# Department of Computer Science & Engineering

# Lab Manual

**Subject Name:**

**Machine Learning I Lab**

**Subject Code:**

**17BTIS612/17MIIA612**

**Class: -** **Branch: -**

**Prepared By: - Nagesh Jadhav**

**Year: - Third Year-Computer**

**Contributors: - Nagesh Jadhav**

**Examinations: - CA (40) and Final Exam (60)**

**Required H/W and S/W: - 64 bit processor based machine and Anaconda Navigator, Python**

# Syllabus

| Course Code | Course Title | | | | Category |
|---|---|---|---|---|---|
| 17BTIS612/17 MIIA612 | Programming Lab-IV | | | | Core |
| Contact Hours per Week | | | CA | FE | Credits |
| L | T | D/P | | | |
| 0 | 0 | 4 | 40 | 60 | 2 |
| Prerequisite: Linear algebra, calculus, Python | | | | | |

**Course Objectives:**

- Use of Data sets in implementing the machine learning algorithms
- Implement the machine learning concepts and algorithms in any suitable language of choice.
- Develop machine learning and soft computing models
- Students will be explored to the interconnection and integration of the physical world and the Cloud services for IOT. And they able to design & develop IOT Devices for given application.

## A. Machine Learning

1. A. Installation and Configuration of machine learning environment with Anaconda on windows or Ubuntu (Jupyter notebook, spyder, python, pycharm)
   B. Installation, Confguration and checking the current Version of numpy, scipy, scikit, pandas and matplot lib using anaconda prompt and list their uses in machine learning.

2. Download the any dataset from UCI or Data.org or from any other data repositories and perform the basic data pre-processing steps using python/R .

3. Download the any dataset from UCI or Data.org or from any other data repositories and Implement Single and multilayer perceptron on a dataset.
   A. MLP Classifier
   B. MLP Regressor
   Display the loss graph for the same. Also generate classification report in terms of confusion matrix, precision, recall, and f1 score.

4. Develop a Bayesian classifier IRIS dataset dataset

5. Prepare the Correlated dataset of your own example or Download the any one of the UCI ML data Set in which find the correlated data and Find the best fit line for the data.

6. Using inbuilt dataset of Breast cancer from scikit learn Implement PCA algorithm.

7. Implement decision tree classification/regression technique for given dataset
   Developed model should be able to answer the given queries.
8. Implement SVM Classifier or Regression for given dataset
9. Implement K means algorithm for multidimensional data
10. Download the famous dataset of iris and Implement KNN algorithm to predict the class to which these plants belong, Calculate the performance matric and compare the error rate with K value( K value range 1 and).
11. Implement CNN for MNIST/CIFAR10 dataset using tensorflow
12. Perform basic image processing using opencv

---

**B. Internet of Things (Elective –I)**

1. A. Prepare Case study on Arduino Raspberry Pi and Beagle Bone Black.
   B. Study and Install and Configure IDE of Arduino or Raspberry Pi or BBB
2. A. Write program using Arduino or Raspberry Pi IDE for Blink LED
   B. Interface the analog/digital temperature sensor using Arduino or Raspberry pi to Measure the current temperature
3. Study and Implement RFID, NFC using Arduino or Raspberry PI.
4. Study and Implement Zigbee Protocol using Raspberry Pi.
5. Study and Implement MQTT Protocol using Arduino or Raspberry PI.

**C: Soft Computing (Elective –I)**

1. Compute the new membership values for fuzzy set C based on the fuzzy set A and B using specified Fuzzy Operations: Union, intersection, complement and disjunctive sum. Set A (providing details about colors) and B (providing details related texture) are given below: A= {(Red, 0.4), (Green, 0.6), (Blue, 0.8)} B= {(Homogeneity, 0.5), (contrast, 0.63), (correlation, 0.86)}
2. Implement fuzzy c-means algorithm and rough k-means algorithm for library data and compare results based on multiple measures like accuracy, precision, recall.
3. Perform experiment using GA for bus driver scheduling.
4. Implement PSO algorithm for Wireless sensor network application.

---

**Course Outcomes:**

- Understand the implementation procedures for the machine learning algorithms.
- Design programs for various Learning algorithms.
- Apply appropriate data sets to the Machine Learning algorithms.
- Identify and apply Machine Learning algorithms to solve real world problems
- Understand and Use of Various development boards, Arduino, Raspberry Pi and BBB for IOT application.
- Ability to develop programs using Arduino IDE and Arduino Board/ Raspberry Pi for various IOT Application.

---

**References**

- Ethem Alpaydin, ―Introduction to Machine Learning 3e (Adaptive Computation and Machine Learning Series)‖, Third Edition, MIT Press, 2014
- Stephen Marsland, ―Machine Learning – An Algorithmic Perspective‖, Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
- Aurélien Géron- Hands-On Machine Learning with Scikit-Learn & TensorFlow, O'reilly, First edition, 2017
- Colin Dow - Internet of Things Programming Projects_ Build modern IoT solutions with the Raspberry Pi 3 and Python-Packt Publishing (2018)
- (Learning path) Cox, Tim_ Fernandes, Steven Lawrence_ Vaish, Diwakar_ Yamanoor, Sai_ Yamanoor, Srihari - Getting started with Python for the Internet of Things _ leverage the full potential of Python

## ASSIGNMENT 1

## Problem Statement-

Installation and Configuration of machine learning environment with Anaconda on windows or Ubuntu (Jupyter notebook)

## Objective

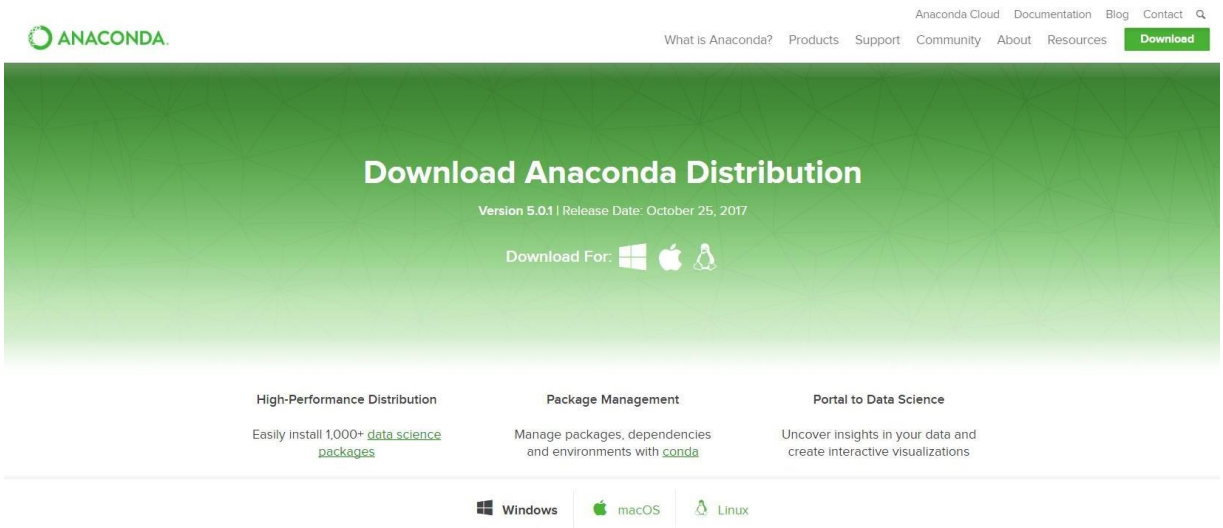
To install anaconda and configure it with Python

## Theory

Installation Of Anaconda in Windows

1. Visit the Anaconda downloads page

Go to the following link: Anaconda.com/downloads

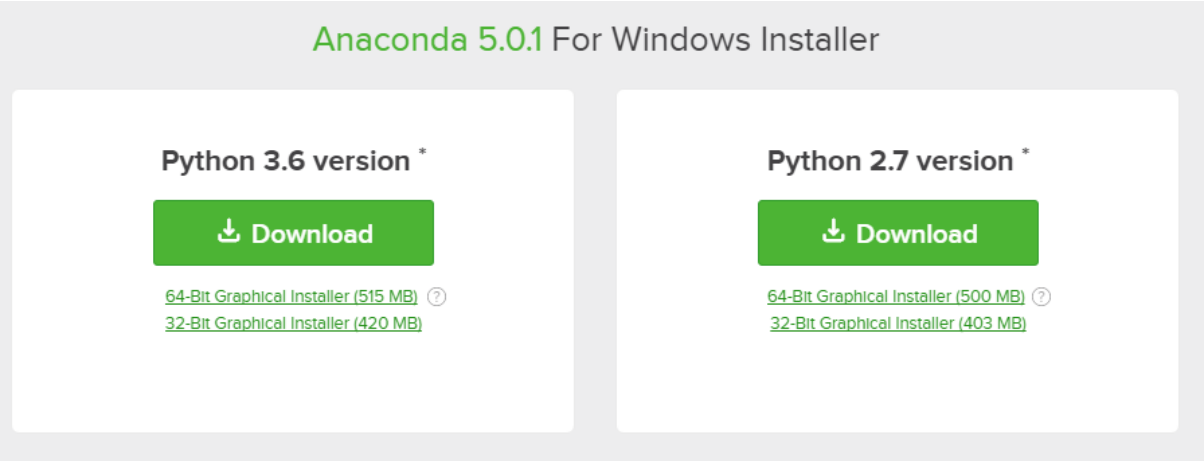



2. Select Windows

Select Windows where the three operating systems are listed.

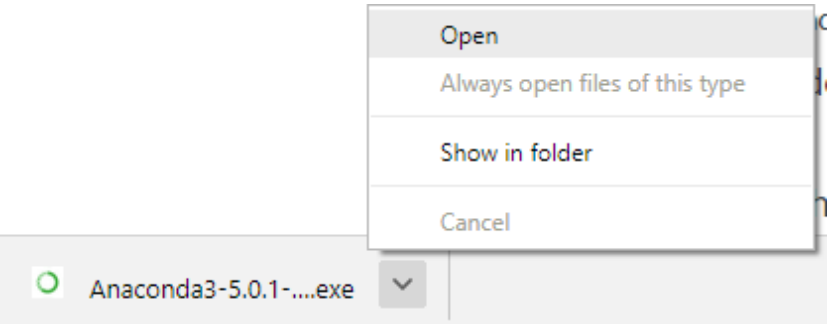
3. Download

Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.

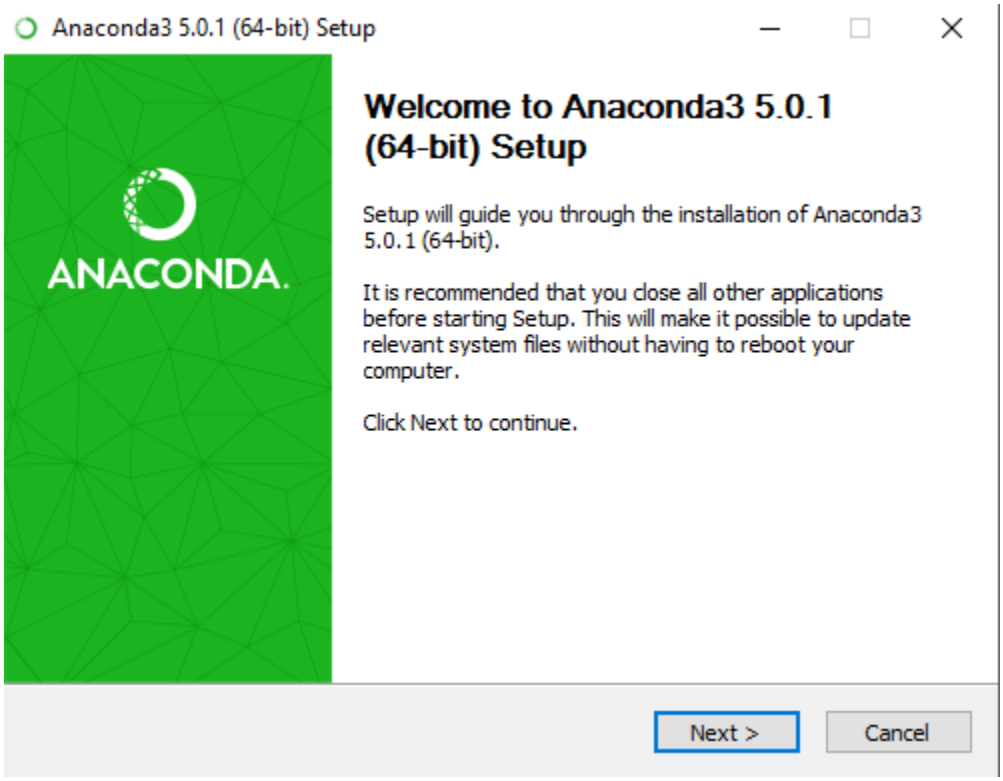



4. Open and run the installer

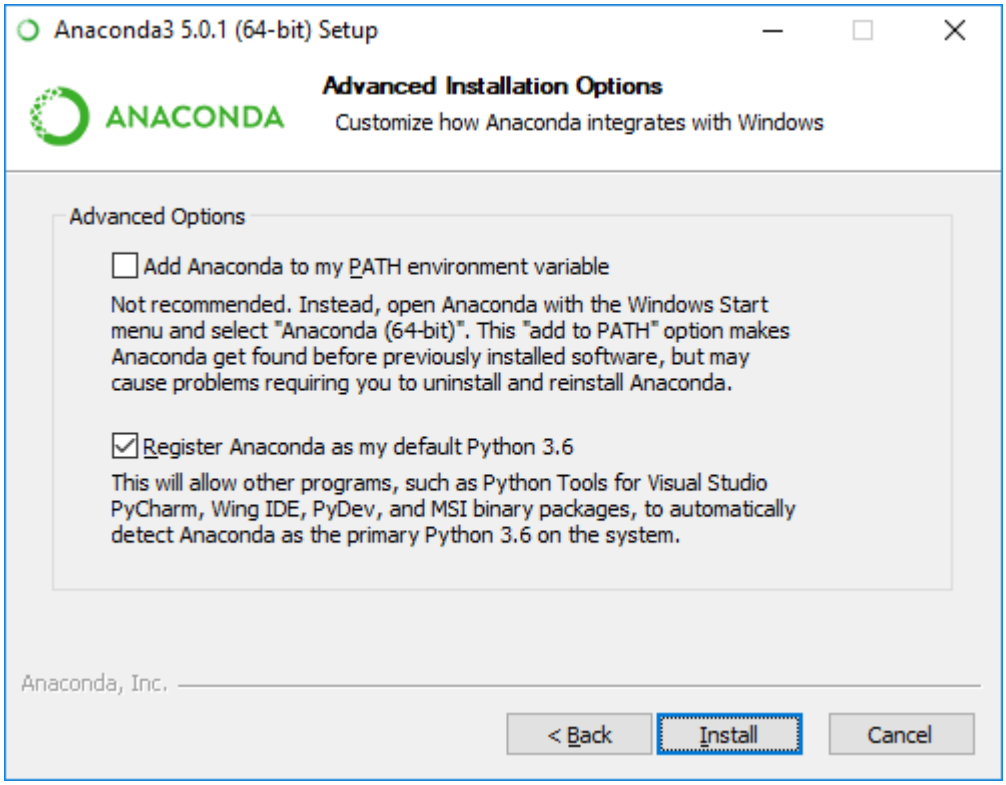
Once the download completes, open and run the ***.exe*** installer



At the beginning of the install, you need to click **Next** to confirm the installation.



At the Advanced Installation Options screen, I recommend that you **do not check** "Add Anaconda to my PATH environment variable"



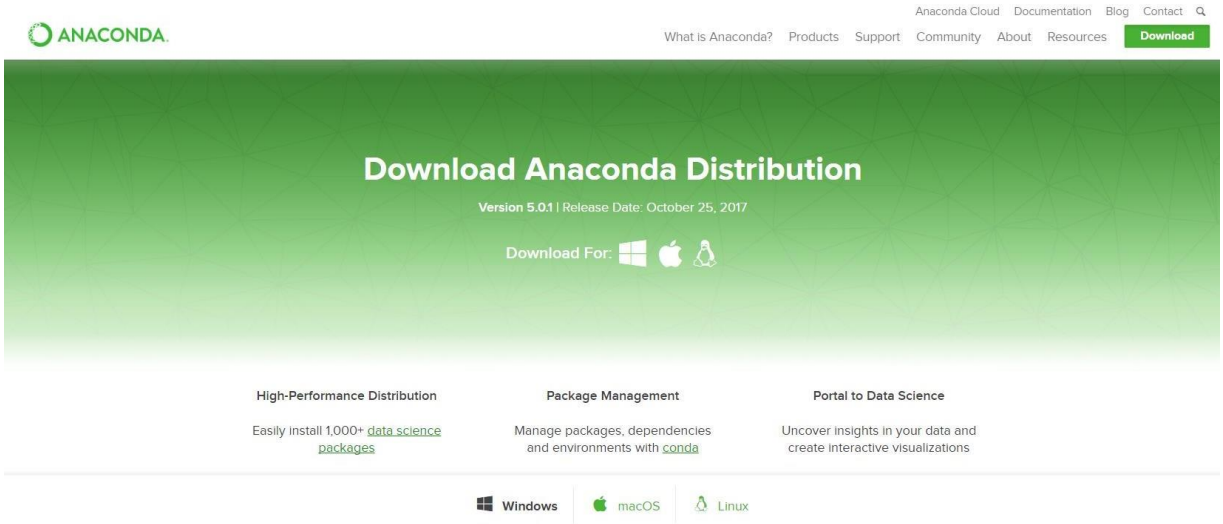

Installation Of Anaconda in Ubuntu

This section details the installation of the Anaconda distribution of Python on Linux, specifically Ubuntu 18.04, but the instructions should work for other Debian-based Linux distributions as well.

Ubuntu 18.04 comes pre-installed with Python (Version 3.6) and legacy Python (Version 2.7). You can confirm the legacy version of Python is installed by opening up a terminal.

1. Visit the Anaconda downloads page

Go to the following link: Anaconda.com/downloads

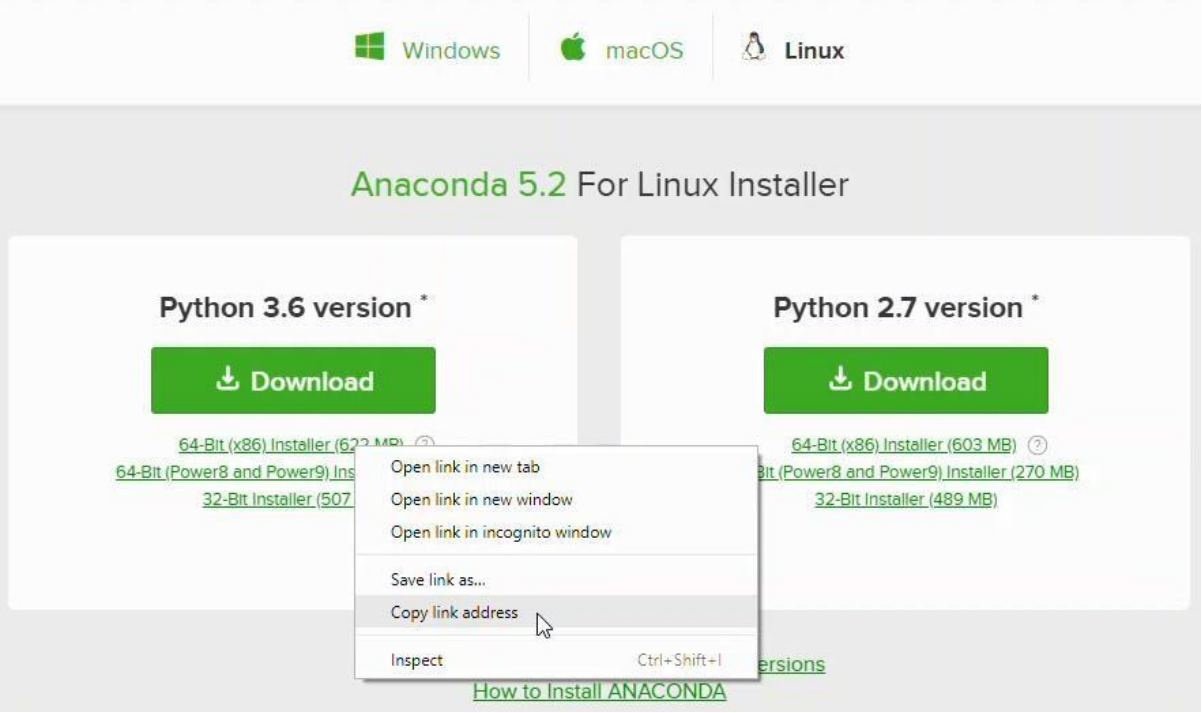



2. Select Linux

On the downloads page, select the Linux operating system.

3. Copy the bash (.sh file) installer link

In the **Python 3.6 Version*** box, right-click on the [64-Bit(x86) Installer] link. Select [copy link address].




4. Use wget to download the bash installer

Now that the bash installer (.sh file) link is stored on the clipboard, use wget to download the installer script. In a terminal, cd into the home directory and make a new directory called tmp. cd into tmp and use wget to download the installer. Although the installer is a bash script, it is still quite large and the download will not be immediate (Note the link below includes <release>. the specific release depends on when you download the installer)

```
$ cd ~
$ mkdir tmp
$ cd tmp
$ https://repo.continuum.io/archive/Anaconda3<release>.sh
```

**5.** Run the bash script to install **Anaconda3**

With the bash installer script downloaded, run the ***.sh*** script to install **Anaconda3**. Ensure you are in the directory where the installer script downloaded:

```
$ ls
Anaconda3-5.2.0-Linux-x86_64.sh
```

Run the installer script with bash.

```
$ bash Anaconda3-5.2.0-Linux-x86_64.sh
```

Accept the Licence Agreement and allow Anaconda to be added to your PATH. By adding Anaconda to your PATH, the Anaconda distribution of Python will be called when you type $ python in a terminal.

6. source the .bash-rc file to add Anaconda to your PATH

Now that **Anaconda3** is installed and **Anaconda3** is added to our PATH, source the .bashrc file to load the new PATH environment variable into the current terminal session. Note the .bashrc file is in the home directory. You can see it with $ ls -a.

```
$ cd ~
$ source .bashrc
```

Installation Of NumPy

Command – pip3 install numpy

To check the version

Import numpy as np

Installation of Scipy-stack

Command – pip3 install scipy-stack

NUMPY/SCIPY:-

NumPy is a Python library, which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc.

- NumPy array can also be used as an efficient multi-dimensional container for generic .data.
- The ndarray (NumPy Array) is a multidimensional array used to store values of same datatype. These arrays are indexed just like Sequences, starts with zero.
- The ndarrays are better than regular arrays in terms of faster computation and ease of manipulation.
- In different algorithms of Machine Learning like K-means Clustering, Random Forest etc. we have to store the values in an array. So, instead of using regular array, ndarray helps us to manipulate and execute easily.

Installation of scikit

Command pip3 install -u scikit-learn

SCIKIT:-

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization.

Installation of Pandas

Command- pip3 install pandas

PANDAS:-

- Merging and Joining Data Sets.
- Reshaping & pivoting Data Sets.
- Inserting & deleting columns in Data Structure.
- Aligning data & dealing with missing data.
- Iterating over a Data set.
- Analyzing Time Series.
- Filtering Data around a condition.
- Arranging Data in an ascending & descending.
- Reading from flies with CSV, TXT, XLSX, other formats.
- Manipulating Data using integrated indexing for DataFrame objects.
- Generating Data range, date shifting, lagging, converting frequency, and other other Time Series functionality.
- Subsettting fancy indexing, & label based slicing Data Sets that are large in size.
- Performing split apply combine on Data Sets using the group by engine.

With Python Pandas, it is easier to clean & wrangle with your Data. features of Pandas make it a great choice for Data Science and Analysis.

Installation of Matplotlib

Command- pip3 install matplotlib

MATPLOTLIB:-

- Matplotlib is a visualization library in Python for 2D plots of arrays. It consists of several plots like line, bar, scatter, histogram etc.
- Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.It can also be used with graphics toolkits like PyQt and wxPython.
- One of the advantage of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals.

### Importing and Checking the version of libraries:

```
>>> import pandas
>>> pandas.__version__
'1.3.3'
>>> import sklearn
>>> sklearn.__version__
'1.0.2'
>>> import numpy
>>> numpy.__version__
'1.20.2'
>>> import matplotlib
>>> matplotlib.__version__
'3.5.1'
>>>
```

### Conclusion

Thus we have successfully installed Anaconda on windows and Ubuntu; We also have described the libraries and the installation of the libraries.

## Assignment 2

Aim: Download any dataset from UCI or Data.org or from any data repositories and perform the basic data pre-processing steps using Python/R

Objectives:

1. Learn to pre-process dataset
2. Learn to use pandas and sklearn

### Theory

#### Basic steps

**Step 1 :** Import the libraries

**Step 2 :** Import the data-set

**Step 3 :** Check out the missing values

**Step 4 :** See the Categorical Values

**Step 5 :** Splitting the data-set into Training and Test Set

### Data cleaning:

The main aim of Data Cleaning is to identify and remove errors & duplicate data, in order to create a reliable dataset. This improves the quality of the training data for analytics and enables accurate decision-making.

Needless to say, data cleansing is a time-consuming process and most data scientists spend an enormous amount of time in enhancing the quality of the data. However, there are various methods to identify and classify data for data cleansing.

There are mainly two distinct techniques, namely Qualitative and Quantitative techniques to classify data errors. Qualitative techniques involve rules, constraints, and patterns to identify errors.

On the other hand, Quantitative techniques employ statistical techniques to identify errors in the trained data.

Normalisation:

**Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.**

Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

Standardisation:

**Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.**

Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

$\mu$ is the mean of the feature values and $\sigma$ is the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=
0)
```

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation
and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

```
Imputer(missing_values='NaN', strategy='mean', axis=0)
```

Imputation transformer for completing missing values.

pandas.read_csv()

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

**Program and Output:**

1) Importing the libraries, reading the dataset file:

## Import the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

## Reading the dataset

```
df = pd.read_csv('breast-cancer.data')
df.head()
```

| | no-recurrence-events | 30-39 | premeno | 30-34 | 0-2 | no | 3 | left | left_low | no.1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | no-recurrence-events | 40-49 | premeno | 20-24 | 0-2 | no | 2 | right | right_up | no |
| **1** | no-recurrence-events | 40-49 | premeno | 20-24 | 0-2 | no | 2 | left | left_low | no |
| **2** | no-recurrence-events | 60-69 | ge40 | 15-19 | 0-2 | no | 2 | right | left_up | no |
| **3** | no-recurrence-events | 40-49 | premeno | 0-4 | 0-2 | no | 2 | right | right_low | no |
| **4** | no-recurrence-events | 60-69 | ge40 | 15-19 | 0-2 | no | 2 | left | left_low | no |

2) Encoding of data and slicing of dataset:

## LabelEncoding of data

```
df2 = df.apply(LabelEncoder().fit_transform)
df2.head()
```

| | no-recurrence-events | 30-39 | premeno | 30-34 | 0-2 | no | 3 | left | left_low | no.1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 2 | 2 | 3 | 0 | 1 | 1 | 1 | 5 | 0 |
| **1** | 0 | 2 | 2 | 3 | 0 | 1 | 1 | 0 | 2 | 0 |
| **2** | 0 | 4 | 0 | 2 | 0 | 1 | 1 | 1 | 3 | 0 |
| **3** | 0 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 4 | 0 |
| **4** | 0 | 4 | 0 | 2 | 0 | 1 | 1 | 0 | 2 | 0 |

## Slicing the dataset into input and output

```
x = df2.iloc[:, 0:9]
y = df2.iloc[:, 9]
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
280    0
281    1
282    0
283    0
```

3) Splitting of data into training and testing and Checking for null values:

## Splitting the dataset for training and testing

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,rando
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(228, 9)
(57, 9)
(228,)
(57,)
```

## Checking for null values

```
print(df.isnull().sum())
```

```
no-recurrence-events    0
30-39                   0
premeno                 0
30-34                   0
0-2                     0
no                      0
3                       0
left                    0
left_low                0
no.1                    0
dtype: int64
```

Conclusion:

Thus we have successfully implemented pre-processing operations on a dataset

**Assignment 3**

**Aim:** Download the any dataset from UCI or Data.org or from any other data repositories and Implement Single and multilayer perceptron on a dataset.

**Objectives:**

1. To learn about classification and regression

2. To learn MLP and backpropagation

3. To demonstrate and analyse the results

Theory:

Regression:

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.

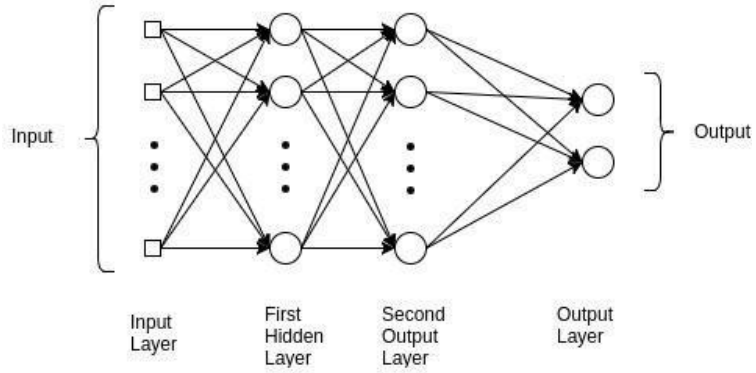
Classification:

A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. For example, when filtering emails “spam” or “not spam”, when looking at transaction data, “fraudulent”, or “authorized”.

Multilayer Perceptron:

In the Multilayer perceptron, there can more than one linear layer (combinations of **neurons**). If we take the simple example the three-layer network, first layer will be the *input layer* and last will be *output layer* and middle layer will be called *hidden layer.* We feed our input data into the input layer and take the

output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.




BackPropagation Algorithm:

**The algorithm is used to effectively train a neural network through a method called chain rule.** In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

In other words, **backpropagation aims to minimize the cost function by adjusting network's weights and biases.** The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

Activation function

Activation functions also known non- linearity, describe the input-output relations in a non-linear way. This gives the model power to be more flexible in describing arbitrary relations. Here are some popular activation functions Sigmoid, Relu, and TanH. I will describe these in my next blog.

**Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

**Recall** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it's above 0.5.

Recall = TP/TP+FN

**F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. In our case, F1 score is 0.701.
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Confusion matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

| | *Class 1 Predicted* | *Class 2 Predicted* |
|---|---|---|
| **Class 1 Actual** | **TP** | **FN** |
| **Class 2 Actual** | **FP** | **TN** |

Code and output:

MLP Classifier:

## Import the libraries

```
import pandas as pd
import numpy as np
from sklearn import preprocessing,datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
```

## Creating a dataframe

```
df = pd.read_csv("breast-cancer.data")
df
```

| | no-recurrence-events | 30-39 | premeno | 30-34 | 0-2 | no | 3 | left | left_low | no.1 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | no-recurrence-events | 40-49 | premeno | 20-24 | 0-2 | no | 2 | right | right_up | no |
| 1 | no-recurrence-events | 40-49 | premeno | 20-24 | 0-2 | no | 2 | left | left_low | no |
| 2 | no-recurrence-events | 60-69 | ge40 | 15-19 | 0-2 | no | 2 | right | left_up | no |
| 3 | no-recurrence-events | 40-49 | premeno | 0-4 | 0-2 | no | 2 | right | right_low | no |
| 4 | no-recurrence-events | 60-69 | ge40 | 15-19 | 0-2 | no | 2 | left | left_low | no |

## Encoding labels

```
df2 = df.apply(LabelEncoder().fit_transform)
df2
```

| | no-recurrence-events | 30-39 | premeno | 30-34 | 0-2 | no | 3 | left | left_low | no.1 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | 2 | 3 | 0 | 1 | 1 | 1 | 5 | 0 |
| 1 | 0 | 2 | 2 | 3 | 0 | 1 | 1 | 0 | 2 | 0 |
| 2 | 0 | 4 | 0 | 2 | 0 | 1 | 1 | 1 | 3 | 0 |
| 3 | 0 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 4 | 0 |
| 4 | 0 | 4 | 0 | 2 | 0 | 1 | 1 | 0 | 2 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 280 | 1 | 1 | 2 | 5 | 0 | 1 | 1 | 0 | 3 | 0 |
| 281 | 1 | 1 | 2 | 3 | 0 | 1 | 2 | 0 | 3 | 1 |
| 282 | 1 | 4 | 0 | 3 | 0 | 1 | 0 | 1 | 3 | 0 |
| 283 | 1 | 2 | 0 | 5 | 4 | 1 | 2 | 0 | 2 | 0 |
| 284 | 1 | 3 | 0 | 5 | 4 | 1 | 2 | 0 | 2 | 0 |

285 rows × 10 columns

## Slicing the dataset into input and output

```
x = df2.iloc[:, 1:9]
x.head()
```

| | 30-39 | premeno | 30-34 | 0-2 | no | 3 | left | left_low |
|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 2 | 3 | 0 | 1 | 1 | 1 | 5 |
| **1** | 2 | 2 | 3 | 0 | 1 | 1 | 0 | 2 |
| **2** | 4 | 0 | 2 | 0 | 1 | 1 | 1 | 3 |
| **3** | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 4 |
| **4** | 4 | 0 | 2 | 0 | 1 | 1 | 0 | 2 |

```
y = df2.iloc[:, 9]
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
280    0
281    1
282    0
283    0
284    0
Name: no.1, Length: 285, dtype: int32
```

## Splitting the dataset for training and testing

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(199, 8)
(86, 8)
(199,)
(86,)
```

## Fitting the model

```
model = MLPClassifier(random_state=0, max_iter=2000).fit(x_train, y_train)
```

# Printing model score for testing and training data

```
print(model.score(x_train, y_train))
```

```
0.9798994974874372
```

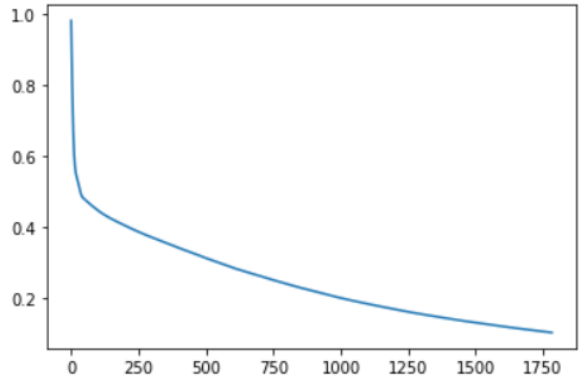
```
print(model.score(x_test, y_test))
```

```
0.7209302325581395
```

```
prd_r = model.predict(x_test)
test_acc = accuracy_score(y_test, prd_r) * 100.
loss_values = model.loss_curve_
```

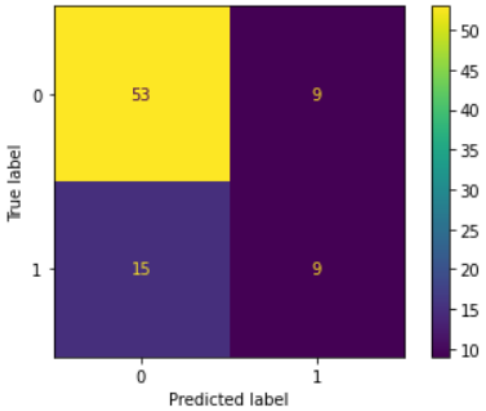
## plotting loss curve

```
import matplotlib.pyplot as plt

plt.plot(loss_values)
plt.show()
```




## plotting Confusion Matrix

```
import matplotlib.pyplot as plt
predictions = model.predict(x_test)
cm = confusion_matrix(y_test, predictions, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
disp.plot()

plt.show()
```



MLP Regressor:

## Import the libraries

```
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

## Creating the dataframe

```
df = pd.read_csv('housing.csv')
df.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | medi |
|---|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | |
| 1 | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | |
| 2 | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | |
| 3 | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | |
| 4 | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | |

## Cleaning the dataset

```
df_clean = df.dropna(axis=1, how="any")
```

```
df_clean.shape
```

```
(20640, 9)
```

```
df_clean.head()
```

| | longitude | latitude | housing_median_age | total_rooms |
|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41.0 | 880.0 |
| 1 | -122.22 | 37.86 | 21.0 | 7099.0 |
| 2 | -122.24 | 37.85 | 52.0 | 1467.0 |
| 3 | -122.25 | 37.85 | 52.0 | 1274.0 |
| 4 | -122.25 | 37.85 | 52.0 | 1627.0 |

```
df_clean2 = df.dropna(how="any")
```

```
df_clean2.shape
```

```
(20433, 10)
```

```
df_clean2.head()
```

```
df_clean2.head()
```

| | longitude | latitude | housing_median_age | total_rooms |
|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 |

```
x = df_clean2.iloc[:, :8]
```

```
y = df_clean2["median_house_value"].values
```

## Standardisation of Dataset

```
obj = StandardScaler()
x_ = obj.fit_transform(x)
```

## Splitting the dataset for training and testing

```
x_train, x_test, y_train, y_test = train_test_split(x_, y, test_size=0.2, random_state=47)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(16346, 8)
(4087, 8)
(16346,)
(4087,)
```

## Fitting the model

```
model = MLPRegressor(
    hidden_layer_sizes=(200,),
    activation="relu", solver="lbfgs",
    learning_rate="adaptive",
    verbose=True, max_iter=1500
)
```

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

```
D:\anaconda3\envs\tf\lib\site-packages\sklearn\neural_network\_multilayer_percept
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
MLPRegressor(hidden_layer_sizes=(200,), learning_rate='adaptive', max_iter=1500,
             solver='lbfgs', verbose=True)
```

## Printing the accuracy of model

```
print(model.score(x_train, y_train))
```

```
0.8366757437519187
```

```
print(model.score(x_test, y_test))
```

```
0.7958244331849722
```

## Conclusion

Thus we have successfully completed the implementation of Multilayer perceptron.

Assignment 4

Aim: Develop a Bayesian classifier on breast cancer dataset

**Objectives**

1. To learn bayes theorem

2. To implement Bayesian classifier

Theory

Bayes Theorem

Bayes' Theorem is a way of finding a probability when we know certain other probabilities.

The formula is:

P(A|B) = *P(A) P(B|A)***P(B)**

Which tells us: how often A happens *given that B happens*, written **P(A|B)**,
When we know: how often B happens *given that A happens*, written **P(B|A)**
and how likely A is on its own, written **P(A)**
and how likely B is on its own, written **P(B)**

Bayes Classifier with example

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models.[1] But they could be coupled with Kernel density estimation and achieve higher accuracy levels.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a

particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Code and Output:

## Import the libraries

```
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot
```

## Creating a dataframe

```
df = pd.read_csv('breast-cancer.csv')
df.head()
```

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | compactness_mean |
|---|---|---|---|---|---|---|---|---|
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 |
| 1 | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 |

5 rows × 32 columns

## Slicing the dataset into input and output

```
x = df.drop(['diagnosis'], axis=1)
y = df.iloc[:, 1]
y
```

```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

## Splitting the dataset for training and testing

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

## Standardization of dataset

```
scaler = StandardScaler()
x_train_std = scaler.fit_transform(x_train)
```

```
x_test_std = scaler.transform(x_test)
```

## Fitting the model

```
model = GaussianNB()
model.fit(x_train_std, y_train)
```

```
GaussianNB()
```

## Printing model score for testing and training data

```
print(model.score(x_train_std, y_train))
```

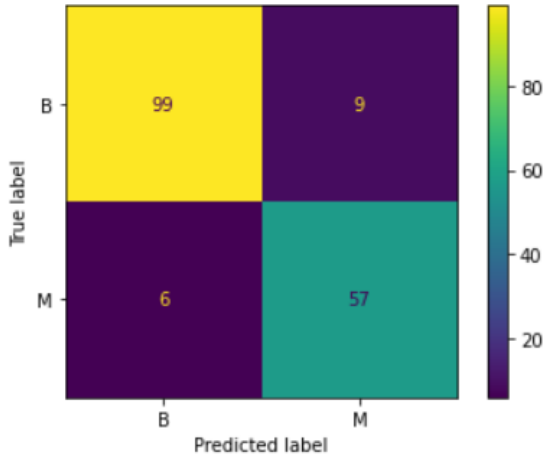
```
0.9422110552763819
```

```
print(model.score(x_test_std, y_test))
```

```
0.9122807017543859
```

## plotting Confusion Matrix

```
import matplotlib.pyplot as plt
predictions = model.predict(x_test_std)
cm = confusion_matrix(y_test, predictions, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
disp.plot()

plt.show()
```




Conclusion

Thus we have successfully completed the implementation of Naïve Bayes Gaussian Classifier.

**Assignment 8**

Aim: Implement SVM Classifier or Regression for given dataset

**Objectives:**

1. To learn SVM and kernel functions

2. To implement SVM classifier

Theory:

SVM:

**support-vector machines** (**SVMs**, also **support-vector networks**[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Kernel function

In machine learning, **kernel methods** are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified *feature map*: in contrast, kernel methods require only a user-specified *kernel*, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, *implicit* feature space without ever computing the coordinates of the data in that space, but rather by simply

computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "**kernel trick**".[1] Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

## Kernel trick

The kernel trick seems to be one of the most confusing concepts in statistics and machine learning; it first appears to be genuine mathematical sorcery, not to mention the problem of lexical ambiguity (does kernel refer to: a non-parametric way to estimate a probability density (statistics), the set of vectors $\mathbf{v}$ for which a linear transformation T maps to the zero vector — i.e. $T(\mathbf{v}) = 0$ (linear algebra), the set of elements in a group G that are mapped to the identity element by a homomorphism between groups (group theory), the core of a computer operating system (computer science), or something to do with the seeds of nuts or fruit?).

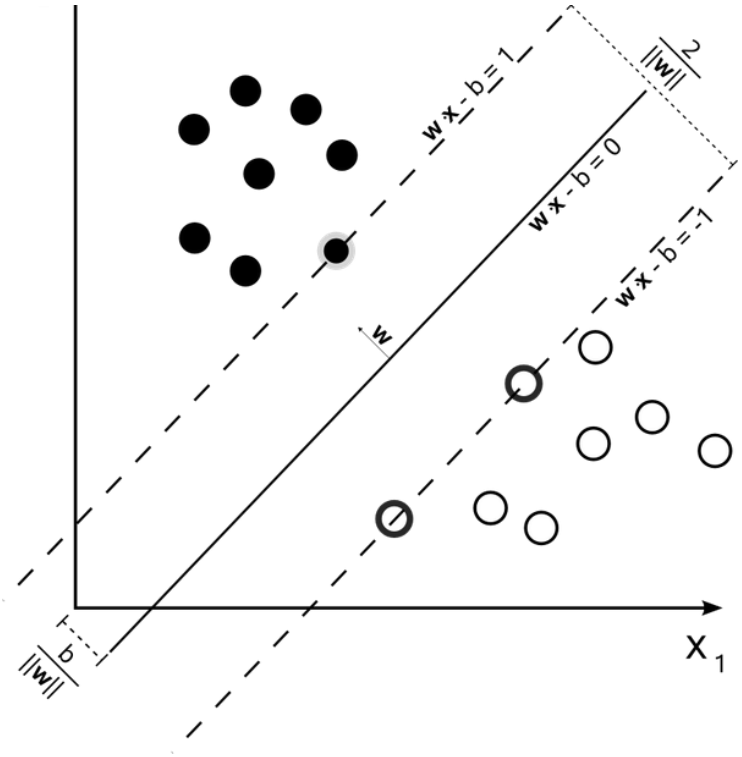
The kernel trick also illustrates some fundamental ideas about different ways to represent data and how machine learning algorithms “see” these different data representations. And finally, the seeming mathematical sleight of hand in the kernel trick just begs one to further explore what it actually means.

## Significance of SVM

it capable of doing both classification and regression. In this post I'll focus on using SVM for classification. In particular I'll be focusing on non-linear SVM, or SVM using a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The benefit is that you can capture much more complex relationships between your datapoints without having to perform difficult transformations on your own. The downside is that the training time is much longer as it's much more computationally intensive.

## Originally Answered: what is the purpose of support vector in SVM?

A support vector machine attempts to find the line that "best" separates two classes of points. By "best", we mean the line that results in the **largest margin** between the two classes. The points that lie on this margin are the *support vectors*.




Here we have three support vectors.

The nice thing about acknowledging these support vectors is that we can then formulate the problem of finding the "maximum-margin hyperplane" (the line that best separates the two classes) as an optimization problem that only considers these support vectors. So we can effectively throw out the vast majority of our data, which makes the classification process go much faster than, say, a neural network.

More importantly, by presenting the problem in terms of the support vectors (the so-called *dual form*), we can apply what's called the *kernel trick* to effectively transform the SVM into a non-linear classifier.

Code and output:

## Import the libraries

```
import pandas as pd
from sklearn.metrics import accuracy_score,confusion_matrix,ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

## Creating a dataframe

```
dataframe = pd.read_csv('titanic.csv')
df = dataframe.dropna()
df
```

| | sex | age | sibsp | parch | fare | embarked | class | who | alone | survived |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | male | 22.0 | 1 | 0 | 7.2500 | S | Third | man | False | 0 |
| **1** | female | 38.0 | 1 | 0 | 71.2833 | C | First | woman | False | 1 |
| **2** | female | 26.0 | 0 | 0 | 7.9250 | S | Third | woman | True | 1 |
| **3** | female | 35.0 | 1 | 0 | 53.1000 | S | First | woman | False | 1 |
| **4** | male | 35.0 | 0 | 0 | 8.0500 | S | Third | man | True | 0 |

## Encoding Labels

```
label_encoder = LabelEncoder()

df2 = df.apply(LabelEncoder().fit_transform)

df2
```

| | sex | age | sibsp | parch | fare | embarked | class | who | alone | survived |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 28 | 1 | 0 | 16 | 2 | 2 | 1 | 0 | 0 |
| **1** | 0 | 51 | 1 | 0 | 180 | 0 | 0 | 2 | 0 | 1 |
| **2** | 0 | 34 | 0 | 0 | 32 | 2 | 2 | 2 | 1 | 1 |
| **3** | 0 | 47 | 1 | 0 | 163 | 2 | 0 | 2 | 0 | 1 |
| **4** | 1 | 47 | 0 | 0 | 34 | 2 | 2 | 1 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **885** | 0 | 52 | 0 | 5 | 128 | 1 | 2 | 2 | 0 | 0 |
| **886** | 1 | 35 | 0 | 0 | 72 | 2 | 1 | 1 | 1 | 0 |
| **887** | 0 | 24 | 0 | 0 | 130 | 2 | 0 | 2 | 1 | 1 |
| **889** | 1 | 34 | 0 | 0 | 130 | 0 | 0 | 1 | 1 | 1 |
| **890** | 1 | 42 | 0 | 0 | 23 | 1 | 2 | 1 | 1 | 0 |

712 rows × 10 columns

## Slicing the dataset for input and output

```
x = df2.iloc[:, 0:9]
x.head()
```

| | sex | age | sibsp | parch | fare | embarked | class | who | alone |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 28 | 1 | 0 | 16 | 2 | 2 | 1 | 0 |
| **1** | 0 | 51 | 1 | 0 | 180 | 0 | 0 | 2 | 0 |
| **2** | 0 | 34 | 0 | 0 | 32 | 2 | 2 | 2 | 1 |
| **3** | 0 | 47 | 1 | 0 | 163 | 2 | 0 | 2 | 0 |
| **4** | 1 | 47 | 0 | 0 | 34 | 2 | 2 | 1 | 1 |

```
y = df2.iloc[:, 9]
y.head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: survived, dtype: int64
```

## Standardization of dataset

```
scalar = StandardScaler()
x_std = scalar.fit_transform(x)
```

## Splitting of dataset

```
x_train, x_test, y_train, y_test = train_test_split(x_std, y, test_size=0.3, random_state=0)
```

## Fitting the model

```
svc = SVC()
model = SVC(kernel = "linear", C=1.0)
```

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

```
SVC(kernel='linear')
```

## Printing model score for training and testing data

```
print(model.score(x_train, y_train))
```

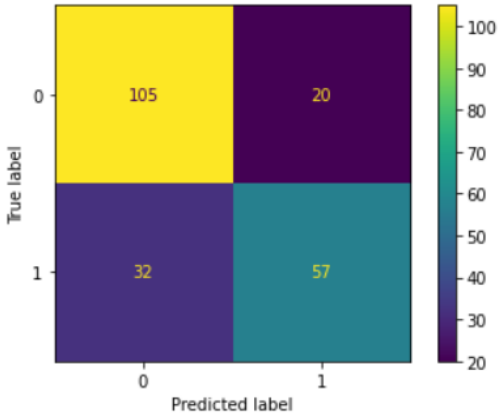
```
0.8012048192771084
```

```
print(model.score(x_test, y_test))
```

```
0.7570093457943925
```

## Confusion Matrix

```
import matplotlib.pyplot as plt
predictions = model.predict(x_test)
cm = confusion_matrix(y_test, predictions, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
disp.plot()

plt.show()
```




Conclusion:

Thus we have successfully completed the implementation of Support Vector Machine