# Numpy

# Libraries

- pip install numpy (Already in Anaconda)
- pip install matplotlib (Already in Anaconda)
- pip install pandas (Already in Anaconda)
- pip install scikit-learn

# Introduction

- NumPy is a Python package which stands for 'Numerical Python'.
- It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

# Operations using NumPy

- Using NumPy, a developer can perform the following operations :-
  - Mathematical and logical operations on arrays.
  - Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
- **ndarray Object**
  - The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index. Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**). Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.

# Datatypes

- **bool:** Boolean (True or False) stored as a byte
- **int_ :** Default integer type (same as C long; normally either int64 or int32)
- **int8:** Byte (-128 to 127)
- **int16:** Integer (-32768 to 32767)
- **int32:** Integer (-2147483648 to 2147483647)
- **int64:** Integer (-9223372036854775808 to 9223372036854775807)
- **uint8:** Unsigned integer (0 to 255)
- **uint16:** Unsigned integer (0 to 65535)
- **uint32:** Unsigned integer (0 to 4294967295)
- **uint64:** Unsigned integer (0 to 18446744073709551615)
- **float_:** Shorthand for float64

# NumPy - Array Attributes

- **ndarray.shape**: This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

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

**Output:** (2, 3)

**6**

**ndarray**

# Reshape the ndarray

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

- **NumPy also provides a reshape function to resize an array.**

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

# Resize

```
import numpy as np
a=np.array([[0,1],[2,3]])
b=np.resize(a,(2,3))
```

Output:

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

# arange function

- **numpy.arange:** This function returns an **ndarray** object containing evenly spaced values within a given range. The format of the function is as follows −
  - numpy.arange(start, stop, step, dtype)
- **Program**

```
import numpy as np
x = np.arange(10,20,2)
print( x)
```

# arange function

- This array attribute returns the number of array dimensions.
- **an array of evenly spaced numbers**

```
import numpy as np
a = np.arange(24)
print (a)
```

Output:

```
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

# Program

```
# this is one dimensional array
import numpy as np
a = np.arange(24)
print(a.ndim)

# now reshape it
b = a.reshape(2,4,3)
print( b)
print(b.ndim)
# b is having three dimensions
```

# numpy.itemsize

- This array attribute returns the length of each element of array in bytes.

**Program:**

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print( x.itemsize)
```

# Program

```
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.float32)
print(x.itemsize)
```

# NumPy - Array Creation Routines

- **numpy.empty:** It creates an uninitialized array of specified shape and dtype. It uses the following constructor −
  - numpy.empty(shape, dtype = float, order = 'C/F')
  - ‘C’ for C-style row-major array, ‘F’ for FORTRAN style column-major array
- Program:

```
import numpy as np
x = np.empty([3,2], dtype = int)
print (x)
```

# numpy.zeros

- Returns a new array of specified size, filled with zeros.
  - numpy.zeros(shape, dtype = float, order = 'C')
- Program

```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print (x)
```

# Program

```
import numpy as np
x = np.zeros((5,), dtype = np.int)
print (x)
```

# numpy.ones

- Returns a new array of specified size and type, filled with ones.
  - numpy.ones(shape, dtype = None, order = 'C')
- Program:

```
import numpy as np
x = np.ones([2,2], dtype = int)
print (x)
```

# NumPy - Array From Existing Data

- **numpy.asarray:** This routine is useful for converting Python sequence into ndarray.
  - numpy.asarray(a, dtype = None, order = None)

**Program to convert list into arrray**

```
import numpy as np
x = [1,2,3]
a = np.asarray(x)
b=list(a) # from array to list
print (a)
print(type(a))
print(type(b))
```

- **numpy.linspace**: This function is similar to **arange()** function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows −

Program:

```
import numpy as np
x = np.linspace(10,20,5)
print (x)
```

**Output:** [10. 12.5 15. 17.5 20.]

# Arithmetic operation

- Np.Sum(a): sum of all elements of a matrix
- Np.sum(a, axis=0): Sum of elements of a matrix column-wise
- Np.sum(a,axis=1): Sum of elements of a matrix row-wise

Program

```
Import numpy as np
a=[[0,1],[0,5]]
print(np.sum(a))
print(np.sum(a,axis=0))
print(np.sum(a,axis=1))
```

Output:

```
6
[0,6]
[1,5]
```

# Matrix operations

- add(): addition of two matrices
- Subtract(): Subtraction of two matrices
- multiply(): Multiplication of two matrices
- divide(): Division of two matrices
- dot():

Program:

```
Import numpy as np
A=np.array[[0,1,2],[2,3,4]]
B=np.array([4,5,6])
print(np.add(a,b))
print(np.subtract(a,b))
print(np.multiply(a,b))
print(np.divide(a,b))
print(np.dot(a,b))
```

# Statistical functions

- **numpy.amin() and numpy.amax()**: These functions return the minimum and the maximum from the elements in the given array along the specified axis.

**Program**

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print(np.amin(a,axis=1))#  0 for column 1 for row
print(np.amin(a,axis=0))
print(np.amax(a,axis=1))
print(np.amax(a,axis=0))
```

# numpy.ptp()

- The **numpy.ptp()** function returns the range (maximum-minimum) of values along an axis.

Program

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print(np.ptp(a))
print(np.ptp(a,axis=1))
print(np.ptp(a,axis=0))
```

# numpy.median()

- **Median** is defined as the value separating the higher half of a data sample from the lower half.

Program

```
import numpy as np
a= np.array([[30,65,70],[80,95,10],[50,90,60]])
print(np.median(a))
print(np.median(a, axis=0))
print(np.median(a,axis=1))
```

# numpy.mean()

- The **numpy.mean()** function returns the arithmetic mean of elements in the array. If the axis is mentioned, it is calculated along it.

Program

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(np.mean(a))
print(np.mean(a,axis=0))
print(np.mean(a,axis=1))
```

# numpy.average()

- The **numpy.average()** function computes the weighted average of elements in an array according to their respective weight given in another array.
- Considering an array [1,2,3,4] and corresponding weights [4,3,2,1], the weighted average is calculated by adding the product of the corresponding elements and dividing the sum by the sum of weights.
- Weighted average = (1*4+2*3+3*2+4*1)/(4+3+2+1)

# Program

```
import numpy as np
a = np.array([1,2,3,4])
print(np.average(a))
wts = np.array([4,3,2,1])
print(np.average(a,weights=wts))
```

# Standard Deviation

- Standard deviation is the square root of the average of squared deviations from mean. The formula for standard deviation is as follows –
  - std = sqrt(mean(abs(x - x.mean())**2))

Program

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

# Variance

- Variance is the average of squared deviations, i.e., **mean(abs(x - x.mean())**2)**. In other words, the standard deviation is the square root of variance.

Program

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

# Broadcasting

- The term **broadcasting** refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations.
- Arithmetic operations on arrays are usually done on corresponding elements.
- If two arrays are of exactly the same shape, then these operations are smoothly performed.

# Program

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
Print( c)
```

Output:

```
[10  40  90  160]
```

- If the dimensions of two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability.
- The smaller array is **broadcast** to the size of the larger array so that they have compatible shapes.

- Broadcasting is possible if the following rules are satisfied –
  - Array with smaller **ndim** than the other is prepended with '1' in its shape.
  - Size in each dimension of the output shape is maximum of the input sizes in that dimension.
  - An input can be used in calculation, if its size in a particular dimension matches the output size or its value is exactly 1.
  - If an input has a dimension size of 1, the first data entry in that dimension is used for all calculations along that dimension.

# Example

```
import numpy as np
a=np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20
.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])
print(a)
print(b)
print(a+b)
```

# BroadCasting

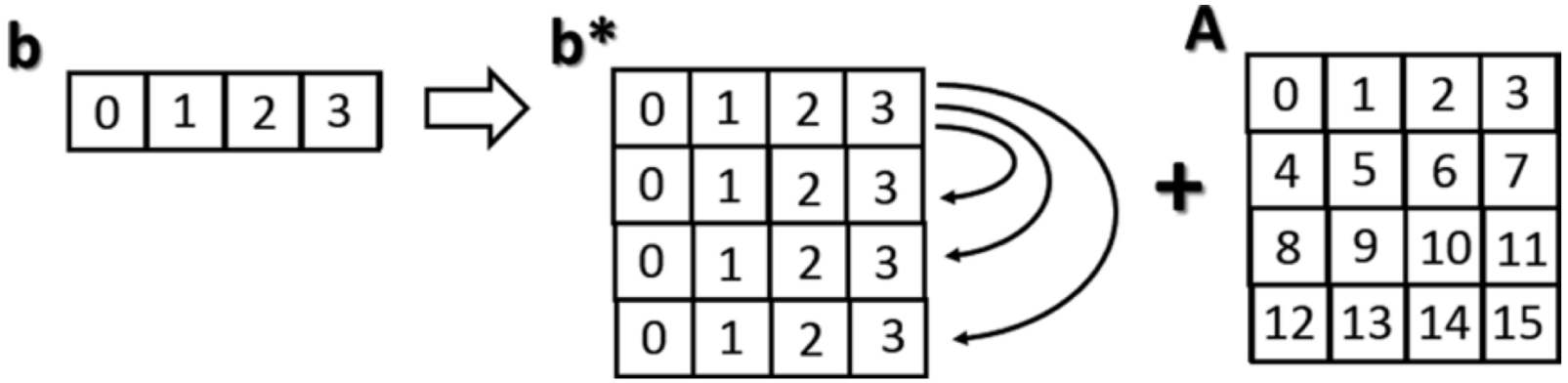



***Figure 3-5.*** *An application of the second broadcasting rule*

```
>>> m = np.arange(6).reshape(3, 1, 2)
>>> n = np.arange(6).reshape(3, 2, 1)
>>> m
array([[[0, 1]],

       [[2, 3]],

       [[4, 5]]])
>>> n
array([[[0],
        [1]],

       [[2],
        [3]],

       [[4],
        [5]]])
```

Broadcasted to →

```
m* = [[[0,1],
       [0,1]],
      [[2,3],
       [2,3]],
      [[4,5],
       [4,5]]]
```

```
n* = [[[0,0],
       [1,1]],
      [[2,2],
       [3,3]],
      [[4,4],
       [5,5]]]
```

# Pandas

- Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Time Series functionality.
- High performance merging and joining of data.

# Python Pandas - Series

- Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.
  - pandas.Series( data, index, dtype, copy)

Program

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
Print( s)
```

# Structured Arrays

So far in the various examples in previous sections, you saw monodimensional and two-dimensional arrays. Actually, NumPy provides the possibility of creating arrays that are much more complex not only in size, but in the structure itself, called precisely **structured arrays**. This type of array contains structs or records instead of the individual items.

For example, you can create a simple array of structs as items. Thanks to the **dtype** option, you can specify a list of comma-separated specifiers to indicate the elements that will constitute the struct, along with its data type and order.

```
bytes                b1
int                  i1, i2, i4, i8
unsigned ints        u1, u2, u4, u8
floats               f2, f4, f8
complex              c8, c16
fixed length strings a<n>
```

For example, if you want to specify a struct consisting of an integer, a character string of length 6 and a Boolean value, you will specify the three types of data in the dtype option with the right order using the corresponding specifiers.

```
>>> structured = np.array([(1, 'First', 0.5, 1+2j),(2, 'Second', 1.3, 2-2j),
(3, 'Third', 0.8, 1+3j)],dtype=('i2, a6, f4, c8'))
>>> structured
array([(1, 'First', 0.5, (1+2j)),
       (2, 'Second', 1.2999999523162842, (2-2j)),
       (3, 'Third', 0.800000011920929, (1+3j))],
      dtype=[('f0', '<i2'), ('f1', 'S6'), ('f2', '<f4'), ('f3', '<c8')])
```

# Python Pandas - DataFrame

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- **Features of DataFrame**
  - Potentially columns are of different types
  - Size – Mutable
  - Labeled axes (rows and columns)
  - Can Perform Arithmetic operations on rows and columns

# The DataFrame

The **DataFrame** is a tabular data structure very similar to the Spreadsheet (the most familiar are Excel spreadsheets). This data structure is designed to extend the case of the Series to multiple dimensions. In fact, the DataFrame consists of an ordered collection of columns (see Figure 4-2), each of which can contain a value of different type (numeric, string, Boolean, etc.).

**DataFrame**

| index | color | object | price |
|---|---|---|---|
| 0 | blue | ball | 1.2 |
| 1 | green | pen | 1.0 |
| 2 | yellow | pencil | 0.6 |
| 3 | red | paper | 0.9 |
| 4 | white | mug | 1.7 |

```
>>> data = {'color' : ['blue','green','yellow','red','white'],
                       'object' : ['ball','pen','pencil','paper','mug'],
                       'price' : [1.2,1.0,0.6,0.9,1.7]}
```

# Example

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print (df)
```

# Create a DataFrame from Dict of Series

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print (df)
```

# Column Selection

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print df ['one']
```

# Column Addition

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
# Adding a new column to an existing DataFrame object with column label by passing new series
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print (df)
 print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']
print (df)
```

# Column Deletion

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' :
pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']), 'three' :
pd.Series([10,20,30], index=['a','b','c'])}
df = pd.DataFrame(d)
print ("Our dataframe is:")
print (df)
 # using del function
print ("Deleting the first column using DEL function:")
del df['one']
print (df)
 # using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print (df)
```

# Row Selection, Addition, and Deletion

- Rows can be selected by passing row label to a **loc** function.

Program

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
Print( df.loc['b'])
```

# Selection by integer location

- Rows can be selected by passing integer location to an **iloc** function.

Program

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
print( df.iloc[2])
```

# Slice Rows

- Multiple rows can be selected using ‘ : ’ operator.

Program

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
   'd'])}
df = pd.DataFrame(d)
print (df[2:4])
```

# Addition of Rows

- Add new rows to a DataFrame using the **append** function.

Program

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
print (df)
```

# Deletion of Rows

- Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

Program

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
# Drop rows with label 0
df = df.drop(0)
print (df)
```

# Python Pandas - IO Tools

- The two functions for reading text files (or the flat files) are **read_csv()** and **read_table()**. They both use the same parsing code to intelligently convert tabular data into a **DataFrame** object .
- Consider the following temp.csv file

S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000

2,Lee,32,HongKong,3000

3,Steven,43,Bay Area,8300

4,Ram,38,Hyderabad,3900

# read.csv

```
import pandas as pd
df=pd.read_csv("temp.csv")
print (df)
```

# Converters

```
import pandas as pd
df = pd.read_csv("temp.csv", dtype={'Salary': np.float64})
print (df.dtypes)
```

# header_names

```
import pandas as pd
df=pd.read_csv("temp.csv", names=['a', 'b', 'c','d','e'])
print (df)
```

# skiprows

```
import pandas as pd
df=pd.read_csv("temp.csv", skiprows=2)
print (df)
```

# Write to .csv file

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df =
pd.DataFrame(data,columns=['Name','Age'])
print (df)
df.to_csv('temp.csv')
```

# Data visualization using Matplotlib

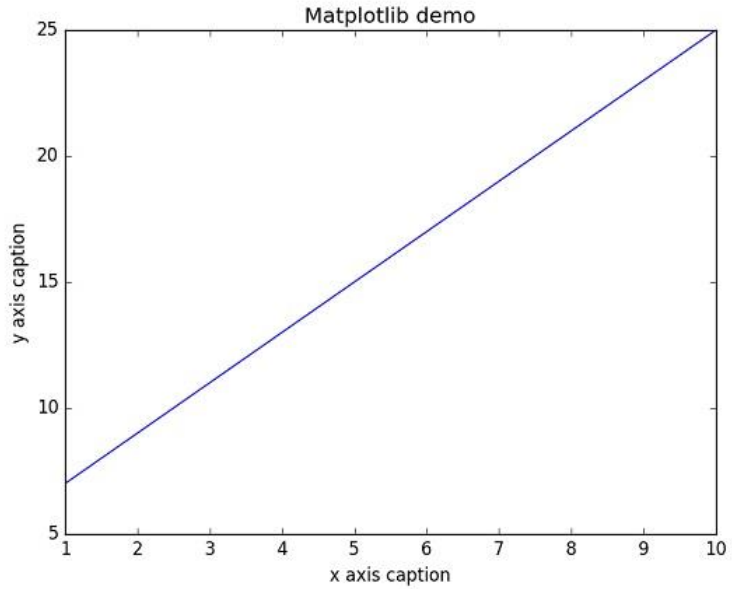
- Matplotlib is a plotting library for Python.
- Conventionally, the package is imported into the Python script by adding the following statement −
  - from matplotlib import pyplot as plt
- Here **pyplot()** is the most important function in matplotlib library, which is used to plot 2D data.

# Example: Line Plot

- Plot the equation **y = 2x + 5**

**Program**

```
import numpy as np
from matplotlib import pyplot as plt
x = np.arange(1,11)
y = 2 * x + 5
plt.title(“Line Plot")
plt.xlabel("x axis")
plt.ylabel("y axis”)
plt.plot(x,y)
plt.show()
```



- Instead of the linear graph, the values can be displayed discretely by adding a format string to the **plot()** function.
- ‘-’ : Solid line style
- ‘- -’ : Dashed line style
- ‘-.’ : Dash-dot line style
- ‘:’ : Dotted line style
- ‘.’ : Point marker
- ‘,’ : Pixel marker
- **'o‘**: Circle marker
- **'v‘**: Triangle_down marker
- **'^‘**: Triangle_up marker
- **'<‘**: Triangle_left marker
- **'>‘**:Triangle_right marker
- **'s‘**: Square marker
- **'p‘**: Pentagon marker
- **'*‘**: Star marker

The following color abbreviations are also defined.

| Character | Color |
|---|---|
| 'b' | Blue |
| 'g' | Green |
| 'r' | Red |
| 'c' | Cyan |
| 'm' | Magenta |
| 'y' | Yellow |
| 'k' | Black |
| 'w' | White |

# Example

```
import numpy as np
from matplotlib import pyplot as plt
x = np.arange(1,11)
y = 2 * x + 5
plt.title(“Line Plot")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y,"ob")
plt.show()
```

# Sine Wave Plot

```
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()
```

# subplot()

- The subplot() function allows you to plot different things in the same figure. In the following script, **sine** and **cosine values** are plotted.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)
# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')
# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
# Show the figure.
plt.show()
```

# bar()

- The **pyplot submodule** provides **bar()** function to generate bar graphs.

Program

```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]
x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center') plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

# Histogram

- NumPy has a **numpy.histogram()** function that is a graphical representation of the frequency distribution of data. Rectangles of equal horizontal size corresponding to class interval called **bin** and **variable height** corresponding to frequency.

Program

```
import numpy as np
a=np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("histogram")
plt.show()
```

# Pie Chart

- Pie charts are good to show proportional data of different categories and figures are usually in percentages.

Program

```
import matplotlib.pyplot as plt
province_population = [12344408, 2441523, 30523371, 110012442, 47886051]
activities = ['Balochistan', 'Gilgit-Baltistan', 'Khyber Pakhtunkhwa', 'Punjab', 'Sindh']
plt.pie(province_population, labels=activities, startangle=90, autopct='%.1f%%')
plt.title('Pakistan Population Province Wise')
plt.show()
```

# scikit

- SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the BSD licensed library to perform Mathematical, Scientific and Engineering Computations.
- The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation.

# SciPy - Cluster

- **K-means clustering** is a method for finding clusters and cluster centers in a set of unlabelled data.
- Intuitively, we might think of a cluster as – comprising of a group of data points, whose inter-point distances are small compared with the distances to points outside of the cluster.
- Given an initial set of K centers, the K-means algorithm iterates the following two steps −
  - For each center, the subset of training points (its cluster) that is closer to it is identified than any other center.
  - The mean of each feature for the data points in each cluster are computed, and this mean vector becomes the new center for that cluster.
- These two steps are iterated until the centers no longer move or the assignments no longer change.
- Then, a new point **x** can be assigned to the cluster of the closest prototype.

# SciPy - Interpolate

- Interpolation is the process of finding a value between two points on a line or a curve.
- To help us remember what it means, we should think of the first part of the word, 'inter,' as meaning 'enter,' which reminds us to look 'inside' the data we originally had.

Program

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(0, 4, 12)
y = np.cos(x**2/3+4)
print (x,y)
plt.plot(x, y,’o’)
plt.show()
```

# SciPy - Ndimage

- The SciPy ndimage submodule is dedicated to image processing. Here, ndimage means an n-dimensional image.
- Some of the most common tasks in image processing are as follows &miuns;
  - Input/Output, displaying images
  - Basic manipulations − Cropping, flipping, rotating, etc.
  - Image filtering − De-noising, sharpening, etc.
  - Image segmentation − Labeling pixels corresponding to different objects
  - Classification
  - Feature extraction
  - Registration

# Application of Tensorflow

- **It's a powerful machine learning framework**
- It is used for deep learning.
- It helps classify and cluster data like that with sometimes superhuman accuracy.
- **RankBrain:** A large-scale deployment of deep neural nets for search ranking on Google
- **SmartReply:** Deep LSTM model to automatically generate email responses
- **On-Device Computer Vision for OCR:** On-device computer vision model to do optical character recognition to enable real-time translation.
- **Massively Multitask Networks for Drug Discovery:** A deep neural network model for identifying promising drug candidates.

# Application of Keras

- Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.
- Weights are downloaded automatically when instantiating a model.
- **Models for image classification with weights trained on ImageNet:**
  - Xception
  - VGG16
  - VGG19
  - ResNet50
  - InceptionV3
  - InceptionResNetV2
  - MobileNet
  - DenseNet
  - NASNet
  - MobileNetV2

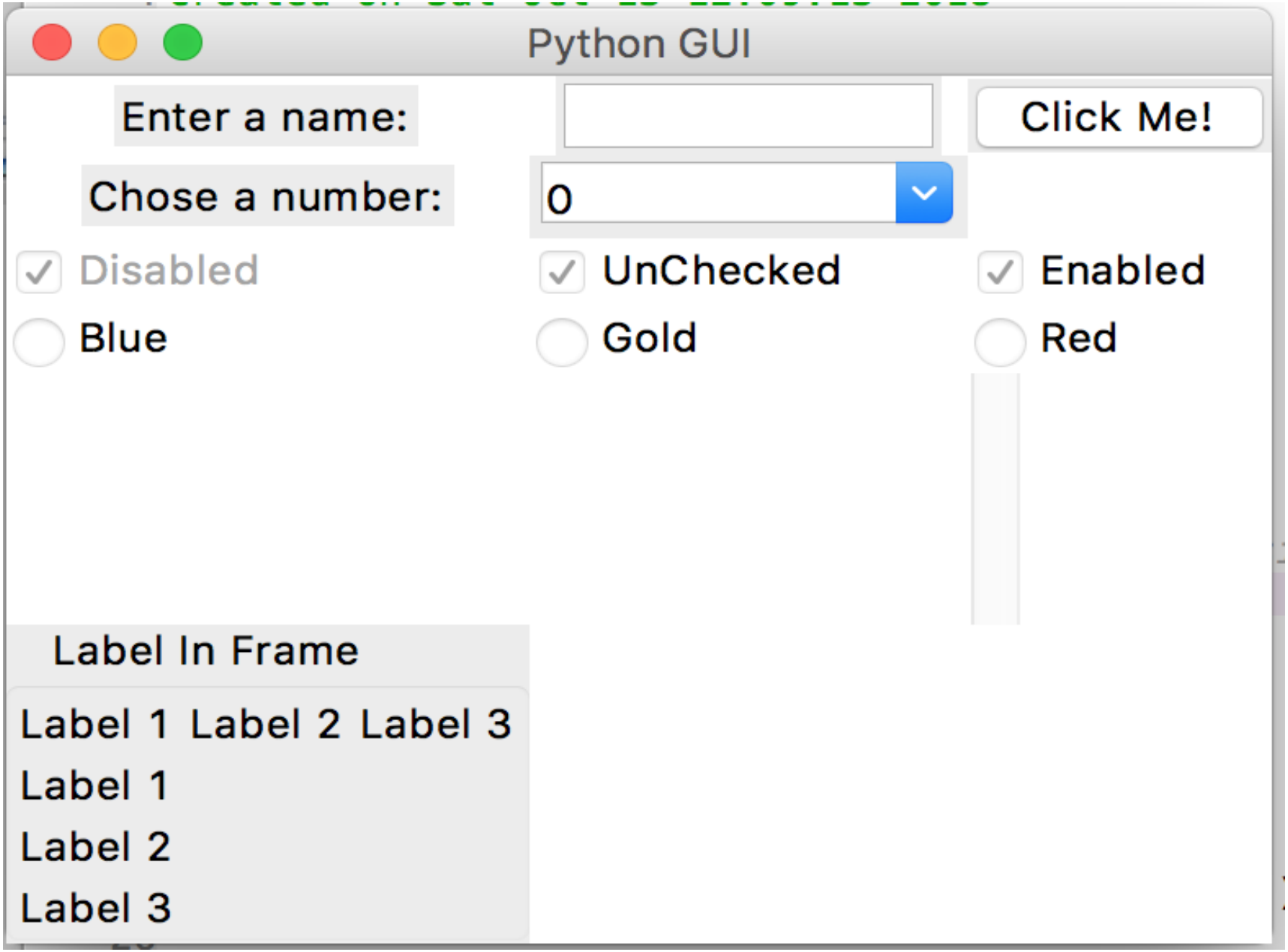
- **Keras has broad adoption in the industry and the research community**

# Application of Scikit-Learn

- Simple and efficient tools for data mining and data analysis
- Built on NumPy, SciPy, and matplotlib
- Applications:
  - Classification: identifying to which category an object belongs to.
  - Regression: Predicting a continuous-valued attribute associated with an object.
  - Clustering: Automatic grouping of similar objects into sets.
  - Dimensionality reduction: Reducing the number of random variables to consider.
  - Model Selection: Comparing, validating and choosing parameters and models.
  - Preprocessing: Feature extraction and normalization.

# Intelligent Systems Track

- Soft Computing Techniques
- Machine Learning Foundation
- Advanced Machine Learning
- Pattern Recognition


Python GUI
Enter a name:
Click Me!
Chose a number:
0
Disabled
UnChecked
Enabled
Blue
Gold
Red
Label In Frame
Label 1 Label 2 Label 3
Label 1
Label 2
Label 3