UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

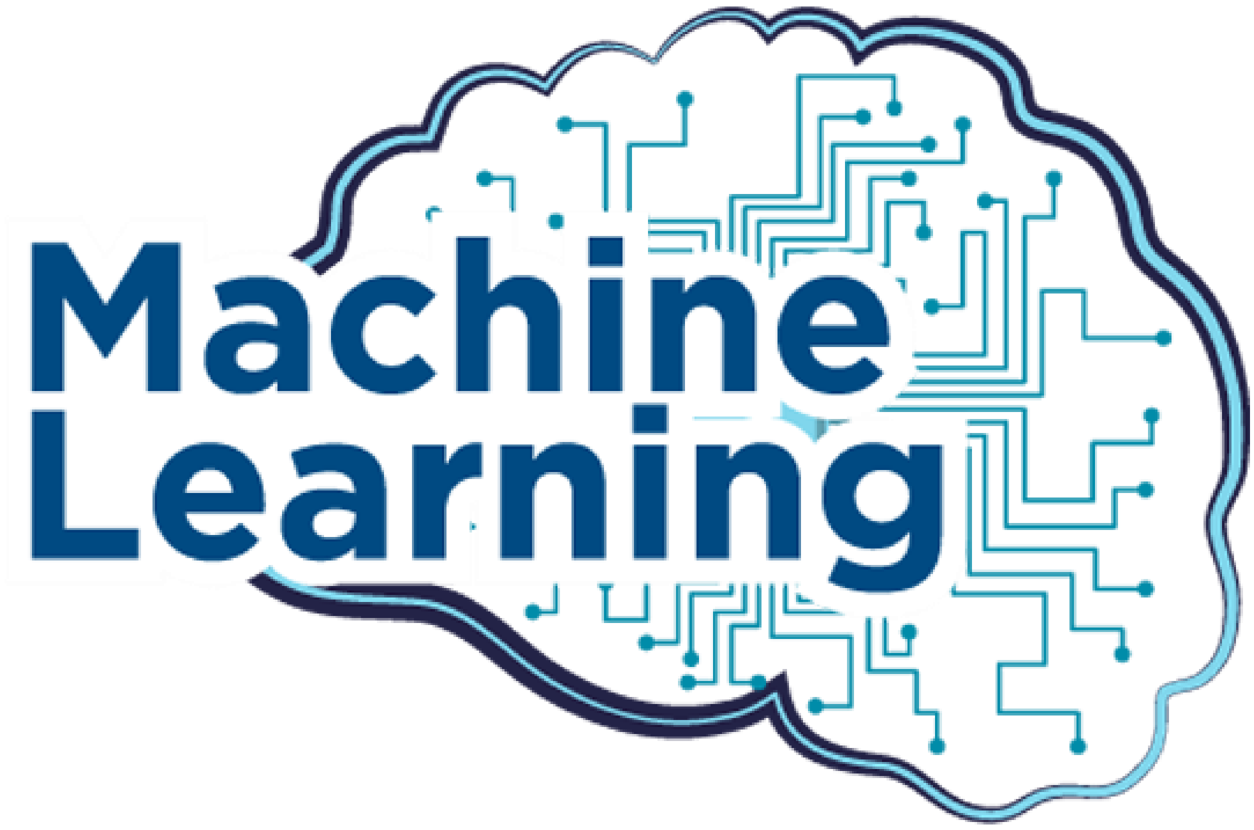
FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Machine Learning

Introduction to Pythons Libraries

(NUMPY,PANDAS)



Dated:

9th Jan, 2024 to 16th Jan, 2024

Semester:

2024

Lab Instructor: Sheharyar Khan

Objectives:-

The objectives of this session are: -

1. NumPy is a Python library.
2. NumPy is used for working with arrays.
3. NumPy is short for "Numerical Python".
4. Pandas is a Python library.
5. Pandas is used to analyze data.

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Installation of NumPy

If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

C:\Users\*Your Name*>pip install numpy

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

Now NumPy is imported and ready to use.

Example 1

```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

NumPy as np

NumPy is usually imported under the np alias.

Create an alias with the as keyword while importing:

```
import numpy as np
```

Example 2

Now the NumPy package can be referred to as np instead of numpy.

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

Checking NumPy Version

The version string is stored under __version__ attribute.

```
import numpy as np

print(np.__version__)
```

NumPy Creating Arrays

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

Example 3

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example 4

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))
```

```
print(arr)
```

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example 5

```
import numpy as np

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

print(arr[0])
```

Test Your Skills.

Rewrite above example 5 to access the second index.

Get third and fourth elements from the following array and add them.

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example 6

Access the element on the first row, second column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
```

```
print('2nd element on 1st row: ', arr[0, 1])
```

Access the element on the 2nd row, 5th column:

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example 7

Access the third element of the second array of the first array:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

Negative Indexing

Use negative indexing to access an array from the end.

Example 8

Print the last element from the 2nd dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Array Slicing

Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [*start*:*end*].

We can also define the step, like this: [*start*:*end*:*step*].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example 9

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

Test Your Skill

Slice elements from index 4 to the end of the array:

Slice elements from the beginning to index 4 (not included):

NumPy Data Types

Data Types in Python

By default Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array:

Get the data type of an array object:

Example 10

```
import numpy as np

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

print(arr.dtype)
```

Example 11

Get the data type of an array containing strings:

```
import numpy as np

arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)
```

NumPy Array Copy vs View

The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy *owns* the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

Example 12

Make a copy, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

Example 13

Make a view, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

NumPy Sorting Arrays

Sorting Arrays

Sorting means putting elements in an *ordered sequence*.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

Example 14

```
import numpy as np

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

print(np.sort(arr))
```

Example 15

Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

NumPy Array Iterating

Iterating Arrays

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.

Example 16

Iterate on the elements of the following 1-D array:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in arr:
  print(x)
```

Random Numbers in NumPy

What is a Random Number?

Random number does NOT mean a different number every time. Random means something that can not be predicted logically.

Pseudo Random and True Random.

Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.

If there is a program to generate random number it can be predicted, thus it is not truly random.

Random numbers generated through a generation algorithm are called *pseudo random*.

Can we make truly random numbers?

Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc.

We do not need truly random numbers, unless it is related to security (e.g. encryption keys) or the basis of application is the randomness (e.g. Digital roulette wheels).

Generate Random Number

NumPy offers the random module to work with random numbers.

Example 17

Generate a random integer from 0 to 100:

```
from numpy import random

x = random.randint(100)

print(x)
```

What is Data Distribution?

Data Distribution is a list of all possible values, and how often each value occurs.

Such lists are important when working with statistics and data science.

The random module offer methods that returns randomly generated data distributions.

Random Distribution

A random distribution is a set of random numbers that follow a certain *probability density function*.

We can generate random numbers based on defined probabilities using the choice() method of the random module.

The choice() method allows us to specify the probability for each value.

The probability is set by a number between 0 and 1, where 0 means that the value will never occur and 1 means that the value will always occur.

Example 18

Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

The probability for the value to be 3 is set to be 0.1

The probability for the value to be 5 is set to be 0.3

The probability for the value to be 7 is set to be 0.6

The probability for the value to be 9 is set to be 0

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

Seaborn

Visualize Distributions With Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Install Seaborn.

If you have Python and PIP already installed on a system, install it using this command:

C:\Users\*Your Name*>pip install seaborn

If you use Jupyter, install Seaborn using this command:

C:\Users\Your Name>!pip install seaborn

Distplots

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

Import Matplotlib

Import the pyplot object of the Matplotlib module in your code using the following statement:

```
import matplotlib.pyplot as plt
```

Import Seaborn

Import the Seaborn module in your code using the following statement:

```
import seaborn as sns
```

Example 19

Plotting a Distplot

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Example 20

Plotting a Distplot Without the Histogram

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

Introduction to Pandas

What is Pandas?

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.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

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

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Example 21

Now Pandas is imported and ready to use

```
import pandas

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

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

Example 22

```
import pandas as pd

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

```
myvar = pd.DataFrame(mydataset)

print(myvar)
```

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

Example 23

In our examples we will be using a CSV file called 'data.csv'.

Load the CSV into a DataFrame:

```
import pandas as pd

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

print(df.to_string())
```

use to_string() to print the entire DataFrame.

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

Example 24

Print the DataFrame without the to_string() method:

```
import pandas as pd

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

print(df)
```

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.

Get a quick overview by printing the first 10 rows of the DataFrame:

Example 25

```
import pandas as pd

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

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

Example 26

Print the first 5 rows of the DataFrame:

```
import pandas as pd

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

print(df.head())
```

There is also a tail() method for viewing the *last* rows of the DataFrame.

The tail() method returns the headers and a specified number of rows, starting from the bottom.

Example

Print the last 5 rows of the DataFrame:

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

Info About the Data

The DataFrames object has a method called info(), that gives you more information about the data set.

Example

Print information about the data:

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

Null Values

The info() method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

Which means that there are 5 rows with no value at all, in the "Calories" column, for whatever reason.

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values.

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

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
```

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

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

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

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

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

Example 27

```
import pandas as pd

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

new_df = df.dropna()

print(new_df.to_string())
```

If you want to change the original DataFrame, use the inplace = True argument:

Remove all rows with NULL values:

Example 28

```
import pandas as pd

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

df.dropna(inplace = True)

print(df.to_string())
```

Replace Empty Values

Another way of dealing with empty cells is to insert a *new* value instead.

This way you do not have to delete entire rows just because of some empty cells.

The fillna() method allows us to replace empty cells with a value:

Example 29

Replace NULL values with the number 130:

```
import pandas as pd

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

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

Replace Only For Specified Columns

The example above replaces all empty cells in the whole Data Frame.

To only replace empty values for one column, specify the *column name* for the DataFrame:

Example 30

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

```
import pandas as pd

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

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

Replace Using Mean, Median, or Mode

A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column:

Example 31

Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

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

```
x = df["Calories"].mean()

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

Example 32

Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

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

x = df["Calories"].median()

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

Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Convert Into a Correct Format

In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

```
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      20201226   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
```

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a to_datetime() method for this:

Example 33

Convert to date:

```
import pandas as pd

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

df['Date'] = pd.to_datetime(df['Date'])

print(df.to_string())
```

Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

```
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      20201226    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
```

How can we fix wrong values, like the one for "Duration" in row 7?

Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

Set "Duration" = 45 in row 7:

```
df.loc[7, 'Duration'] = 45
```

For small data sets you might be able to replace the wrong data one by one, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example 34

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

Example 36

Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Duplicate Rows

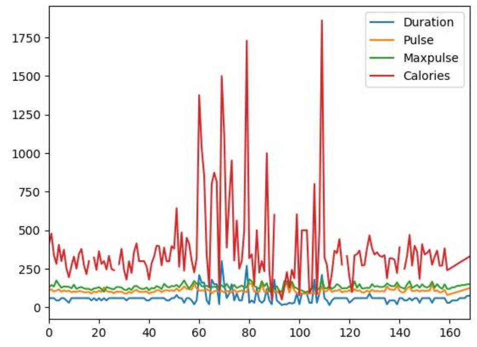
Example 37

Returns True for every row that is a duplicate, otherwise False:

print(df.duplicated())

df.drop_duplicates(inplace = True)

Pandas - Plotting



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.

Example 38
Import pyplot from Matplotlib and visualize our DataFrame:

```
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'

Example 39

```
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()
```

Introduction to Machine Learning

Machine Learning is making the computer learn from studying data and statistics.

Machine Learning is a step into the direction of artificial intelligence (AI).

Machine Learning is a program that analyses data and learns to predict the outcome.

Data Set

In the mind of a computer, a data set is any collection of data. It can be anything from an array to a complete database.

Example of an array:

[99,86,87,88,111,86,103,87,94,78,77,85,86]

Example of a database:

Carname	Color	Age	Speed	AutoPass
BMW	Red	5	99	Y
Volvo	black	7	86	Y
VW	gray	8	87	N
VW	white	7	88	Y
Ford	white	2	111	Y
VW	white	17	86	Y
Tesla	Red	2	103	Y
BMW	black	9	87	Y
Volvo	gray	4	94	N
Ford	white	11	78	N
Toyota	gray	12	77	N
VW	white	9	85	N
Toyota	blue	6	86	Y

By looking at the array, we can guess that the average value is probably around 80 or 90, and we are also able to determine the highest value and the lowest value, but what else can we do?

And by looking at the database we can see that the most popular color is white, and the oldest car is 17 years, but what if we could predict if a car had an AutoPass, just by looking at the other values?

That is what Machine Learning is for! Analyzing data and predicting the outcome!(See you in Next Lab Manual 02)

TASKS

1. **Run all Example 40 in the Lab Manual 01 and save the code and Output?**
2. **Perform Below Tasks (Pandas)**
   a. Create a Pandas DataFrame from a dictionary.
   b. Create a Pandas Series from a list.
   c. Access a specific column in a DataFrame.
   d. Add a new column to an existing DataFrame.
3. **Perform below Tasks**
   a. Handle missing values in a DataFrame.
   b. Remove duplicate rows from a DataFrame.
   c. Rename columns in a DataFrame.
4. **Do as Directed (Numpy)**
   a. Create a 1D and 2D NumPy array.
   b. Reshape an array.
   c. Slice and index arrays.
   d. Perform element-wise operations (addition, subtraction, multiplication, etc.) on arrays.
   e. Use NumPy functions like `np.sum()`, `np.mean()`, and `np.max()` on arrays.
   f. Perform matrix multiplication.
   g. Calculate mean, median, and standard deviation of an array.
   h. Calculate correlation and covariance between two arrays.

THE END