

DATA VISUALIZATION Techniques in matplotlib

Project Report



March 12, 2022

RAHUL SINGH

Singhrahulsid1909@gmail.com

**DECLARATION**

I SHIV SHARMA hereby declare that the Project Report titled ‘**Data Visualization Techniques in Matplotlib**’ is the original work done by me after three months certified course at Brillica Services Private Ltd.

I also declare that I have strictly observed reporting ethics and duly discharged copy-right obligation and properly referred all outsoursing of materials used in this report and nothing is confidential in this report in respect of the company of my internship. I take the responsibility for legal and ethical requirements regarding this report

Date-

Signature of the Student

**(Shiv Sharma)**

**ABSTRACT**

Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations. These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand.

Data visualization can be utilized for a variety of purposes, and it’s important to note that is not only reserved for use by data teams. Management also leverages it to convey organizational structure and hierarchy while data analysts and data scientists use it to discover and explain patterns and trends.

Here in this project we will practice some of the data visualizations in matplotlib **Scatter, Lines, Area, Bars, Histogram, Hexabin, Pie** and many more.

We will get introduce to Machine Learning and Python programming which are the base of this project and then dataset used in the project.

We will practice from installing matplotlib to how it works and its nature.

How flexible matplotlib is and how we can use it to visualize the 2D data in different forms and at the end we will discuss the output, problem, result and discussion.

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my project guide Akansha ma’am ,Bhumika ma’am and the Management of Brillica services who gave me the golden opportunity to do this wonderful project on the topic ‘**Data Visualization Techniques in Matplotlib**’, which also helped me in doing a lot of researech and i came to know about so many new things, I am really thankfull to them.

Secondly I would also like to thank my parents and member of my family, who have always supported me morally as well as Financially.

At last gratitude goes to all my friends and batchmates who helped me a lot in finalizing this projet within the limited time and also to me my self for always being motivated and contineously acquiring knowledge.

Thank You

**(RAHUL SINGH)**

**CONTENTS**

**Pg.No**

DECLARATION 1

ABSTRACT 2

ACKNOWLEDGEMENT 3

CONTENTS 4

**CHAPTER: 1 INTRODUCTION 5**

Data Visualization and Matplotlib 5-7

**CHAPTER: 2 DATA VISUALIZATION TECHNIQUES IN MATPLOTLIB 8-30**

1. Types 5
2. Importing Matplotlib 9
3. Visualization of Data using Matplotlib 23-30

**CHAPTER: 3 MACHINE LEARNING 31-36**

1. Types 33
2. Application of Meachine Learning 34
3. Models 34-35

**CHAPTER: 4 LANGUAGE USED: PYTHON 37-40**

PROBLEM DEFINITION 41

ABOUT DATASET 41

**CHAPTER: 5 METHODOLOGY USED 42**

1. Exploratory Data Analysis (EDA) 42-45
2. Feature scaling 46-47
3. Data visualization 47

TRAINING THE MODEL 47-50

RESULT AND DISCUSSION 50

BIBLIOGRAPHY 51

-**CHAPTER 1**

**Introduction to Data Visualization and Matplotlib:**

Data Visualization is an important part of business activities as organizations nowadays collect a huge amount of data. Sensors all over the world are collecting climate data, user data through clicks, car data for prediction of steering wheels etc. All of these data collected hold key insights for businesses and visualizations make these insights easy to interpret.Data is only as good as it’s presented.

**Why are visualizations important?**

Visualizations are the easiest way to analyse and absorb information. Visuals help to easily understand the complex problem. They help in identifying patterns, relationships, and outliers in data. It helps in understanding business problems better and quickly. It helps to build a compelling story based on visuals. Insights gathered from the visuals help in building strategies for businesses. It is also a precursor to many high-level data analysis for Exploratory Data Analysis (EDA) and Machine Learning (ML).

Human beings are visual creatures. Countless studies show how our brain is wired for the visual, and processes everything faster when it is through the eye.

“Even if your role does not directly involve the nuts and bolts of data science, it is useful to know what data visualization can do and how it is realized in the real world.”- Ramie Jacobson

Data visualizations in python can be done via many packages. We’ll be discussing of matplotlib package. It can be used in Python scripts, Jupyter notebook, and web application servers.

There are many kinds of data visualizations, each of which serves a specific professional purpose. Some of the most popular techniques for conveying information are:

* Histograms
* Waterfall charts
* Area charts
* Scatter plots
* Infographics
* Maps
* Pie charts
* Bar charts
* Box-and-whisker plots
* Heat maps

Because we live in an increasingly visual culture, those who know how to present information in visually engaging stories have the power not only to help make sense of past events but to offer predictions for the future as well.

There are many data visualization tools and libraries currently available to help transform raw data into visual stories that convey actionable insights. This project will explore Matplotlib, a Python data visualization library, and the benefits and (few) drawbacks to using it at your workplace.

**What is Matplotlib?**

A good starting point for those who want to learn about Matplotlib is by taking a closer look at Python. Python is a high-level, object-oriented programming language whose straightforward syntax lends itself to readability. Because its basis is English syntax, Python is one of the easiest coding languages to learn. This multipurpose programming language is applicable to almost any situation that uses data, lines of code, or mathematical computations. It allows users to perform advanced data manipulations as well as numerical analysis by using data frames.

Matplotlib is a two-dimensional Python data visualization and plotting library. It was written by John Hunter in 2002. In Hunter’s words, “Matplotlib tries to make easy things easy and hard things possible.” This multi-platform library was created on NumPy arrays and was intended to work with the SciPy stack. It is used in Python and IPython shells, as well as web application servers and Jupyter Notebook.

Matplotlib allows users to write a single script that can be used for flexible data parsing and plotting. This free, open-source library supports many output types, which allows it to be used on any operating system. In addition, it’s helpful for modeling machine learning technologies.

Matplotlib is particularly suited to working with numerical information that needs to be visually conveyed. It is able to create publishable, high-quality graphs with much less effort than other data visualization tools. Matplotlib is used by Data Analysts around the world to design engaging and stunning figures, charts, and graphs. This extensive library can change even the most minute details of a figure to enhance the subsequent visualization.

Many companies and businesses use Matplotlib for their data visualization needs, such as Nordstrom, WellsFargo, and Cigna.

**How is Matplotlib Used by Analysts?**

Matplotlib has a wide range of uses for those working in data science or data analytics. The following are a few of its most popular features for Analysts looking to create visualizations based on data:

* Plots can be easily made using Matplotlib, since most plots are created by following the exact same steps.
* Matplotlib comes with several plot options. These allow users to identify patterns and trends, and to make correlations.
* This library includes an object-oriented API, which is useful for embedding plots into various applications.
* Matplotlib is useful for those who wish to create bar graphs to compare and contrast data in different categories or track changes during a given period of time.
* When working with Matplotlib’s scatter plots, it’s easy to spot outliers.
* In situations where numerical proportion must be communicated, Data Analysts can create pie charts using Matplotlib. These charts depict the proportions of a part to the whole.
* Matplotlib is a powerful tool for designing histograms, which are essential for counting the variables in a plot.
* Those who wish to monitor changes over time for multiple related groups can use Matplotlib to create area plots.

**Matplotlib Toolkits**

Although Matplotlib isn’t part of Python’s standard libraries, users can download toolkits such as the following to increase Matplotlib’s utility and functionality:

* Mplot3d helps users create three-dimensional plots.
* Basemap is a useful map plotting toolkit. It provides a variety of map projections, political boundaries, and even coastlines.
* Cartopy is a mapping library that includes object-oriented map projection definitions. It also offers arbitrary line, polygon, image, and point capabilities.
* Microsoft Excel tools allow Matplotlib to exchange data with Excel.
* Real-World Scenarios for Using Matplotlib
* Here are just a few of the real-world applications of Matplotlib:
* This library allows users to plot data from a website or database.
* Basemap can be used to plot geographical data in Matplotlib.
* Curve fitting allows users to plot extrapolated data.
* Data from CSV files can be plotted.
* The data extracted by parsing an Apache log file can be plotted using Matplotlib.

When used in concert with other Python libraries and data visualization tools, Matplotlib provides users with a powerful and helpful way to analyse and visualize data.

**CHAPTER 2**

**Data Visualization Techniques in Matplotlib:**

Matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

We will use the pip command to install this module



**!**pip install matplotlib

Requirement already satisfied: matplotlib in c:\users\shiv\anaconda3\lib\site-packages (3.4.3)

Requirement already satisfied: numpy>=1.16 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (1.20.3)

Requirement already satisfied: python-dateutil>=2.7 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (2.8.2)

Requirement already satisfied: cycler>=0.10 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (0.10.0)

Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (1.3.1)

Requirement already satisfied: pyparsing>=2.2.1 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (3.0.4)

Requirement already satisfied: pillow>=6.2.0 in c:\users\shiv\anaconda3\lib\site-packages (from matplotlib) (8.4.0)

Requirement already satisfied: six in c:\users\shiv\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib) (1.16.0)

**Pyplot**

Pyplot Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are Line Plot, Histogram, Scatter, 3D Plot, Image, Contour, and Polar.

In [3]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

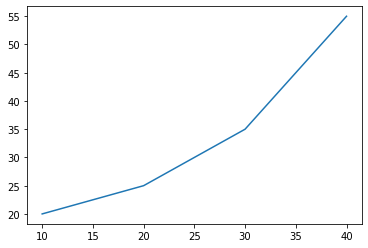
x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# plotting the data*

plt.plot(x, y)

plt.show()



Now let see how to add some basic elements like title, legends, labels to the graph.

**Adding Title**

The title() method in matplotlib module is used to specify the title of the visualization depicted and displays the title using various attributes.

Syntax:matplotlib.pyplot.title(label, fontdict=None, loc=’center’, pad=None, \*\*kwargs)

In [4]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

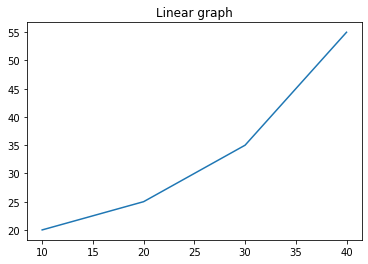
*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph")

plt.show()



We can also change the appearance of the title by using the parameters of this function.

In [7]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

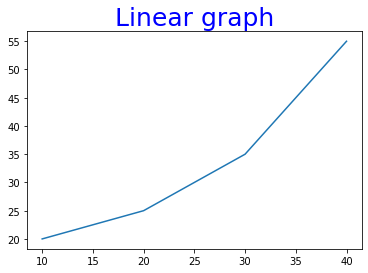
*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph", fontsize**=**25, color**=**"blue")

plt.show()



**Adding X Label and Y Label**



In layman’s terms, the X label and the Y label are the titles given to X-axis and Y-axis respectively. These can be added to the graph by using the xlabel() and ylabel() methods.

​

Syntax:

​

matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, **\*\*kwargs)**

In [9]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph", fontsize**=**25, color**=**"blue")

*# Adding label on the y-axis*

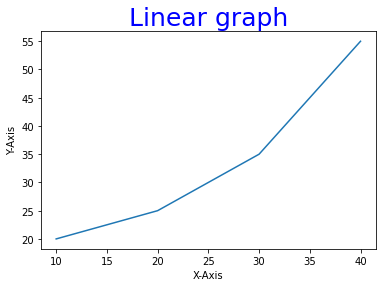
plt.ylabel('Y-Axis')

*# Adding label on the x-axis*

plt.xlabel('X-Axis')

plt.show()

​



**Setting Limits and Tick labels**

You might have seen that Matplotlib automatically sets the values and the markers(points) of the X and Y axis, however, it is possible to set the limit and markers manually. xlim() and ylim() functions are used to set the limits of the X-axis and Y-axis respectively. Similarly, xticks() and yticks() functions are used to set tick labels.

Example: In this example, we will be changing the limit of Y-axis and will be setting the labels for X-axis.

In [10]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph", fontsize**=**25, color**=**"blue")

*# Adding label on the y-axis*

plt.ylabel('Y-Axis')

*# Adding label on the x-axis*

plt.xlabel('X-Axis')

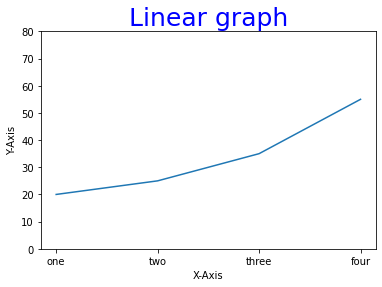
*# Setting the limit of y-axis*

plt.ylim(0, 80)

*# setting the labels of x-axis*

plt.xticks(x, labels**=**["one", "two", "three", "four"])

plt.show()



**Adding Legends**

A legend is an area describing the elements of the graph. In simple terms, it reflects the data displayed in the graph’s Y-axis. It generally appears as the box containing a small sample of each color on the graph and a small description of what this data means. The attribute bbox\_to\_anchor=(x, y) of legend() function is used to specify the coordinates of the legend, and the attribute ncol represents the number of columns that the legend has. Its default value is 1.

Syntax:

matplotlib.pyplot.legend([“name1”, “name2”], bbox\_to\_anchor=(x, y), ncol=1)

In [13]:



*# import matplotlib.pyplot as plt*

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph", fontsize**=**25, color**=**"blue")

*# Adding label on the y-axis*

plt.ylabel('Y-Axis')

*# Adding label on the x-axis*

plt.xlabel('X-Axis')

*# Setting the limit of y-axis*

plt.ylim(0, 80)

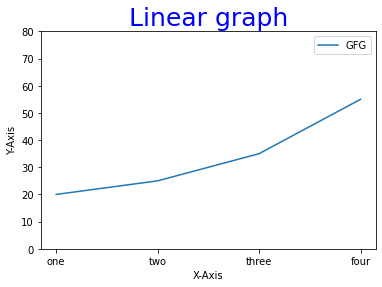
*# setting the labels of x-axis*

plt.xticks(x, labels**=**["one", "two", "three", "four"])

*# Adding legends*

plt.legend(["GFG"])

plt.show()



Before moving any further with Matplotlib let’s discuss some important classes that will be used further in the tutorial. These classes are –

Figure Axes Note: Matplotlib take care of the creation of inbuilt defaults like Figure and Axes.

**Figure class**

Consider the figure class as the overall window or page on which everything is drawn. It is a top-level container that contains one or more axes. A figure can be created using the figure() method.

Syntax:

class matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None, linewidth=0.0, frameon=None, subplotpars=None, tight\_layout=None, constrained\_layout=None)

In [15]:



*# Python program to show pyplot module*

**import** matplotlib.pyplot **as** plt

**from** matplotlib.figure **import** Figure

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# Creating a new figure with width = 7 inches*

*# and height = 5 inches with face color as*

*# green, edgecolor as red and the line width*

*# of the edge as 7*

fig **=** plt.figure(figsize **=**(7, 5), facecolor**=**'g',

edgecolor**=**'b', linewidth**=**7)

*# Creating a new axes for the figure*

ax **=** fig.add\_axes([1, 1, 1, 1])

*# Adding the data to be plotted*

ax.plot(x, y)

*# Adding title to the plot*

plt.title("Linear graph", fontsize**=**25, color**=**"yellow")

*# Adding label on the y-axis*

plt.ylabel('Y-Axis')

*# Adding label on the x-axis*

plt.xlabel('X-Axis')

*# Setting the limit of y-axis*

plt.ylim(0, 80)

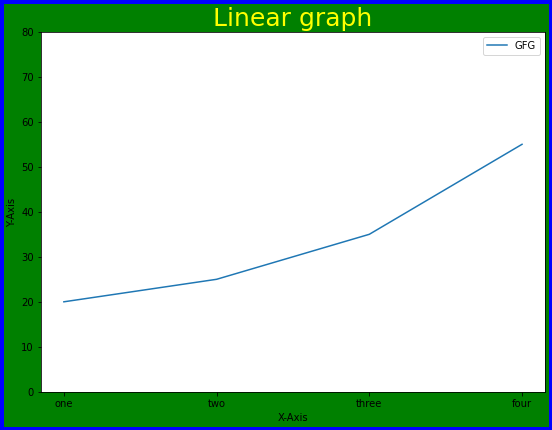
*# setting the labels of x-axis*

plt.xticks(x, labels**=**["one", "two", "three", "four"])

*# Adding legends*

plt.legend(["GFG"])

plt.show()



**Axes Class**

Axes class is the most basic and flexible unit for creating sub-plots. A given figure may contain many axes, but a given axes can only be present in one figure. The axes() function creates the axes object.

Syntax:

axes([left, bottom, width, height])

Just like pyplot class, axes class also provides methods for adding titles, legends, limits, labels, etc. Let’s see a few of them –

Adding Title – ax.set\_title() Adding X Label and Y label – ax.set\_xlabel(), ax.set\_ylabel() Setting Limits – ax.set\_xlim(), ax.set\_ylim() Tick labels – ax.set\_xticklabels(), ax.set\_yticklabels() Adding Legends – ax.legend()

In [17]:



*# Python program to show pyplot module*

**import** matplotlib.pyplot **as** plt

**from** matplotlib.figure **import** Figure

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

fig **=** plt.figure(figsize **=** (5, 4))

*# Adding the axes to the figure*

ax **=** fig.add\_axes([1, 1, 1, 1])

*# plotting 1st dataset to the figure*

ax1 **=** ax.plot(x, y)

*# plotting 2nd dataset to the figure*

ax2 **=** ax.plot(y, x)

*# Setting Title*

ax.set\_title("Linear Graph")

*# Setting Label*

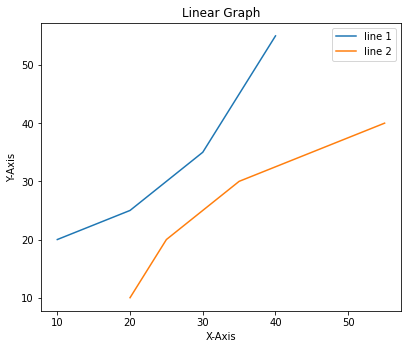
ax.set\_xlabel("X-Axis")

ax.set\_ylabel("Y-Axis")

*# Adding Legend*

ax.legend(labels **=** ('line 1', 'line 2'))

plt.show()



**Multiple Plots**

We have learned about the basic components of a graph that can be added so that it can convey more information. One method can be by calling the plot function again and again with a different set of values as shown in the above example. Now let’s see how to plot multiple graphs using some functions and also how to plot subplots.

Method 1: Using the add\_axes() method

The add\_axes() method is used to add axes to the figure. This is a method of figure class

Syntax:

add\_axes(self, *args, \**kwargs)

In [18]:



*# Python program to show pyplot module*

**import** matplotlib.pyplot **as** plt

**from** matplotlib.figure **import** Figure

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# Creating a new figure with width = 5 inches*

*# and height = 4 inches*

fig **=** plt.figure(figsize **=**(5, 4))

*# Creating first axes for the figure*

ax1 **=** fig.add\_axes([0.1, 0.1, 0.8, 0.8])

*# Creating second axes for the figure*

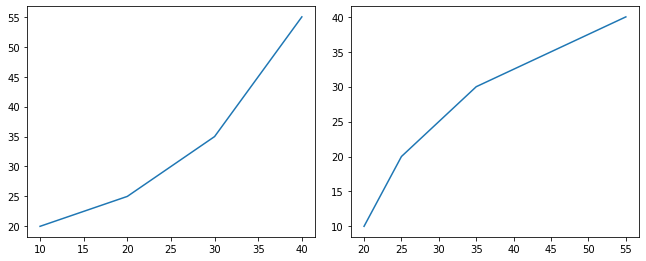
ax2 **=** fig.add\_axes([1, 0.1, 0.8, 0.8])

*# Adding the data to be plotted*

ax1.plot(x, y)

ax2.plot(y, x)

plt.show()



Method 2: Using subplot() method.

This method adds another plot at the specified grid position in the current figure.

Syntax:

subplot(nrows, ncols, index, \*\*kwargs)

subplot(pos, \*\*kwargs)

subplot(ax)

In [19]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# Creating figure object*

plt.figure()

*# addind first subplot*

plt.subplot(121)

plt.plot(x, y)

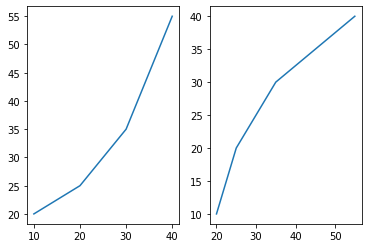
*# addding second subplot*

plt.subplot(122)

plt.plot(y, x)

Out[19]:

[<matplotlib.lines.Line2D at 0x20784611be0>]



Method 3: Using subplots() method

This function is used to create figures and multiple subplots at the same time.

Syntax:

matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot\_kw=None, gridspec\_kw=None, \*\*fig\_kw)

In [20]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# Creating the figure and subplots*

*# according the argument passed*

fig, axes **=** plt.subplots(1, 2)

*# plotting the data in the*

*# 1st subplot*

axes[0].plot(x, y)

*# plotting the data in the 1st*

*# subplot only*

axes[0].plot(y, x)

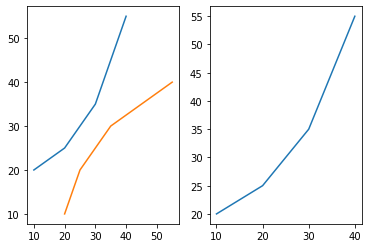
*# plotting the data in the 2nd*

*# subplot only*

axes[1].plot(x, y)

Out[20]:

[<matplotlib.lines.Line2D at 0x20783b96e20>]



Method 4: Using subplot2grid() method

This function creates axes object at a specified location inside a grid and also helps in spanning the axes object across multiple rows or columns. In simpler words, this function is used to create multiple charts within the same figure.

Syntax:

Plt.subplot2grid(shape, location, rowspan, colspan)

In [21]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# adding the subplots*

axes1 **=** plt.subplot2grid (

(7, 1), (0, 0), rowspan **=** 2, colspan **=** 1)

axes2 **=** plt.subplot2grid (

(7, 1), (2, 0), rowspan **=** 2, colspan **=** 1)

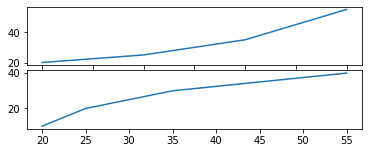
*# plotting the data*

axes1.plot(x, y)

axes2.plot(y, x)

Out[21]:

[<matplotlib.lines.Line2D at 0x207843832e0>]



In [29]:



**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

*# Reading the iris.csv file*

data **=** pd.read\_csv('iris.csv')

*# initializing the data*

x **=** data['sepal\_length']

y **=** data['petal\_width']

*# plotting the data*

plt.bar(x, y)

*# Adding title to the plot*

plt.title("iris Dataset")

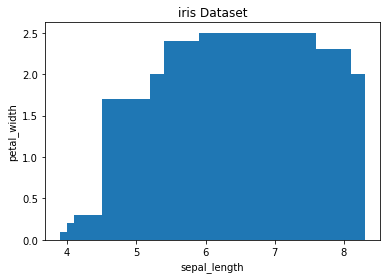
*# Adding label on the y-axis*

plt.ylabel('petal\_width')

*# Adding label on the x-axis*

plt.xlabel('sepal\_length')

plt.show()



**Different types of Matplotlib Plots**

Matplotlib supports a variety of plots including line charts, bar charts, histograms, scatter plots, etc. We will discuss the most commonly used charts in this project with the help of some good examples and will also see how to customize each plot.

Note: Some elements like axis, color are common to each plot whereas some elements are pot specific.

**Line Chart**



Line chart is one of the basic plots and can be created using the plot() function. It is used to represent a relationship between two data X and Y on a different axis.

​

Syntax:

​

matplotlib.pyplot.plot(\\*args, scalex=True, scaley=True, data=None, \\*\\*kwargs)

In [24]:



**import** matplotlib.pyplot **as** plt

*# initializing the data*

x **=** [10, 20, 30, 40]

y **=** [20, 25, 35, 55]

*# plotting the data*

plt.plot(x, y)

*# Adding title to the plot*

plt.title("Line Chart")

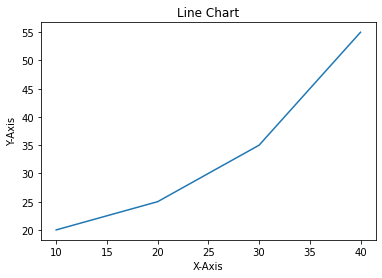
*# Adding label on the y-axis*

plt.ylabel('Y-Axis')

*# Adding label on the x-axis*

plt.xlabel('X-Axis')

plt.show()



**Bar Chart**

A 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. It can be created using the bar() method.

In the below example, we will use the iris dataset.

**Histogram**

A histogram is basically used to represent data provided in a form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. The hist() function is used to compute and create histogram of x.

Syntax:

matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype=’bar’, align=’mid’, orientation=’vertical’, rwidth=None, log=False, color=None, label=None, stacked=False, \*, data=None, \*\*kwargs)

In [31]:



**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

*# Reading the iris.csv file*

data **=** pd.read\_csv('iris.csv')

*# initializing the data*

x **=** data['sepal\_length']

*# plotting the data*

plt.hist(x)

*# Adding title to the plot*

plt.title("iris Dataset")

*# Adding label on the y-axis*

plt.ylabel('Frequency')

*# Adding label on the x-axis*

plt.xlabel('sepal\_length')

plt.show()



**Scatter Plot**

Scatter plots are used to observe relationships between variables. The scatter() method in the matplotlib library is used to draw a scatter plot.

Syntax:

matplotlib.pyplot.scatter(x\_axis\_data, y\_axis\_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None

In [36]:



​

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

*# Reading the iris.csv file*

data **=** pd.read\_csv('iris.csv')

*# initializing the data*

x **=** data['sepal\_length']

y **=** data['petal\_width']

*# plotting the data*

plt.scatter(x, y)

*# Adding title to the plot*

plt.title("iris Dataset")

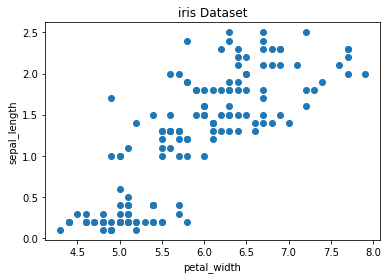
*# Adding label on the y-axis*

plt.ylabel('sepal\_length')

*# Adding label on the x-axis*

plt.xlabel('petal\_width')

plt.show()



**Pie Chart**

Pie chart is a circular chart used to display only one series of data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. It can be created using the pie() method.

Syntax:

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

For ploting the piechart we here we are using another data 'tips' as to get better understanding.

In [39]:



**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

*# Reading the heart.csv file*

data **=** pd.read\_csv('heart.csv')

*# initializing the data*

sex **=** ['male', 'female']

data **=** [70,30]

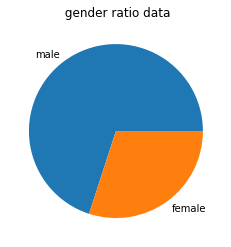
*# plotting the data*

plt.pie(data, labels**=**sex)

*# Adding title to the plot*

plt.title(" gender ratio data")

plt.show()



In [40]:



**import** matplotlib.pyplot **as** plt

*# Creating data*

year **=** ['2010', '2002', '2004', '2006', '2008']

production **=** [25, 15, 35, 30, 10]

*# Plotting barchart*

plt.bar(year, production)

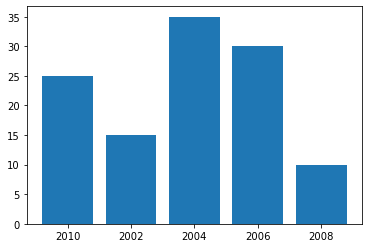
*# Saving the figure.*

plt.savefig("output.jpg")

*# Saving figure by changing parameter values*

plt.savefig("output1", facecolor**=**'y', bbox\_inches**=**"tight",

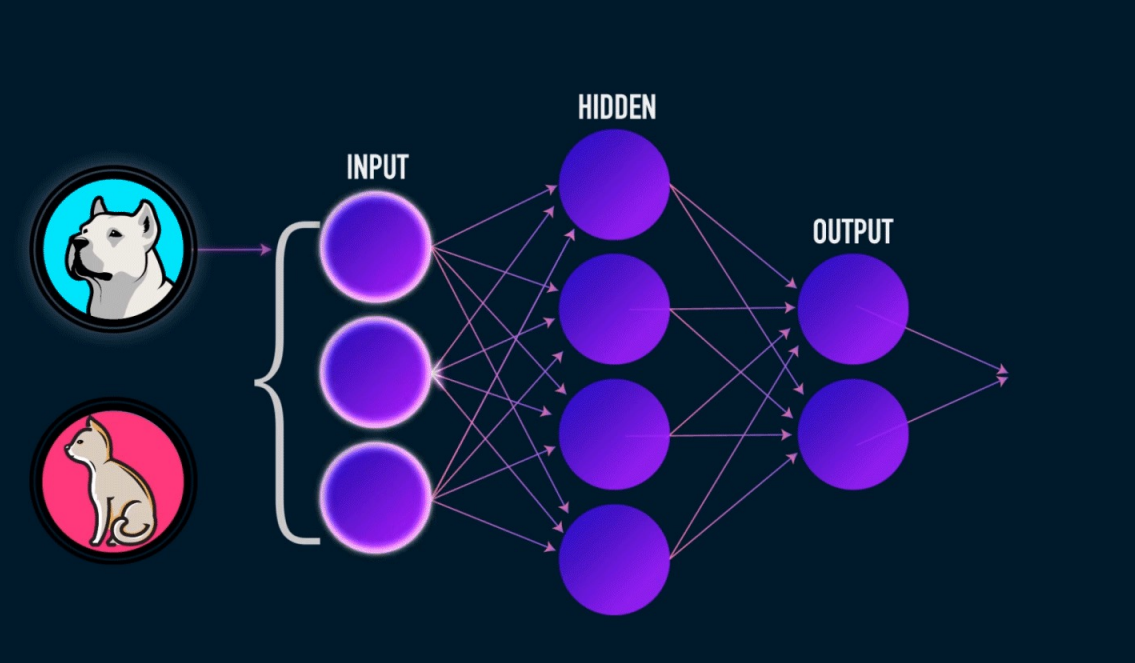
pad\_inches**=**0.3, transparent**=True**)



* **CHAPTER 3**

**MACHINE LEARNING**

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

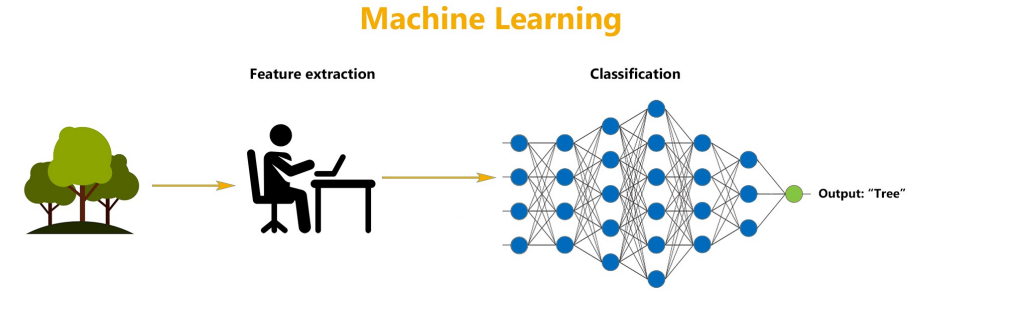


Need for Machine Learning Lack of human expertise The very first scenario in which we want a machine to learn and take data- driven decisions, can be the domain where there is a lack of human expertise. The examples can be navigations in unknown territories or spatial planets. Dynamic scenarios There are some scenarios which are dynamic in nature i.e. they keep changing over time. In case of these scenarios and behaviours, we want a machine to learn and take data-driven decisions. Some of the examples can be network connectivity and availability of infrastructure in an organization. Difficulty in translating expertise into computational tasks There can be various domains in which humans have their expertise; however, they are unable to translate this expertise into computational tasks. In such circumstances we want machine learning. The examples can be the domains of speech recognition, cognitive tasks etc.

**Definition**

❑ Professor Mitchell defines ML as - “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

❑ The main components of any learning algorithm are - Task(T), Performance(P) and experience (E). In this context, we can simplify this definition as, ML is a field of AI consisting of learning algorithms that − Improve their performance (P) At executing some task (T) Over time with experience (E).

  
**How Machine Learning works?**

* A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it.
* The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately**.**

**Features of ML**

* Machine learning uses data to detect various patterns in a given dataset.
* It can learn from past data and improve automatically.
* It is a data-driven technology.
* Machine learning is much similar to data mining as it also deals with the huge amount of the data.

**Need of ML**

■ We can train machine learning algorithms by providing them the huge amount of data and let them explore the data, construct the models, and predict the required output automatically.

■ The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

■ The importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in self-driving cars, cyber fraud detection, face recognition, and friend suggestion by Facebook, etc.

■ Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyse the user interest and recommend product accordingly.

Following are some key points which show the importance of Machine Learning:

– Rapid increment in the production of data

– Solving complex problems, which are difficult for a human

– Decision making in various sector including finance

– Finding hidden patterns and extracting useful information from data.

**Classification of ML algorithms**

At broad level, ML algorithms can be classified as –

■ Supervised Learning

■ Unsupervised Learning

■ Reinforcement Learning

**Supervised Learning**

* Supervised learning is a type of machine learning method in which we provide sample labelled data to the machine learning system in order to train it, and on that basis, it predicts the output.
* The goal of supervised learning is to map input data with the output data.
* The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is spam filtering.
* Supervised learning can be grouped further in two categories of algorithms: – Classification – Regression

**Unsupervised Learning**

* Unsupervised learning is a learning method in which a machine learns without any supervision
* The training is provided to the machine with the set of data that has not been labelled , classified, or categorized, and the algorithm needs to act on that data without any supervision.
* The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.
* In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data.
* It can be further classifieds into two categories of algorithms: – Clustering – Association

**Reinforced Learning**

* Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action.
* The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it.
* The goal of an agent is to get the most reward points, and hence, it improves its performance.
* The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning

**Applications of ML**

Some of the popular real-world applications are as follows -

* Emotion analysis
* Error detection and prevention
* Stock market analysis and forecasting
* Speech recognition
* Customer segmentation
* Object recognition
* Fraud detection
* Recommendation of products to customer in online shopping

**ML Models**

Alternatively, we can break down machine learning models into five types. This approach gives a more specific and in-depth look at machine learning characteristics.

* **Classification Models**

Classification predicts the class or type of an object according to a finite number of options. The classification output variable is always a category. For example, is this email spam or not?

* **Regression Models**

Regression is a problem set where output variables can assume continuous values. For example, predicting the per barrel price of oil on the commodity market is a standard regression task. Regression models get further split into:

o   Decision Trees

o   Random Forests

o   Linear Regression

* **Clustering**

This model involves gathering similar objects into groups. This process helps identify similar objects automatically without human intervention. Effective supervised machine learning models, including models that need to be trained with labeled or manually curated data, need homogeneous data, and clustering provides a smarter way to do it.

* **Dimensionality Reduction**

Sometimes, the number of possible variables in real-world data sets is too high, which leads to problems. Not all those countless variables even contribute significantly to the goal. Thus, we turn to dimensionality reduction, which preserves variances with a smaller number of variables.

* **Deep Learning**

This machine learning type involves neural networks. Neural networks are networks of mathematical equations. The network takes input variables, runs them through the equations, and produces output variables. The most significant deep learning models are:

* Autoencoders
* Boltzmann Machine
* Convolution Neural Networks
* Multi-layer perceptron
* Recurrent Neural Networks

**How to Build a Machine Learning Model**

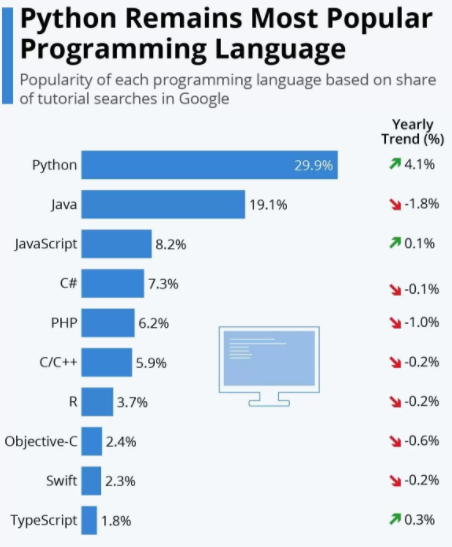
There are seven steps to building a good machine learning model.

1. Understand the business problem and what constitutes success. You need to understand a problem before you can fix it. This understanding involves working with the project owner and establishing the requirements and objectives. Then, figure out what parts of the business objective need a machine learning solution and how do you know when you’ve succeeded.
2. Understand the data and identify it. Machine learning models rely on clean, plentiful training data to learn. Figure out what kinds of data you need and if it’s in good enough shape for the project. It would help establish where the data comes from, how much you need, and its condition. Furthermore, you must understand how and if the machine learning model will work with real-time data.
3. Collect and prepare your data. Now that you know your data sources, you need to process the data into something suitable for machine learning training. This process includes collecting the data from its many sources, standardizing it, finding and replacing erroneous information, removing duplicate and extraneous information, and dividing the data into training, test, and validation sets.
4. Train your model. Now comes the fun part. You must train your model to learn from the good quality data you’ve collected and processed. This step involves choosing a model technique, model training, selecting algorithms, and model optimization. Consult the machine learning model types mentioned above for your options.
5. Evaluate the model’s performance and set up benchmarks. This step is analogous to the quality assurance aspect of application development. You must evaluate your model’s performance against the established requirements and metrics, which in turn determines how well you can expect it to work in the real world.
6. Try out the model and make sure it performs as expected. This step is alternately known as operationalizing the model. Next, deploy the model in a way that you can continually measure and monitor its performance. Cloud environments are ideal for this. Next, develop benchmarks that you can use to measure future iterations of your model. Then, continuously iterate your model’s various aspects to improve its overall performance.
7. Keep adjusting and iterating your model. Keep monitoring and improving your model. After all, technologies advance and change, business requirements evolve, and the real world occasionally throws a wrench into things. Any of these factors could potentially mean new requirements. Keep improving the model’s accuracy and performance. Think of your machine learning model as a mobile app. The application will always need tweaking, updating, and improving. The same thing applies to your machine learning model.

**CHAPTER 4**

**Language used :Python**

Python was officially born on February 20, 1991, with version number 0.9.0 and has taken a tremendous growth path to become the most popular language for the last 5 years in a row (2012 to 2016). Its application cuts across various areas such as website development, mobile apps development, scientific and numeric computing, desktop GUI, and complex software development. Even though Python is a more general-purpose programming and scripting language, it has been gaining popularity over the past 5 years among data scientists and Machine Learning engineers.

****

There are well-designed development environments such as IPython Notebook and Spyder that allow for a quick introspection of the data and enable developing of machine learning models interactively. Powerful modules such as NumPy and Pandas exist for the efficient use of numeric data. Scientific computing is made easy with SciPy package. A number of primary machine learning algorithms have been efficiently implemented in scikit-learn (also known as sklearn). HadooPy, PySpark provides seamless work experience with big data technology stacks. Cython and Numba modules allow executing Python code in par with the speed of C code. Modules such as nosetest emphasize high-quality, continuous integration tests, and automatic deployment. Combining all of the above has made many machine learning engineers embrace Python as the choice of language to explore data, identify patterns, and build and deploy models to the production environment. Most importantly the business-friendly licenses for various key Python packages are encouraging the collaboration of businesses and the open source community for the benefit of both worlds. Overall the Python programming ecosystem allows for quick results and happy programmers. We have been seeing the trend of developers being part of the open source community to contribute to the bug fixes and new algorithms for the use by the global community, at the same time protecting the core IP of the respective company they work for

**Python 2.7.x or Python 3.4.x?**

Python 3.4.x is the latest version and comes with nicer, consistent functionalities! However, there is very limited third-party module support for it, and this will be the trend for at least a couple of more years. However, all major frameworks still run on version 2.7.x and are likely to continue to do so for a significant amount of time. Therefore, it is advised to start with Python 2, for the fact that it is the most widely used version for building machine learning systems as of today

**Indentation**

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents its semantic structure. This feature is sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule). Some other languages use indentation this way; but in most, indentation has no semantic meaning. The recommended indent size is four spaces.

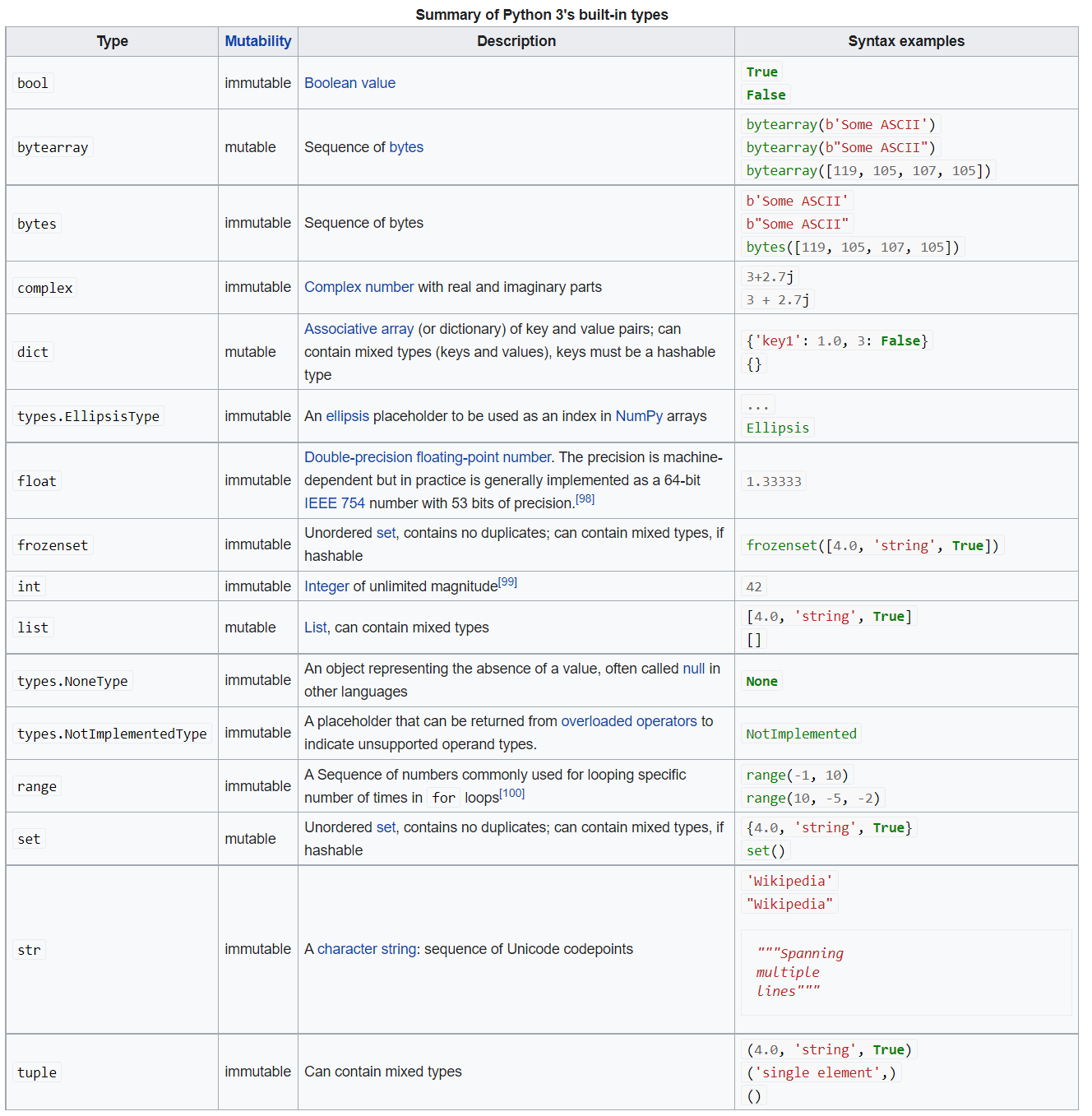
**Statements and control flow**

Python's [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) include:

* The [assignment](https://en.wikipedia.org/wiki/Assignment_(computer_science)) statement, using a single equals sign =
* The [if](https://en.wikipedia.org/wiki/If-then-else) statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if)
* The [for](https://en.wikipedia.org/wiki/Foreach#Python) statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block
* The [while](https://en.wikipedia.org/wiki/While_loop#Python) statement, which executes a block of code as long as its condition is true
* The [try](https://en.wikipedia.org/wiki/Exception_handling_syntax#Python) statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block is always run regardless of how the block exits
* The raise statement, used to raise a specified exception or re-raise a caught exception
* The class statement, which executes a block of code and attaches its local namespace to a [class](https://en.wikipedia.org/wiki/Class_(computer_science)), for use in object-oriented programming
* The def statement, which defines a [function](https://en.wikipedia.org/wiki/Function_(computing)) or [method](https://en.wikipedia.org/wiki/Method_(computing))
* The [with](https://en.wikipedia.org/wiki/Dispose_pattern#Language_constructs) statement, which encloses a code block within a context manager (for example, acquiring a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) before it is run, then releasing the lock; or opening and closing a [file](https://en.wikipedia.org/wiki/Computer_file)), allowing [resource-acquisition-is-initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization) (RAII)-like behavior and replacing a common try/finally idiom
* The [break](https://en.wikipedia.org/wiki/Break_statement) statement, which exits a loop
* The continue statement, which skips the current iteration and continues with the next
* The del statement, which removes a variable—deleting the reference from the name to the value, and producing an error if the variable is referred to before it is redefined
* The pass statement, serving as a [NOP](https://en.wikipedia.org/wiki/NOP_(code)), syntactically needed to create an empty code block
* The [assert](https://en.wikipedia.org/wiki/Assertion_(programming)) statement, used in debugging to check for conditions that should apply
* The yield statement, which returns a value from a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)#Python) function (and also an operator); used to implement [coroutines](https://en.wikipedia.org/wiki/Coroutine)
* The return statement, used to return a value from a function

**Arithmetic operations**

Python has the usual symbols for arithmetic operators (+, -, \*, /), the floor division operator // and the modulo operation % (where the remainder can be negative, e.g. 4 % -3 == -2). It also has \*\* for exponentiation, e.g. 5\*\*3 == 125 and 9\*\*0.5 == 3.0, and a matrix‑multiplication operator @ . These operators work like in traditional math; with the same precedence rules, the operators infix (+ and - can also be unary to represent positive and negative numbers respectively).



**First Code**

Hello world program:

print('Hello, world!')

**Problem Definition**

This study has been conducted purely to understand Machine learning Language python and how we visualize data in matplotlib.

The study is restricted to four types of data visualization linear, bar, pie chart, histogram

The study is limited to the Programming language Python

Detailed study of the topic was not possible due to limited size of the project.

Suggestions and conclusions are based on mix of the past, current & limited data.

**About dataset**

A data set (or dataset) is a collection of [data](https://en.wikipedia.org/wiki/Data). In the case of tabular data, a data set corresponds to one or more [database tables](https://en.wikipedia.org/wiki/Table_(database)), where every [column](https://en.wikipedia.org/wiki/Column_(database)) of a table represents a particular [variable](https://en.wikipedia.org/wiki/Variable_(computer_science)), and each [row](https://en.wikipedia.org/wiki/Row_(database)) corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Data sets can also consist of a collection of documents or files.

**Dataset used in project:**

Iris data set- iris.csv

Heart data set- heart.csv

**CHAPTER 5**

**METHODOLOGY USED**

**Observational**

Working of machine learning how a raw input data gives an analysed output in terms of picture recorgnition.

**Experimental**

Visualized an original dataset in terms of bars and charts fer better understanding.

**Simulation**

Used scatter plot, subplot, multiplot

**Derived**

Successfully installed and imported matplotlib and worked on it with python programming.

* 1. **Exploratory Data Analysis (EDA)**

EDA is all about understanding your data by employing summarizing and visualizing techniques. At a high level the EDA can be performed in two folds, that is, univariate analysis and multivariate analysis.

Let’s learn and consider an example dataset to learn practicality. Iris dataset is one of a well-known datasets used extensively in pattern recognition literature. It is hosted at UC Irvine Machine Learning Repository. The dataset contains petal length, petal width, sepal length, and sepal width measurement for three types of iris flowers, that is, setosa, versicolor, and virginica.



**Univariate Analysis**

Individual variables are analysed in isolation to have a better understanding about them. Pandas provide the describe function to create summary statistics in tabular format for all variables. These statistics are very useful for numerical types of variables to understand any quality issues such as missing values and the presence of outliers. See Listings **3-4 and 3-5**

**Listing 3-4. Univariate analysis**

from sklearn import datasets

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

iris = datasets.load\_iris()

# Let's convert to dataframe

iris = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],

columns= iris['feature\_names'] + ['species'])

# replace the values with class labels

iris.species = np.where(iris.species == 0.0, 'setosa', np.where(iris.

species==1.0,'versicolor', 'virginica'))

# let's remove spaces from column name

iris.columns = iris.columns.str.replace(' ','')

iris.describe()

**#----output----**

sepallength(cm)sepalwidth(cm)petallength(cm) petalwidth(cm)

Count 150.00 150.00

150.00 150.00

Mean5.84 3.05 3.75 1.19

std 0.82 0.43 1.76 0.76

min 4.30 2.00 1.00 0.10

25% 5.10 2.80 1.60 0.30

50% 5.80 3.00 4.35 1.30

75% 6.40 3.30 5.10 1.80

max 7.90 4.40 6.90 2.50

**The columns ‘species’ is categorical, so lets check the frequency distribution for each**

**category.**

print iris['species'].value\_counts()

**#----output----**

Setosa 50

versicolor 50

virginica 50

Pandas supports plotting functions to quick visualization on attributes. We can see

from the plot that 'species' has 3 category with 50 records each.

**Listing 3-5. Pandas dataframe visualization**

# Set the size of the plot

plt.figsize(15, 8)

iris.hist() # plot histogram

plt.suptitle("Histogram", fontsize=16) # use suptitle to add title to all sublots

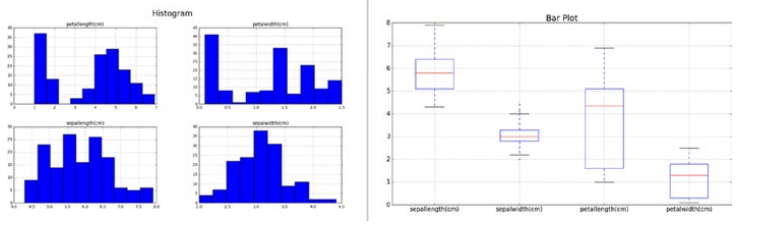
plt.show()

iris.boxplot() # plot boxplot

plt.title("Bar Plot", fontsize=16)

plt.show()

**#----output----**



**Multivariate Analysis**

In multivariate analysis you try to establish a sense of relationship of all variables with

one other.

Let’s understand the mean of each feature by species type. **See Listing 3-6.**

**Listing 3-6. Multivariate analysis**

# print the mean for each column by species

iris.groupby(by = "species").mean()

# plot for mean of each feature for each label class

iris.groupby(by = "species").mean().plot(kind="bar")

plt.title('Class vs Measurements')

plt.ylabel('mean measurement(cm)')

plt.xticks(rotation=0) # manage the xticks rotation

plt.grid(True)

# Use bbox\_to\_anchor option to place the legend outside plot area to be tidy

plt.legend(loc="upper left", bbox\_to\_anchor=(1,1))

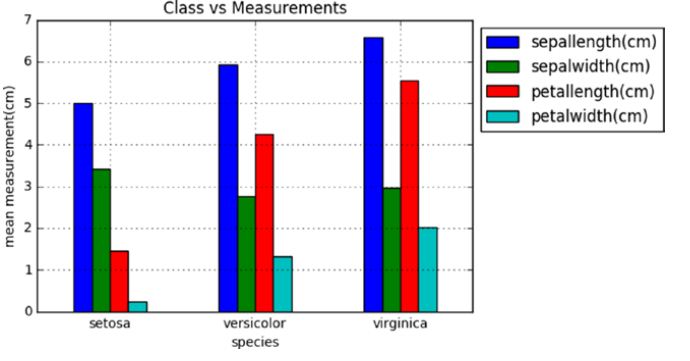
**#----output----**

sepallength(cm) sepalwidth(cm) petallength(cm) petalwidth(cm)

setosa 5.006 3.418 1.464 0.244

versicolor 5.936 2.770 4.260 1.326

virginica 6.588 2.974 5.552 2.026



* 1. **Feature Scaling** is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing.

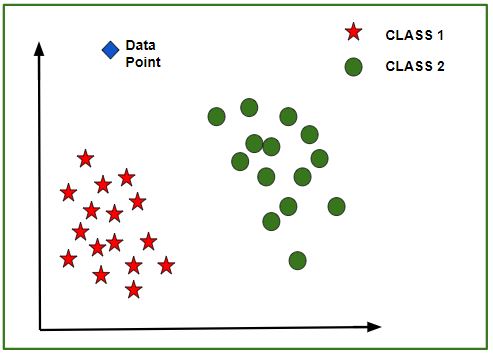
**Working:**  
Given a data-set with features- *Age*, *Salary*, *BHK Apartment* with the data size of 5000 people, each having these independent data features.

Each data point is labeled as:

* **Class1- YES** (means with the given *Age*, *Salary*, *BHK Apartment* feature value one can buy the property)
* **Class2- NO** (means with the given *Age*, *Salary*, *BHK Apartment* feature value one can’t buy the property).

Using a dataset to train the model, one aims to build a model that can predict whether one can buy a property or not with given feature values.

Once the model is trained, an N-dimensional (where N is the no. of features present in the dataset) graph with data points from the given dataset, can be created. The figure given below is an ideal representation of the model.



As shown in the figure, star data points belong to **Class1 – Yes** and circles represent **Class2 – No** labels, and the model gets trained using these data points. Now a new data point (diamond as shown in the figure) is given and it has different independent values for the 3 features (*Age*, *Salary*, *BHK Apartment*) mentioned above. The model has to predict whether this data point belongs to Yes or No.

**Prediction of the class of new data points:**   
The model calculates the distance of this data point from the centroid of each class group. Finally, this data point will belong to that class, which will have a minimum centroid distance from it.   
The distance can be calculated between centroid and data point using these methods-

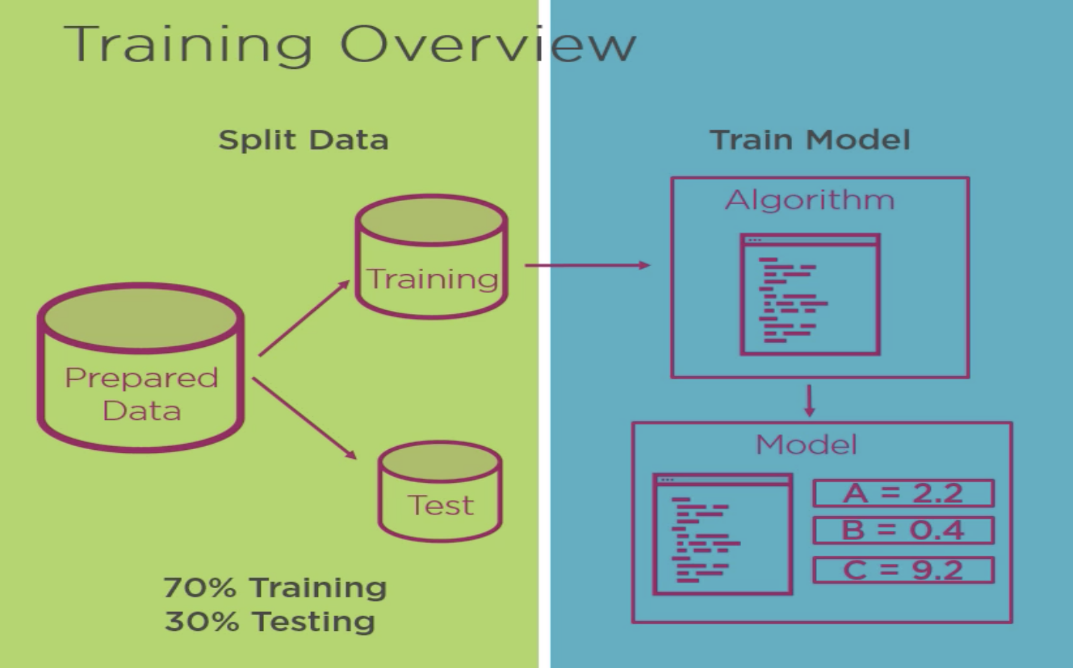
* **Euclidean Distance:** It is the square root of the sum of squares of differences between the coordinates (feature values – *Age*, *Salary*, *BHK Apartment*) of data point and centroid of each class. This formula is given by the Pythagorean theorem.   
  where x is Data Point value, y is Centroid value and k is no. of feature values, Example: given data set has k = 3
* **Manhattan Distance:** It is calculated as the sum of absolute differences between the coordinates (feature values) of data point and centroid of each class.
* **Minkowski Distance:** It is a generalization of the above two methods. As shown in the figure, different values can be used for finding r.
  1. **Data Visualization** is an important part of business activities as organizations nowadays collect a huge amount of data. Sensors all over the world are collecting climate data, user data through clicks, car data for prediction of steering wheels etc. All of these data collected hold key insights for businesses and visualizations make these insights easy to interpret.Data is only as good as it’s presented.

**Training model**

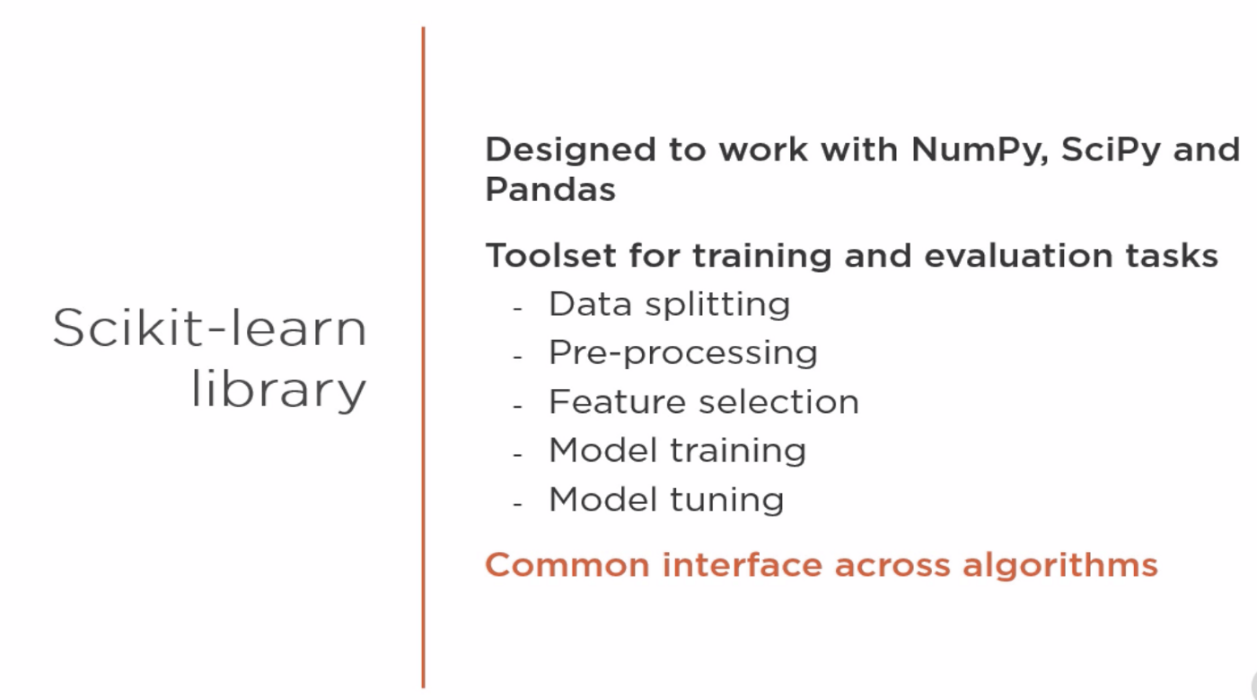
here algorithm is “**Naive Bayes**” and data is the “**prepared data**” which we have already selected in the previous articles.

So, we are going to provide the “**prepared data**” to the “**Naive Bayes**” algorithm to get the trained model in the end.

And we are going to split the “**prepared data**” into two parts: 70% training data and rest 30% testing data. The reason because the data we have is all related to real world entities and hence, we need some part of data to be available for testing the model so that it can make the accurate prediction about the people will develop the diabetes.



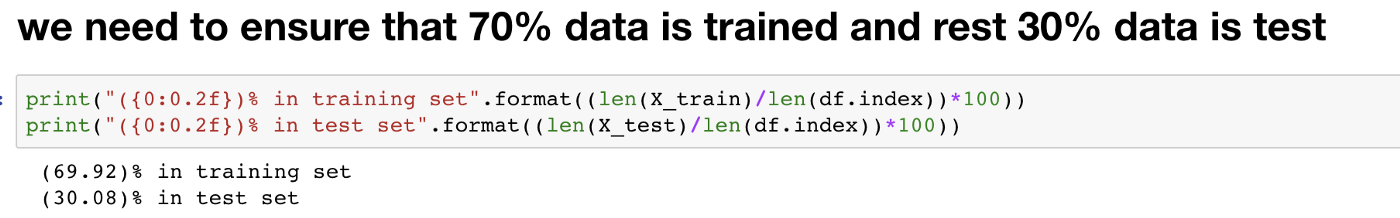
And in python we have the scikit-learn package to get the trained model:





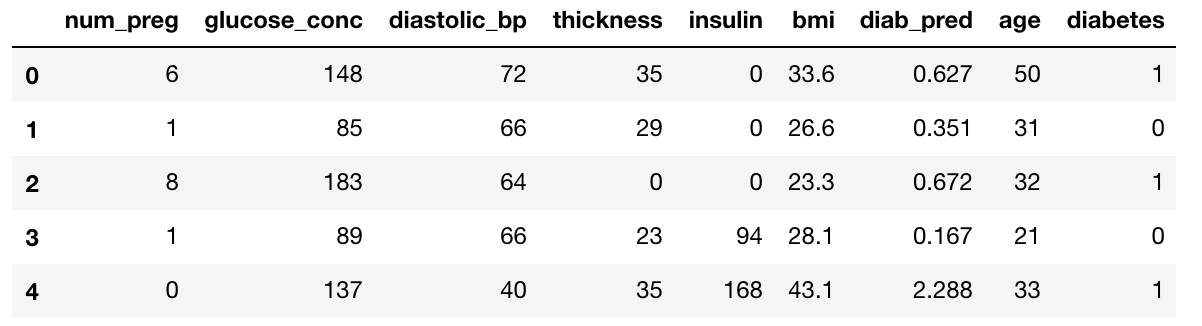
In the above code we have taken the columns in our data and defining predicted class which has two values 0 for not diabetes and 1 for diabetes. We have defined split test size = 0.30 just because we need 30% data for testing and random\_state = 42 because to pass any numerical value to indicate we need to split the data.

And to ensure the split:



Now, we have the data divided. Let’s run the code.

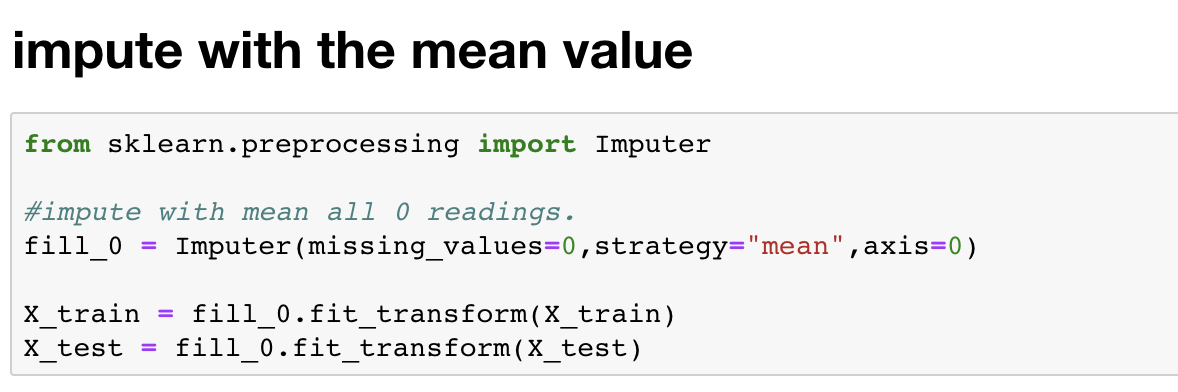
df.head()



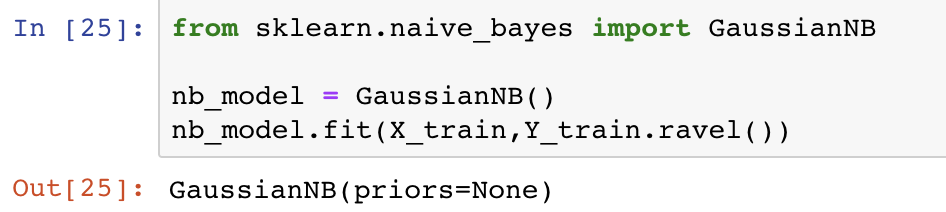
Above we can see the so many zero values for the columns such as thickness, num\_preg and insulin and these can impact the accuracy of our model. So, we have the following options:

* Ignore the values.
* Drop the observations (rows).
* Replace the values (impute).

But, we will go with the last option because we can’t delete the missing or zero values which can a great bias in the need. So, we going to replace the values with the mean of the values of that particular column and we call this operation **Imputing**and in python we have **Imputer**class for the same.



Finally, we have reached the last step to get the trained model by importing the **Naive Bayes**using python.



In the output above, we have the trained model which we will test it in the next article and finally will use it to make the prediction.

**Result and Discussion**

We have found that how easy it is to work using matplotlib with python programming.

We got to knew that python is the most popular programming language.

Working behind machine learning .

How we train models through images.

Deep learning is

* advanced
* Requires large data
* Takes longer to train

Whereas Machine gives

* lesser accuracy
* Takes less time to train
* Limited tuning capabilities

**Bibliography**

Mastering Machine Learning **Author: Manohar Swamynathan**

Kaggle

GIThub

Wikki

GeeksforGeeks

IBM

Anaconda

Jupyter

THANK YOU.