Unit-II
Python libraries suitable for Machine Learning

Computer Department

AVPTI Rajkot

Unit Outcomes

- Program using Python Libraries

Topics

- Numpy
- Panda
- Matplotlib
- sklearn

Numpy

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python".
- 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.
- It is written partially in Python, but most of the parts that require fast computation are written in C or C++.

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.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.
- The source code for NumPy is located at this github repository https://github.com/numpy/numpy
- Github enables many people to work on the same codebase.

Installation of NumPy

- C:\Users\Your Name>pip install numpy
- You can use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.
- Once Numpy Installed you have to import it
- import numpy…..is used to import Numpy library

Create and Access array…

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

- Accessing Array by referring to its index number
- import numpy as np
  ```
  arr = np.array([1, 2, 3, 4])
  print(arr[0])
  ```

Stacking

- Stacking is same as concatenation, the only difference is that stacking is done along a new axis.
- We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, i.e. stacking.
- We can use stack() with passing arrays and axis to perform stacking.by default axis is equal to 0.

-
  ```
  import numpy as np
  arr1 = np.array([1, 2, 3])
  arr2 = np.array([4, 5, 6])
  arr = np.stack((arr1, arr2), axis=1)
  print(arr)
  ```

Spitting

- Splitting is reverse operation of Joining.
- Joining merges multiple arrays into one and Splitting breaks one array into multiple.
- array_split() is used to split arrays with passing array that we want 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)
```

Maths Functions

- **numpy.add()** function is used when we want to compute the addition of two array. It add arguments element-wise.
- **numpy.subtract()** function is used when we want to compute the difference of two array.It returns the difference of arr1 and arr2, element-wise.
- **numpy.multiply()** function is used when we want to compute the multiplication of two array. It returns the product of arr1 and arr2, element-wise.

Continue…

- **numpy.divide()** function is used to divide the two arrays.
- Array element from first array is divided by elements from second array(all happens element-wise). Both arr1 and arr2 must have same shape and element in arr2 must not be zero,otherwise it will raise an error.
- **numpy.pow()** function is used to compute the first array elements raised to powers from the second array elements, element-wise. Both arrays must be having the same shape and each element of the first array must be raised to the corresponding positive value from the second array.
- **numpy.mod()** function is used to find modulo between two arrays element wise.It returns element-wise remainder of division between two array arr1 and arr2

Statistics Functions

- numpy.amax()computes the maximum of an array or the maximum of an array along a specified axis.

-
  ```
  import numpy as np
  arr = np.array([1,2,5,6,0])
   print(np.amax(arr))
  ```

- numpy.amin()computes the minimum of an array or the minimum of an array along a specified axis.

-
  ```
  import numpy as np
  arr = np.array([1,2,5,6,0])
   print(np.amin(arr))
  ```

Continue…

- numpy.mean() function returns the average of the array elements.
- The sum of elements, along with an axis divided by the number of elements, is known as **arithmetic mean**. The numpy.mean() function is used to compute the arithmetic mean along the specified axis
- import numpy as np

```
arr = np.array([2, 7, 5, 8, 9,4])
arr1 = np.mean(arr)
print(arr1)
```

Continue…

- numpy.median() function is used to compute the median of the given NumPy array .The term median is the value separating the higher half from the lower half of a data sample in other words median is a value in the middle when you sorted the values in the ascending order.
- import numpy as np

```
arr = np.array([2, 7, 5, 8, 9,4])
arr1 = np.median(arr)
print(arr1)
```

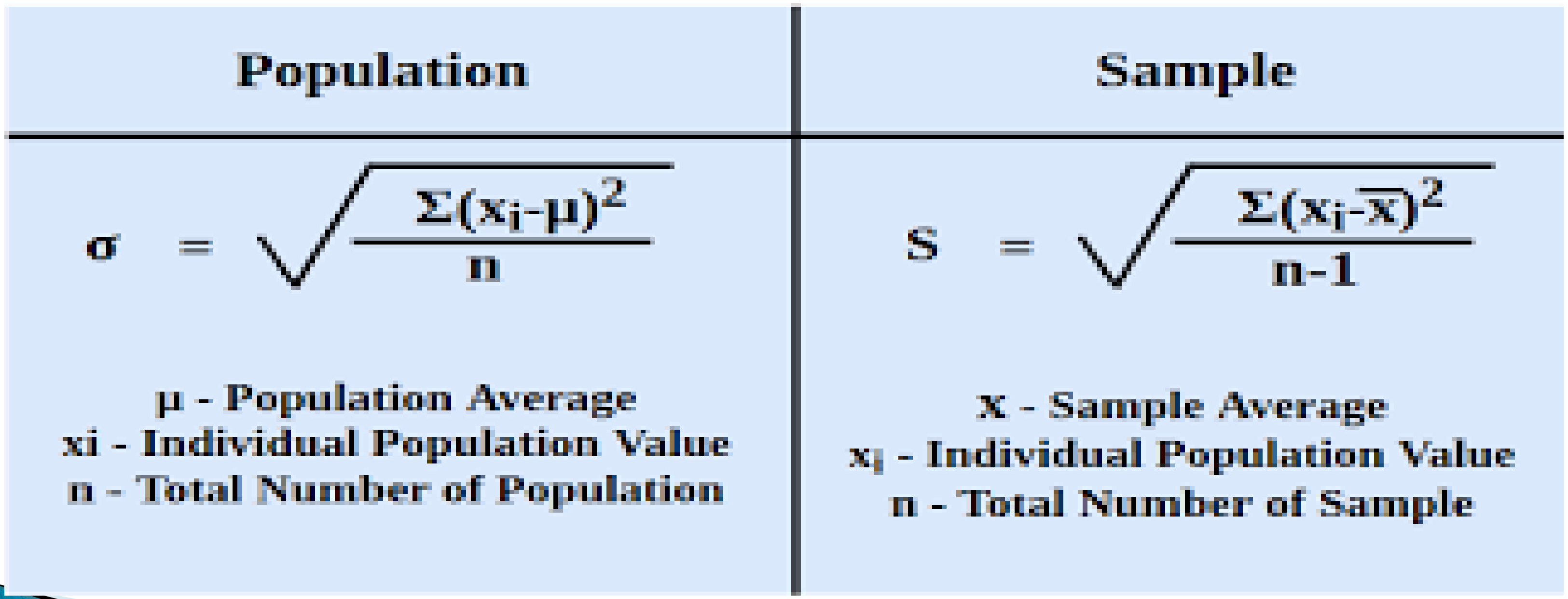
Continue…

- **numpy.std()** Compute the standard deviation along the specified axis.
- Returns the standard deviation, a measure of the spread of a distribution, of the array elements. The standard deviation is computed for the flattened array by default, otherwise over the specified axis.
- import numpy as np

```
arr = np.array([2, 7, 5, 8, 9,4])
arr1 = np.std(arr)
print(arr1)
```

Continue...

- Standard deviation calculates the extent to which the values differ from the average. Standard Deviation, the most widely used measure of dispersion, is based on all values.



Population	Sample
$$\sigma = \sqrt{\frac{\Sigma(x_i-\mu)^2}{n}}$$	$$S = \sqrt{\frac{\Sigma(x_i-\overline{x})^2}{n-1}}$$
μ - Population Average xi - Individual Population Value n - Total Number of Population	x - Sample Average x_i - Individual Population Value n - Total Number of Sample

Continue…

- **numpy.var()** Compute the variance along the specified axis.
- Returns the variance of the array elements, a measure of the spread of a distribution. The variance is computed for the flattened array by default, otherwise over the specified axis.
- It is a square of standard deviation.
- import numpy as np

```
arr = np.array([2, 7, 5, 8, 9,4])
arr1 = np.var(arr)
print(arr1)
```

Continue…

- **numpy.average()** used for calculating the weighted average along the specified axis.
- if the weights are not supplied for np.average(), it acts similarly to np.mean()
- Weighted average is an average in which each quantity to be averaged is assigned a weight. These weightings determine the relative importance of each quantity on average.

Continue…

- For example, if we need to find the average of 10, 13, and 25 on a simple average, it is (10 + 13 + 25) / 3 = 48 / 3 = 16.
- Now we took the same example with weight. Let's say that the weight of number 10 is 25%, 13 is 30%, and 25 is 45%. Weighted average of the above three numbers of would-be = (10 * 25%) + (13 * 30%) + (25 * 45%) = 2.5 + 3.9 + 11.25 = 17.65.
- **Weighted Average Formula = W_1X_1 + W_2X_2 +……+W_nX_n Here, w = respective weight (in percentage), x = value**

Continue…

- import numpy as np
  a = np.array([1,2,3,4])
  print (np.average(a))   ….o/p is 2.5
  wts = np.array([4,3,2,1])
  print (np.average(a,weights = wts)) ….o/p is 2

Weighted avg=sum of(element*weight)/sum of weight

Continue…

- numpy.ptp() function is used to **return a range of values along an axis.**
- "**ptp**" stands for **peak to peak.**
- The range can be calculated using **range= maximum_value - minimum_value.**
- import numpy as np
  inp = [[15, 18, 16, 63, 44], [19, 4, 29, 5, 20], [24, 4, 54, 6, 4,]]
  print(np.ptp(inp)) ….o/p is 63-4=59

Pandas

- Pandas is a Python library used for working with data sets.
- **pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- 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.

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.
- The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas
- Pandas is one of the most popular Python packages used in data science. Pandas offer a powerful, and flexible data structure (**Dataframe & Series**) to manipulate, and analyze the data. Visualization is the best way to interpret the data.

Installation of Pandas

- C:\Users\Your Name>pip install pandas
- You can use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.
- Once Pandas Installed you have to import it
- import pandas …..is used to import Pandas library
- import pandas as pd….. Now the Pandas package can be referred to as pd instead of pandas.

Series: Series()

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.
- Create a simple Pandas Series from a list:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

- 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.

```
print(myvar[0])
```

Continue…

- 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)
  ```
- When you have created labels, you can access an item by referring to the label.

```
print(myvar["y"])
```

Continue…

- Key/Value Objects as Series
- You can also use a key/value object, like a dictionary, when creating a Series.
- Create a simple Pandas Series from a dictionary:
- import pandas as pd
  calories = {"day1": 420, "day2": 380, "day3": 390}
  myvar = pd.Series(calories)
  print(myvar)
- Create a Series using only data from "day1" and "day2":

  myvar = pd.Series(calories, index = ["day1", "day2"])

Dataframes: DataFrames()

- Data sets in Pandas are usually multi-dimensional tables, called DataFrames.
- Series is like a column, a DataFrame is the whole table.
- A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.
- Create a simple Pandas DataFrame:
- 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)
```

Continue…

- Pandas use the loc attribute to return one or more specified row(s)
- #refer to the row index:
  print(df.loc[0])
- #use a list of indexes:
  print(df.loc[[0, 1]])
- Named Indexes

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

Continue...

- Add a list of names to give each row a name:

-
  ```
  import pandas as pd
  data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
  }
  df = pd.DataFrame(data, index =
  ["day1", "day2", "day3"])
  print(df)
  ```

-
  ```
  #refer to the named index:
  print(df.loc["day2"])
  ```

Read CSV File: read_csv()

- 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.
- In our examples we will be using a CSV file called 'data.csv'.
- If your data sets are stored in a file, Pandas can load them into a DataFrame.
- Load the CSV into a DataFrame:

-
  ```
  import pandas as pd
  df = pd.read_csv('data.csv')
  print(df)
  ```
- If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

Continue...

- import pandas as pd
  df = pd.read_csv('data.csv')
  print(df.to_string())
- Use to_string() to print the entire DataFrame.
- The number of rows returned is defined in Pandas option settings.
- You can check your system's maximum rows with the pd.options.display.max_rows ststement.
- import pandas as pd
  print(pd.options.display.max_rows)

Cleaning Empty Cells: dropna()

- Pandas is one of the packages that makes importing and analyzing data much easier. Sometimes CSV file has null values, which are later displayed as NaN in Pandas DataFrame.
- Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways.
- Remove all rows wit NULL values from the DataFrame.

-
  ```
  import pandas as pd
  df = pd.read_csv('data.csv')
  newdf = df.dropna()
  ```

Continue...

- *dataframe*.dropna(axis, how, thresh, subset, inplace)

Parameter	Value	Description
axis	0 1 'index' 'columns'	Optional, default 0. 0 and 'index'removes ROWS that contains NULL values 1 and 'columns' removes COLUMNS that contains NULL values
how	'all' 'any'	Optional, default 'any'. Specifies whether to remove the row or column when ALL values are NULL, or if ANY value is NULL.
thresh	*Number*	Optional, Specifies the number of NOT NULL values required to keep the row.
subset	*List*	Optional, specifies where to look for NULL values
inplace	True False	Optional, default False. If True: the removing is done on the current DataFrame. If False: returns a copy where the removing is done.

Cleaning Wrong Data: drop()

- The drop() method removes the specified row or column.
- By specifying the column axis (axis=‘columns’) it removes the specified column.
- By specifying the row axis (axis=‘index’) it removes the specified row.

Continue…

- *dataframe*.drop(*labels*, axis, index, columns, level, inplace., errors)

Parameter	Value	Description
labels		Optional, The labels or indexes to drop. If more than one, specify them in a list.
axis	0 1 'index' 'columns'	Optional, Which axis to check, default 0.
index	*String* *List*	Optional, Specifies the name of the rows to drop. Can be used instead of the labels parameter.
columns	*String* *List*	Optional, Specifies the name of the columns to drop. Can be used instead of the labels parameter.
level	*Number* *level name*	Optional, default None. Specifies which level (in a hierarchical multi index) to check along
inplace	True False	Optional, default False. If True: the removing is done on the current DataFrame. If False: returns a copy where the removing is done.
errors	'ignore' 'raise'	Optional, default 'ignore'. Specifies whether to ignore errors or not

Removing Duplicates: duplicated()

- The duplicated() method returns a Series with True and False values that describe which rows in the DataFrame are duplicated and not.
- Syntax……. *dataframe*.duplicated(subset, keep)
- Use the subset parameter to specify which columns to include when looking for duplicates. By default all columns are included.
- By default, the first occurrence of two or more duplicates will be set to False.
- Set the keep parameter to False to also set the first occurrence to True.
- It return a Series with a boolean value for each row in the DataFrame.

Continue…

Parameter	Value	Description
subset	column label(s)	Optional. A String, or a list, of the column names to include when looking for duplicates. Default subset=None (meaning no subset is specified, and all columns should be included.
keep	'first' 'last' False	Optional, default 'first'. Specifies how to deal with duplicates: 'first' means set the first occurrence to False, the rest to True. 'last' means set the last occurrence to False, the rest to True. False means set all occurrences to True.

Pandas Plotting: plot()

- We can plot a Dataframe using **the plot()** method.
- Pandas uses the plot() method to create diagrams.
- We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.
- There are a number of plots available to interpret the data. Each graph is used for a purpose. Some of the plots are BarPlots, ScatterPlots, and Histograms, etc.

Continue…

- import pandas as pd
  import matplotlib.pyplot as plt
  df = pd.read_csv('data.csv')
  df.plot(kind = 'scatter', x = 'Duration', y
  = 'Calories',color='red')
  plt.show()
- You can use kind is equal to line,bar,hist for different types of graphs and have to modifies the other arguments as per graph type.

Matplotlib

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter.
- 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 an amazing visualization library in Python for 2D plots of arrays.
- Matplotlib is a multi-platform data visualization library built on NumPy arrays.
- Matplotlib consists of several plots like line, bar, scatter, histogram etc.
- The source code for Matplotlib is located at github repository https://github.com/matplotlib/matplotlib

Continue…

- Install it using this command:

  C:\Users\Your Name>pip install matplotlib
- You can use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.
- import matplotlib…..for importing
- print(matplotlib.__version__)….for version check
- The version string is stored under __version__ attribute.

Matplotlib Pyplot.plot()

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias.
- import matplotlib.pyplot as plt
- The plot() function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the **x-axis**.
- Parameter 2 is an array containing the points on the **y-axis**.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

Continue…

- Draw a line in a diagram from position (1, 3) to position (8, 10)

-
  ```
  import matplotlib.pyplot as plt
  import numpy as np
  xpoints = np.array([1, 8])
  ypoints = np.array([3, 10])
  plt.plot(xpoints, ypoints)  // plot line
  plt.show()
  ```
- plt.plot(xpoints, ypoints, 'o')….to plot only markers.
- You can also pass multiple point and make X-axis default.
- You can use the keyword argument marker to emphasize each point with a specified marker

-
  ```
  plt.plot(ypoints, marker = 'o')
  plt.plot(ypoints, marker = '*')
  ```

Show: show()

- The **show() function** in pyplot module of matplotlib library is used to display all figures.
- plt.show()starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure or figures.

Labels: xlabel(), ylabel()

- With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.
- With Pyplot, you can use the title() function to set a title for the plot.
-
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

Continue…

- You can use the fontdict parameter in xlabel(),ylabel() and title() to set font properties for the title and labels.

-
  ```
  font1 = {'family':'serif','color':'blue','size':20}
  font2 = {'family':'serif','color':'darkred','size':15}

  plt.title("Sports Watch Data", fontdict = font1)
  plt.xlabel("Average Pulse", fontdict = font2)
  plt.ylabel("Calorie Burnage", fontdict = font2)
  ```

Grid: grid()

- With Pyplot, you can use the grid() function to add grid lines to the plot.

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

  x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
  y =
  np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

  plt.title("Sports Watch Data")
  plt.xlabel("Average Pulse")
  plt.ylabel("Calorie Burnage")

  plt.plot(x, y)
  plt.grid()
  plt.show()
  ```

Continue…

- plt.grid(axis = 'x')….for x-axis gridline
- plt.grid(axis = ‘y')….for y-axis gridline
- Default is both gridline.
- You can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*).
- plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

Bars: bar()

- With Pyplot, you can use the bar() function to draw bar graphs.

-
  ```
  import matplotlib.pyplot as plt
  import numpy as np
  x = np.array(["A", "B", "C", "D"])
  y = np.array([3, 8, 1, 10])
  plt.bar(x,y)
  plt.show()
  ```
- The categories and their values represented by the *first* and *second* argument as arrays.
- If you want the bars to be displayed horizontally instead of vertically, use the barh() function.

Continue…

- plt.bar(x, y, color = "red")
- plt.bar(x, y, width = 0.1)
- The barh() takes the keyword argument height() to set the height of the bars.
- plt.barh(x, y, height = 0.1)
- The default width and height value is 0.8

Histogram: hist()

- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- A histogram is an accurate representation of the distribution of numerical data.
- 10 students having SPI 9 to 10
- 20 students having SPI 8 to 9 etc.
- In Matplotlib, we use the hist() function to create histograms.
- The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

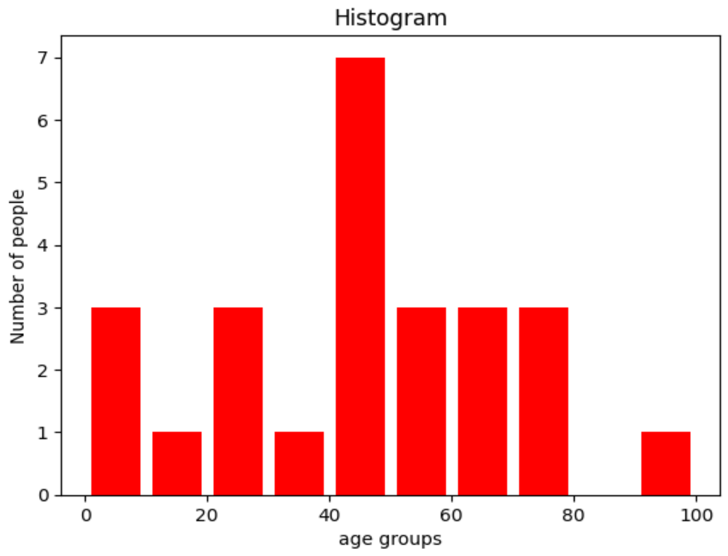
Continue…

- The **matplotlib.pyplot.hist()** function plots a histogram. It computes and draws the histogram of x.
- This method accept the following parameters
  - **x** : This parameter are the sequence of data.
  - **bins** : This parameter is an optional parameter and it contains the integer or sequence or string.
  - **range** : This parameter is an optional parameter and it the lower and upper range of the bins.
  - **histtype** : This parameter is an optional parameter and it is used to draw type of histogram. {‘bar’, ‘barstacked’, ‘step’, ‘stepfilled’}
  - rwidth,color,label are some other parameters.

Continue…

- import matplotlib.pyplot as plt
  import numpy as np

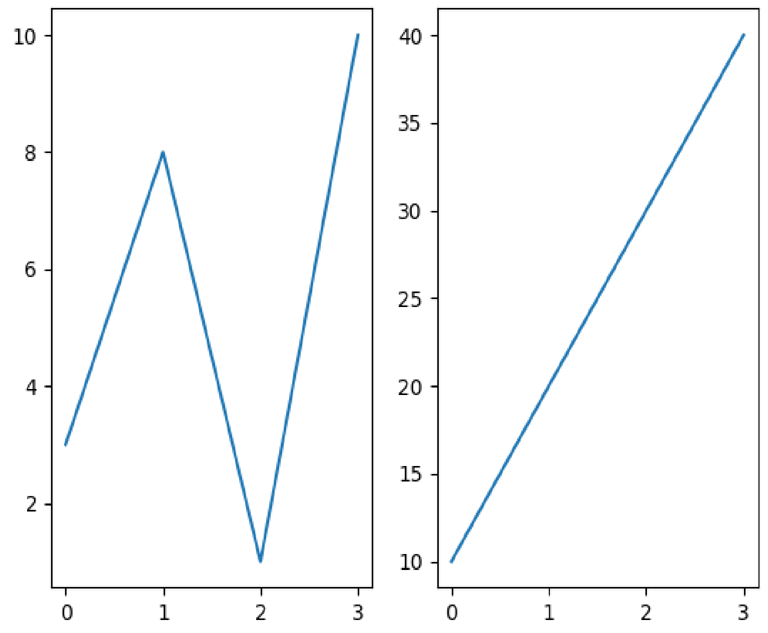
```
population_age=[21,53,60,49,25,27,30,42,40,1,2,102,95
,8,15,105,70,65,55,70,75,60,52,44,43,42,45]
bins=[0,10,20,30,40,50,60,70,80,90,100]

plt.hist(population_age,bins,histtype=
'bar',rwidth=0.8,color='red')

plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```



Subplot: subplot()

- With the subplot() function you can draw multiple plots in one figure.
- import matplotlib.pyplot as plt
  import numpy as np

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```

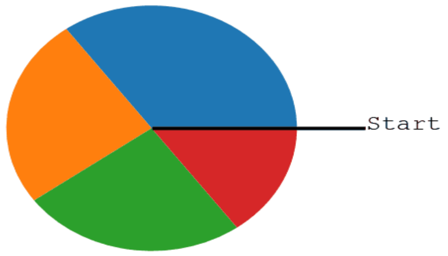


Continue…

- The subplot() function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the *first* and *second* argument.
- The third argument represents the index of the current plot.
- plt.subplot(1, 2, 1)
  #the figure has 1 row, 2 columns, and this plot is the *first* plot.
- You can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.

pie chart: pie()

- With Pyplot, you can use the pie() function to draw pie charts.
- A pie chart, sometimes called a circle chart, is a way of summarizing a set of nominal data or displaying the different values of a given variable (e.g. percentage distribution).
- In This type of chart circle is divided into a series of segments.
- Each segment is known as a wedge.
-
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



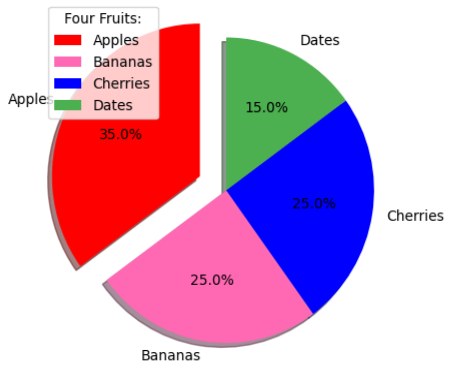
Continue…

- **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.
  startangle is used to defined an angle in degrees, default angle is 0.

Continue…

- mylabels =["Apples", "Bananas", "Cherries", "Dates"]
  myexplode = [0.2, 0, 0, 0]
  mycolors = [“red", "hotpink", "b", "#4CAF50"]

  plt.pie(y, labels = mylabels,explode = myexplode,colors = mycolors,startangle = 90,shadow = True, autopct='%1.1f%%')
- plt.legend()
  To add a list of explanation for each wedge
- plt.legend(title = "Four Fruits:")
  To add a header to the legend



Save the plotted images into pdf: savefig()

- savefig() method is used to save the figure created after plotting data. The figure created can be saved to our local machines by using this method.
- savefig(fname, dpi=None, facecolor='w', edgecolor='w',orientation='portrait', papertype=None, format=None,
transparent=False, bbox_inches=None,
pad_inches=0.1)

Continue…

- **fname**: This refers to a file name or file location.
- **dpi**: This is the number of dots per inch.
- **facecolor**: This controls the background color of the saved image. By default, the color will be white.
- **edgecolor**: This is used to add the border to the figure. By default, this is white.
- **orientation**: This can be a landscape or portrait orientation.
- **papertype**: This tells us which type of paper we want to use, like a letter, legal, a0 to a10, or something else.
- **format**: This is the file format.
- **transparent**: This makes the picture's background transparent.
- **bbox_inches**: This tells us how the image will be fit.
- **pad_inches**: This indicates the padding around the image.

Continue…

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

population_age=[21,53,60,49,25,27,30,42,40,1,2,102,95,8,15,105,70
,65,55,70,75,60,52,44,43,42,45]
bins=[0,10,20,30,40,50,60,70,80,90,100]

plt.hist(population_age,bins,histtype= 'bar',rwidth=0.8,color='red')
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')

plt.savefig("squares1.pdf",bbox_inches ="tight",
pad_inches = 1,transparent = True,facecolor ="g",
edgecolor ='w',
orientation ='landscape')

plt.show()
```

Sklearn....scikit-learn...scikit(SciPy Toolkit)

- 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.
- Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, dimensionality reduction and statistical tools for analyzing these models.
- This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.
- Scikit-learn is a community effort and anyone can contribute to it. This project is hosted on https://github.com/scikit-learn/scikit-learn.

Continue…

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
```

Using conda

- ➢ Following command can be used to install scikit-learn via conda

```
conda install scikit-learn
```

sklearn Features

- Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −
- **Supervised Learning algorithms** − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.
- **Unsupervised Learning algorithms** − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.
- **Clustering** − This model is used for grouping unlabeled data.
- **Cross Validation** − It is used to check the accuracy of supervised models on unseen data.
- **Dimensionality Reduction** − It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.
- **Ensemble methods** − As name suggest, it is used for combining the predictions of multiple supervised models.
- **Feature extraction** − It is used to extract the features from data to define the attributes in image and text data.
- **Feature selection** − It is used to identify useful attributes to create supervised models.
- **Open Source** − It is open source library

Steps to build a model sklearn

- Importing All the Required Libraries
- Loading the dataset
- Understanding the dataset
- Data preprocessing
- Data visualization
- Split the data set in training and testing sets
- Define a model
- Train a model
- Model evaluation
- Model prediction

Continue…

- from sklearn.model_selection import train_test_split

  from sklearn.linear_model import

  LinearRegression

  import pandas as pdd
- # Loading the dataset

  data_h = pdd.read_csv('kc_house_data.csv')
- # Selecting the features and target variable Features1 = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'zipcode']

  target = 'price'

  X1 = data_h[features1]

  y1 = data_h[target]

Continue…

- # We will perform the data splitting into training and testing sets

```
X_train, X_test, y_train, y_test =
train_test_split(X1, y1, test_size=0.2,
random_state=42)
```

- # instance of the Linear Regression model creation

```
model = LinearRegression()
```

- # Training the model

```
model.fit(X_train, y_train)
```

- # Making predictions on the test set

```
y_pred = model.predict(X_test)
```

Continue…

- # Evaluating the model

  score = model.score(X_test, y_test)

  print("Model Score:", score)

- # Predicting the price of a new house

 new_house = pdd.DataFrame({'bedrooms': [2], 'bathrooms': [2.5], 'sqft_living': [600], 'sqft_lot': [600], 'floors': [2], 'zipcode': [98008]})

 predicted_price = model.predict(new_house)

 print("Predicted Price:", predicted_price[0])

Happy Learning