Analysing Data Using Pandas
a Vademecum

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

Load data from CSV

```
import pandas as pd

df = pd.read_csv('../rollercoasters.csv')
df
```

Load data from mongoldb

```python
import pymongo
import pandas as pd
from pymongo import MongoClient

client = MongoClient()
db = client.database_name
collection = db.collection_name

# Query MongoDB with specific fields and convert to DataFrame
results = collection.find({sport: 'tennis', tournament:'davis cup'},
      # Fetch only sport and tournament excluding '_id'
      {'_id': 0, 'sport': 1, 'tournament': 1})

data = pd.DataFrame(list(results))
```

Some recommendations

- Query Optimization:
  - Depending on the size of the collection, fetching all data at once might not be efficient
  - You might want to use query filters (like collection.find({}))
    - to fetch only the necessary data or limit the number of results using limit()
- Data Types:
  - MongoDB stores data in BSON format, so ensure that the data types are appropriate when converted to a DataFrame.
- Memory Usage:
  - If the collection is large, fetching all the data at once might consume a lot of memory
  - Consider fetching data in batches

UNIVERSITÀ DI TORINO

Check the shape

- **`df.shape`**
  - will give you the number of rows and columns
- why you should use it?
  - so to make sure that the data you have loaded is what you expect
    - a typical error is to have no or very few rows
      - typically a wrong query

```
In 140   1  df.shape
            Executed at 2023.11.24 14:49:27 in 10ms

Out 140      (142, 21)
```

UNIVERSITÀ DI TORINO

Checking the types

- **`df.types`**
  - why you should use it:
    - some fields may just have an apparent correct type:
      - e.g. some dates may appear as "12/03/2024"
        - but some of them may be represented as a string rather than a daytime
          - check the types to make sure they are what you expect

```
In 141   1  df.dtypes
Executed at 2023.11.24 14:49:28 in 13ms

Out 141     max_speed              int64
            avg_speed              int64
            ride_time              int64
            ride_length            int64
            max_pos_gs           float64
            max_neg_gs           float64
            max_lateral_gs       float64
            total_air_time       float64
            drops                  int64
            highest_drop_height    int64
            inversions             int64
            dtype: object
```

To convert

- How to convert:
  - in general:
    - `df[column_name] = df[column_name].astype('int')`
      - replace **`int`** with the type you need
  - to convert to a date time:
    - `pd.to_datetime(df[column_name])`
  - This will modify the entire column

UNIVERSITÀ DI TORINO

Accessing

- A column

```
df['custom_design']
```

- A row

```
df.loc[0]
```

```
In 142  1  df['custom_design']
Executed at 2023.11.24 14:49:29 in 76ms

Out 142
0      0
1      0
2      0
3      0
4      0
      ..
137    1
138    1
139    1
140    1
141    1
Name: custom_design, Length: 142, dtype: int64
```

```
In 143  1  df.loc[0]
Executed at 2023.11.24 14:49:30 in 14ms

Out 143
max_speed                39
avg_speed                12
ride_time                63
ride_length            1496
max_pos_gs             2.59
max_neg_gs            -0.27
max_lateral_gs         1.71
total_air_time          0.0
drops                     2
highest_drop_height      19
inversions                0
Name: 0, dtype: object
```

Setting a column as an index

- Normally the indexes in a data frame are integer from 0 to num_columns
  - however we can assign specific indexes to each row, typically by using some unique ides, generally provided by a database relation
    - e.g.

```
df.set_index('park_id')
```

- this removes park_id from the columns
  - simplifying any queries and results
- and identifies each element row using its id

Checking the name of the columns

```
df.columns
```

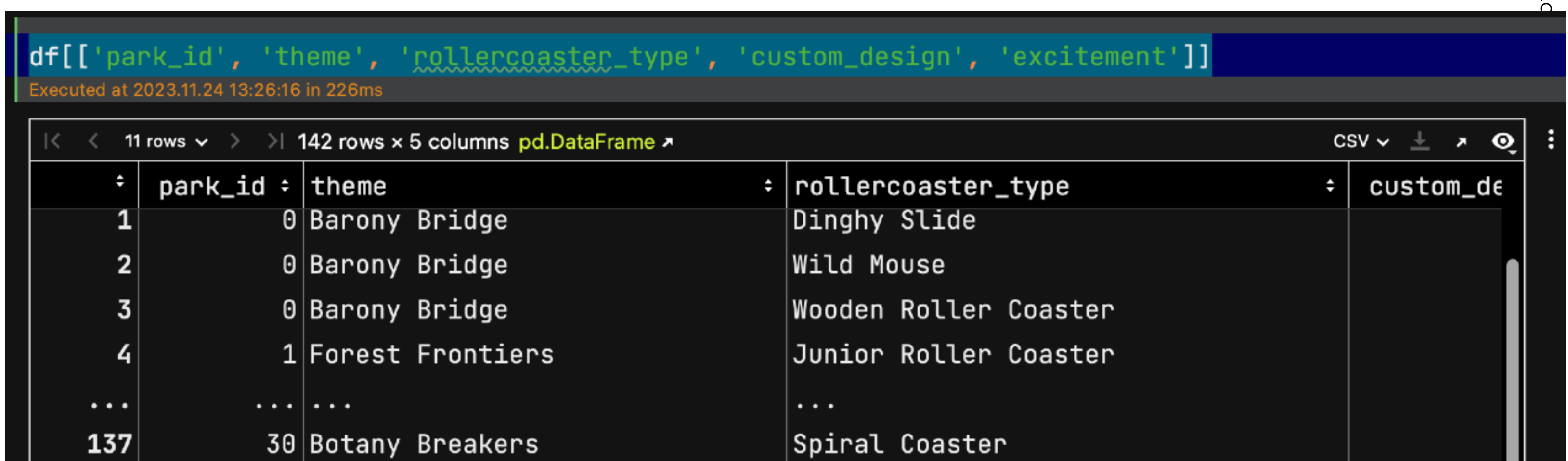
- why it is important
  - to remember their name when you type commands that require the name of columns
  - to change the names in order to make the code easier to read
    - e.g. a column originally called AVG_TMP could be renamed into Average_Temperature
      - making the programme easier to read and interpret

UNIVERSITÀ
DI TORINO

Creating a view on a DF

```
df[['park_id', 'theme', 'rollercoaster_type',
     'custom_design', 'excitement']]
```

- Why you should use it
  - to work with smaller (apparent) data and focus only on the parts of the data you really need
    - if you do not need the rest of the data, just reassign the view to the original df variable
      - `df = df[['park_id',…`



```
df[['park_id', 'theme', 'rollercoaster_type', 'custom_design', 'excitement']]
```

Executed at 2023.11.24 13:26:16 in 226ms

11 rows 142 rows × 5 columns pd.DataFrame CSV

	park_id	theme	rollercoaster_type	custom_de
1	0	Barony Bridge	Dinghy Slide	
2	0	Barony Bridge	Wild Mouse	
3	0	Barony Bridge	Wooden Roller Coaster	
4	1	Forest Frontiers	Junior Roller Coaster	
...	...	...	...	
137	30	Botany Breakers	Spiral Coaster	

UNIVERSITÀ DI TORINO

Understanding the data

- Describing a number column

```
df['max_speed'].describe()
```

- Why you should use it?
  - to check the actual distribution of the data. For example it returns the 25/50/75 percentile mean
    - which tells you if some data has outliers
      - in the image, to 25% corresponds an increase by 4 or 5
        - but the max is 89, so it is probably an outlier

```
df['max_speed'].describe()
Executed at 2023.11.24 13:26:16 in 215ms

count    142.000000
mean      43.570423
std        8.980197
min       29.000000
25%       38.000000
50%       42.500000
75%       47.000000
max       89.000000
Name: max_speed, dtype: float64
```

Get all rows that match a condition

- You must create a filter
  - `df[condition]`
- and then apply it to the dataframe
  - `df[filter]`
- so in total it is
  - `df[[condition]]`
- e.g.

```
df.loc[df['max_speed'] > 42]
```

```
df.loc[df['max_speed'] > 42]
```

Executed at 2023.11.24 14:55:30 in 36ms

11 rows ⌄ 71 rows × 21 columns pd.DataFrame ↗ CSV ⌄

	park_id	theme	rollercoaster_type	custo
3	0	Barony Bridge	Wooden Roller Coaster	

Better: use query

- This is easier to use

```
df.query('max_speed>42')
```

- Note that use of the quotes are around the entire condition

```
df.loc[df['max_speed'] > 42]
```

Executed at 2023.11.24 14:55:30 in 36ms

11 rows 71 rows × 21 columns pd.DataFrame CSV

	park_id	theme	rollercoaster_type	custo
3	0	Barony Bridge	Wooden Roller Coaster	
6	2	Haunted Harbour	Wooden Roller Coaster	
13	4	Pacific Pyramids	Vertical Drop Coaster	
15	5	Mel's World	Inverted Roller Coaster	
17	5	Mel's World	Suspended Swinging Coaster	

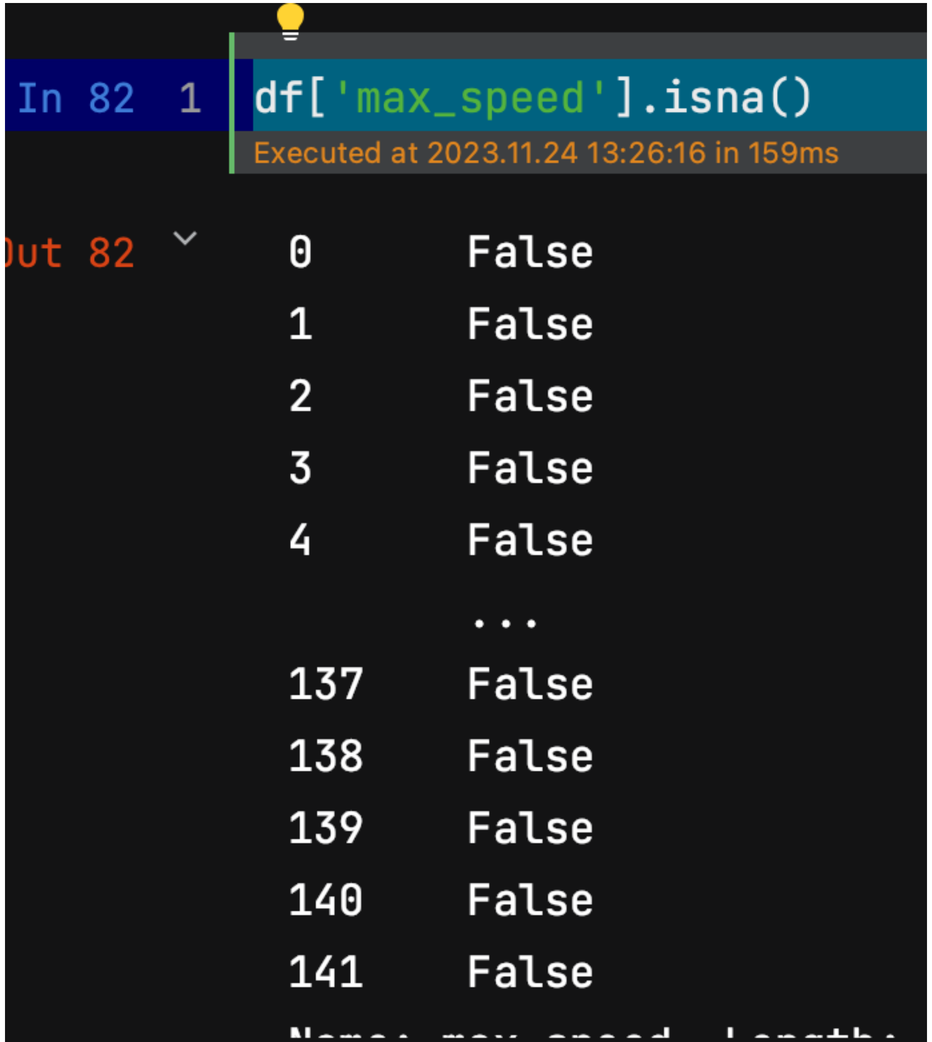
Checking for invalid values

- Some values in some rows can be invalid
  - i.e. None, NaT, Nan…

```
df.isna()
```

- will return all the rows with a None value
  - not very useful. Typically you would check the value in specific columns

```
df['max_speed'].isna()
```



UNIVERSITÀ DI TORINO

What to do with Nas?

- Several strategies, e.g.
  - remove the rows from teh df (note the negation of the filter using ~)

```
df = df.loc[~df['max_speed'].isna()]
```

  - insert the average of the column

```
df.loc[df['max_speed'].isna()].max_speed = df['max_speed'].mean()
```

Finding Duplicates

- Why you should use it?
  - because duplicates
    - can be errors (information repeated)
    - can influence measures such as means and standard deviation
  - two types of duplications
    - full duplication (entire element - typically an error in input), The filter is: `df.duplicated()`
    - partial duplication
      - the significant columns such as id and name are identical
      - some minor information is different
        - e.g. date of creation - in that case it may mean that there is uncertainty on the information
        - the filter is:

```
df.duplicated(subset=['column_1, column_2, …, column_n'])
```

UNIVERSITÀ DI TORINO

Inspecting duplicates

```python
# you can do loc on it because it is a filter
df.loc[~df.duplicated(subset=['theme', 'rollercoaster_type'])]
```

```python
df.loc[df.duplicated(subset=['theme', 'rollercoaster_type'])]
```

Executed at 2023.11.24 13:26:16 in 284ms

Out 90

7 rows × 21 columns pd.DataFrame

	park_id	theme	rollercoaster_type	custom
36	10	Karts And Coasters	Wooden Roller Coaster	
40	11	Three Monkeys Park	Looping Roller Coaster	
41	11	Three Monkeys Park	Looping Roller Coaster	
46	12	Crumbly Woods	Corkscrew Roller Coaster	
52	14	Canry Mines	Vertical Drop Coaster	
115	26	Paradise Pier	Looping Roller Coaster	
120	27	Swamp Cove	Inverted Roller Coaster	

Check the duplicates and remove them

```
df = df.loc[~df.duplicated(subset=['theme', 'rollercoaster_type'])]
```

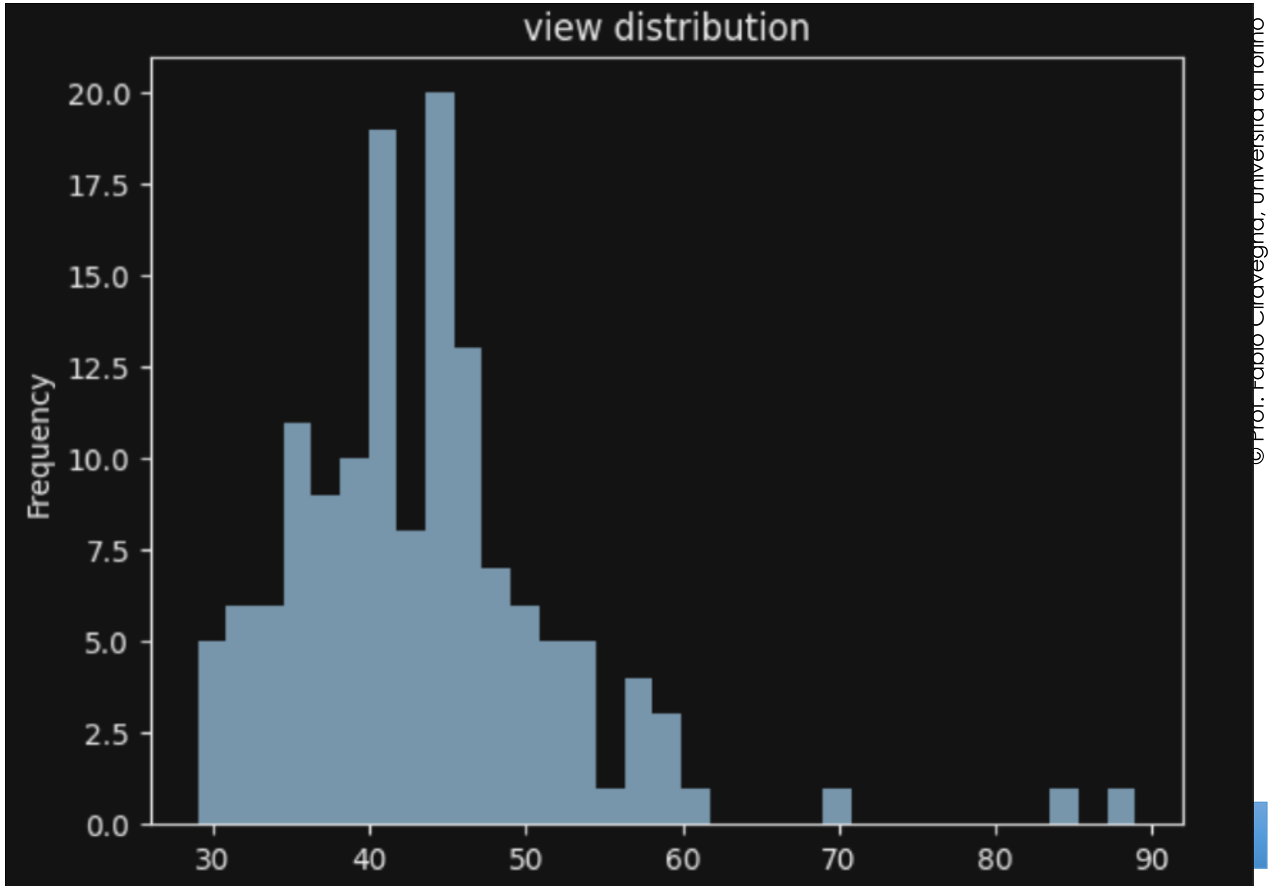
Univariate analysis

- you can analyse the distribution of a column by writing
  - **`df[column_name].value_counts()`**
- e.g. how many rollercoasters were built per year
  - you may want to reduce that to the top 10
    - just use **`.head`** on the value_counts call
      - and if you want to plot it, you can
-

Studying the distribution: Histograms

```
import matplotlib.pyplot as plt

ax = df['max_speed'].plot(kind='hist', bins=33, title='view distribution')
ax.x_label = "Max Speed"
ax.y_label = "Total number"
plt.show()
```



This will create an histogram dividing the data into 33 bins regularly paced

The image shows a generally normal distribution with a few outliers on the right hand side
(the latter ones are elements of interest to investigate)

Check the outliers

- Being an outlier does not mean being an error. Some outliers are just fully justified
  - check them one by one

```
df.loc[df['max_speed']>80]
```

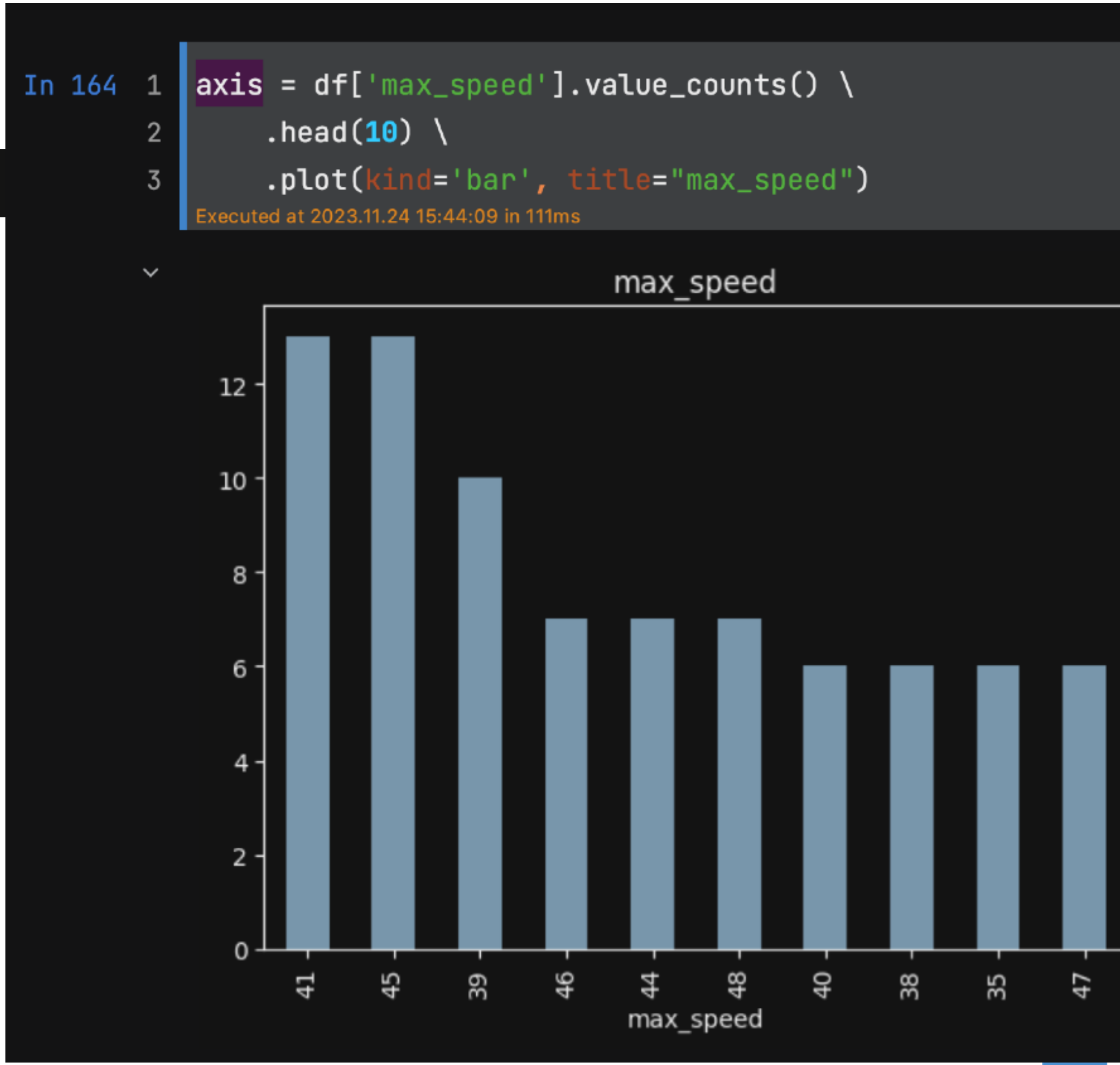
```
In 9  1  df.loc[df['max_speed']>80]
Executed at 2023.11.27 12:43:01 in 40ms
```

Out 9

2 rows × 21 columns pd.DataFrame

	park_id	theme	rollercoaster_type	custom_design	excitement	excite
66	19	Razor Rocks	Air Powered Vertical Coaster	0	8.42	Very H
85	21	Vertigo Views	Hypercoaster	0	9.48	Very H

```python
axis = df['max_speed'].value_counts() \
    .head(10) \
    .plot(kind='bar', title="max_speed"
```



Note!
The values on the x axis are sorted by the value of the y values

How to fix the outliers"

- How to fix an outlier
  - remove all elements above a specific reasonable manual value

```
df = df.loc[~df['max_speed']>1000]
```

  - insert a reasonable manual value into these elements

```
df.loc[df['max_speed']>1000].max_speed = 1000
```

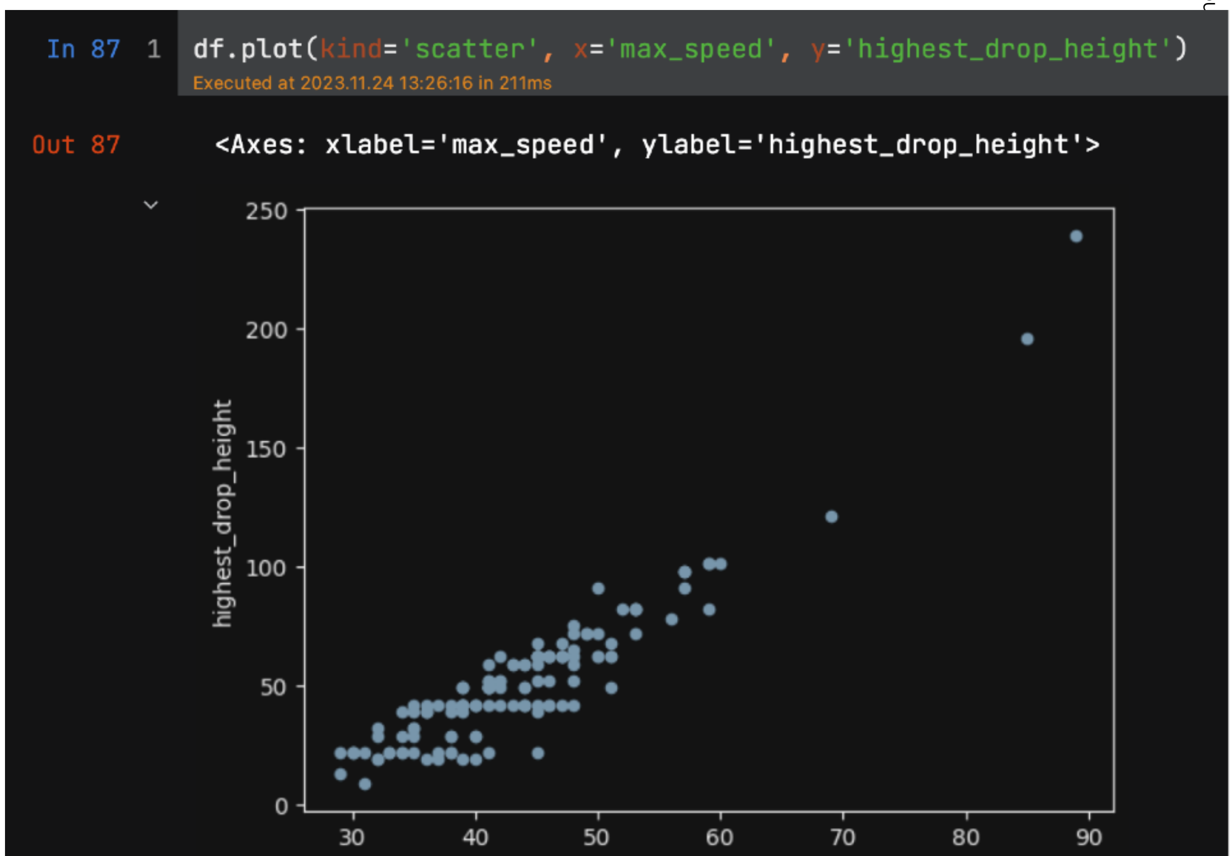
I have not checked these operations
there may be minor errors

Analysing correlations

- you can use scatterplots to see correlation between two variables (e.g. max speed and highest drop height)

```
df.plot(kind='scatter', x='max_speed', y='highest_drop_height')
```

- why you should use it?
  - a correlation between two variables is an important information about the data
    - it shows if two variables may be dependent or not
      - note! correlation is not causation!!!
        - it is just an hypothesis to be verified!



Correlations using the confusion matrix

```
df[['nausea', 'excitement']].dropna().corr()
```

```
# correlations
```

In 166

```
df[['nausea', 'excitement']].dropna().corr()
```

Executed at 2023.11.24 15:47:03 in 38ms

Out 166

2 rows ⌄ 2 rows × 2 columns pd.DataFrame ↗

	nausea	excitement
nausea	1.000000	0.349712
excitement	0.349712	1.000000

Group by and aggregation

- Grouping allow to study the rows that meet a specific condition, e.g. the same value on a column

```
df.groupby('theme')
```

- Groups can be described (as we did for columns)

```
df.groupby('theme').describe()
```

Out 97

1-11 30 rows × 128 columns pd.DataFrame

CSV

theme	park_id count	mean	std	min	25%	50%
Adrenaline Heights	6.0	29.0	0.000000	29.0	29.0	29
Arid Heights	9.0	20.0	0.000000	20.0	20.0	20
Barony Bridge	4.0	0.0	0.000000	0.0	0.0	0
Botany Breakers	8.0	30.0	0.000000	30.0	30.0	30
Bumbly Bazaar	4.0	24.0	0.000000	24.0	24.0	24
Butterfly Dam	3.0	17.0	0.000000	17.0	17.0	17

Groups by multiple columns values

```
df.groupby(['country_of_origin', 'programming_language']).describe(
```

```
In 170
df.groupby(['country_of_origin', 'programming_language']).describe()

Executed at 2023.11.24 15:51:56 in 28ms
```

Out 170

10 rows ⌄ 10 rows × 4 columns pd.DataFrame ↗ CSV ⌄

country_of_origin	programming_language	experience count	unique	top
Italy	Pyhon	1	1	pro
Nigeria	Java	2	2	good
	Pyhon	1	1	good
The Netherlands	Javascript	1	1	good
	Kotlin	2	2	good

getting a group

```
df.groupby('country_of_origin').get_group('USA')
```

- it returns a group that can be accessed as a data frame

```
In 133  df.groupby('country_of_origin').get_group('USA')
Executed at 2023.11.24 13:51:44 in 18ms
Out 133
```

2 rows × 3 columns pd.DataFrame

	programming_language	country_of_origin	experience
0	Pyhon	USA	intermediate
4	Javascript	USA	intermediate

- e.g.

```
df.groupby('country_of_origin').get_group('Nigeria')['experience'].value_counts()
```

A complex example

```
# Filter the DataFrame for programmers who know Kotlin
kotlin_programmers = df[df['programming_language'] == 'Kotlin']

# Group by country and count Kotlin programmers for each country
kotlin_programmers_by_country =
kotlin_programmers.groupby('country_of_origin').size().reset_index(
    name='kotlin_programmers_count')

# Calculate the total programmers for each country
total_programmers_by_country =
df.groupby('country_of_origin').size().reset_index(name='total_programmers')

# Merge the two dataframes on 'country_of_origin'
merged_df = pd.merge(kotlin_programmers_by_country, total_programmers_by_country,
on='country_of_origin')

# Calculate the percentage for each country
merged_df['percentage_kotlin_programmers'] = (merged_df['kotlin_programmers_count'] / merged_df[
    'total_programmers']) * 100
merged_df
```

UNIVERSITÀ
DI TORINO

Questions?