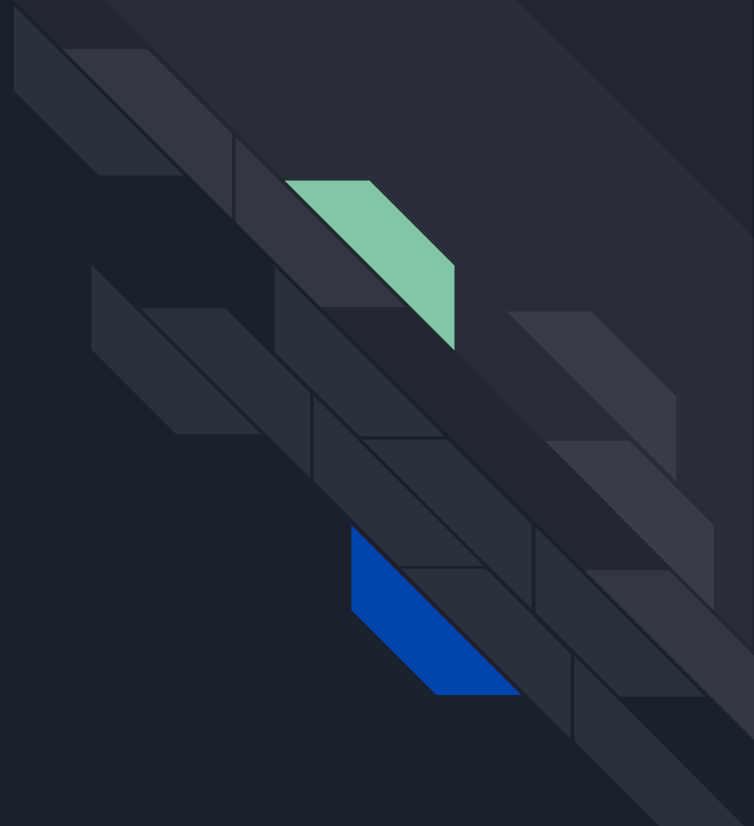


Introduction to Data Science with Python

Chapter 3



Topics of the course

2. Chapter

3. Chapter

4. Chapter

Python Fundamentals

Basic concepts,
Variables, basic data
structures, functions

Data Wrangling & Simple visualizations

How to process data with
pandas and visualize it
with matplotlib

Visualizations & Modelling

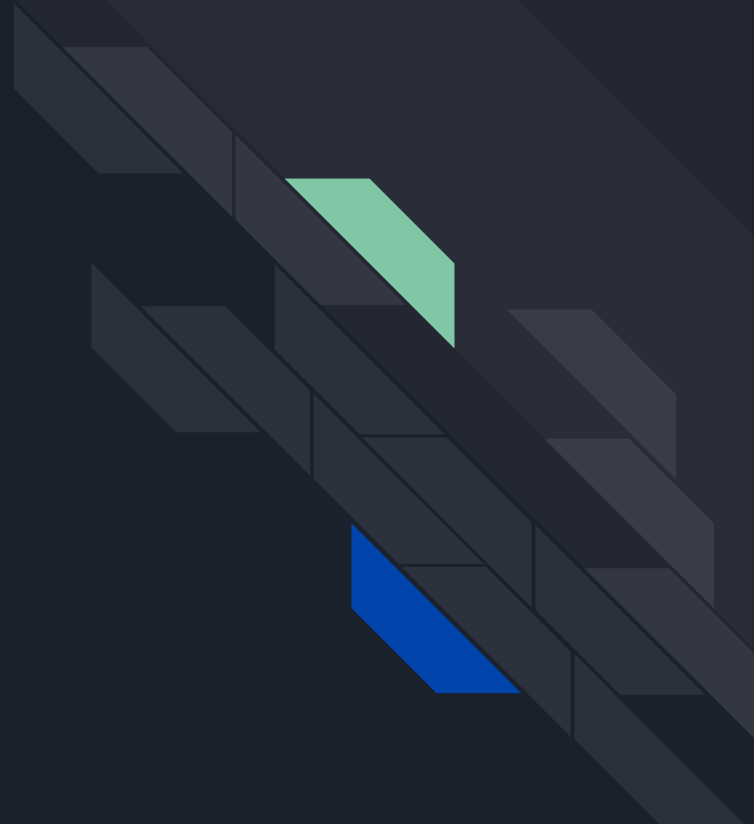
More plots with matplotlib
and seaborn and an
introduction to modelling



What you learn in this Chapter...

- ... how to use Python libraries
- ... work with data in a DataFrame
- ... filter, merge and group your data
- ... visualize data with simple plots

Data Wrangling & Visualization **Libraries**



Libraries

- A collection of functions is bundled in a **library**
- we import these libraries and can use the defined functions
- Some libraries come with a Python installation, some need to be installed

matplotlib



... for plotting and visualization

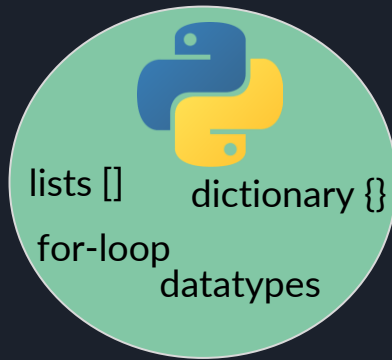
pandas

... for working with tabular data (Excel-files, csv-files,...)

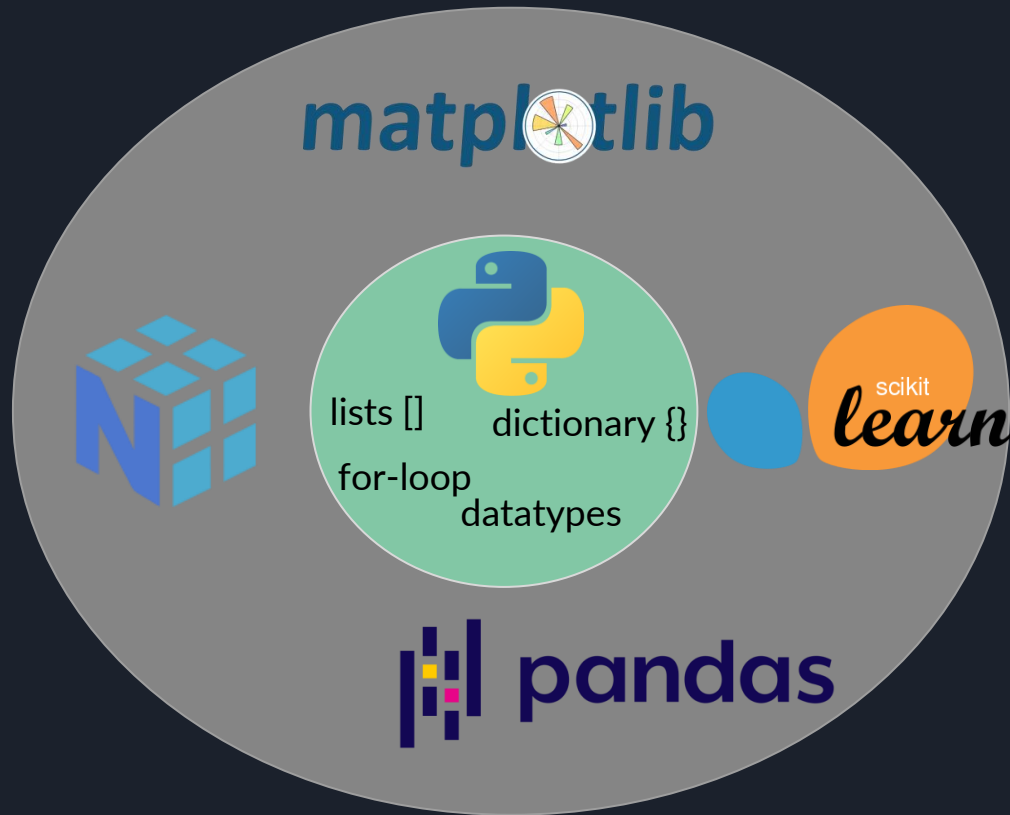


... creating machine learning models

Libraries



Libraries





Function & Methods

Functions

```
a = function_name(parameter)
```




Function & Methods

Functions

```
a = function_name(parameter)
```

Methods

```
a = "a string!"  
a = a.upper() # 'A STRING!'
```



Import Libraries



```
import library
```

```
a = library.function_name()
```



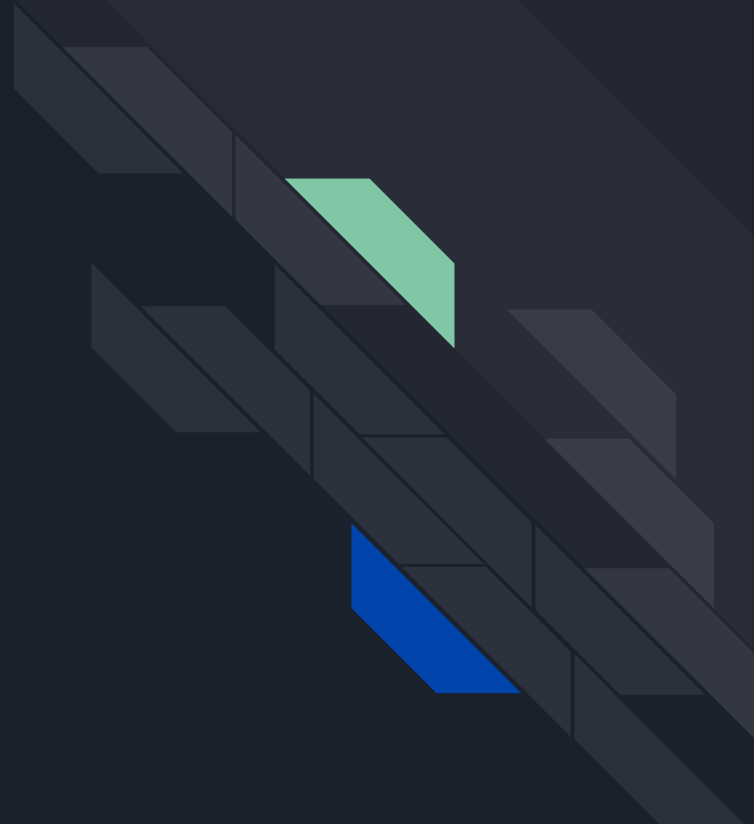
Import Libraries



```
import library as l  
  
a = l.function_name()
```

Data Wrangling & Visualization

Arrays with Numpy





NumPy - library



- Library for scientific computing
- Work with lists, matrices or higher dimensional structures
- NumPy lists have much more functionality than usual lists

```
import numpy as np
```

```
a = np.array([1,2,3,4])
```



NumPy - library



- Library for scientific computing
- Work with lists, matrices or higher dimensional structures
- NumPy lists have much more functionality than usual lists

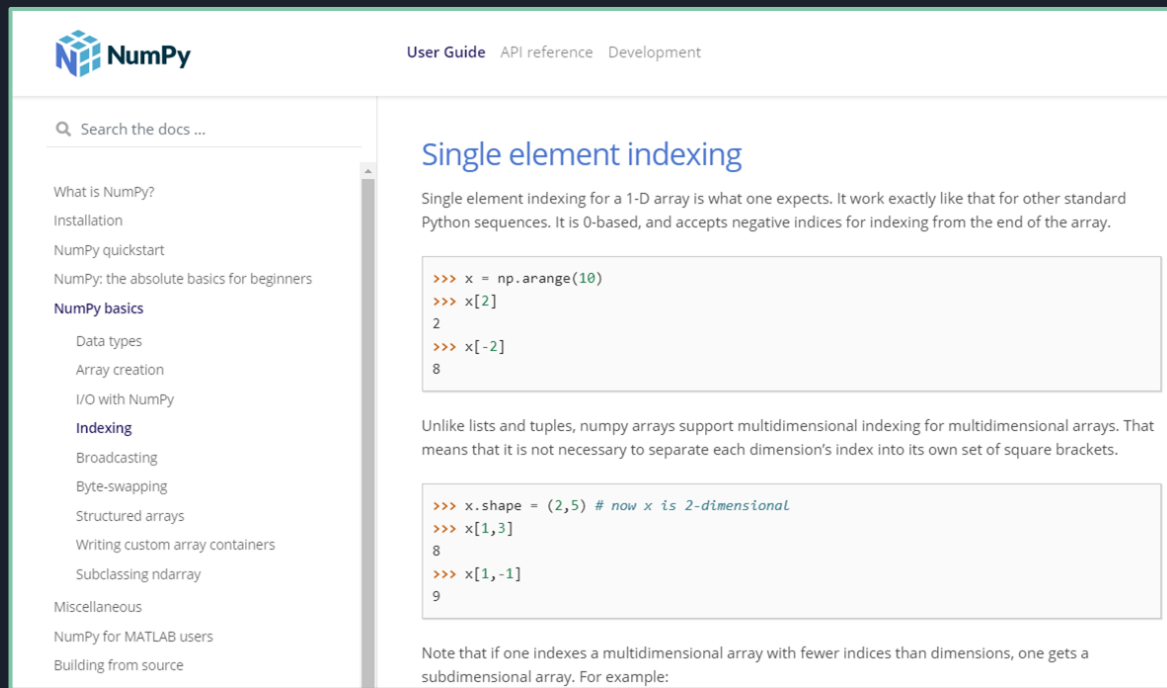
```
import numpy as np

a = np.array([1,2,3,4])
a.sum() # 10
a.mean() # 2.5
a.std() # 1.118...
```



Documentation

- Explanations to the methods and functions
- Often include examples and tutorials
- <https://numpy.org/doc/>



The screenshot displays the NumPy documentation website. At the top, the NumPy logo is on the left, and navigation links for 'User Guide', 'API reference', and 'Development' are on the right. A search bar is positioned below the logo. The left sidebar contains a table of contents with categories like 'What is NumPy?', 'Installation', 'NumPy quickstart', 'NumPy: the absolute basics for beginners', 'NumPy basics' (which is expanded to show sub-topics like 'Data types', 'Array creation', 'I/O with NumPy', 'Indexing', 'Broadcasting', 'Byte-swapping', 'Structured arrays', 'Writing custom array containers', 'Subclassing ndarray', and 'Miscellaneous'), 'NumPy for MATLAB users', and 'Building from source'. The main content area is titled 'Single element indexing' and includes a paragraph explaining that single element indexing for a 1-D array works like standard Python sequences, being 0-based and accepting negative indices. Below this, two code blocks show examples: the first creates an array `x = np.arange(10)` and shows `x[2]` returning 2 and `x[-2]` returning 8; the second shows a 2D array `x` with shape (2,5) and demonstrates indexing `x[1,3]` to get 8 and `x[1,-1]` to get 9. A note at the bottom states that indexing a multidimensional array with fewer indices than dimensions results in a subdimensional array.

NumPy

User Guide API reference Development

Search the docs ...

What is NumPy?
Installation
NumPy quickstart
NumPy: the absolute basics for beginners
NumPy basics
Data types
Array creation
I/O with NumPy
Indexing
Broadcasting
Byte-swapping
Structured arrays
Writing custom array containers
Subclassing ndarray
Miscellaneous
NumPy for MATLAB users
Building from source

Single element indexing

Single element indexing for a 1-D array is what one expects. It work exactly like that for other standard Python sequences. It is 0-based, and accepts negative indices for indexing from the end of the array.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

Unlike lists and tuples, numpy arrays support multidimensional indexing for multidimensional arrays. That means that it is not necessary to separate each dimension's index into its own set of square brackets.

```
>>> x.shape = (2,5) # now x is 2-dimensional
>>> x[1,3]
8
>>> x[1,-1]
9
```

Note that if one indexes a multidimensional array with fewer indices than dimensions, one gets a subdimensional array. For example:



NumPy - Append



```
import numpy as np

a = np.array([1,2,3,4])
np.append(a, [5,6,7]) # array([1, 2, 3, 4, 5, 6, 7])
```


NumPy - Append



```
import numpy as np

a = np.array([1,2,3,4])
np.append(a, [5,6,7]) # array([1, 2, 3, 4, 5, 6, 7])
a = np.append(a, [5,6,7])
```



Compute with arrays - Broadcasting



```
import numpy as np
a = np.array([1,2,3])
a * 2 # array([2, 4, 6])
a ** 2 # array([1, 4, 9])
a - 1 # array([0, 1, 2])
```



Compute with arrays



```
import numpy as np
a = np.array([1,2,3])
b = np.array([1,1,1])
a + b # array([2, 3, 4])
```



Indexing

```
import numpy as np

a = np.array([1,2,3,4])
a[0:2] # array([1,2])
```





Boolean Indexing



```
import numpy as np

a = np.array([1,2,3,4])
a[[True,True,False,False]] # array([1,2])
```



Boolean Indexing



```
import numpy as np

a = np.array([1,2,3,4])
a <= 2 # [True, True, False, False]
```

Boolean Indexing

```
import numpy as np

a = np.array([1,2,3,4])
a <= 2 # [True, True, False, False]
a[a<=2] # array([1,2])
```



Exercise 1

```
import numpy as np
```

```
a = np.array([1,2,3,4])
```

```
a = np.append(a, [5,6,7])# [1,2,3,4,5,6,7]
```

```
a[a<=2] # [1,2] (boolean indexing)
```

```
a.sum() # 28
```


Data Wrangling & Visualization

Tabular Data with Pandas





pandas - library



```
import pandas as pd
```

```
data = {  
    "Name": ["Clara", "Tom", "Sarah", "John"],  
    "Age"  : [20, 24, 19, 21]}
```

```
df = pd.DataFrame(data)
```

	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

DataFrame - data structure





Read data

- Pandas can read data from files
- Various data formats possible



```
df = pd.read_csv('path_to_file')  
df = pd.read_excel('path_to_file')  
df = pd.read_sql_table('postgres://db')
```

First look at the data



```
df.head()
```

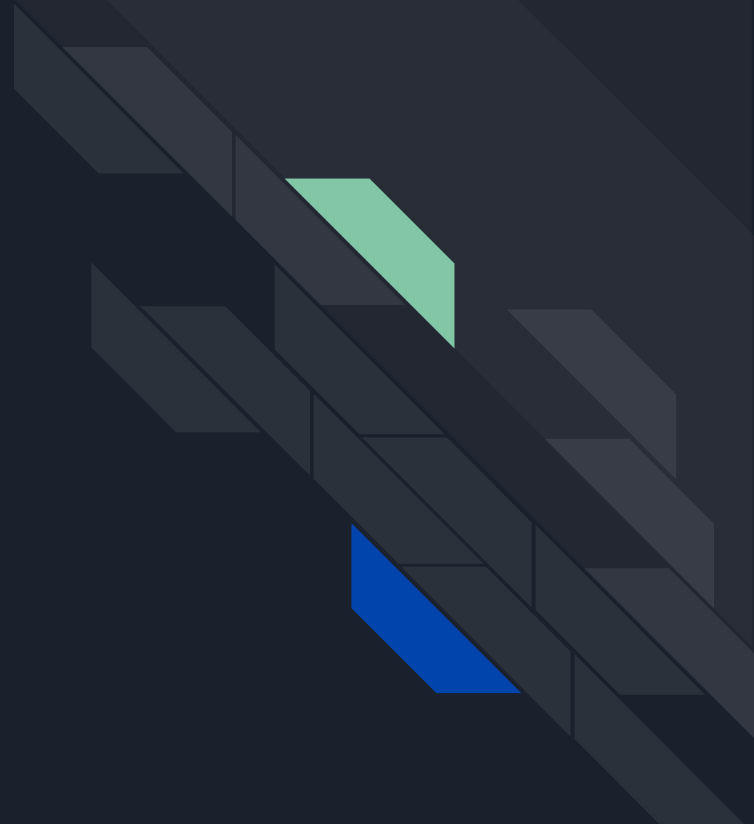
	id	price	neighbourhood_group_cleansed	latitude
0	28684898	\$50.00	Neukölln	52.473978
1	22607348	\$10.00	Treptow - Köpenick	52.468095
2	21019199	\$35.00	Neukölln	52.481810
3	21919556	\$99.00	Pankow	52.537269
4	4820648	\$39.00	Friedrichshain-Kreuzberg	52.491483



```
df.describe()
```

	id	price	latitude	longitude	bathrooms
count	1.353100e+04	13531.000000	13531.000000	13531.000000	13508.000000
mean	1.573089e+07	70.082625	52.509956	13.405871	1.095203
std	8.580394e+06	255.451132	0.030773	0.058517	0.335469
min	2.695000e+03	0.000000	52.346203	13.103557	0.000000
25%	8.041528e+06	30.000000	52.489082	13.374950	1.000000
50%	1.697254e+07	45.000000	52.509229	13.416764	1.000000
75%	2.264464e+07	70.000000	52.532808	13.439258	1.000000
max	2.986735e+07	9000.000000	52.651670	13.721671	8.500000

Select Data



Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)
```



Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)  
df["Name"]
```



Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)
df.iloc[0:3]
```



Add data to a DataFrame

idx	Name	Age	Grade
0	Clara	20	1
1	Tom	24	2
2	Sarah	19	3
3	John	21	4

```
df["Grade"] = [1,2,3,4]
```



Compute with data in a DataFrame


idx	Name	Age	Grade
0	Clara	20	2
1	Tom	24	3
2	Sarah	19	4
3	John	21	5

```
df["Grade"] = df["Grade"] + 1
```



Drop data

idx	Name	Age	Grade
0	Clara	20	1
1	Tom	24	2
2	Sarah	19	3
3	John	21	4



```
# Drop column  
df = df.drop(column="Grade")
```

Drop data

idx	Name	Age	Grade
2	Sarah	19	3
3	John	21	4

Drop column

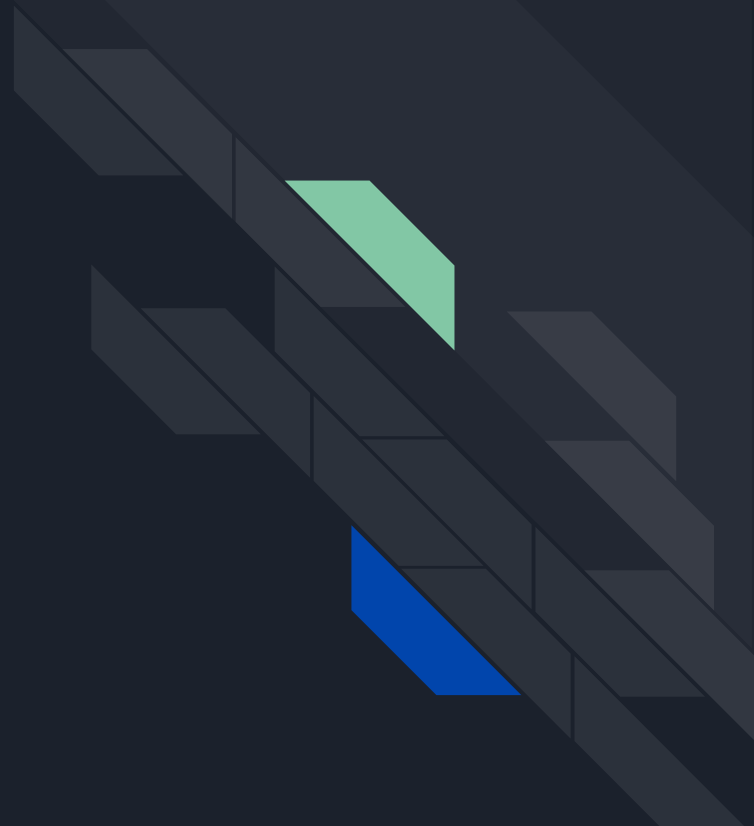
```
df = df.drop(column="Grade")
```

Drop row

```
df.drop([0,1], inplace=True)
```




Filter data



Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	John	21	5




```
keep_rows = [True, False,  
             True, False]
```

```
df[keep_rows]
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
1	Tom	24	3
2	Sarah	19	4
3	John	21	5



```
df["Age"] <= 20  
→ [True, False, True, False]
```


Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	John	21	5

```
df[ df["Age"] <= 20 ]
```

```
[True, False, True, False]
```



Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2

```
df[
    (df["Age"] <= 20) &
    (df["Grade"] < 3)
]
```



Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
df[
    (df["Age"] == 20) |
    (df["Grade"] == 4)
]
```



Combine filtering with other methods

idx	Name	Age	Grade
0	Clara	20	2
2	John	21	5

```
df[df["Age"] <= 20]
```

This is again a DataFrame



Combine filtering with other methods

idx	Name	Age	Grade
0	Clara	20	2
2	John	21	5

```
df[df["Age"]<=20]["Age"].mean()
```



Exercise 2

```
import pandas as pd
```

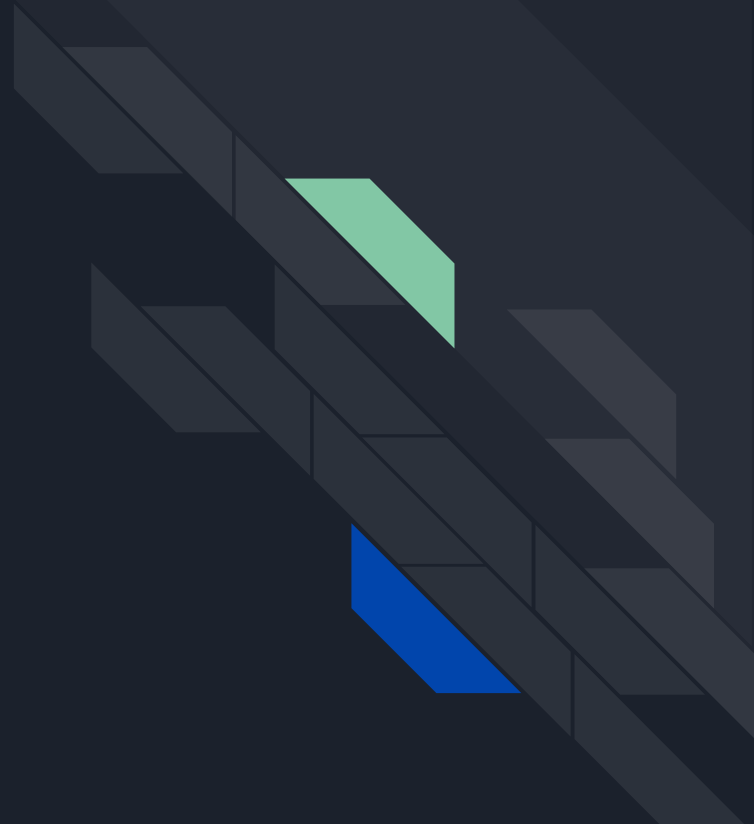
```
df = pd.DataFrame(data) # create dataframe
```

```
df['new_column'] = [1,2,3]
```

```
df.sort_values(by='column_name') # sort
```

```
df[df['age']<=20] # filter data
```

Pandas methods





Apply pandas methods

- Pandas has large amount of commonly used methods
- Can be applied to single column or whole data frame

```
df["Grade"].mean()  
df["Grade"].std()  
df["Grade"].sum()
```





Often used methods

```
df["Grade"].mean()
```

```
df["Grade"].sum()
```

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

```
df.sort_values(by="Grade")
```

```
df.groupby(by="Grade").sum()
```

```
df1.merge(df2)
```

```
df.drop(columns=["Grade"])
```

```
df["Grade"].replace(5, "Failed")
```



Group-By

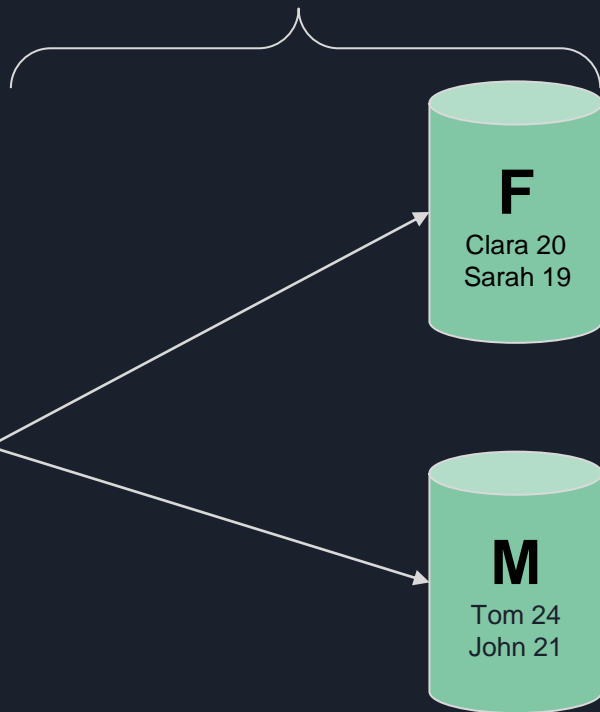
idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M

What is the average age per gender?

Group-By

```
df.groupby(by='Gender').mean()
```

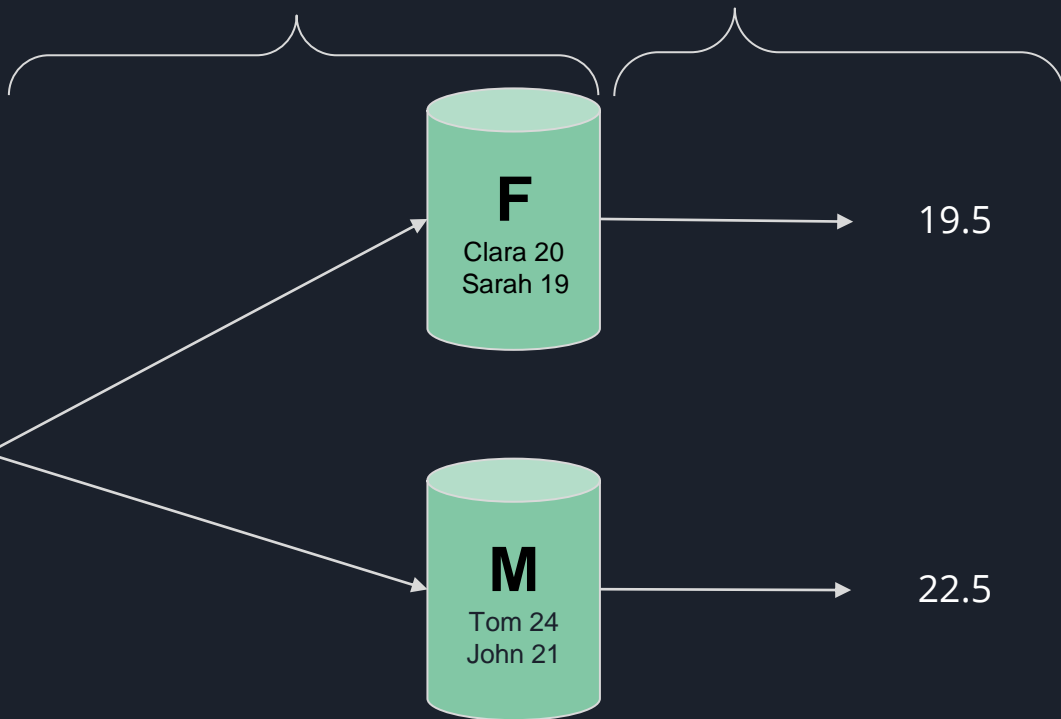
idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M



Group-By

```
df.groupby(by='Gender').mean()
```

idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M





Merge DataFrame's


Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

Name	Subject
Sarah	Physics
Tom	Politics
John	English



Name	Age	Grade	Subject
Tom	24	3	Politics
Sarah	19	4	Physics

Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

```
df1.merge(df2, on='Name')
```

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Name	Age	Grade	Subject
Tom	24	3	Politics
Sarah	19	4	Physics

Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

```
df1.merge(df2, how='left',  
on='Name')
```

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Name	Age	Grade	Subject
Tom	24	3	Politics
Sarah	19	4	Physics
Clara	20	2	-

Exercise 3

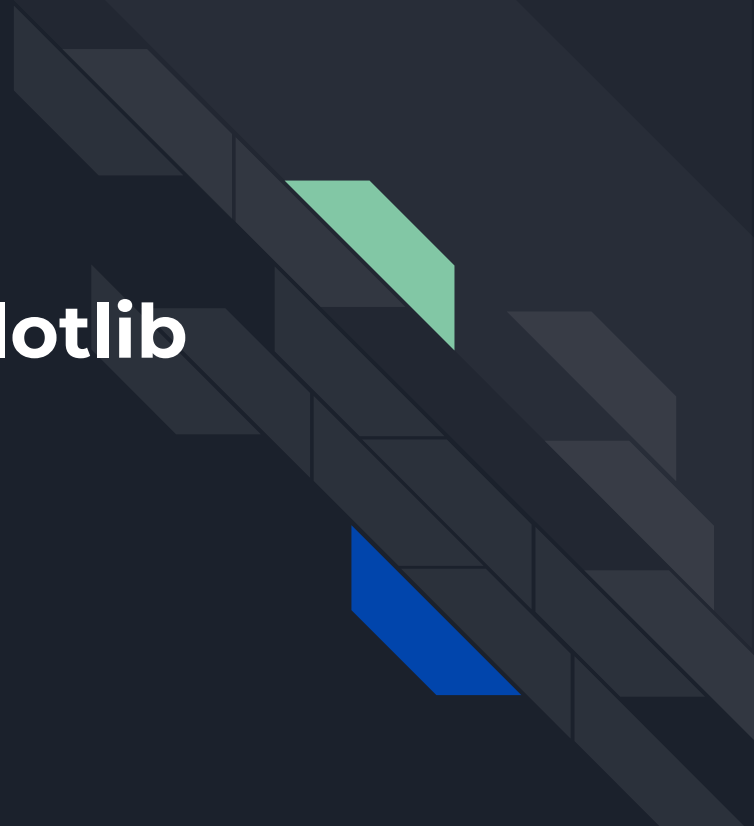
```
import pandas as pd
```

```
df = df.groupby(by='column_name')
```

```
df = df.merge(df2, on='column_name', how='left')
```

Data Wrangling & Visualization

Visualizations with Matplotlib



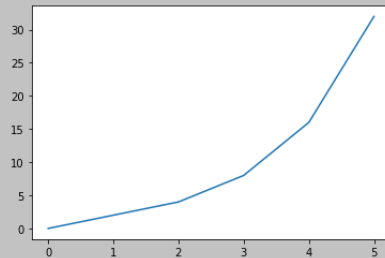
matplotlib - library

- data visualization tool
- generate highly customizable plots
- good integration with pandas



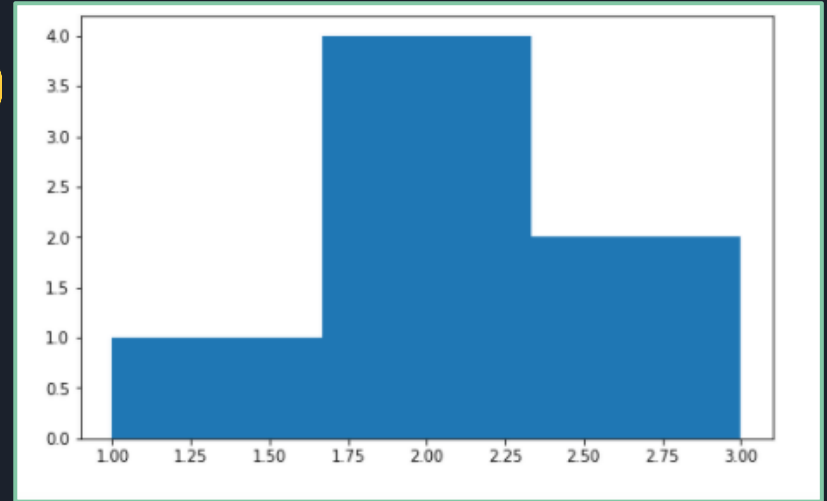
```
import matplotlib.pyplot as plt  
x = [0,1,2,3,4,5]  
y = [0,2,4,8,16,32]  
plt.plot(x,y)
```

Plot:



Histogram

```
import matplotlib.pyplot as plt  
x = [1,2,2,2,2,3,3]  
plt.hist(x, bins=3)
```

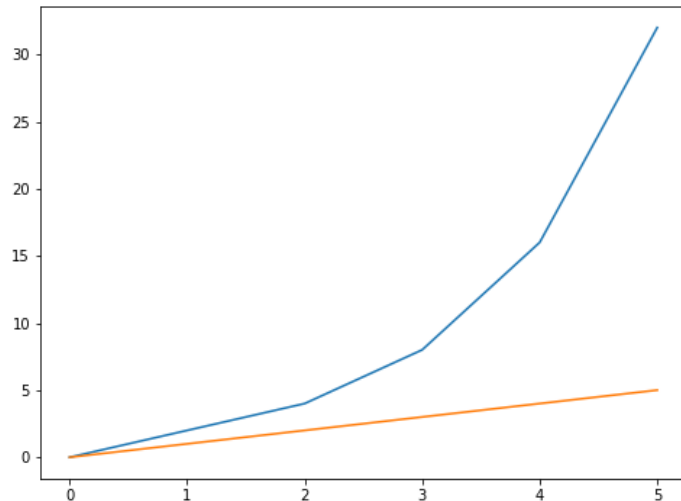


Customize plots



```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y)
plt.plot(x,y2)
```



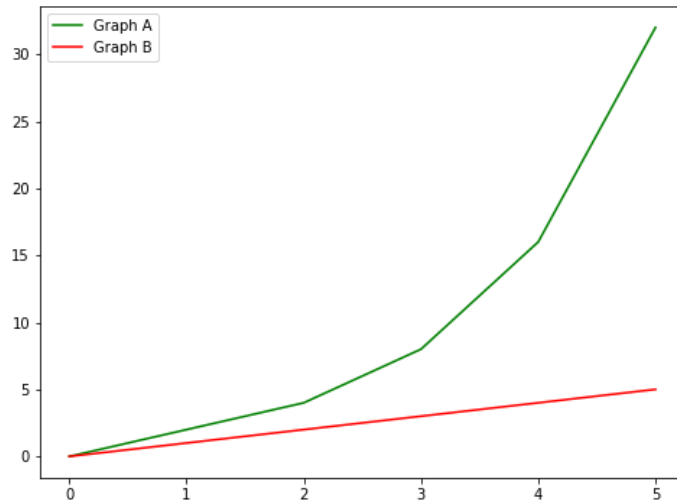
Customize plots



```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')

plt.legend()
```



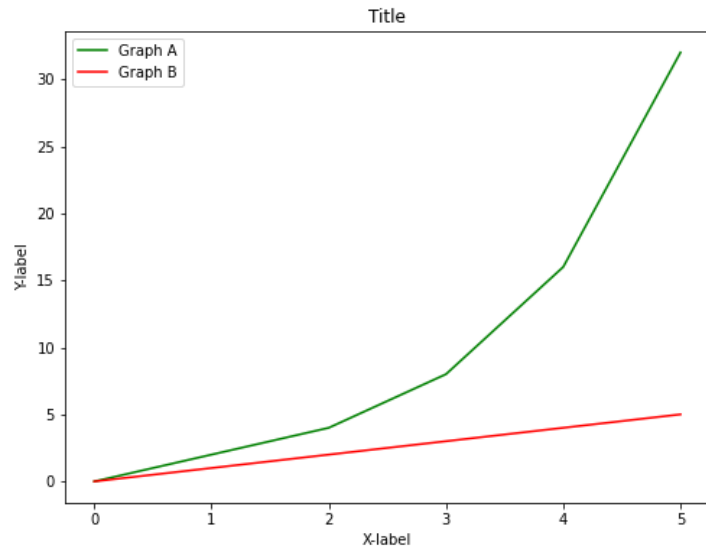
Customize plots



```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')

plt.legend()
plt.xlabel("X-label")
plt.ylabel("Y-label")
plt.title("Title")
```



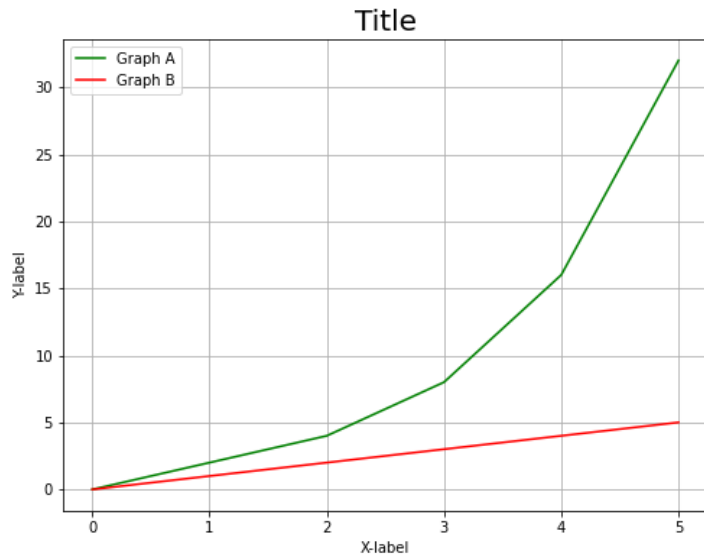
Customize plots



```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')


plt.legend()
plt.xlabel("X-label")
plt.ylabel("Y-label")
plt.title("Title",
{'fontname':'DejaVu Sans', 'size':'20'})
plt.grid()
```





pandas & matplotlib

- Pandas and matplotlib work very well together
- We can pass columns of a DataFrame to matplotlib



```
import matplotlib.pyplot as plt  
df = pd.DataFrame(data)  
plt.hist(df["Age"])
```

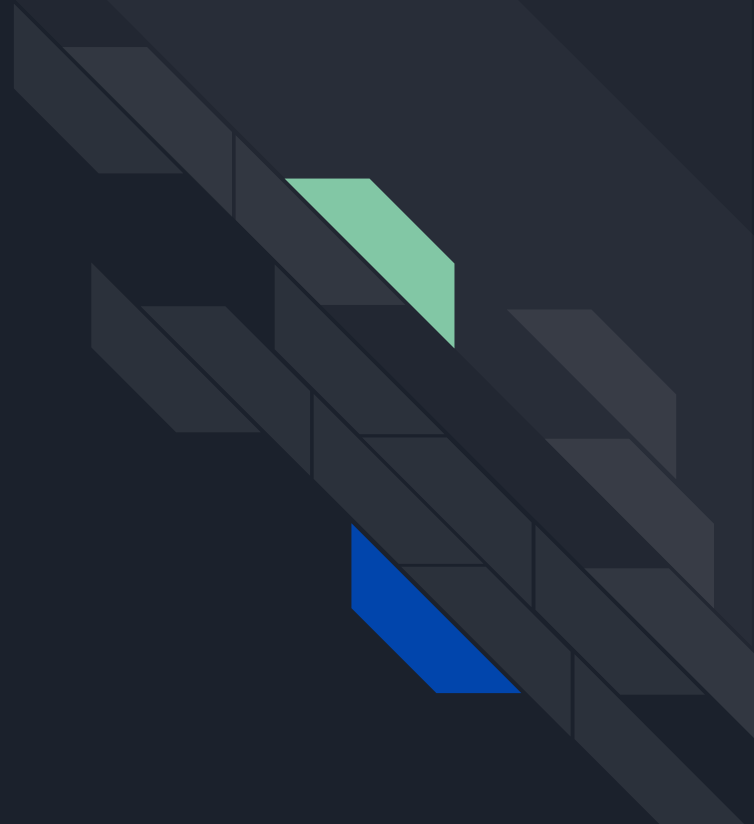
Exercise 4

```
import matplotlib.pyplot as plt  
df = pd.DataFrame(data)  
plt.hist(df["Age"])
```



Data Wrangling & Visualization

Clean Data with Pandas






Clean data

Data comes often in an untidy form, therefore some data cleaning is necessary

Name	Town
Clara	Frankfurt a.M.
Sarah	Frankfurt am Main
John	Berlin



```
df1['Town'].str.replace(  
    'a.M.',  
    'am Main', inplace=True)
```



Clean data

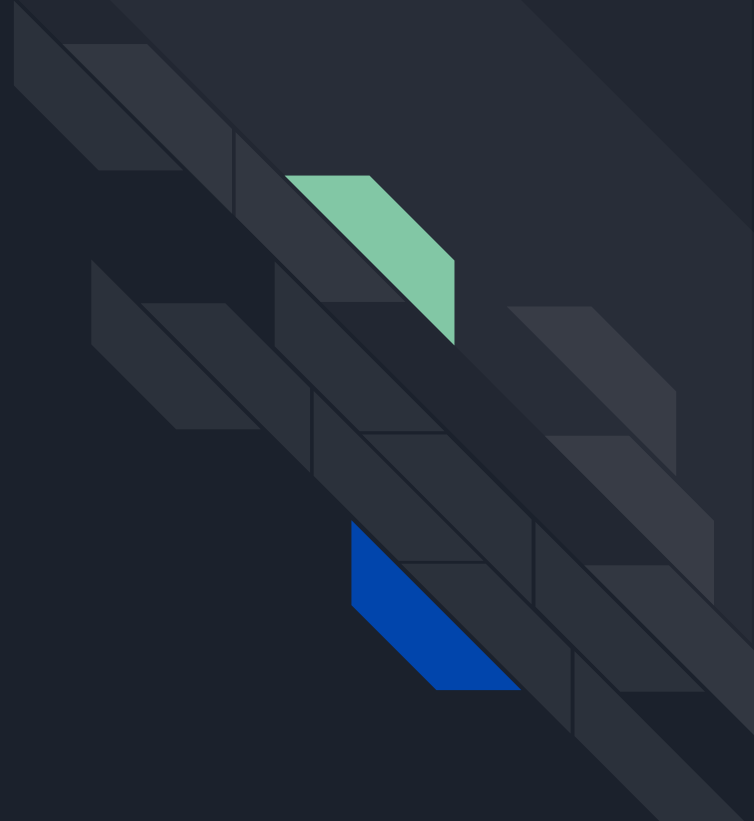
Data comes often in an untidy form, therefore some data cleaning is necessary

Name	Subject
Clara	Physics
Sarah	physics
John	Math

```
df1['Subject'].str.lower(inplace=True)
```



Working with dates





QUIZ

What kind of data type is this : "27-03-2021" ?

- a) integer b) float c) string d) date

Transform string to datetime object



```
df['Birthday'] = pd.to_datetime(df['Birthday'])
```

Name	Birthday
Clara	"20/10/1995"
Sarah	"10/01/1999"
John	"05/03/2001"

Transform string to datetime object



```
df['Birthday'] = pd.to_datetime(df['Birthday'], format="%d/%m/%y")
```

Name	Birthday
Clara	"20/10/1995"
Sarah	"10/01/1999"
John	"05/03/2001"

Transform string to datetime object



```
df['Birthday'] = pd.to_datetime(df['Birthday'], format="%m-%d-%y")
```

Name	Birthday
Clara	"10-10-1995"
Sarah	"10-01-1999"
John	"05-03-2001"

Transform string to datetime object



```
df['day'] = df['Birthday'].dt.day  
df['weekday'] = df['Birthday'].dt.weekday  
df['month'] = df['Birthday'].dt.month
```

Name	Birthday	day	weekday	month
Clara	"10-10-1995"	10	1	10
Sarah	"10-01-1999"	1	4	10
John	"05-03-2001"	3	3	5

Set date as index

date	City	Temperature
"20-04-2021"	Frankfurt	10
"20-04-2021"	Berlin	11
"20-04-2021"	Munich	12
"21-04-2021"	Frankfurt	11
"21-04-2021"	Berlin	13
"21-04-2021"	Munich	14
"22-04-2021"	Frankfurt	9
"21-04-2021"	Berlin	10

```
df = df.set_index(['date'])
```

