Marwadi University

01CE1705-Programming with Python

Unit-8 Data Analytics and Visualization

Prof. Chetan Chudasama
Computer Engineering Department

Introduction of 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.

Which Language is NumPy written in?

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Numpy vs List:

- Numpy data structure better in:

  size:-Numpy data structure takes less space

  performance:-it is faster than list

  functionality:-numpy have optimized functions such as linear algebra operation built in.

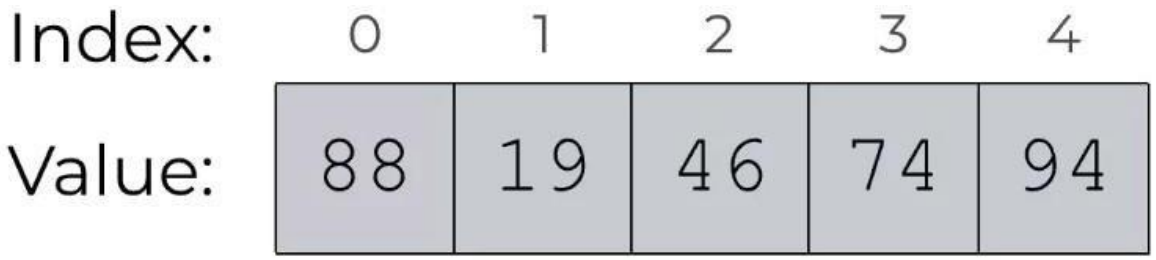
Installation of numpy

Marwadi University

- If you have Python and PIP already installed on a system, then installation of NumPy is very easy. Install it using this command: pip install numpy
- If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.
- Once NumPy is installed, import it in your applications by adding the import keyword. import numpy
- NumPy is usually imported under the np alias. Alias are an alternate name.
- import numpy as np
- Now the numpy package can be referred to as np instead of numpy.

Numpy Array

- A NumPy array is a collection of elements that have the same data type.
- NumPy arrays must contain data all of the same type. That means that if your NumPy array contains integers, all of the values must be integers. If it contains floating point numbers, all of the values must be floats.
- Each of the element inside numpy array have an address. We call that address an index.



- Many data structures in Python have indexes, and the indexes of a NumPy array essentially work the same.
- Just like other Python structures that have indexes, the indexes of a NumPy array begin at zero

How to create numpy array:-

- There are a lot of ways to create a NumPy array.

•We can create array using built-in array() function.

Example:

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

- The array object in NumPy is called ndarray.
- To create an ndarray, we can pass a list or tuple into array() method and it will be converted into an ndarray.

Dimensions in Arrays:-

1-Dimensional numpy array:-

- We can use array() function, we will need to provide list of elements as the argument to the function.

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

2-Dimensional numpy array:-

- We can create 2-dimensional array using np.array() function, we need to pass in a list of lists.

```
import numpy as np
np.array([[1,2,3],[4,5,6]]
```

- Inside of the call to np.array(), there is a list of two lists: [[1,2,3],[4,5,6]]. The first list is [1,2,3] and the second list is [4,5,6]. Those two lists are contained inside of a larger list; a list of lists.

- Then that list of lists is passed to the array function, which creates a 2-dimensional NumPy array.

3-Dimensional numpy array:-

•An array that has 2-D arrays as its elements is called 3-D array.

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

Array creation using built-in function

arange():-

•It returns evenly spaced numeric values within an interval.

Syntax:-numpy.arange(start,stop,step,dtype)

Parameters:-

It accepts the following parameters.

start: The starting of an interval. The default is 0.Optional

stop: represents the value at which the interval ends excluding this value. Required

step: The number by which the interval values change. Optional

dtype: the data type of the numpy array items. Optional

Example:-

```
import numpy as np
a=np.arange(5)#return [0,1,2,3,4]
a=np.arange(2,5]#return [2,3,4]
a=np.arange(1,10,2)#return [1,3,5,7,9]
a=np.arange(1,10,2,dtype="float")#return [1.,3.,5.,7.,9.]
```

zeros():-

•We can create an array with all zeros using this function.

Syntax:-numpy.zeros(shape, dtype=float, order='C')

Parameters:-

shape:-int or tuple. This parameter is used to define the dimensions of the array. This parameter is used for the shape in which we want to create an array, such as (3,2) or 2.

dtype: data-type(optional).This parameter is used to define the desired data-type for the array. By default, the data-type is float.

order:-(optional).This parameter is used to define the order in which we want to store data in memory either row-major(C-style) or column-major(Fortran-style)

Example:-

```
np.zeros(4)#[0.,0.,0.,0.]
np.zeros((4,3))#4 rows 3 column all filled with 0's
np.zeros((4,3),dtype="int")
```

ones():-

- The ones() function is used to get a new array of given shape and type, filled with ones.

Syntax:-numpy.ones(shape, dtype=None, order='C')

Parameters:-

shape:-Shape of the new array, e.g., (2, 3) or 2.

dtype:-The desired data-type for the array.Default is float

order:-Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory

Example:-

```
np.ones(4)#[1.,1.,1.,1.]
np.ones((4,3))#4 rows 3 column all filled with 1's
np.ones((4,3),dtype="int")
```

linspace():-

- The linspace() function returns evenly spaced numbers over a specified interval [start, stop].
- The endpoint of the interval can optionally be excluded.

Syntax:-numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

Parameters:-

start:-(Required).The starting value of the sequence.

stop:-(Required).The end value of the sequence.

num:-Number of samples to generate. Default is 50. Must be non-negative.

endpoint:-Default is True. If False stop value is not included.

retstep:-If True, return (samples, step), where step is the spacing between samples.

dtype:-It represents the data type of the array items.

Example:-

np.linspace(1,10)

np.linspace(1,10,num=5)

np.linspace(1,10,num=5,endpoint=False)

np.linspace(1,10,num=5,endpoint=False,retstep=True)

np.linspace(1,10,num=5,endpoint=False,retstep=True,dtype="int")

eye():-

- The eye() function is used to create a 2-D array with ones on the diagonal and zeros elsewhere.

Syntax:-numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

Parameters:-

N:-Number of rows in the output.

M-Number of columns in the output. If None, defaults to N

k:-Index of the diagonal: 0 (the default) refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal.

dtype:-Data-type of the returned array.

order:-Whether the output should be stored in row-major (C-style) or column-major (Fortran-style) order in memory.

Example:-

```
np.eye(4)
np.eye(4,3)
np.eye(4,3,1)
np.eye(4,3,1,"int")
```

Attributes of numpy array

ndim

- It represents the number of dimensions (axes) of the ndarray.

•For example, 1 means that the array is 1D,2 means that the array is 2D and so on.

Example:-

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
print('Number of dimensions in x:', x.ndim)
y = np.array([[11, 12, 13, 14], [32, 33, 34, 35]])
print('Number of dimensions in y:', y.ndim)
z = np.array([[[11, 12, 13, 14], [32, 33, 34, 35]],
              [[55, 56, 57, 58], [59, 60, 61, 62]]])
print('Number of dimensions in z:', z.ndim)
```

Output:-
Number of dimensions in x: 1
Number of dimensions in y: 2
Number of dimensions in z: 3

shape:-

- This attribute returns the dimension of the array.
- It is a tuple of integers that indicates the size of your NumPy array. For example, if you create a matrix with n rows and m columns, the shape will be (n * m)

Example:-

```
a=np.array([[1,2],[3,4]])
a.shape#return (2,2)
b=np.array([1,2,3,4])
b.shape#return (4,)
```

size:-

- This attribute shows the number of elements present in the array.
- It is equal to the product of elements of the shape.
- For 2-dimensional array [[2,3],[4,5]],shape is (2,2),size will be product of 2 and 2.

Example:-

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
print('Number of elements in x:', x.size)
y = np.array([[11, 12, 13, 14], [32, 33, 34, 35]])
print('Number of elements in y:', y.size)
z = np.array([[[11, 12, 13, 14], [32, 33, 34, 35]],
              [[55, 56, 57, 58], [59, 60, 61, 62]]])
print('Number of elements in z:', z.size)
```

Output:-
Number of elements in x: 5
Number of elements in y: 8
Number of elements in z: 16

dtype:-

- This attribute gives the data type of Numpy array.
- In NumPy array, all the elements have the same data type.

Example:-

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
print('The data type of array x is', x.dtype)
y = np.array([[1 + 2j, 14.478], [5 + 11j, 34]])
print('The data type of array y is', y.dtype)
```

```
z = np.array([[[11.147, 12, 13, 14], [32.984, 33, 34, 35]],
              [[55, 56.951, 57, 58.369], [59, 60, 61, 62]]])
print('The data type of array z is', z.dtype)
```

Output:-

The data type of array x is int64

The data type of array y is complex128

The data type of array z is float64

itemsize:-

- itemsize returns the size (in bytes) of each element of a NumPy array.

e.g. for this NumPy array [[3,4,6], [0,8,1]], itemsize will be 4, because this array consists of integers and size of integer is 4 bytes.

T:-This attributes convert rows into columns and columns into row.

Example:

```
import numpy as np
x = np.array([[12, 14, 43], [511, 34, 2], [21, 5, 90]])
print('Array x:\n', x)
print('The transpose of array x is\n', x.T)
```

Output:-

```
Array x:
array([[ 12,  14,  43],
       [511,  34,   2],
       [ 21,   5,  90]])
The transpose of array x is
array([[ 12, 511,  21],
       [ 14,  34,   5],
       [ 43,   2,  90]])
```

Array Manipulation

- Performing arithmetic operation on numpy array is the one way to manipulate the array. Similar to that there are many ways we discuss few of them here.
- We discuss about
    - reshaping arrays
    - array flattening
    - dimension shuffling
    - joining array
    - splitting arrays
    - add/remove elements to array

reshape():-

•The reshape() function is used to give a new shape to an array without changing its data.

Syntax:-numpy.reshape(a, newshape, order='C')

Parameters:-
a:-Array to be reshaped.
newshape:-if we are arranging an array with 10 elements then shaping
it like numpy.reshape(4, 8) is wrong; we can do numpy.reshape(2, 5) or (5, 2)
order:-(Optional) C,F

Example:-

```
import numpy as np
a=np.arange(1,11)
print(np.reshape(a,(5,2)))#order="F" check this argument also
```

resize():-

- The resize() function is used to create a new array with the specified shape.
- If the new array is larger than the original array, then the new array is filled with repeated copies of a.

Syntax:-numpy.resize(a, new_shape)

Parameters:-

a:-Array to be resized

new_shape:-Shape of resized array

- The new array is formed from the data in the old array, repeated if necessary to fill out the required number of elements. The data are repeated in the order that they are stored in memory.

Example:-

```
import numpy as np
a=np.arange(1,11)
print(np.resize(a,(4,3)))
```

flatten():-

- The flatten() function is used to get a copy of an given array collapsed into one dimension.

Syntax:-ndarray.flatten(order='C')

Parameters:-‘C’ means to flatten in row-major (C-style) order. ‘F’ means to flatten in column-major (Fortran- style) order.

Example:-

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
print(a.flatten(order="F"))
```

concatenate():-

- This function combines NumPy arrays together.
- This function is basically used for joining two or more arrays of the same shape along a specified axis.
- This function can operate both vertically and horizontally. This means we can concatenate arrays together horizontally or vertically.

Syntax:-np.concatenate((a1,a2,...),axis)

Parameters:-

a1,a2,....:-Sequence of arrays. Here, a1, a2, a3 ... are the arrays which have the same shape, except in the dimension corresponding to the axis.

axis:-This parameter defines the axis along which the array will be joined. By default, its value is 0.

Example:-

```
import numpy as np
a=np.array([[1,2],[3,4]])
b=np.array([[10,18]])
print(np.concatenate((a,b)))
#print(np.concatenate((a,b),axis=1))
#print(np.concatenate((a,b.T),axis=1))
```

vstack():-

•It is used to stack the sequences of input arrays vertically to make a single array.

Synax:-np.vstack(tup)

Parameters:-

tup:-Tuple containing arrays to be stacked.

Example:-

```
import numpy as np
arr1=np.array([1,2,3,4])
arr2=np.array([5,6,7,8])
print(np.vstack((arr1,arr2)))
```

hstack():-

- It is used to stack the sequences of input arrays horizontally(Column wise) to make single array.

Syntax:-np.hstack(tup)

Parameters:-

tup:-Tuple containing arrays to be stacked.

Example:-

```
import numpy as np
arr1=np.array([1,2,3,4])
arr2=np.array([5,6,7,8])
print(np.hstack((arr1,arr2)))
```

split():-

- This function split an array into more than one (multiple) sub arrays as views. This function divides the array into subarrays

Example:-

```
import numpy as np
a=np.arange(1,10)
print(np.split(a,3))
#print(np.split(a,4)) Error 9 elements is not divide into 4 equal parts
```

insert():-

•It is used to insert values along the given axis before the given indices.

Syntax:-numpy.insert(arr,obj,values,axis=None)

Parameter:-

arr:-Input array

obj:-Object that defines the index before which values is inserted.

values:-Values to insert into arr. If the type of values is different from that of arr, values is converted to the type of arr.

axis:-Axis along which to insert values. If axis is None then arr is flattened first.

- It return a copy of array with values inserted.

```
import numpy as np
arr=np.arange(1,11)
print(arr)
print(np.insert(arr,1,50))
#print(np.insert(arr,1,50.5))
#print(np.insert(arr,(1,3),50))
```

append():-

•The append() function is used to append values to the end of an given array.

Syntax:-numpy.append(arr, values, axis=None)

Parameters:-

arr:-Values are appended to copy of this array.

values:-These values are appended to a copy of arr. It must be of the correct shape (the same shape as arr, excluding axis). If axis is not specified, values can be any shape and will be flattened before use.

axis:-The axis along which values are appended. If axis is not given, both arr and values are flattened before use.

•It return copy of array with values appended to axis.

Example:-

```
import numpy as np
arr=np.arange(1,11)
print(np.append(arr,3452))
```

delete():-

•It is used to delete the elements based on index position.

Syntax:-numpy.delete(arr,obj,axis=None)

Parameters:-

arr:-Input array

obj:-Index position or list of index positions of items to be deleted from the input array

axis:-Axis along which we want to delete. If it is '1' then delete columns, or '0' then delete rows. If the axis is None then return the flattened array.

- It returns a new array with the deletion of sub-arrays along with the specified axis. If the axis is None then return the flattened array.

Example

```
import numpy as np
arr=np.arange(0,10)
print(arr)
print(np.delete(arr,3))
```

Array Indexing

- We 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:-

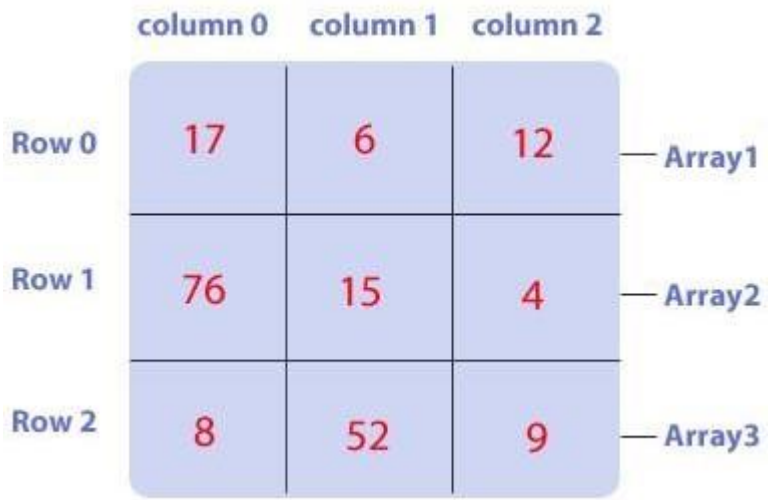
```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Access 2-D array:-

- 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 row represents the dimension and the index represents the column.

Example:-

```
import numpy as np
arr=np.array([[17,6,12],[76,15,4],[8,52,9]])
print(arr[1][2])
print(arr[-3][-1])
```



Array Iterating

- 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:-

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

Iterating 2-D Arrays:-

•In a 2-D array it will go through all the rows.

```
import numpy as np
arr=np.arange(12).reshape(3,4)
```

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

- To return the actual values, the scalars, we have to iterate the arrays in each dimension.

```
import numpy as np
arr=np.arange(12).reshape(3,4)
for row in arr:
    for cell in row:
        print(cell)
```

Iterating Arrays Using nditer()

- NumPy package contains an iterator object numpy.nditer.
- It is an efficient multidimensional iterator object using which it is possible to iterate over an array.
- Each element of an array is visited using Python's standard Iterator interface.

```
import numpy as np
arr=np.arange(12).reshape(3,4)
print(arr)
for number in np.nditer(arr,order='F'):
    print(number)
```

Iteration Using ndenumerate():-

- Enumeration means mentioning sequence number of somethings one by one.
- Sometimes we require corresponding index of the element while iterating, the ndenumerate() method can be used for those usecases.

```
import numpy as np
arr=np.arange(12).reshape(3,4)
print(arr)
for index,number in np.ndenumerate(arr):
    print(index,number)
```

Introduction to pandas

- Pandas is an open source python library that makes data science easy and effective.

What is Data science:-Data science or data analytics is a process of analyzing large set of data points to get answers on questions related to that data set.

- It provides high performance data manipulation in python.
- The name of pandas derived from the word Panel Data, which means an econometrics from multidimensional data.
- It was developed by Wes McKinney in 2008.
- There are different tools for fast data processing such as Numpy,Scipy,Cpython and pandas. But we prefer pandas because working with pandas is fast and simple as compare to other tools.
- Pandas is built on top of the Numpy package, means Numpy is required for operating the Pandas.

- Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis.

Features:-

- It is high performance data analysis tool.
- It will be used for working with large data sets.
- It supports OR load files with different formats.
- It is more flexible to use.
- Represents data in tabular form(rows and columns)
- It is suitable for working on missing data.
- It is used for indexing,slicing,subsetting large datasets.
- We can merge or join two different data sets easily.
- We can reshape datasets.

Pandas Series

- The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types.
- We can easily convert the list, tuple, and dictionary into series using "series' method.
- The axis labels are collectively called index.
- Pandas Series is nothing but a column in an excel sheet. A Series cannot contain multiple columns.
- Pandas series will be created by CSV file,Excel file,SQL Database,lists,dictionary and an array.

Creating a series from list: In order to create a series from list, we have to first create a list after that we can create a series from list.

Example 1

```
import pandas as pd
result=pd.Series(["IND","AUS","NZ","ENG"])
result
```

Example:2

```
import pandas as pd
result=pd.Series(["IND","AUS","NZ","ENG"],index=["i","ii","iii","iv"])
result
```

Creating a series from dictionary:-In order to create a series from dictionary, we have to first create a dictionary after that we can create a series from dictionary.

Example:3

```
import pandas as pd
data={"IND":2000,"Aus":5000,"NZ":4500,"US":9000}
result=pd.Series(data)
result
```

Creating a series from array:-In order to create a series from array, we have to import a numpy module and have to use array() function.

Example 4:-

```
import pandas as pd,numpy as np
arr=np.array([17,18,3,7,12])
result=pd.Series(arr)
```

Result

Creating a series from csv file:

Example 5:-

```
import pandas as pd
df=pd.read_csv('e:\\weather_data.csv')
result=pd.Series(df['day'])
```

result

- In order to access the series element refers to the index number. Use the index operator [ ] to access an element in a series. The index must be an integer.
- In order to access multiple elements from a series, we use Slice operation.

DataFrame:-

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

Load Files Into a DataFrame:-

- If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example:-

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

Create Dataframe from dictionary:-

```
import pandas as pd
student_info={
    'name':['vaibhav','harsh','jaydeep','bhaumik'],
    'percentage':[77,74,75,65],
    'branch':['CSE','Mech','IC','CSE']
}
df=pd.DataFrame(student_info)
df
```

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

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

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

```
print(df.loc["day2"])
```

Manipulating DataFrame

- The data in the real world is very unpleasant & unordered so by performing certain operations we can make data understandable based on one's requirements, this process of converting unordered data into meaningful information can be done by data manipulation.
- To perform data manipulation first we need to import pandas library and read the dataframe.

```
import pandas as pd
df=pd.read_csv("e:\\weather_info.csv")
df
```

- We can read the dataframe by using head() function also which is having an argument (n) i.e. number of rows to be displayed.

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

- Counting the rows and columns in DataFrame using shape attribute. It returns the no. of rows and columns enclosed in a tuple.

```
df.shape
```

- Counting the rows and columns in DataFrame using shape attribute. It returns the no. of rows and columns enclosed in a tuple.

```
df.shape
```

- Summary of Statistics of DataFrame using describe() method.

```
df.describe()
```

- Dropping the missing values in DataFrame, it can be done using the dropna() method, it removes all the NaN values in the dataframe.

```
df.dropna()
df.dropna(axis=1)
```

- This will drop all the columns with any missing values.

- Sorting the DataFrame using sort_values() method.

```
df.sort_values(by=['temperature'], ascending=True)
```

Importing and Exporting data with Excel file

- For importing an Excel file into Python using Pandas we have to use pandas.read_excel() function.This function return DataFrame.

Example:-

```
import pandas as pd
df = pd.read_excel("marwadi_university.xlsx")
print(df)
```

- Call to_excel() function with the file name to export the DataFrame.First we need to create the DataFrame.

Example:-

```
import pandas as pd
student_info={
'name':['vaibhav','harsh','jay'],
'city':['jamnagar','ahmedabad','baroda'], 'age':[30,28,33] }
```

```
df=pd.DataFrame(student_info)
df.to_excel("new.xlsx")
```

Introduction of Matplotlib:-

- Matplotlib is basically a Multiplatform visualization library for data that is built on NumPy Arrays.
- It was introduced by John Hunter in the year 2002.
- As Matplotlib is a visualization library, these visualizations allow us to represent huge amount of data in visual form like charts and plots.
- Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

- Matplotlib is useful in creating 2D plots from the data in Arrays.
- This library gets easily integrated with the Pandas package which is used for data manipulation.

Installation of Matplotlib:-

- If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.
- Install it using this command:pip install matplotlib
- If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib

- Once Matplotlib is installed, import it in your applications by adding the import module statement:

```
import matplotlib
```

Checking Matplotlib Version

•The version string is stored under ___version__attribute.

```
import matplotlib
print(matplotlib.__version__)
```

Annotation:-

- Annotate means to label something.
- Suppose we draw a VI characteristic graph in this, we label the x-axis as V(voltage) and the y-axis as I(current). Similarly, the function helps us label graphs generated with the help of matplotlib.

Syntax:-matplotlib.pyplot.annotate(s,xy,xytext,arrowprops)

Parameters:-This method accept the following parameters that are described below:

s:-This parameter is the text of the annotation

xy:-This parameter is the point (x, y) to annotate.

xytext:-An optional parameter represents the position where the text along X and Y needs to be placed.

arrowprops:-This parameter is also an optional value and contains “dict” type.

Example:

```
import matplotlib.pyplot as plt
plt.plot([10,20,30,40],marker="o")
plt.xlabel("X  Label")
plt.ylabel("Y Label")
plt.title("Annotate function")
plt.annotate("(1,20)",(1,20),(0.5,25),size=12,color="red",arrowprops={'arrowstyle':'<->'})
plt.show()
```

Legends

- Plot legends give meaning to a visualization, assigning meaning to the various plot elements.
- Legends can be placed in various positions: A legend can be placed inside or outside the chart and the position can be moved.
- In the matplotlib library, there’s a function called legend() which is used to Place a legend on the axes.
- We usually call this legend() function in three different ways:

  legend()#without any argument

  legend(labels)

  legend(handles,labels)

Example

```
import matplotlib.pyplot as plt
import numpy as np
t=np.array([1,2,3,4])
plt.plot(t**2,t,"red",label="squares")
plt.plot(t**3,t,"green",label="cubes")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.title("Legend")
plt.legend()
#plt.legend(["square","cubes"])
plt.show()
```

The Following are some more attributes of function legend() :
shadow:-Whether to draw a shadow behind the legend. It's Default value is None.

fontsize:-It consist of either string value(xx-small,x-small,small,medium,large,x-large,xx-large) or integer value.

facecolor:-The legend’s background color

edgecolor:-The legend’s background patch edge color

markerfirst:-If False, legend marker is placed to the right of the legend label.

frameon:-Whether the legend should be drawn on a patch

Types of charts

Pie chart:-

- A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data.
- The area of slices of the pie represents the percentage of the parts of the data.
- The slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the wedge. The area of a wedge represents the relative percentage of that part with respect to whole data.
- Pie charts are commonly used in business presentations like sales, operations, survey results, resources, etc as they provide a quick summary.

Syntax:-matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)

Parameters:-
data represents the array of data values to be plotted

labels is a list of sequence of strings which sets the label of each wedge.

color attribute is used to provide color to the wedges.

autopct is a string used to label the wedge with their numerical value.

shadow is used to create shadow of wedge.

Example:-

```
import matplotlib.pyplot as plt
slices=[59219,55466,47544,36443,35917]
x=["PHP","SQL","Python","C#.NET","Java"]
y=[0,0,0.1,0,0]
z=['#008fd5','#fc4f30','#e5ae37','#6d904f','#BD57A7']
plt.pie(slices,labels=x,wedgeprops={'edgecolor':'black'},explode=y,shadow=True,autopct='
%1.1f%%',colors=z)
plt.title("Pie Chart")
plt.show()
```

Bar Graph:-

- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.
- The bar plots can be plotted horizontally or vertically.
- A bar chart describes the comparisons between the discrete categories.
- One of the axis of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.
- bar() function is used to draw bar graph.

Syntax:-plt.bar(x,height,width,bottom,align)

- The function creates a bar plot bounded with a rectangle depending on the given parameters.

Example:

```
import matplotlib.pyplot as plt
x=["Computer Engineering","BPharm","MBA"]
h=[540,300,150]
c=["purple","lime","gold"]
plt.bar(x,h,0.4,align="edge",color=c,edgecolor="red",linewidth=3)
plt.xlabel("Courses")
plt.ylabel("Students  enrolled")
plt.title("Marwadi  University")
plt.show()
```

Line plot:-

- A line plot visualizes information as a series of data points called markers connected by straight line segments.
- These are also used to express trends in data over intervals of time.

•Line Plots display a numerical variable on one axis and a categorical variable on the other.

Example:-

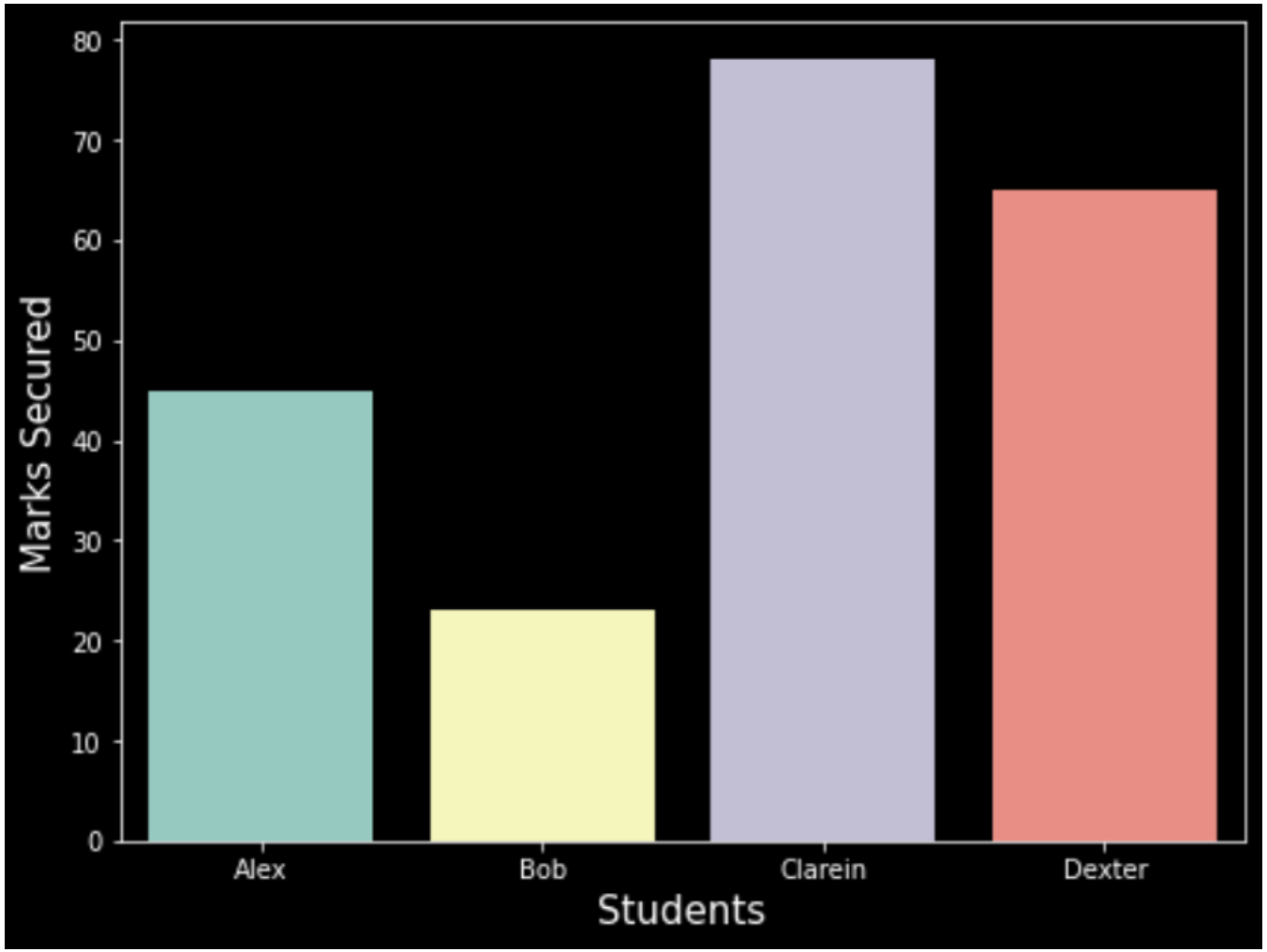
```
import matplotlib.pyplot as plt
year=[2011,2013,2015,2017,2019,2021]
unemployee=[5.5,7,6.2,4.5,2,5]
plt.plot( year, unemployee, marker="o")
plt.xlabel("Year")
plt.ylabel(“Unemployement rate")
plt.title("Indian Unemployement")
plt.show()
```

Plotting directly from Pandas Data Frame

Marwadi University

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = {"Name": ["Alex", "Bob", "Clarein", "Dexter"],
        "Marks": [45, 23, 78, 65]}
df = pd.DataFrame(data, columns=['Name', 'Marks'])

plt.figure(figsize=(8, 6))
plots = sns.barplot(x="Name", y="Marks", data=df)
plt.xlabel("Students", size=15)
plt.ylabel("Marks Secured", size=15)
plt.show()
```


80
70
60
50
40
30
20
10
0
Marks Secured
Alex
Bob
Clarein
Dexter
Students

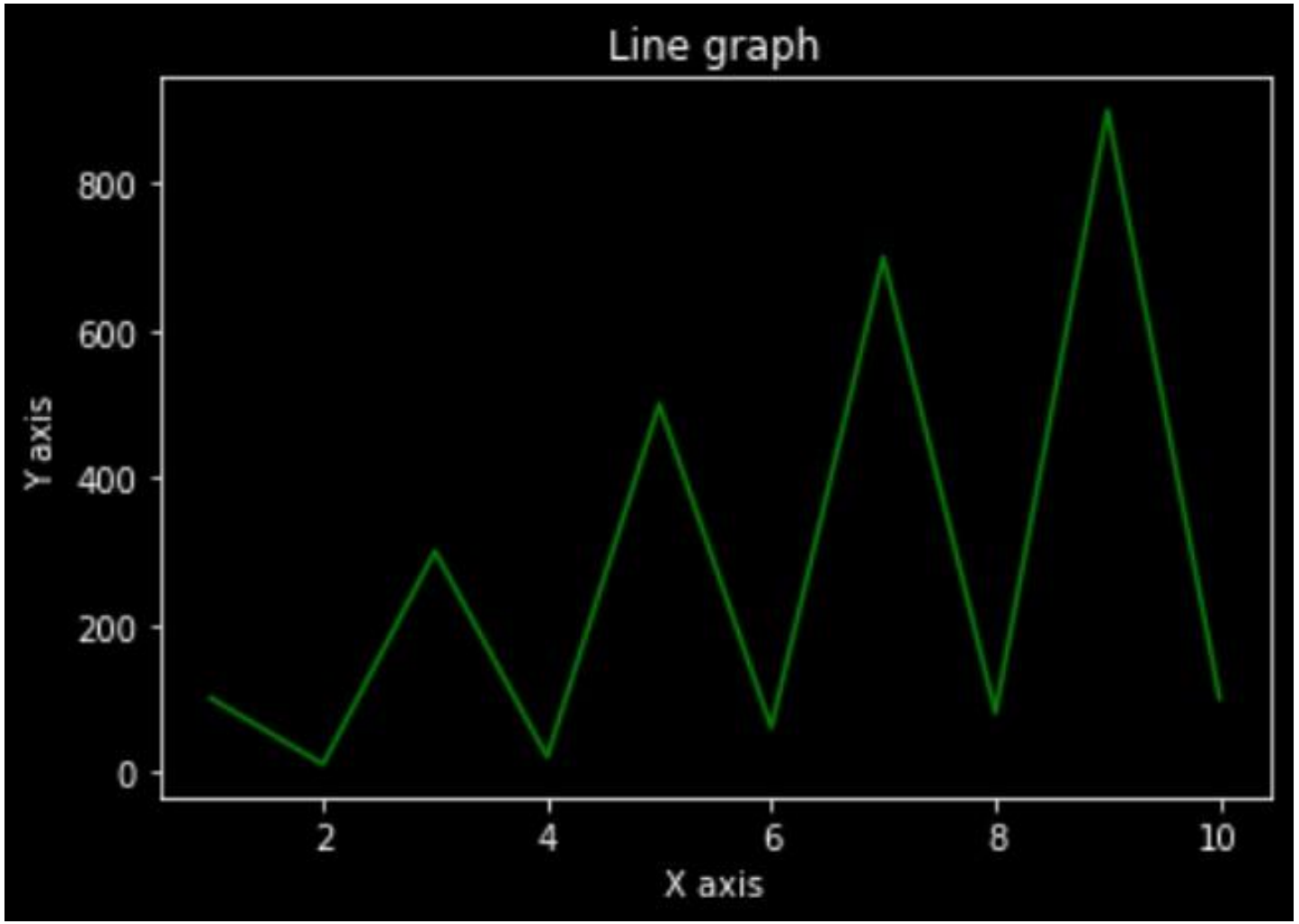
Plotting directly from NumPy Array

Marwadi University

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

x = np.arange(1, 11)
y = np.array([100, 10, 300, 20, 500,
              60, 700, 80, 900, 100])

plt.title("Line graph")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.plot(x, y, color ="green")
plt.show()
```



Marwadi University

Thank You