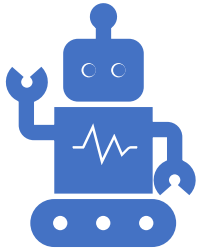
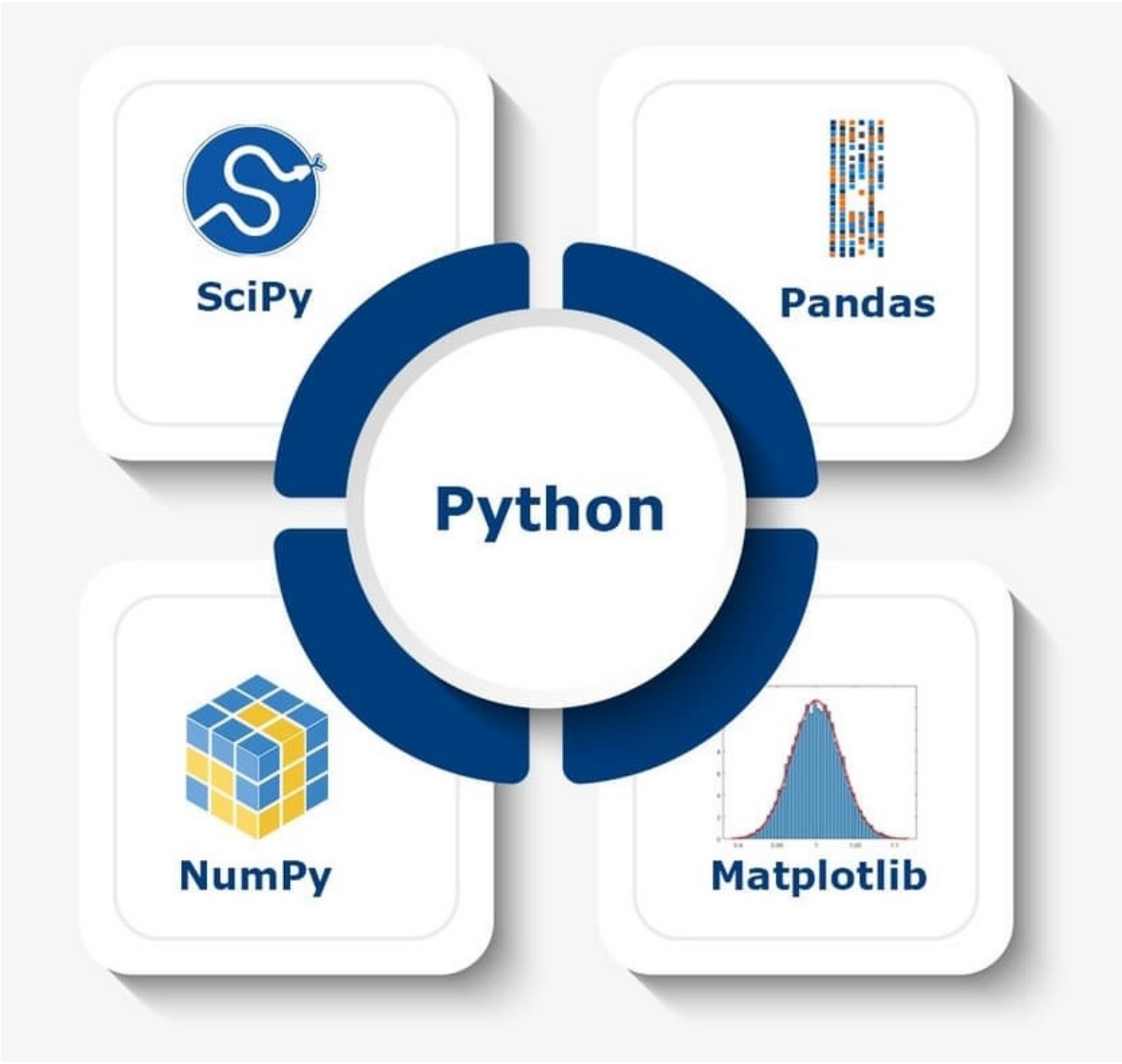


Python for Machine Learning III

Dr. Junya Michanan

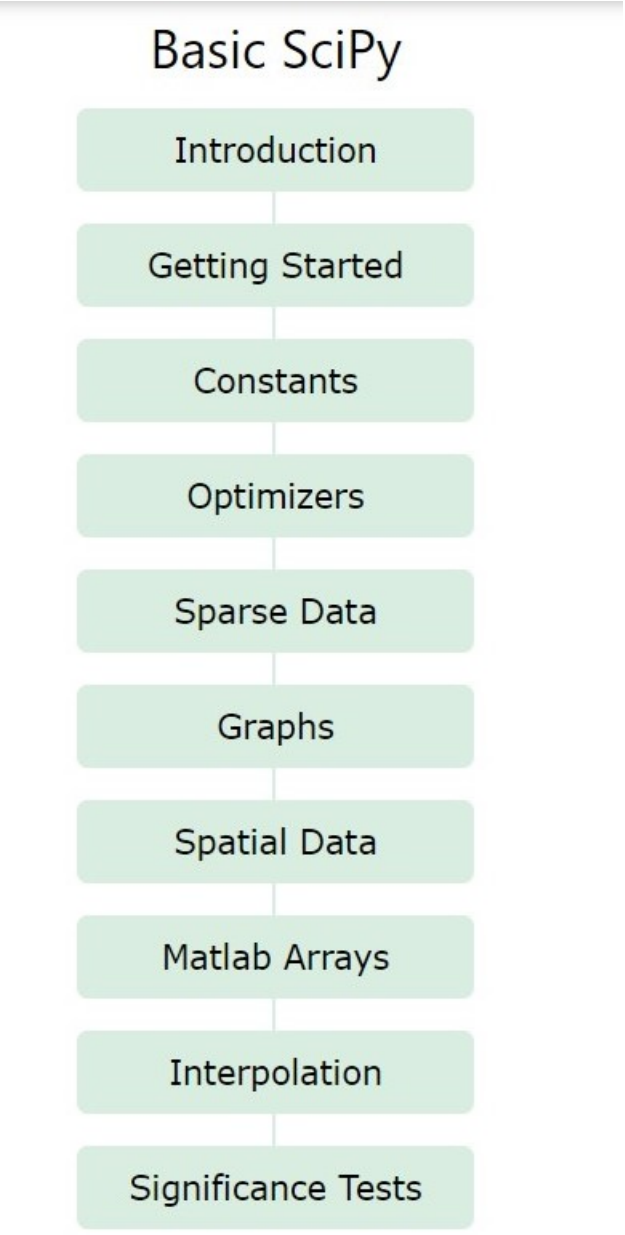
Agenda

- SciPy
- Pandas
- Class Activity
- Homework



What is SciPy?

- SciPy is a scientific computation library that uses NumPy underneath.
- SciPy stands for Scientific Python.
- It provides more utility functions for optimization, stats and signal processing.
- Like NumPy, SciPy is open source so we can use it freely.
- SciPy was created by NumPy's creator Travis Olliphant.



SciPy Getting Started

Installation of SciPy

```
C:\Users\Your Name>pip install scipy
```

Imports of SciPy

Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the from scipy import module statement:

How many cubic meters are in one liter:

```
from scipy import constants

print(constants.liter)
```

SciPy Constants

- As SciPy is more focused on scientific implementations, it provides many built-in scientific constants.
- These constants can be helpful when you are working with Data Science.
- PI is an example of a scientific constant.

```
from scipy import constants

print(constants.pi)
```

Constant Units using: dir()

```
from scipy import constants

print(dir(constants))
```

List of All Constants

- ['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'C2F', 'C2K', 'ConstantWarning', 'F2C', 'F2K', 'G', 'Julian_year', 'K2C', 'K2F', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann', 'Tester', 'Wien', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '_obsolete_constants', 'absolute_import', 'acre', 'alpha', 'angstrom', 'arcmin', 'arcminute', 'arcsec', 'arcsecond', 'astronomical_unit', 'atm', 'atmosphere', 'atomic_mass', 'atto', 'au', 'bar', 'barrel', 'bbl', 'c', 'calorie', 'calorie_IT', 'calorie_th', 'carat', 'centi', 'codata', 'constants', 'convert_temperature', 'day', 'deci', 'degree', 'degree_Fahrenheit', 'deka', 'division', 'dyn', 'dyne', 'e', 'eV', 'electron_mass', 'electron_volt', 'elementary_charge', 'epsilon_0', 'erg', 'exa', 'exbi', 'femto', 'fermi', 'find', 'fine_structure', 'fluid_ounce', 'fluid_ounce_US', 'fluid_ounce_imp', 'foot', 'g', 'gallon', 'gallon_US', 'gallon_imp', 'gas_constant', 'gibi', 'giga', 'golden', 'golden_ratio', 'grain', 'gram', 'gravitational_constant', 'h', 'hbar', 'hectare', 'hecto', 'horsepower', 'hour', 'hp', 'inch', 'k', 'kgf', 'kibi', 'kilo', 'kilogram_force', 'kmh', 'knot', 'lambda2nu', 'lb', 'lbf', 'light_year', 'liter', 'litre', 'long_ton', 'm_e', 'm_n', 'm_p', 'm_u', 'mach', 'mebi', 'mega', 'metric_ton', 'micro', 'micron', 'mil', 'mile', 'milli', 'minute', 'mmHg', 'mph', 'mu_0', 'nano', 'nautical_mile', 'neutron_mass', 'nu2lambda', 'ounce', 'oz', 'parsec', 'pebi', 'peta', 'physical_constants', 'pi', 'pico', 'point', 'pound', 'pound_force', 'precision', 'print_function', 'proton_mass', 'psi', 'pt', 'short_ton', 'sigma', 'speed_of_light', 'speed_of_sound', 'stone', 'survey_foot', 'survey_mile', 'tebi', 'tera', 'test', 'ton_TNT', 'torr', 'troy_ounce', 'troy_pound', 'u', 'unit', 'value', 'week', 'yard', 'year', 'yobi', 'yotta', 'zebi', 'zepto', 'zero_Celsius', 'zetta']

Unit Categories

Metric	Binary	Mass	Angle	Time
Length	Pressure	Volume	Speed	Temperature
	Energy	Power	Force	

Unit Categories

Metric (SI) Prefixes:

Return the specified unit in **meter** (e.g. `centi` returns `0.01`)

```
from scipy import constants

print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0
print(constants.mega)     #1000000.0
print(constants.kilo)     #1000.0
print(constants.hecto)    #100.0
print(constants.deka)     #10.0
print(constants.deci)     #0.1
print(constants.centi)    #0.01
print(constants.milli)    #0.001
print(constants.micro)    #1e-06
print(constants.nano)     #1e-09
print(constants.pico)     #1e-12
print(constants.femto)    #1e-15
print(constants.atto)     #1e-18
print(constants.zepto)    #1e-21
```

BinaryPrefixes:

Return the specified unit in **bytes** (e.g. `kibi` returns `1024`)

```
from scipy import constants

print(constants.kibi)    #1024
print(constants.mebi)    #1048576
print(constants.gibi)    #1073741824
print(constants.tebi)    #1099511627776
print(constants.pebi)    #1125899906842624
print(constants.exbi)    #1152921504606846976
print(constants.zebi)    #1180591620717411303424
print(constants.yobi)    #1208925819614629174706176
```

Speed:

Return the specified unit in **meters per second** (e.g. `speed_of_sound` returns `340.5`)

```
from scipy import constants

print(constants.kmh)             #0.2777777777777778
print(constants.mph)             #0.44703999999999994
print(constants.mach)            #340.5
print(constants.speed_of_sound) #340.5
print(constants.knot)            #0.5144444444444445
```

SciPy Optimizers

- Optimizers are a set of procedures defined in SciPy that either find the minimum value of a function, or the root of an equation.
- Optimizer Functions --essentially, all of the algorithms in Machine Learning are nothing more than a complex equation that needs to be minimized with the help of given data.

Find root of the equation `x + cos(x)`:

```
from scipy.optimize import root
from math import cos

def eqn(x):
  return x + cos(x)

myroot = root(eqn, 0)

print(myroot.x)
```

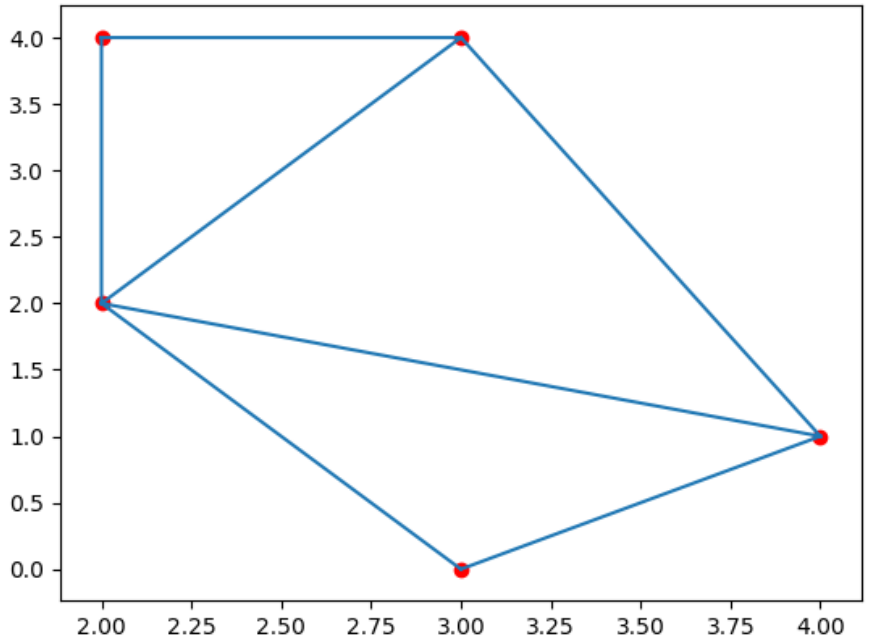
```
[-0.73908513]
```

SciPy Spatial Data

- Spatial data refers to data that is represented in a geometric space.
- E.g. points on a coordinate system.
- We deal with spatial data problems on many tasks.
- E.g. finding if a point is inside a boundary or not.
- SciPy provides us with the module scipy.spatial, which has functions for working with spatial data.

Create a triangulation from following points:

```
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt

points = np.array([
  [2, 4],
  [3, 4],
  [3, 0],
  [2, 2],
  [4, 1]
])

simplices = Delaunay(points).simplices

plt.triplot(points[:, 0], points[:, 1], simplices)
plt.scatter(points[:, 0], points[:, 1], color='r')

plt.show()
```



Statistical Description of Data

number of observations (nobs)

minimum and maximum values = minmax

mean

variance

skewness

kurtosis

Statistical Description of Data

```
import numpy as np
from scipy.stats import describe

v = np.random.normal(size=100)
res = describe(v)

print(res)
```

```
DescribeResult(
  nobs=100,
  minmax=(-2.0991855456740121, 2.1304142707414964),
  mean=0.11503747689121079,
  variance=0.99418092655064605,
  skewness=0.013953400984243667,
  kurtosis=-0.671060517912661
)
```

Normality Tests (Skewness and Kurtosis)

Normality tests are based on the skewness and kurtosis.
The `normaltest()` function returns p value for the null hypothesis:
"x comes from a normal distribution".

Skewness (ความเบ้หรือสมมาตรของข้อมูล):

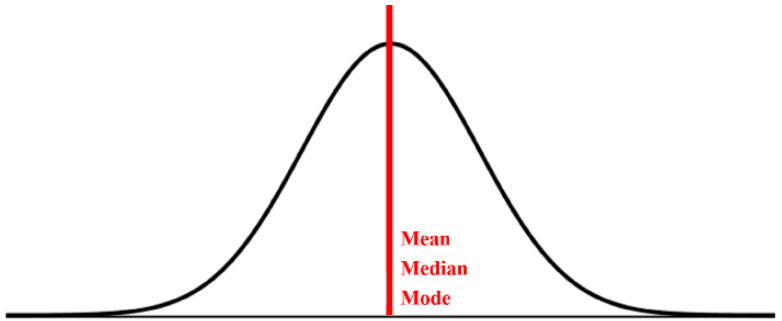
- A measure of symmetry in data.
- For normal distributions it is 0.
- If it is negative, it means the data is skewed left.
- If it is positive it means the data is skewed right.

Kurtosis (ความโด่งของข้อมูล):

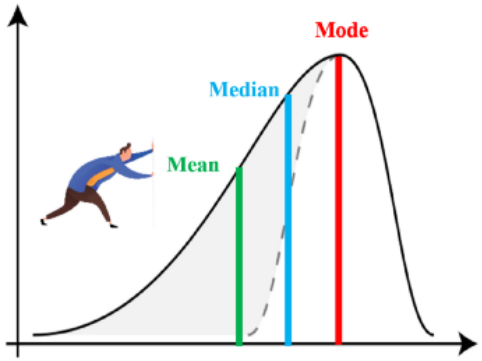
- A measure of whether the data is heavy or lightly tailed to a normal distribution.
- **Positive** kurtosis means heavy tailed.
- **Negative** kurtosis means lightly tailed.

Normality Tests (Skewness and Kurtosis)

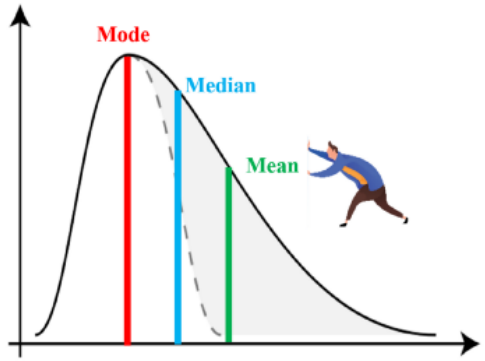
Skewness ความเบ้หรือสมมาตรของข้อมูล



normal distributions

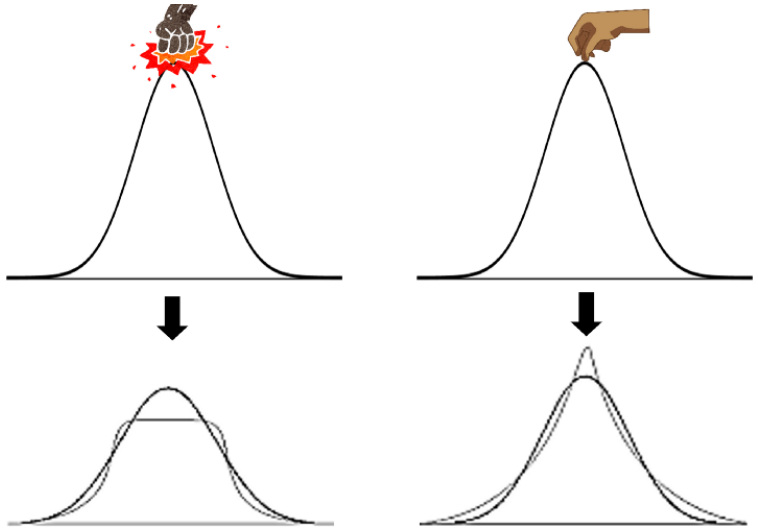


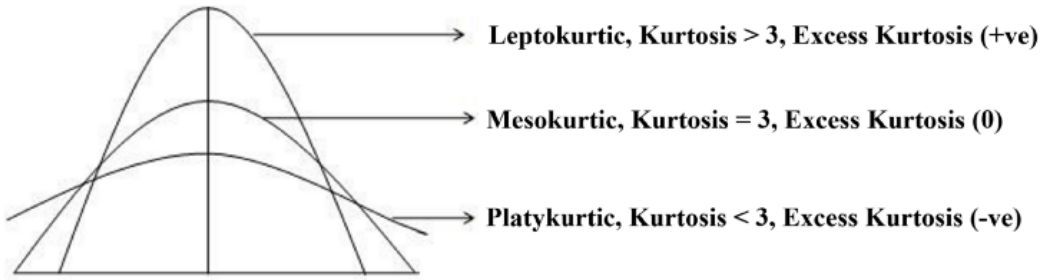
positive



negative

Kurtosis ความโด่งของข้อมูล





Pandas

- Pandas is a Python library.
- Pandas is used to analyze data.
- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Pandas Getting Started

- Installation of Pandas

```
C:\Users\Your Name>pip install pandas
```

- Import Pandas

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Pandas Series

- What is a Series?
  - A Pandas Series is like a column in a table.
  - It is a one-dimensional array holding data of any type.

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

Key/Value Objects as Series

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

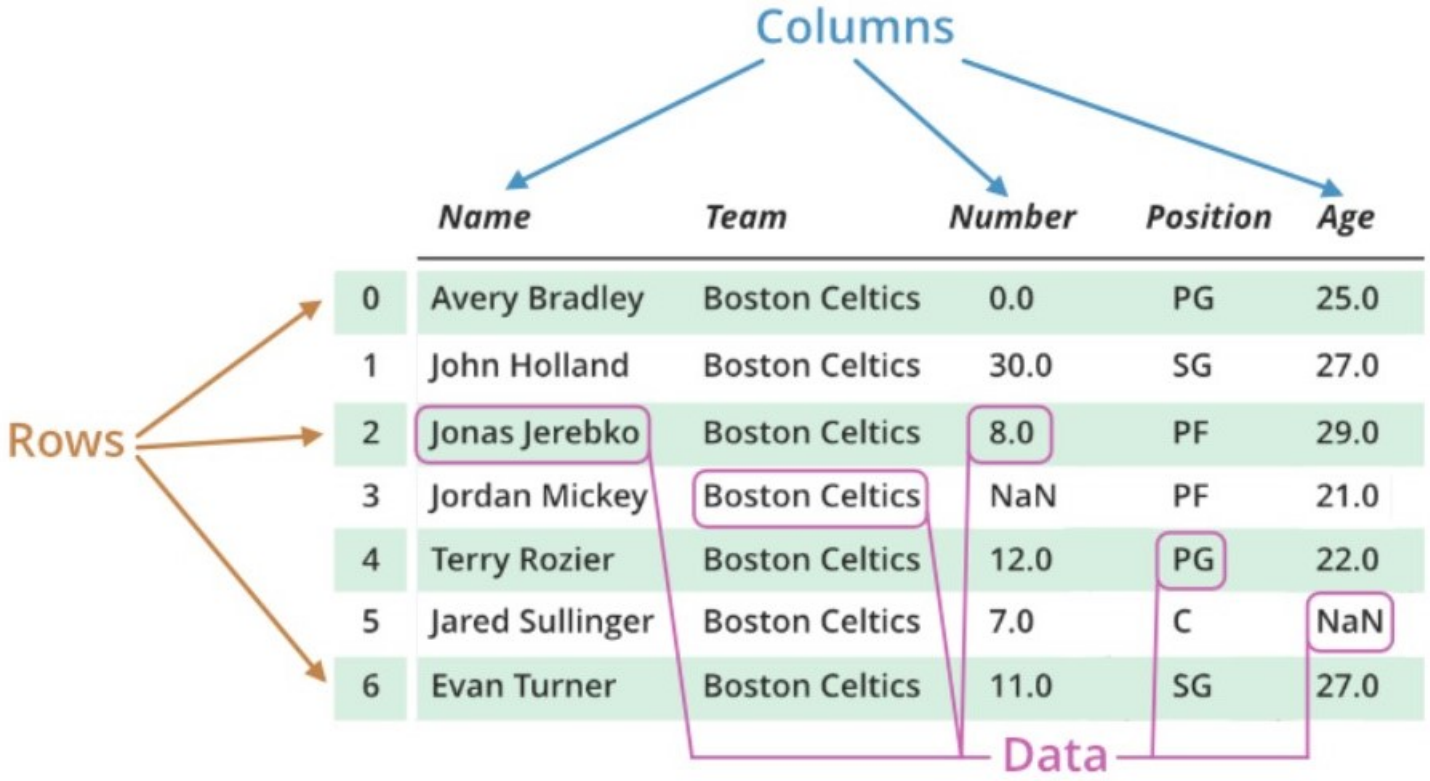
```
day1    420
day2    380
day3    390
dtype: int64
```

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

Pandas DataFrames

- Data sets in Pandas are usually multi-dimensional tables, called DataFrames.
- Series is like a column, a DataFrame is the whole table.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Pandas DataFrames

Create a DataFrame from two Series:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.
Pandas use the `loc` attribute to return one or more specified row(s)

```
#refer to the row index:
print(df.loc[0])
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

```
   calories  duration
0       420        50
1       380        40
```

Pandas DataFrames

- Named Indexes

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Locate Named Indexes

Use the named index in the `loc` attribute to return the specified row(s).

```
#refer to the named index:
print(df.loc["day2"])
```

```
calories    380
duration     40
Name: day2, dtype: int64
```

Pandas Read CSV (comma separated values)

- Downoad CSV file: https://www.w3schools.com/python/pandas/data.csv

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

```
print(df)
```

```
Duration,Pulse,Maxpulse,Calories
60,110,130,409.1
60,117,145,479.0
60,103,135,340.0
45,109,175,282.4
45,117,148,406.0
60,102,127,300.0
60,110,136,374.0
45,104,134,253.3
...
60,110,145,300.0
60,115,145,310.2
75,120,150,320.4
75,125,150,330.4
```

data.csv

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
...
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Pandas Read JSON

- Download JSON file link: https://www.w3schools.com/python/pandas/data.js

```
import pandas as pd

df = pd.read_json('data.json')

print(df.to_string())
```

```
print(df)
```

```
{
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    ...
    "165":300.4,
    "166":310.2,
    "167":320.4,
    "168":330.4
  }
}
```

data.json

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0

...

165	60	110	145	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
..	...	...	...	...
164	60	105	140	290.8
165	60	110	145	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

[169 rows x 4 columns]

```
import pandas as pd

data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

Dictionary as JSON

- **JSON = Python Dictionary**
- JSON objects have the same format as Python dictionaries.

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.5
```

Pandas - Analyzing DataFrames

- Viewing the Data
  - One of the most used method for getting a quick overview of the DataFrame, is the head() method.
  - The head() method returns the headers and a specified number of rows, starting from the top, returns the first 5 rows as the default.

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head())
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0

```
print(df.head(10))
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
5	60	102	127	300.5
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0

```
print(df.tail())
```

	Duration	Pulse	Maxpulse	Calories
164	60	105	140	290.8
165	60	110	145	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

Info About the Data

```
print(df.info())
```

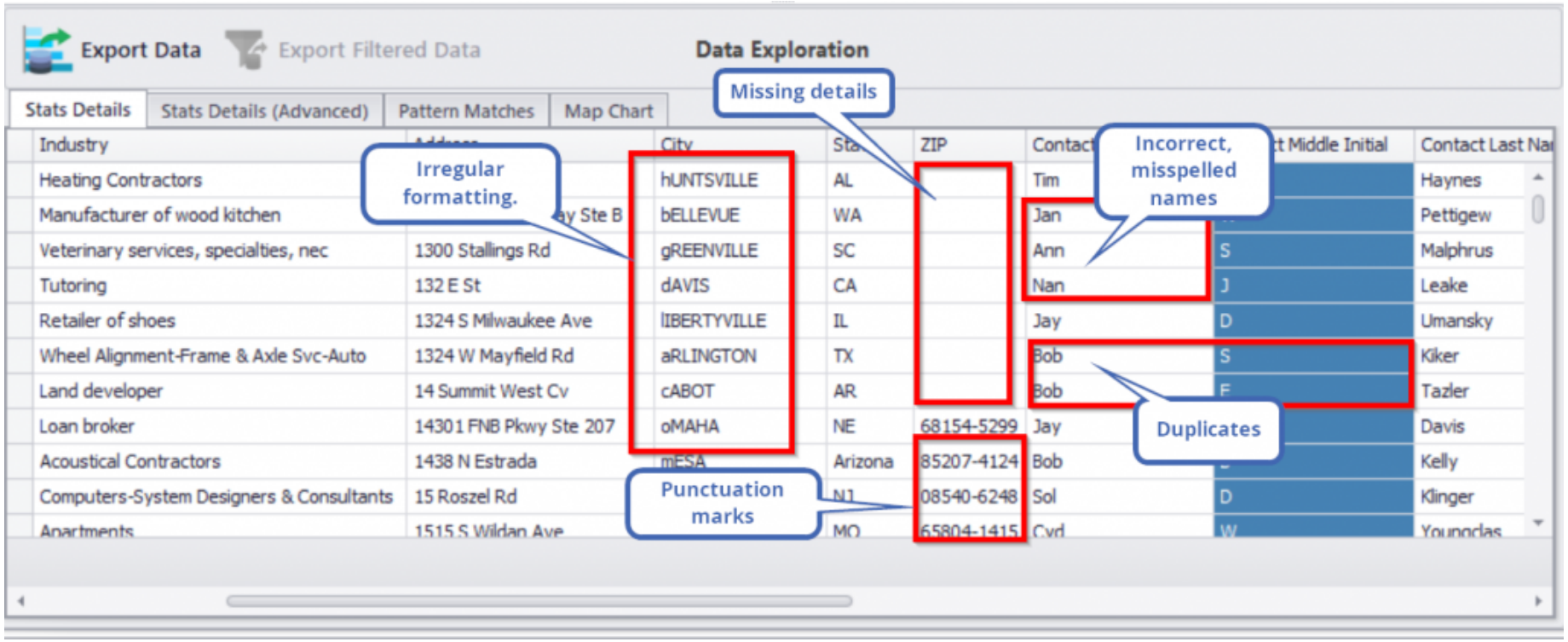
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

there are 5 rows with no value at all, in the "Calories" column

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called *cleaning data*.

Pandas - Cleaning Data

- Data cleaning means fixing bad data in your data set.
- Bad data could be:
  - Empty cells (ไม่มีข้อมูล)
  - Data in wrong format (ผิดรูปแบบ)
  - Wrong data (ไม่ถูกต้อง)
  - Duplicates (ซ้ำซ้อน)



Our Data Set

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	2020/12/26	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

1. The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

2. The data set contains wrong format ("Date" in row 26).

3. The data set contains wrong data ("Duration" in row 7).

4. The data set contains duplicates (row 11 and 12).

Cleaning Data

- Download ink of dirty data: https://www.w3schools.com/python/pandas/dirtydata.csv

1. Pandas - Cleaning Empty Cells

```
import pandas as pd

df = pd.read_csv('dirtydata.csv')

new_df = df.dropna()

print(new_df.to_string())
```

#Notice in the result that some rows have been removed (row 18, 22 and 28).

#These rows had cells with empty values.

By default, the `dropna()` method returns a *new* DataFrame, and will not change the original.

Remove all rows with NULL values:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Now, the `dropna(inplace = True)` will NOT return a new DataFrame, but it will remove all rows containg NULL values from the original DataFrame.

Cleaning Data

Pandas - Replace Empty Values

```
import pandas as pd

df = pd.read_csv('dirtydata.csv')

df.fillna(130, inplace = True)
```

Replace NULL values with the number 130

Replace Using Mean, Median, or Mode

```
import pandas as pd

df = pd.read_csv('dirtydata.csv')

x = df["Calories"].mean() #or median, mode

df["Calories"].fillna(x, inplace = True)
```

Replace Only For a Specified Columns

```
import pandas as pd

df = pd.read_csv('dirtydata.csv')

df["Calories"].fillna(130, inplace = True)
```

Replace NULL values in the "Calories" columns with the number 130

Mean = the average value (the sum of all values divided by number of values).

Median = the value in the middle, after you have sorted all values ascending.

Mode = the value that appears most frequently.

Pandas - Data Correlations

- Finding Relationships
- A great aspect of the Pandas module is the corr() method.
- The corr() method calculates the relationship between each column in your data set.

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.corr())
```

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922721
Pulse	-0.155408	1.000000	0.786535	0.025120
Maxpulse	0.009403	0.786535	1.000000	0.203814
Calories	0.922721	0.025120	0.203814	1.000000

Perfect Correlation:

We can see that "Duration" and "Duration" got the number `1.000000`, which makes sense, each column always has a perfect relationship with itself.

Good Correlation:

"Duration" and "Calories" got a `0.922721` correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.

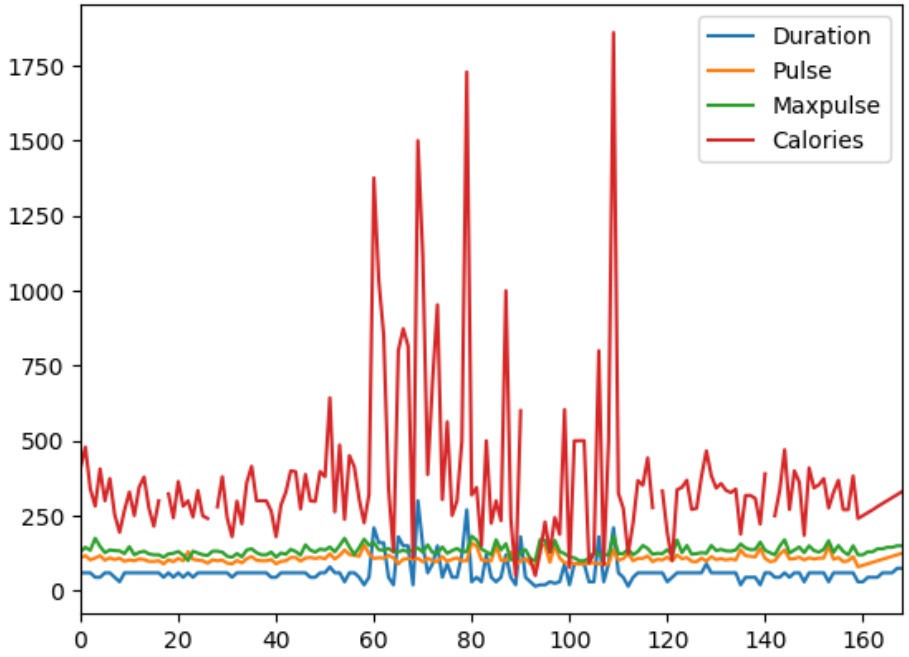
Bad Correlation:

"Duration" and "Maxpulse" got a `0.009403` correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

Pandas - Plotting

- Pandas uses the plot() method to create diagrams.
- We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

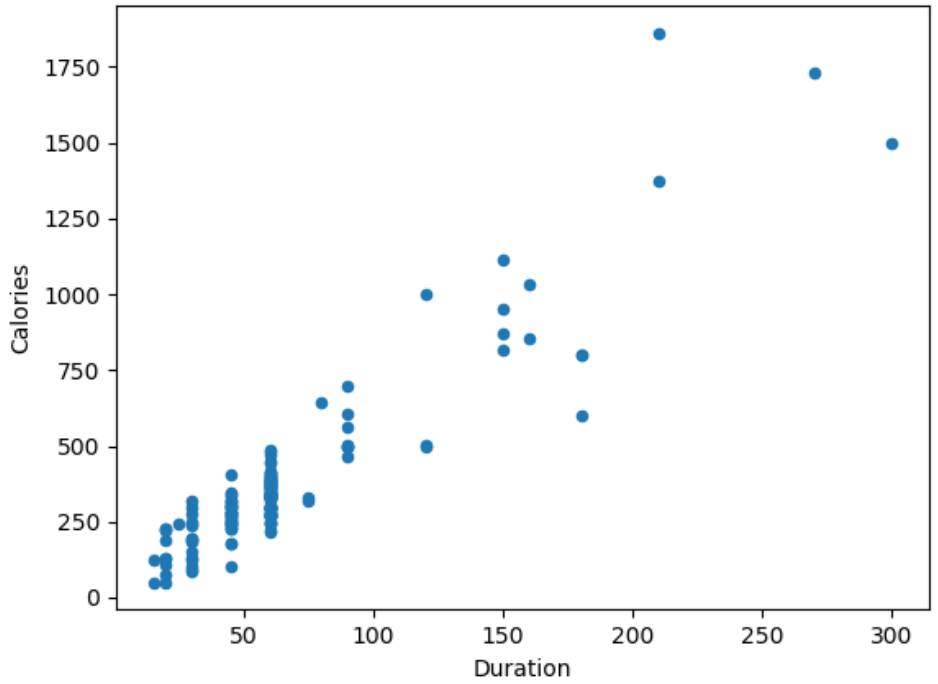
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()

plt.show()
```



Scatter Plot

- Specify that you want a scatter plot with the kind argument:
  - kind = 'scatter'
- A scatter plot needs an x- and a y-axis.
- In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.
- Include the x and y arguments like this:
  - x = 'Duration', y = 'Calories'

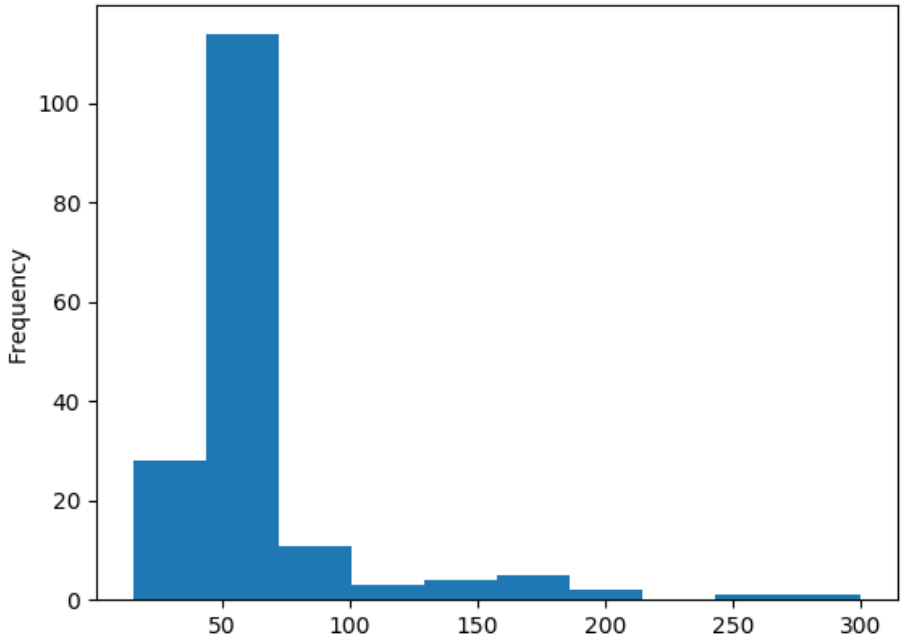
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```



Histogram

- Use the kind argument to specify that you want a histogram:
  - kind = 'hist'
- A histogram needs only one column.
- A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

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

df = pd.read_csv('data.csv')

df["Duration"].plot(kind = 'hist')

plt.show()
```



Class 4’s Activity - Code Learning (10 points)

- Gather all the codes into .py or .ipynb files
- Run the codes and make sure they execute successfully
- Capture the images of the codes and the running results
- Submit the files and screen shots to E-Learning class assignment

SciPy Exercises – 10 Points

- Finish SciPy Exercises in w3schools.com
  - Go to https://www.w3schools.com/python/scipy/exercise.php
    - SCIPY Constants
    - SCIPY Optimizers
  - Copy all the codes and put in JupyterNotebook (.ipynb) or .py
  - Capture screens with results and submit along with the code files
  - Combine all the exercise into 1 file (add comments to separate the exercises)

Pandas Exercises – 10 Points

- Finish Pandas Exercises in w3schools.com
  - Go to https://www.w3schools.com/python/pandas/exercise.asp
    - Finish Pandas Series
    - Finish Pandas DataFrames
    - Finish Pandas Data Cleaning
    - Finish Pandas Corellations
    - Finish Pandas Plotting
  - Copy all the codes and put in JupyterNotebook (.ipynb) or .py
  - Capture screens with results and submit along with the code files
  - Combine all the exercise into 1 file (add comments to separate the exercises)