20AM3609-DATA SCIENCE

Module-1 Syllabus

Introduction to Data Science & Programming Tools for Data Science

Concept of Data Science, Traits of Big data, Analysis vs Reporting, Toolkits using Python:, NumPy, Pandas, Scikit-learn, Matplotlib, Visualizing Data: Bar Charts, Line Charts, Scatterplot. Working with data: Reading Files, Scraping the Web, Using APIs (Example: Using the Twitter APIs), Cleaning and Munging, Manipulating Data, Rescaling, Dimensionality Reduction, Principal Component Analysis, Feature extraction

Introduction to Data Science & Programming Tools for Data Science

CONCEPT OF DATA SCIENCE

- **Data Science** is the field that helps in extracting meaningful insights from data using programming skills, domain knowledge, and mathematical and statistical knowledge. It helps to analyze the raw data and find the hidden patterns.

(OR)

  - **Data Science** is the area of study that involves extracting insights from vast amounts of data by using various scientific methods, algorithms, and processes.
  - It allows you to extract knowledge from structured or unstructured data.
- **Data scientist** is someone who extracts insights from messy data.
- Statistics, Visualization, Deep Learning, Machine Learning are important Data Science concepts.

Why Data Science?

Future of Data Science 2030 is estimated to bring opportunities in various areas of banking, finance, insurance, entertainment, telecommunication, automobile, etc. **A data scientist will help grow an organization by assisting them in making better decisions**.

- Empowering Management and Officers to Make Better Decisions
  - to generate insights an organization can use to learn more about its customers and audience.
- Directing Actions Based on Trends—which in Turn Help to Define Goals
- Recruiting the Right Talent for the Organization

TRAITS OF BIG DATA

Today we are living in the data world. Large amount of data are generated in the data world through the data generators.

Data generators- How we are getting huge amount of data?

1. Sensors
2. CCTV cameras
3. Social Media
4. Online Shopping
5. Airlines
6. Hospital data

In 1990, HD capacity 1GB to 20 GB

RAM-64-28MB

Reading capacity- 10kbps

In 2014, HD capacity-min 1TB

RAM-4-16GB

Reading capacity-100Mbps

Example:-

Farmer-rice packets story (10,20,50 and 200).

If it exceeds the limit then a separate godown will be created and maintained.

Similarly hard disk varies from 1TB,2TB…..100TB.

If the Hard Disk do not have enough capacity. We need Data Centers.

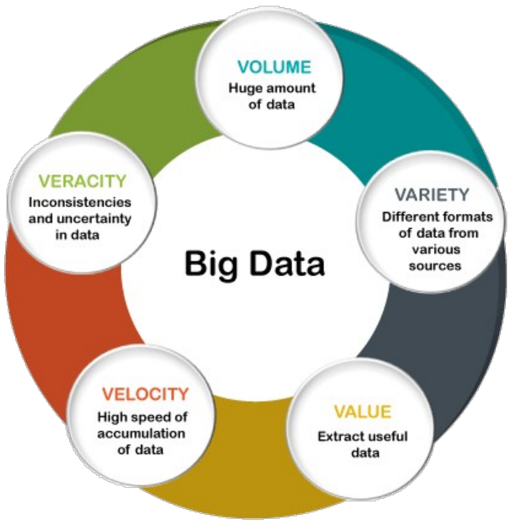
Example: IBM, EMC server.

When we need data we have to fetch it from server then process quit.

Managing data is difficult because huge amount of data can be generated and that data should not be discarded. After identifying all the potential sources of data then a technique need to store and manage the data because processing speed is less when getting huge amount of data.

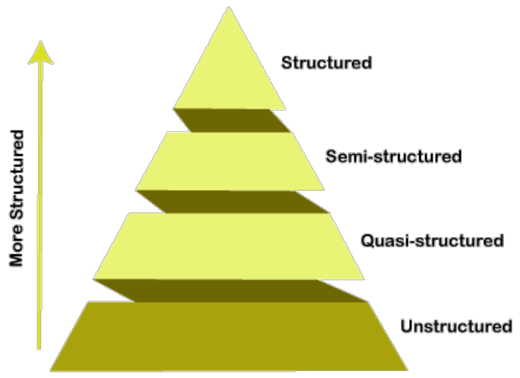
Big Data

The data which is beyond the storage capacity and which is beyond the processing power is called as big data. It is a collection of data from many different sources.


VOLUME
Huge amount of data
VARIETY
Different formats of data from various sources
VALUE
Extract useful data
VELOCITY
High speed of accumulation of data
VERACITY
Inconsistencies and uncertainty in data
Big Data

It has 5 characteristics (5V's of big data)

- Volume-the size and amounts of big data
- Velocity- the speed at which the data is getting generated.
- Variety- Big Data can be **structured, unstructured, and semi-structured** that are being collected from different sources.



a. **Structured data:** In Structured schema, along with all the required columns. It is in a tabular form. Structured Data is stored in the relational database management system.

b. **Semi-structured:** In Semi-structured, the schema is not appropriately defined, e.g., **JSON, XML, CSV, TSV**, and **email**. OLTP (**Online Transaction Processing**) systems are built to work with semi-structured data. It is stored in relations, i.e., **tables**.

c. **Unstructured Data**: All the **unstructured files, log files, audio files**, and **image** files are included in the unstructured data. Some organizations have much data available, but they did not know how to **derive** the value of data since the data is raw.

d. **Quasi-structured Data:**The data format contains textual data with inconsistent data formats that are formatted with effort and time with some tools.

Example: Web server logs, i.e., the log file is created and maintained by some server that contains a list of **activities**.

- Value- It is **valuable** and **reliable** data that we **store, process**, and also **analyze**.
- Veracity-

  Veracity means how much the data is reliable. It has many ways to filter or translate the data. Veracity is the process of being able to handle and manage data efficiently.

  For example, **Facebook posts** with hashtags.

ANALYSIS VS REPORTING

- Analysis transforms data and information into insights.
- Data reporting is the process of collecting and formatting raw data and translating it into a digestible format to assess the ongoing performance of your organization. It provides invaluable insights into trends and helps create strategies to help improve operations, customer satisfaction, growth, and other business metrics.

Analytics includes

- Questioning the data
- Understanding the data
- Investigating the data

Reporting includes

- Gathering the required information
- Organizing
- Summarizing
- Presenting

TOOLKITS USING PYTHON

NumPy, Pandas, Scikit-learn, Matplotlib

NumPy

NumPy is a Python library. NumPy is used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python.

Installation of NumPy

Install it using this command:

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

Import NumPy

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

```
import numpy
```

Example

```
import numpy

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

print(arr)
```

Output:-

```
[1 2 3 4 5]
```

NumPy as np

NumPy is usually imported under the np alias.

```
import numpy as np
```

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

Example:-

```
import numpy as np

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

print(arr)
```

Output:

```
[1 2 3 4 5]
```

Checking NumPy Version

The version string is stored under __version__ attribute.

```
import numpy as np

print(np.__version__)
```

Output:

```
1.21.6
```

NumPy Creating Arrays

Create a NumPy ndarray Object

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

Example:-

```
import numpy as np

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

print(arr)

print(type(arr))
```

Output:-

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Dimensions in Arrays

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

```
import numpy as np
```

arr = np.array(42)

print(arr)

Output:

```
42
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

import numpy as np

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

print(arr)

Output:

```
[1 2 3 4 5]
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

import numpy as np

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

print(arr)

Output:-

```
 [[1 2 3]
 [4 5 6]]
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

import numpy as np

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

print(arr)

Output:-

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Output:-

```
0
1
2
3
```

NumPy Array Indexing

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:
import numpy as np

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

print(arr[0])

Output:
1

Example

```
import numpy as np

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

print(arr[1])
```

Using the NumPy functions

NumPy has a variety of built-in functions to create an array.

a. Creating one-dimensional array in NumPy

For 1-D arrays the most common function is **np.arange(..)**, passing any value create an array from 0 to that number.

```
import numpy as np

array=np.arange(20)

array
```

Output

```
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,12, 13, 14, 15, 16, 17, 18, 19])
```

We can check the dimensions by using **array.shape.**
Output

```
(20,)
```

To access the element of the array we can specify its non-negative index.
array[3]
Output

```
3
```

b. Creating two-dimensional arrays in NumPy

We can use reshape()function along with the arange() function to create a 2D array. The reshape()function specifies the rows and columns.

```
array=np.arange(20).reshape(4,5)
```

Output

```
array([[ 0, 1, 2, 3, 4],
[ 5, 6, 7, 8, 9],
[10, 11, 12, 13, 14]
[15, 16, 17, 18, 19]])
```

In 2-D arrays to access the elements, we will require to specify two values for row and column respectively.

Similarly, we can create 3-D and more by increasing the number of parameters in the reshape()function.

c. Using other NumPy functions

We can use other functions like and to quickly create filled arrays.

```
np.zeros((2,4))

np.ones((3,6))
```

Output

```
array([[0., 0., 0., 0.],
[0., 0., 0., 0.]])

array([[1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1.],
[1., 1., 1., 1., 1., 1.]])
```

The .empty() function creates an array with random variables and the full() function creates an n*n array with the given value.

```
np.empty((2,3))

np.full((2,2), 3)
```

Output

```
array([[2.69893675e-316, 0.00000000e+000, 0.00000000e+000],
[0.00000000e+000, 0.00000000e+000, 0.00000000e+000]])
array([[3, 3],
[3, 3]])
```

The .eye(,)function creates an array with diagonals as 1 and other
values as 0. The .linspace(, ,)
function outputs an equally spaced array.

```
np.eye(3,3)

np.linspace(0, 10, num=4)
```

Output

```
array([[1., 0., 0.],
[0., 1., 0.],
[0., 0., 1.]])
array([ 0. , 3.33333333, 6.66666667, 10. ])
```

2**. Conversion from Python structure like lists**

We can use the Python lists to create arrays by passing a list to the array function. We can also directly create an array of elements by passing a list.

```
array=np.array([4,5,6])

array

list=[4,5,6]

list
```

Output

```
array([4, 5, 6])[4, 5, 6]
```

3. Using other library functions

We can use the function “random” to create an array with random values between 0 and 1. This is a useful case in scenarios with random nature.

```
np.random.random((2,3))
```

Output

```
array([[0.42593659, 0.91495384, 0.05727104],
[0.3754818 , 0.63166016, 0.5901392 ]])
```

NumPy Array Indexing

Indexing of the array has to be proper in order to access and manipulate its values. Indexing can be done through:

- **Slicing –** we perform slicing on NumPy arrays with the declaration of a slice for all the dimensions.
- **Integer array Indexing–** users can pass lists for one to one mapping of corresponding elements for each dimension.
- **Boolean Array Indexing–** we can pick elements after satisfying a particular Boolean condition.

NumPy Basic Array Operations

There is a vast range of built-in operations that we can perform on these arrays.

1. ndim – It returns the dimensions of the array.
2. itemsize – It calculates the byte size of each element.
3. dtype – It can determine the data type of the element.
4. reshape – It provides a new view.
5. slicing – It extracts a particular set of elements.
6. linspace – Returns evenly spaced elements.

7. max/min , sum, sqrt
8. ravel – It converts the array into a single line.
There are also a few Special Operations like sine, cosine, tan, log, etc.

Checking Array Dimensions in NumPy

We can determine the NumPy array dimensions using the ndim attribute. The argument return an integer that indicates the array dimension.

```
import numpy as np

a = np.array(10)

b = np.array([1,1,1,1])

c = np.array([[1, 1, 1], [2,2,2]])

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

print(a.ndim)

print(b.ndim)

print(c.ndim)

print(d.ndim)
```

Output

```
0 1 2 3
```

Higher Dimensional Arrays in NumPy

In Numpy we can have arrays with any number of dimensions. There can be arrays with a high number of dimensions. We define the number of dimensions with the ndim argument.

```
import numpy as np

arr = np.array([1, 1, 1, 1, 1], ndmin=10)

print(arr)

print('number of dimensions :', arr.ndim)
```

Output

```
[[[[[[[[[[1 1 1 1 1]]]]]]]]]] number of dimensions : 10
```

In the above example, the innermost dimension (10th dim) has 5 elements, the 9th dim has 1 element that is the vector, the 8th dim has 1 element that is the matrix with the vector, the 7th dim has 1 element that is 3D array 6th dim has 1 element that is a 4D array and so on.

Indexing and Slicing in NumPy

These are two very important concepts. It is useful when we want to work with sub-arrays.
Indexing starts with zero as the first index. We can retrieve element values by its index value. For 2 or more dimensional arrays, we have to specify 2 or more indices. Indexing can be of two types

1**. Integer array indexing**

We pass lists for indexing in all the dimensions. Then one to one mapping occurs for the creation of a new array.

2. Boolean array indexing

In this type of indexing, we carry out a condition check. If the boolean condition satisfies we create an array of those elements.

```
import numpy as np

arr=([1,2,5,6,7])

arr[3]
```

Output

```
6
```

Slicing is similar to indexing, but it retrieves a string of values. The range is defined by the starting and ending indices. It is similar to lists in Python. The arrays can also be sliced. The arrays can be single or multidimensional. We can specify slices for all the dimensions.

```
import numpy as np

arr=([1,2,5,6,7])

arr[2:5]
```

Output

```
[5, 6, 7]
```

Refer
https://towardsdatascience.com/27-things-that-a-beginner-needs-to-know-about-numpy-edda217fb662

Example Programs

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.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])
```

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

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.

```
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])
```

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

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.

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

NumPy Array Slicing

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*].

```
import numpy as np

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

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

Output:

```
[2 3 4 5]
```

```
import numpy as np

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

print(arr[4:])
```

Output:

```
[5 6 7]
```

```
import numpy as np

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

print(arr[:4])
```

```
[1 2 3 4]
```

NumPy Array Copy vs 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.

COPY:

```
import numpy as np
arr=np.array([1, 2, 3, 4, 5])
x=arr.copy()
arr[0]= 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

VIEW:

```
import numpy as np
arr=np.array([1, 2, 3, 4, 5])
x=arr.view()
arr[0]= 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

Make Changes in the VIEW:

```
import numpy as np
arr=np.array([1, 2, 3, 4, 5])
x=arr.view()
x[0]= 31
print(arr)
print(x)
```

```
[31  2  3  4  5]
[31  2  3  4  5]
```

NumPy Array Shape

The shape of an array is the number of elements in each dimension. NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

```
[[[[[1234]]]]]
shape of array : (1, 1, 1, 1, 4)
```

NumPy Array Reshaping

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D

```
import numpy as np

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

newarr=arr.reshape(4, 3)

print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Reshape From 1-D to 3-D

```
import numpy as np

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

newarr=arr.reshape(2, 3, 2)

print(newarr)
```

```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
```

```
  [ 9 10]
  [11 12]]]
```

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.

```
import numpy as np
arr=np.array([1, 2, 3])
for x in arr:
  print(x)
```

Output:-

```
1
2
3
```

Iterating 2-D Arrays

```
import numpy as np

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

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

Output:

```
[123]
[4 5 6]
```

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array.

```
import numpy as np

arr1=np.array([1, 2, 3])

arr2=np.array([4, 5, 6])

arr=np.concatenate((arr1,arr2))
```

```
print(arr)
```

Output:-

```
[1 2 3 4 5 6]
```

```
import numpy as np

arr1=np.array([[1, 2],[3, 4]])

arr2= np.array([[5, 6],[7, 8]])

arr=np.concatenate((arr1,arr2),axis=1)

print(arr)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

Splitting NumPy Arrays

Splitting is reverse operation of Joining. Joining merges multiple arrays into one and Splitting breaks one array into multiple. We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

```
import numpy as np
arr=np.array([1, 2, 3, 4, 5, 6])
newarr= np.array_split(arr, 3)
print(newarr)
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

NumPy Searching Arrays

You can search an array for a certain value, and return the indexes that get a match. To search an array, use the where() method.

```
import numpy as np

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

```
x= np.where(arr== 4)

print(x)
```

```
(array([3, 5, 6]),)
```

NumPy 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:

```
import numpy as np

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

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

```
[0 1 2 3]
```

```
import numpy as np

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

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

```
['apple' 'banana' 'cherry']
```

Sort a Boolean Array

```
import numpy as np

arr = np.array([True, False, True])

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

```
[False True True]
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

```
import numpy as np
```

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

print(np.sort(arr))

```
[[2 3 4]
 [0 1 5]]
```

Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called filtering. In NumPy, you filter an array using a boolean index list.

import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)

output:

```
[41 43]
```

PANDAS

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

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

Install it using this command:

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

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

```
import pandas
```

Example:

```
import pandas as pd

mydataset = {  'cars': ["BMW", "Volvo", "Ford"],

                     'passings': [3, 7, 2]

}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Output:

```
   cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

Pandas as pd

Pandas is usually imported under the pd alias.

alias: In Python alias are an alternate name for referring to the same thing.

```
import pandas as pd
```

Checking Pandas Version

The version string is stored under __version__ attribute.

import pandas as pd

print(pd.__version__)

Output:-

```
1.0.3
```

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
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar[0])

```
1
```

Create Labels

With the index argument, you can name your own labels.

```
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
```

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar["y"])
```

Output:-

```
7
```

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

```
import pandas as pd

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

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Output:-

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

Locate Row

Pandas use the loc attribute to return one or more specified row(s)

```
import pandas as pd

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

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df.loc[0])
```

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

Return row 0 and 1:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
```

df = pd.DataFrame(data)

print(df.loc[[0, 1]])

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

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.

Data.csv

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

60,102,127,300.5

60,110,136,374.0

45,104,134,253.3

30,109,133,195.1

60,98,124,269.0

60,103,147,329.3

60,100,120,250.7

60,106,128,345.3

60,104,132,379.3

60,98,123,275.0

60,98,120,215.2

60,100,120,300.0

45,90,112,

60,103,123,323.0

```
45,97,125,243.0
60,108,131,364.2
45,100,119,282.0
60,130,101,300.0
45,105,132,246.0
60,102,126,334.5
60,100,120,250.0
60,92,118,241.0
60,103,132
60,100,132,280.0
60,102,129,380.3
60,92,115,243.0
45,90,112,180.1
60,101,124,299.0
60,93,113,223.0
60,107,136,361.0
60,114,140,415.0
60,102,127,300.5
60,100,120,300.1
60,100,120,300.0
45,104,129,266.0
45,90,112,180.1
60,98,126,286.0
60,100,122,329.4
60,111,138,400.0
60,111,131,397.0
60,99,119,273.0
60,109,153,387.6
45,111,136,300.0
45,108,129,298.0
60,111,139,397.6
```

```
60,107,136,380.2

80,123,146,643.1

60,106,130,263.0

60,118,151,486.0

30,136,175,238.0

60,121,146,450.7

60,118,121,413.0

45,115,144,305.0

20,153,172,226.4

45,123,152,321.0

210,108,160,1376.0

160,110,137,1034.4

160,109,135,853.0

45,118,141,341.0

20,110,130,131.4

180,90,130,800.4

150,105,135,873.4

150,107,130,816.0

20,106,136,110.4

300,108,143,1500.2

150,97,129,1115.0

60,109,153,387.6

90,100,127,700.0

150,97,127,953.2

45,114,146,304.0

90,98,125,563.2

45,105,134,251.0

45,110,141,300.0

120,100,130,500.4

270,100,131,1729.0

30,159,182,319.2
```

```
45,149,169,344.0
30,103,139,151.1
120,100,130,500.0
45,100,120,225.3
30,151,170,300.1
45,102,136,234.0
120,100,157,1000.1
45,129,103,242.0
20,83,107,50.3
180,101,127,600.1
45,107,137,
30,90,107,105.3
15,80,100,50.5
20,150,171,127.4
20,151,168,229.4
30,95,128,128.2
25,152,168,244.2
30,109,131,188.2
90,93,124,604.1
20,95,112,77.7
90,90,110,500.0
90,90,100,500.0
90,90,100,500.4
30,92,108,92.7
30,93,128,124.0
180,90,120,800.3
30,90,120,86.2
90,90,120,500.3
210,137,184,1860.4
60,102,124,325.2
45,107,124,275.0
```

```
15,124,139,124.2
45,100,120,225.3
60,108,131,367.6
60,108,151,351.7
60,116,141,443.0
60,97,122,277.4
60,105,125,
60,103,124,332.7
30,112,137,193.9
45,100,120,100.7
60,119,169,336.7
60,107,127,344.9
60,111,151,368.5
60,98,122,271.0
60,97,124,275.3
60,109,127,382.0
90,99,125,466.4
60,114,151,384.0
60,104,134,342.5
60,107,138,357.5
60,103,133,335.0
60,106,132,327.5
60,103,136,339.0
20,136,156,189.0
45,117,143,317.7
45,115,137,318.0
45,113,138,308.0
20,141,162,222.4
60,108,135,390.0
60,97,127,
45,100,120,250.4
```

```
45,122,149,335.4
60,136,170,470.2
45,106,126,270.8
60,107,136,400.0
60,112,146,361.9
30,103,127,185.0
60,110,150,409.4
60,106,134,343.0
60,109,129,353.2
60,109,138,374.0
30,150,167,275.8
60,105,128,328.0
60,111,151,368.5
60,97,131,270.4
60,100,120,270.4
60,114,150,382.8
30,80,120,240.9
30,85,120,250.4
45,90,130,260.4
45,95,130,270.0
45,100,140,280.9
60,105,140,290.8
60,110,145,300.4
60,115,145,310.2
75,120,150,320.4
75,125,150,330.4
```

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
```

```
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
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
17         45     90       112       NaN
18         60    103       123     323.0
19         45     97       125     243.0
20         60    108       131     364.2
21         45    100       119     282.0
22         60    130       101     300.0
23         45    105       132     246.0
24         60    102       126     334.5
25         60    100       120     250.0
26         60     92       118     241.0
27         60    103       132       NaN
28         60    100       132     280.0
29         60    102       129     380.3
30         60     92       115     243.0
31         45     90       112     180.1
32         60    101       124     299.0
33         60     93       113     223.0
34         60    107       136     361.0
35         60    114       140     415.0
36         60    102       127     300.5
37         60    100       120     300.1
38         60    100       120     300.0
39         45    104       129     266.0
40         45     90       112     180.1
41         60     98       126     286.0
42         60    100       122     329.4
43         60    111       138     400.0
44         60    111       131     397.0
45         60     99       119     273.0
46         60    109       153     387.6
47         45    111       136     300.0
48         45    108       129     298.0
49         60    111       139     397.6
50         60    107       136     380.2
51         80    123       146     643.1
```

52	60	106	130	263.0
53	60	118	151	486.0
54	30	136	175	238.0
55	60	121	146	450.7
56	60	118	121	413.0
57	45	115	144	305.0
58	20	153	172	226.4
59	45	123	152	321.0
60	210	108	160	1376.0
61	160	110	137	1034.4
62	160	109	135	853.0
63	45	118	141	341.0
64	20	110	130	131.4
65	180	90	130	800.4
66	150	105	135	873.4
67	150	107	130	816.0
68	20	106	136	110.4
69	300	108	143	1500.2
70	150	97	129	1115.0
71	60	109	153	387.6
72	90	100	127	700.0
73	150	97	127	953.2
74	45	114	146	304.0
75	90	98	125	563.2
76	45	105	134	251.0
77	45	110	141	300.0
78	120	100	130	500.4
79	270	100	131	1729.0
80	30	159	182	319.2
81	45	149	169	344.0
82	30	103	139	151.1
83	120	100	130	500.0
84	45	100	120	225.3
85	30	151	170	300.1
86	45	102	136	234.0
87	120	100	157	1000.1
88	45	129	103	242.0
89	20	83	107	50.3
90	180	101	127	600.1
91	45	107	137	NaN
92	30	90	107	105.3
93	15	80	100	50.5
94	20	150	171	127.4
95	20	151	168	229.4
96	30	95	128	128.2
97	25	152	168	244.2
98	30	109	131	188.2
99	90	93	124	604.1
100	20	95	112	77.7
101	90	90	110	500.0
102	90	90	100	500.0
103	90	90	100	500.4
104	30	92	108	92.7

105	30	93	128	124.0
106	180	90	120	800.3
107	30	90	120	86.2
108	90	90	120	500.3
109	210	137	184	1860.4
110	60	102	124	325.2
111	45	107	124	275.0
112	15	124	139	124.2
113	45	100	120	225.3
114	60	108	131	367.6
115	60	108	151	351.7
116	60	116	141	443.0
117	60	97	122	277.4
118	60	105	125	NaN
119	60	103	124	332.7
120	30	112	137	193.9
121	45	100	120	100.7
122	60	119	169	336.7
123	60	107	127	344.9
124	60	111	151	368.5
125	60	98	122	271.0
126	60	97	124	275.3
127	60	109	127	382.0
128	90	99	125	466.4
129	60	114	151	384.0
130	60	104	134	342.5
131	60	107	138	357.5
132	60	103	133	335.0
133	60	106	132	327.5
134	60	103	136	339.0
135	20	136	156	189.0
136	45	117	143	317.7
137	45	115	137	318.0
138	45	113	138	308.0
139	20	141	162	222.4
140	60	108	135	390.0
141	60	97	127	NaN
142	45	100	120	250.4
143	45	122	149	335.4
144	60	136	170	470.2
145	45	106	126	270.8
146	60	107	136	400.0
147	60	112	146	361.9
148	30	103	127	185.0
149	60	110	150	409.4
150	60	106	134	343.0
151	60	109	129	353.2
152	60	109	138	374.0
153	30	150	167	275.8
154	60	105	128	328.0
155	60	111	151	368.5
156	60	97	131	270.4
157	60	100	120	270.4

```
158         60    114        150      382.8
159         30     80        120      240.9
160         30     85        120      250.4
161         45     90        130      260.4
162         45     95        130      270.0
163         45    100        140      280.9
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
```

What is difference between NumPy and pandas?

Pandas is mostly used for data analysis tasks in Python. NumPy is mostly used for working with Numerical values as it makes it easy to apply mathematical functions.

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

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.

```
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:

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

```
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

```
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:

```
import pandas as pd
df = pd.read_csv('data.csv')
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)
```

```
import pandas as pd

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

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

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

```
import pandas as pd

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

x = df["Calories"].mode()[0]

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.

```
import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

Removing Rows

Remove the row by using the dropna() method.

```
df.dropna(subset=['Date'], inplace = True)
```

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".

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:

```
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

Loop through all values in the "Duration" column.

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

```
import pandas as pd
df = pd.read_csv('data.csv')
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
print(df.to_string())
```

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

```
import pandas as pd

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

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

Discovering Duplicates
Duplicate rows are rows that have been registered more than one time.
To discover duplicates, we can use the duplicated() method. The duplicated() method returns a Boolean values for each row.

```
import pandas as pd
```

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

print(df.duplicated())
```

Removing Duplicates
To remove duplicates, use the drop_duplicates() method.

```
import pandas as pd
df = pd.read_csv('data.csv')
df.drop_duplicates(inplace = True)
print(df.to_string())
```

Pandas - Data Correlations
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())
```

we have a dataset of columns temp,humidity,vaporpressure and rainfall

temp,hum,vp-input
rainfall-output

1
Scikit-learn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Installation

If you already installed NumPy and Scipy, following are the two easiest ways to install scikit-learn

using pip

Following command can be used to install scikit-learn via pip

```
pip install -U scikit-learn
```

Scikit Learn - Modelling Process

Dataset Loading
A collection of data is called dataset. It is having the following two components
Features − The variables of data are called its features. They are also known as predictors, inputs or attributes.

- **Feature matrix** − It is the collection of features, in case there are more than one.

- **Feature Names** − It is the list of all the names of the features.

Response − It is the output variable that basically depends upon the feature variables. They are also known as target, label or output.

- **Response Vector** − It is used to represent response column. Generally, we have just one response column.
- **Target Names** − It represent the possible values taken by a response vector.

Splitting the dataset

To check the accuracy of our model, we can split the dataset into two pieces-**a training set** and **a testing set**. Use the training set to train the model and testing set to test the model. After that, we can evaluate how well our model did.

Train the Model

Next, we can use our dataset to train some prediction-model. As discussed, scikit-learn has wide range of **Machine Learning (ML) algorithms** which have a consistent interface for fitting, predicting accuracy, recall etc.

Preprocessing the Data

As we are dealing with lots of data and that data is in raw form, before inputting that data to machine learning algorithms, we need to convert it into meaningful data. This process is called preprocessing the data. Scikit-learn has package named preprocessing for this purpose. The preprocessing package has the following techniques

- **Binarisation**

  This preprocessing technique is used when we need to convert our numerical values into Boolean values.

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
   [2.1, -1.9, 5.5],
   [-1.5, 2.4, 3.5],
   [0.5, -7.9, 5.6],
   [5.9, 2.3, -5.8]]
)
data_binarized = preprocessing.Binarizer(threshold=0.5).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

In the above example, we used threshold value = 0.5 and that is why, all the values above 0.5 would be converted to 1, and all the values below 0.5 would be converted to 0.

```
Binarized data:
[
   [ 1. 0. 1.]
   [ 0. 1. 1.]
   [ 0. 0. 1.]
```

```
  [ 1. 1. 0.]
]
```

Mean Removal
This technique is used to eliminate the mean from feature vector so that every feature centered on zero

```python
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
  [2.1, -1.9, 5.5],
  [-1.5, 2.4, 3.5],
  [0.5, -7.9, 5.6],
  [5.9, 2.3, -5.8]]
)

#displaying the mean and the standard deviation of the input data
print("Mean =", input_data.mean(axis=0))
print("Stddeviation = ", input_data.std(axis=0))
#Removing the mean and the standard deviation of the input data

data_scaled = preprocessing.scale(input_data)
print("Mean_removed =", data_scaled.mean(axis=0))
print("Stddeviation_removed =", data_scaled.std(axis=0))
```

```
Mean = [ 1.75 -1.275 2.2 ]
Stddeviation = [ 2.71431391 4.20022321 4.69414529]
Mean_removed = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]
Stddeviation_removed = [ 1. 1. 1.]
```

Scaling
We use this preprocessing technique for scaling the feature vectors. Scaling of feature vectors is important, because the features should not be synthetically large or small.

```python
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
  [
    [2.1, -1.9, 5.5],
    [-1.5, 2.4, 3.5],
    [0.5, -7.9, 5.6],
    [5.9, 2.3, -5.8]
  ]
)
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
```

```
print ("\nMin max scaled data:\n", data_scaled_minmax)
```

```
Min max scaled data:
[
  [ 0.48648649 0.58252427 0.99122807]
  [ 0. 1. 0.81578947]
  [ 0.27027027 0. 1. ]
  [ 1. 0.99029126 0. ]
]
```

Normalisation

We use this preprocessing technique for modifying the feature vectors. Normalisation of feature vectors is necessary so that the feature vectors can be measured at common scale. There are two types of normalisation as follows -

L1 Normalisation

It is also called Least Absolute Deviations. It modifies the value in such a manner that the sum of the absolute values remains always up to 1 in each row. Following example shows the implementation of L1 normalisation on input data.

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
  [
    [2.1, -1.9, 5.5],
    [-1.5, 2.4, 3.5],
    [0.5, -7.9, 5.6],
    [5.9, 2.3, -5.8]
  ]
)
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
print("\nL1 normalized data:\n", data_normalized_l1)
```

```
L1 normalized data:
[
  [ 0.22105263 -0.2 0.57894737]
  [-0.2027027 0.32432432 0.47297297]
  [ 0.03571429 -0.56428571 0.4 ]
  [ 0.42142857 0.16428571 -0.41428571]
]
```

L2 Normalisation

Also called Least Squares. It modifies the value in such a manner that the sum of the squares remains always up to 1 in each row. Following example shows the implementation of L2 normalisation on input data.

```
import numpy as np
from sklearn import preprocessing
Input_data = np.array(
  [
    [2.1, -1.9, 5.5],
    [-1.5, 2.4, 3.5],
    [0.5, -7.9, 5.6],
    [5.9, 2.3, -5.8]
  ]
)
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nL1 normalized data:\n", data_normalized_l2)
```

```
L2 normalized data:
[
  [ 0.33946114 -0.30713151 0.88906489]
  [-0.33325106 0.53320169 0.7775858 ]
  [ 0.05156558 -0.81473612 0.57753446]
  [ 0.68706914 0.26784051 -0.6754239 ]
]
```

Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.

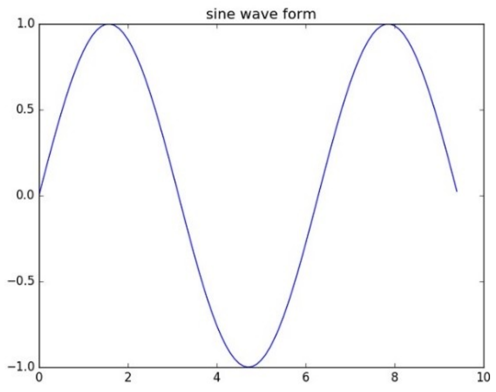
Conventionally, the package is imported into the Python script by adding the following statement −

from matplotlib import pyplot as plt

The following script produces the **sine wave plot** using matplotlib.

Example

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



VISUALIZING DATA

There are two uses for data visualization

- To explore data
- To communicate data

Matplotlib library is used to visualize the data.

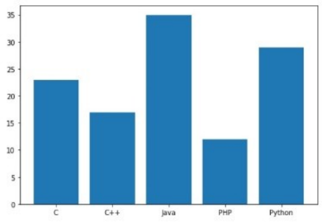
Python –m pip install matplotlib

- **Bar Charts**

  It shows how some quantity varies among some discrete set of items. It can be a good choice for plotting histograms of bucketed numeric values.

  A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```



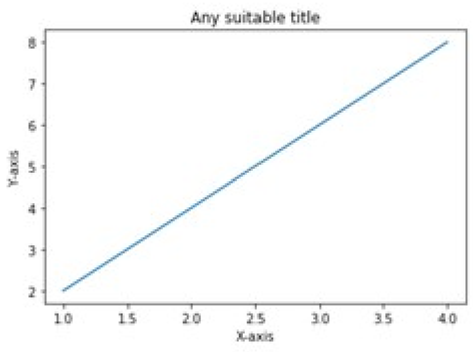
- **Line Charts**

  plt.plot can be used to make line charts. It shows the trends. A line chart, also referred to as a line graph or a line plot, connects a series of data points using a line. This chart type presents sequential values to help you identify trends. Most of the time, the x-axis (horizontal axis) represents a sequential progression of values.

Or

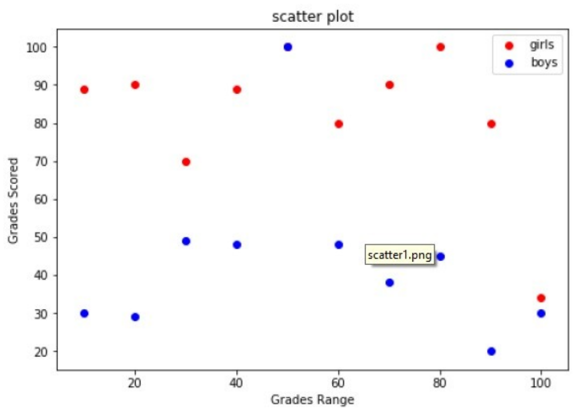
The line chart is a simple two dimensional chart with an X and Y axis, each point representing a single value.

```
import matplotlib.pyplot as plt
import numpy as np


# Define X and Y variable data
x = np.array([1, 2, 3, 4])
y = x*2

plt.plot(x, y)
plt.xlabel("X-axis")  # add X-axis label
plt.ylabel("Y-axis")  # add Y-axis label
plt.title("Any suitable title")  # add title
plt.show()
```



- **Scatterplot**
    1. Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another.
    2. Each row in the data table is represented by a marker the position depends on its values in the columns set on the X and Y axes.
    3. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot.

```
import matplotlib.pyplot as plt
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(grades_range, girls_grades, color='r')
ax.scatter(grades_range, boys_grades, color='b')
ax.set_xlabel('Grades Range')
ax.set_ylabel('Grades Scored')
ax.set_title('scatter plot')
plt.show()
```



Working with data

Web scraping is about extracting data from the Web. Specifically, a web scraper is a tool that performs web scraping and is generally represented by a script.

Python is one of the easiest and most reliable scripting language for web scraping It comes with a wide variety of web scraping libraries.

(OR)

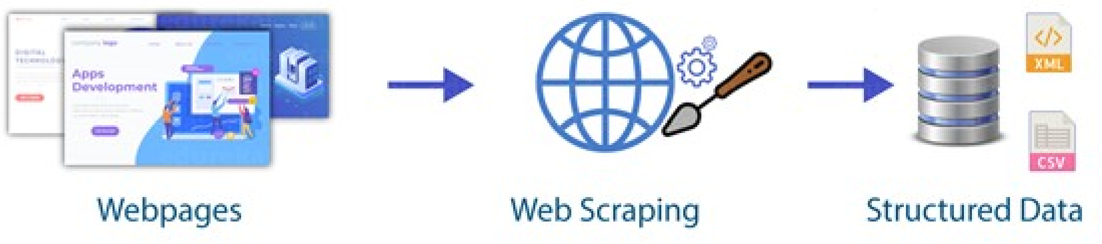
Web scraping (or data scraping) is a technique used to collect content and data from the internet. This data is usually saved in a local file so that it can be manipulated and analyzed as needed.

Web scraping has countless applications, especially within the field of data analytics. Market research companies use scrapers to pull data from social media or online forums for things like customer sentiment analysis. Others scrape data from product sites like Amazon or eBay to support competitor analysis.

- Step 1: Making an HTTP request to a server
- Step 2: Extracting and parsing (or breaking down) the website’s code
- Step 3: Saving the relevant data locally

Web scraping

Web scraping is an automated method used to extract large amounts of data from websites. The data on the websites are unstructured. Web scraping helps collect these unstructured data and store it in a structured form. There are different ways to scrape websites such as online Services, APIs or writing your own code.



Is Web Scraping Legal?

- some websites allow web scraping and some don't.
- To know whether a website allows web scraping or not, you can look at the website's "robots.txt" file.
- You can find this file by appending "/robots.txt" to the URL that you want to scrape.

How to scrape the web (step-by-step)

1. Find the URLs you want to scrape

2. Inspect the page
3. Identify the data you want to extract
4. Write the necessary code

5. Execute the code
6. Storing the data

Real time Applications of Web Scraping

Web Scraping has multiple applications across various industries.

1. Price Monitoring

Web Scraping can be used by companies to scrap the product data for their products and competing products as well to see how it impacts their pricing strategies. Companies can use this data to fix the optimal pricing for their products so that they can obtain maximum revenue.

2. Market Research

Web scraping can be used for market research by companies. High-quality web scraped data obtained in large volumes can be very helpful for companies in analyzing consumer trends and understanding which direction the company should move in the future.

3. News Monitoring

Web scraping news sites can provide detailed reports on the current news to a company. This is even more essential for companies that are frequently in the news or that depend on daily news for their day-to-day functioning. After all, news reports can make or break a company in a single day!

4. Sentiment Analysis

If companies want to understand the general sentiment for their products among their consumers, then Sentiment Analysis is a must. Companies can use web scraping to collect data from social media websites such as Facebook and Twitter as to what the general sentiment about their products is. This will help them in creating products that people desire and moving ahead of their competition.

5. Email Marketing

Companies can also use Web scraping for email marketing. They can collect Email ID's from various sites using web scraping and then send bulk promotional and marketing Emails to all the people owning these Email ID's.

Modules needed

- bs4: Beautiful Soup(bs4) is a Python library for pulling data out of HTML and XML files. This module does not come built-in with Python. To install this type the below command in the terminal.

  pip install bs4

- requests: Request allows you to send HTTP/1.1 requests extremely easily. This module also does not come built-in with Python. To install this type the below command in the terminal.

  pip install requests

Beautiful Soup is a Python library for parsing structured data. It allows you to interact with HTML in a similar way to how you interact with a web page using developer tools. The library exposes a couple of intuitive functions you can use to explore the HTML you received.

Example:-

```python
import requests
from bs4 import BeautifulSoup

URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)
```

```
soup = BeautifulSoup(page.content, "html.parser")
```

->page.content, which is the HTML content you scraped as its input.

->"html.parser", makes sure that you use the appropriate parser for HTML content.

Find Elements by ID

In an HTML web page, every element can have an id attribute. As the name already suggests, that id attribute makes the element uniquely identifiable on the page. You can begin to parse your page by selecting a specific element by its ID.

```
<div id="ResultsContainer">
  <!-- all the job listings -->
</div>
```

Beautiful Soup allows you to find that specific HTML element by its ID:

```
results = soup.find(id="ResultsContainer")
```

For easier viewing, you can prettify any Beautiful Soup object when you print it out. If you call .prettify() on the results variable that you just assigned above, then you'll see all the HTML contained within the <div>:

```
print(results.prettify())
```

Find Elements by HTML Class Name

```
job_elements = results.find_all("div", class_="card-content")
```

find_all() on a Beautiful Soup object, which returns an iterable containing all the HTML for all the job listings displayed on that page.

```
for job_element in job_elements:
    print(job_element, end="\n"*2)
```

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element)
    print(company_element)
    print(location_element)
    print()
```

Each job_element is another BeautifulSoup() object. Therefore, you can use the same methods on it as you did on its parent element, results.

```
<h2 class="title is-5">Senior Python Developer</h2>
<h3 class="subtitle is-6 company">Payne, Roberts and Davis</h3>
<p class="location">Stewartbury, AA</p>
```

Extract Text From HTML Elements

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text)
    print(company_element.text)
    print(location_element.text)
    print()
```

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    print()
```

O/p

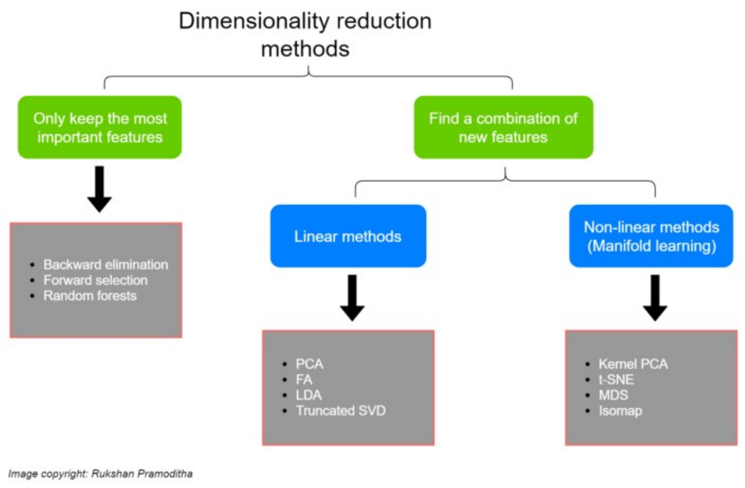
```
Senior Python Developer
Payne, Roberts and Davis
Stewartbury, AA

Energy engineer
Vasquez-Davidson
Christopherville, AA
```

```
Legal executive
Jackson, Chambers and Levy
Port Ericaburgh, AA
```

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of features in a dataset while retaining as much information as possible.

This can be done to reduce the complexity of a model, improve the performance of a learning algorithm, or make it easier to visualize the data.


Dimensionality reduction methods
Only keep the most important features
Find a combination of new features
Linear methods
Non-linear methods (Manifold learning)
Backward elimination
Forward selection
Random forests
PCA
FA
LDA
Truncated SVD
Kernel PCA
t-SNE
MDS
Isomap
Image copyright: Rukshan Pramoditha

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called **Principal Component Analysis**

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Steps for PCA algorithm

Getting the dataset

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

Representing data into a structure

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

Standardizing the data

In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.

If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

Calculating the Covariance of Z

To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

Calculating the Eigen Values and Eigen Vectors

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

Sorting the Eigen Vectors

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

Calculating the new features Or Principal Components

Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

Remove less or unimportant features from the new dataset.

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.