknown     16
on time      9
Name: count, dtype: int64
```

Just with selecting rows, there are many options to select the columns.

- The simplest syntax is by quoting the column names as a string.

```
df[["origin", "dest"]]
```

```
    origin dest
0      JFK  BUF
1      JFK  SJU
2      JFK  MIA
3      JFK  BQN
4      JFK  MCO
..     ...  ...
303    JFK  IAD
304    JFK  DFW
305    JFK  JAX
306    JFK  ORF
```

- To select columns based on the *type* they hold, use `.selct_dtypes()`.
- Let's take a look at the types of the columns in `df`:

```
df.dtypes
```

```
month                int64
day                  int64
dep_time           float64
sched_dep_time       int64
dep_delay          float64
carrier             object
origin              object
dest                object
sched_hour           int64
dep_status          object
dtype: object
```

All interger columns

```
df.select_dtypes("int")
```

```
     month  day  sched_dep_time  sched_hour
0        9    2            2255          22
1        9    2            2359          23
2        9    2             545           5
3        9    2             545           5
4        9    2             600           6
..     ...  ...             ...         ...
303      9    2            2035          20
304      9    2            1835          18
305      9    2            1935          19
306      9    2            1645          16
307      9    2            1940          19

[308 rows x 4 columns]
```

```python
df.select_dtypes("object")
```

```
    carrier origin dest dep_status
0        B6    JFK  BUF    delayed
1        B6    JFK  SJU    delayed
2        AA    JFK  MIA      early
3        B6    JFK  BQN      early
4        B6    JFK  MCO      early
..      ...    ...  ...        ...
303      9E    JFK  IAD    unknown
304      9E    JFK  DFW    unknown
305      9E    JFK  JAX    unknown
306      MQ    JFK  ORF    unknown
307      MQ    JFK  RDU    unknown

[308 rows x 4 columns]
```

- We can get all columns that begins with "sched":

```
df.loc[:, df.columns.str.startswith("sched")]
```

```
     sched_dep_time  sched_hour
0              2255          22
1              2359          23
2               545           5
3               545           5
4               600           6
..              ...         ...
303            2035          20
304            1835          18
305            1935          19
306            1645          16
307            1940          19
```

- Alternatively, use the `.filter()` method with regular expression.

```
df.filter(regex = "^sched")      # Replace the ^ symbol manually in VSCode
```

```
     sched_dep_time  sched_hour
0              2255          22
1              2359          23
2               545           5
3               545           5
4               600           6
..              ...         ...
303            2035          20
304            1835          18
305            1935          19
306            1645          16
307            1940          19

[308 rows x 2 columns]
```

Renaming columns

There are multiple ways to rename columns.

1. `rename()` a set of columns with a dictionary.
   Example: `{"old_name1": "new_name1", "old_name2": "new_name2"}`.

```python
# Rename columns
df.rename(columns = {"dep_time": "dep",
                     "sched_dep_time": "sched_dep"})
```

```
     month  day    dep  sched_dep  ...  origin dest  sched_hour dep_status
0        9    2    8.0       2255  ...     JFK  BUF          22    delayed
1        9    2   15.0       2359  ...     JFK  SJU          23    delayed
2        9    2  537.0        545  ...     JFK  MIA           5      early
3        9    2  542.0        545  ...     JFK  BQN           5      early
4        9    2  557.0        600  ...     JFK  MCO           6      early
..     ...  ...    ...        ...  ...     ...  ...         ...        ...
303      9    2    NaN       2035  ...     JFK  IAD          20    unknown
304      9    2    NaN       1835  ...     JFK  DFW          18    unknown
```

2. The following is useful if we want to rename **all** columns.

```
# Convert column names to upper case
df.columns = df.columns.str.upper()
df.head(1)
```

```
   MONTH  DAY  DEP_TIME  SCHED_DEP_TIME  ...  ORIGIN DEST SCHED_HOUR DEP_S
0      9    2       8.0            2255  ...     JFK  BUF         22    de

[1 rows x 10 columns]
```

```
# Convert column names back to lower case
df.columns = df.columns.str.lower()
df.head(1)
```

```
   month  day  dep_time  sched_dep_time  ...  origin dest sched_hour dep_s
0      9    2       8.0            2255  ...     JFK  BUF         22    de
```

3. Replace a specific part of column names.

```
# Replace specific parts of column names
df.columns = df.columns.str.replace("_time", "")
df.head(3)
```

```
   month  day    dep  sched_dep  ...  origin dest  sched_hour dep_status
0      9    2    8.0       2255  ...     JFK  BUF          22    delayed
1      9    2   15.0       2359  ...     JFK  SJU          23    delayed
2      9    2  537.0        545  ...     JFK  MIA           5      early

[3 rows x 10 columns]
```

Re-ordering columns

```python
# Select and specify the order of columns
df[["month", "day", "origin", "dest", "dep", "sched_dep", "dep_delay"]]
```

```
     month  day origin dest    dep  sched_dep  dep_delay
0        9    2    JFK  BUF    8.0       2255       73.0
1        9    2    JFK  SJU   15.0       2359       16.0
2        9    2    JFK  MIA  537.0        545       -8.0
3        9    2    JFK  BQN  542.0        545       -3.0
4        9    2    JFK  MCO  557.0        600       -3.0
..     ...  ...    ...  ...    ...        ...        ...
303      9    2    JFK  IAD    NaN       2035        NaN
304      9    2    JFK  DFW    NaN       1835        NaN
305      9    2    JFK  JAX    NaN       1935        NaN
306      9    2    JFK  ORF    NaN       1645        NaN
307      9    2    JFK  RDU    NaN       1940        NaN
```

- Order columns in an alphabetical order.
  - `axis = 1` means the second axis (i.e., columns).

```
# Sort columns in alphabetical order
df.reindex(sorted(df.columns), axis = 1)
```

```
    carrier day    dep  dep_delay ... month origin  sched_dep sched_hour
0        B6   2    8.0       73.0 ...     9    JFK       2255         22
1        B6   2   15.0       16.0 ...     9    JFK       2359         23
2        AA   2  537.0       -8.0 ...     9    JFK        545          5
3        B6   2  542.0       -3.0 ...     9    JFK        545          5
4        B6   2  557.0       -3.0 ...     9    JFK        600          6
..      ...  ...   ...        ... ...   ...    ...        ...        ...
303      9E   2    NaN        NaN ...     9    JFK       2035         20
304      9E   2    NaN        NaN ...     9    JFK       1835         18
305      9E   2    NaN        NaN ...     9    JFK       1935         19
306      MQ   2    NaN        NaN ...     9    JFK       1645         16
307      MQ   2    NaN        NaN ...     9    JFK       1940         19
```

There are often multiple ways for the same task in Python!

To get column names, use `df.columns`.

To get the first column (name `col1`) from `df`:

- `df["col1"]`, `df.loc[:, "col1"]`, `df.iloc[:, 0]`.

To get the first row (name `row1`) from ‘df:

- `df.loc["row1", :]`, `df.iloc[0, :]`.

To get the first value in the first row and first column:

- `df["col1"][0]`, `df.loc["row1", "col1"]`, `df.iloc[0, 0]`.

Continue working on the `flights` data frame.

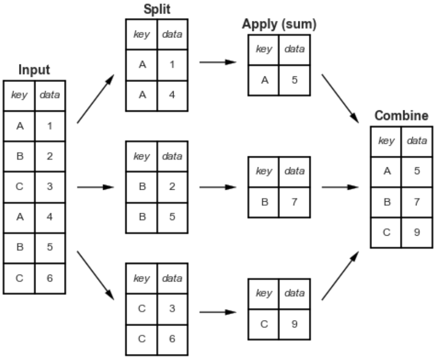
4. What do the missing values in the `dep_time` and `arr_time` columns represent?
5. Compare values in `sched_dep_time`, `dep_time`, and `dep_delay`.
   How would you expect these numbers to be related? What do you actually observe and how do you interpret this?

Grouping and summary statistics

So far, we've learned about working with rows and columns.

`Pandas` gets even more powerful when we add in the ability to **work with groups**.

- The diagram below gives a sense of how these operations can be proceed together.



In the following, we will continue working on the `df` data frame:

- Flights on September 2, 2013 from JFK.

```
df.head()
```

```
   month  day    dep  sched_dep  ...  origin dest  sched_hour dep_status
0      9    2    8.0       2255  ...     JFK  BUF          22    delayed
1      9    2   15.0       2359  ...     JFK  SJU          23    delayed
2      9    2  537.0        545  ...     JFK  MIA           5      early
3      9    2  542.0        545  ...     JFK  BQN           5      early
4      9    2  557.0        600  ...     JFK  MCO           6      early

[5 rows x 10 columns]
```

Let's create a group, select a column, and compute a summary statistics.

- Aggregation always produces a new index on the group level.
- It helps us keep track of the groups we have in the remaining analysis.

```
df.groupby("carrier")[["dep_delay"]].mean()
```

```
          dep_delay
carrier
9E        56.304348
AA        18.142857
B6        36.484127
DL        34.950820
EV       125.333333
HA        -4.000000
MQ        56.235294
UA        16.818182
```

- We can also use `agg()`, which stands for aggregate.

```
df.groupby("carrier")[["dep_delay"]].agg("mean")
```

```
          dep_delay
carrier
9E        56.304348
AA        18.142857
B6        36.484127
DL        34.950820
EV       125.333333
HA        -4.000000
MQ        56.235294
UA        16.818182
US        17.333333
VX        11.888889
```

The syntax is `df.groupby("group_var")[["col_name"]].agg("agg_function")`.

The common functions we can pass to `agg()` are the following:

Aggregation function	**Description**
size()	Number of items
first() , last()	First and last item
mean() , median()	Mean and median
min() , max()	Min and max
std() , var()	Standard deviation and variance
sum()	Sum of all items

Aggregation summarizes each group down to one row.

For multiple operations by group:

```
df.groupby("carrier").agg(
  mean_delay = ("dep_delay", "mean"),
  max_delay = ("dep_delay", "max"))
```

```
         mean_delay  max_delay
carrier
9E        56.304348      296.0
AA        18.142857      100.0
B6        36.484127      307.0
DL        34.950820      299.0
EV       125.333333      222.0
HA        -4.000000       -4.0
MQ        56.235294      167.0
UA        16.818182       89.0
US        17.333333       62.0
VX        11.888889       51.0
```

We can also group a data frame by multiple variables by passing `.groupby()` a list of column names.

- `df.groupby(["var1", "var2"])[["col_name"]].agg("agg_function")`.
- The resulting data frame will have a **multi-index**.

```
df.groupby(["carrier", "origin"])[["dep_delay"]].agg("mean")
```

```
                 dep_delay
carrier origin
9E      JFK      56.304348
AA      JFK      18.142857
B6      JFK      36.484127
DL      JFK      34.950820
EV      JFK     125.333333
HA      JFK      -4.000000
MQ      JFK      56.235294
UA      JFK      16.818182
US      JFK      17.333333
```

- To go back to an index that just informs the position, use `.reset_index()`.

```
df.groupby(["carrier", "origin"])[["dep_delay"]].agg("mean").reset_index()
```

```
  carrier origin   dep_delay
0      9E    JFK   56.304348
1      AA    JFK   18.142857
2      B6    JFK   36.484127
3      DL    JFK   34.950820
4      EV    JFK  125.333333
5      HA    JFK   -4.000000
6      MQ    JFK   56.235294
7      UA    JFK   16.818182
8      US    JFK   17.333333
9      VX    JFK   11.888889
```

- To remove one layer of the index, pass the position you'd like to remove.
- The following example removes the `origin` index and places it to a column.

```
df.groupby(["carrier","origin"])[["dep_delay"]].agg("mean").reset_index(1)
```

```
        origin   dep_delay
carrier
9E         JFK   56.304348
AA         JFK   18.142857
B6         JFK   36.484127
DL         JFK   34.950820
EV         JFK  125.333333
HA         JFK   -4.000000
MQ         JFK   56.235294
UA         JFK   16.818182
US         JFK   17.333333
VX         JFK   11.888889
```

We don't always want to change the index to reflect new groups when performing group-level computations.

- Instead, we can just add the new column to the existing data.
- Use `.transform()` with `.groupby()` in this case.
- Let's compute the worst departure delay on Christmas day by origin airports.

```
df1 = flights.query("month == 12 and day == 25")
df1 = df1[["carrier", "flight", "origin", "sched_dep_time", "dep_delay"]]
df1.head(5)
```

	carrier	flight	origin	sched_dep_time	dep_delay
105232	US	1895	EWR	500	-4.0
105233	UA	1016	EWR	515	9.0
105234	AA	2243	JFK	540	2.0
105235	B6	939	JFK	550	-4.0
105236	AA	301	LGA	600	-4.0

- Next, use `.transform()` to create a new column, and add it to the right of the existing data frame.

```
df1["max_delay"] = df1.groupby("origin")[["dep_delay"]].transform("max")
df1.head(5)
```

```
       carrier  flight origin  sched_dep_time  dep_delay  max_delay
105232      US    1895    EWR             500       -4.0      321.0
105233      UA    1016    EWR             515        9.0      321.0
105234      AA    2243    JFK             540        2.0      234.0
105235      B6     939    JFK             550       -4.0      234.0
105236      AA     301    LGA             600       -4.0      251.0
```

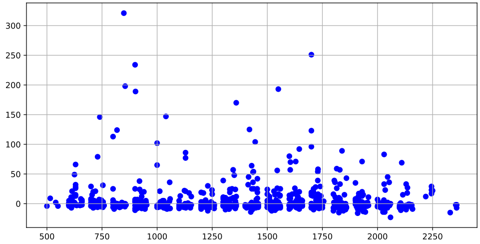
- Compare that to what we would have achieved with `.agg()`.

```
df1 = flights.query("month == 12 and day == 25")
df1 = df1[["carrier", "flight", "origin", "sched_dep_time", "dep_delay"]]
df1.groupby("origin")[["dep_delay"]].agg("max")
```

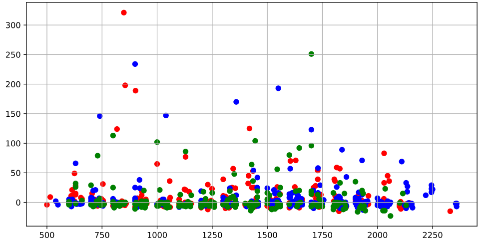
```
        dep_delay
origin
EWR         321.0
JFK         234.0
LGA         251.0
```

Visualizing departure delay by time

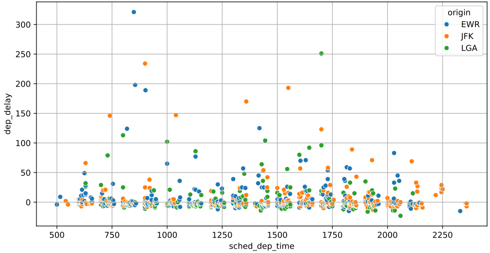
```
import matplotlib.pyplot as plt
plt.figure(figsize = (10, 5))
plt.scatter(df1["sched_dep_time"], df1["dep_delay"], color = "blue")
plt.grid(True)
plt.show()
```



```
colors = {"EWR": "red", "LGA": "green", "JFK": "blue"}
plt.figure(figsize = (10, 5))
for x in df1["origin"].unique():
  df2 = df1[df1["origin"] == x]
  plt.scatter(df2["sched_dep_time"], df2["dep_delay"], color = colors[x])
plt.grid(True)
plt.show()
```



```
import seaborn as sns
plt.figure(figsize = (10, 5))
sns.scatterplot(data = df1,
                x = "sched_dep_time", y = "dep_delay", hue = "origin")
plt.grid(True)
plt.show()
```



The `seaborn` code will be more comprehensible in a few weeks' time.

For now, focus on its purpose:

- It creates a scatterplot with scheduled departure time and departure delay.
- ... also uses color (hue) to denote different flight origins.

Compared to `matplotlib`, `seaborn` offers greater flexibility and more concise code.

We shall explore `seaborn` in more detail in a few weeks' time.

Continue working on the `flights` data.

6. On average, which carrier has the worst departure delays in 2013?
7. Find the most delayed flight to each destination.
8. How do departure and arrival delays vary over the the day?
9. Examine the number of cancelled flights per day. Is there a pattern?