<u>ML Lab Assignments</u>

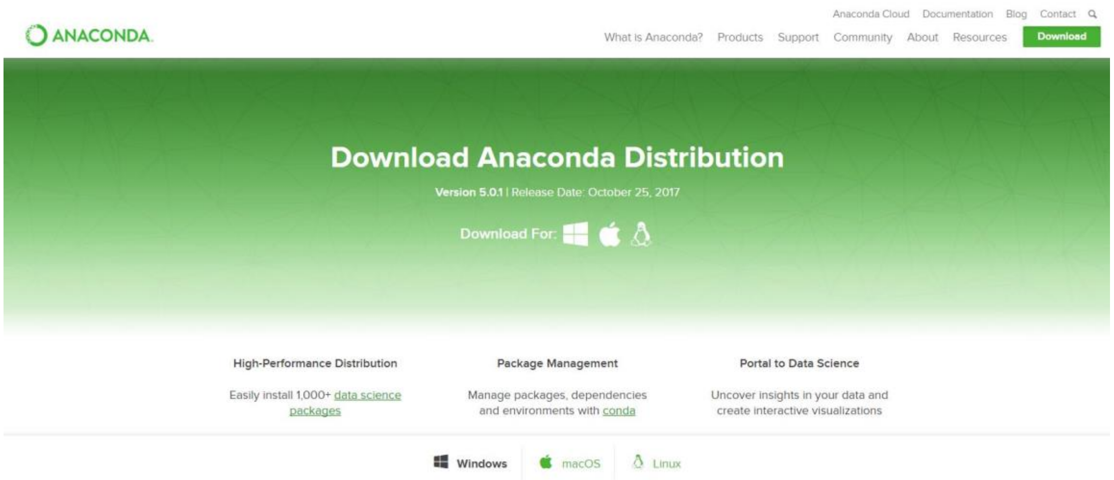
Name: Abhishek Mishra
Class: TY CSE IS 2
Roll No: 2193009

<u>ASSIGNMENT 1</u>

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

Objective: I) To install anaconda and configure it with Python
II) Installation, Configuration and checking the current Version of numpy, scipy, scikit, pandas and matplotlib using anaconda prompt and list their uses in machine learning.

Theory:
Installation Of Anaconda in Windows
1. Visit the Anaconda downloads page
Go to the following link: Anaconda.com/downloads



Download Anaconda Distribution

Version 5.0.1 | Release Date: October 25, 2017

Download For:

High-Performance Distribution
Easily install 1,000+ data science packages

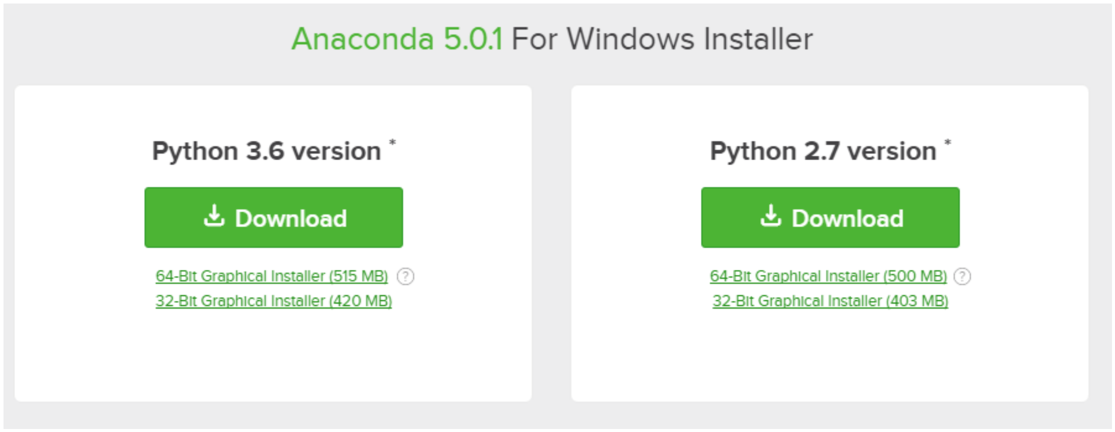
Package Management
Manage packages, dependencies and environments with conda

Portal to Data Science
Uncover insights in your data and create interactive visualizations

2. Select Windows
Select Windows where the three operating systems are listed.

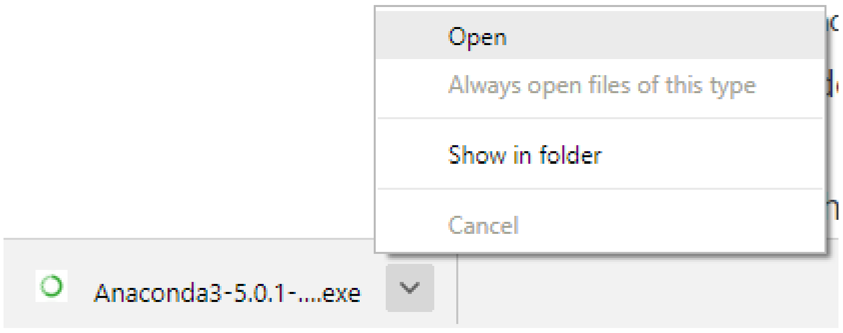
3. Download

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.


Anaconda 5.0.1 For Windows Installer
Python 3.6 version *
Download
64-Bit Graphical Installer (515 MB)
32-Bit Graphical Installer (420 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (500 MB)
32-Bit Graphical Installer (403 MB)

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, It is recommend that you **do not check** "Add Anaconda to my PATH environment variable".

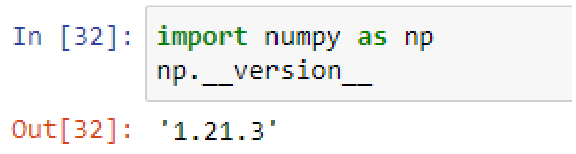


NUMPY/SCIPY:-

Installation Of NumPy

Command – pip3 install numpy

To check the version



```
In [32]: import numpy as np
         np.__version__

Out[32]: '1.21.3'
```

Installation of Scipy-stack

Command – pip3 install scipy-stack

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.

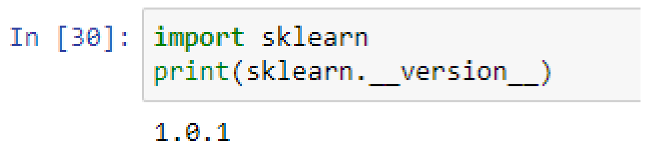
• NumPy array can also be used as an efficient multi-dimensional container for generic .data.

• 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.

SCIKIT:-
Installation of scikit
Command pip3 install -u scikit-learn

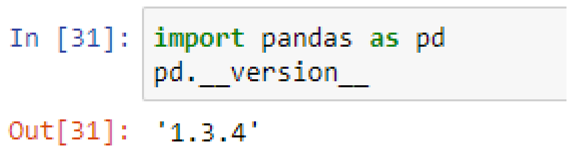


```
In [30]: import sklearn
         print(sklearn.__version__)

         1.0.1
```

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**
• **Pre-processing** , including Min-Max Normalization.

PANDAS:-
Installation of Pandas
Command- pip3 install pandas

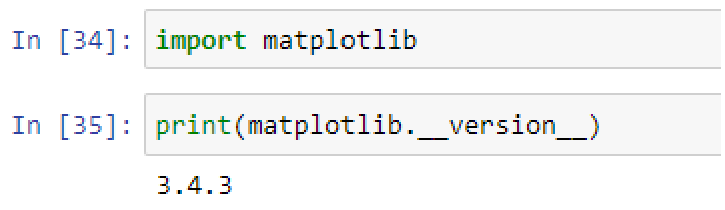


```
In [31]: import pandas as pd
         pd.__version__

Out[31]: '1.3.4'
```

• 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.

MATPLOTLIB:-
Installation of Matplotlib
Command- pip3 install matplotlib



```
In [34]: import matplotlib

In [35]: print(matplotlib.__version__)

         3.4.3
```

• 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.

Programs and Outputs

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.preprocessing import StandardScaler, Normalizer
        from sklearn.metrics import accuracy_score, confusion_matrix
        from sklearn.model_selection import train_test_split
        from sklearn import preprocessing,datasets
```

```
In [2]: # Dataset creation
        np.random.seed(42)
        x = np.random.random ((10,5))
        x
```

```
Out[2]: array([[0.37454012, 0.95071431, 0.73199394, 0.59865848, 0.15601864],
               [0.15599452, 0.05808361, 0.86617615, 0.60111501, 0.70807258],
               [0.02058449, 0.96990985, 0.83244264, 0.21233911, 0.18182497],
               [0.18340451, 0.30424224, 0.52475643, 0.43194502, 0.29122914],
               [0.61185289, 0.13949386, 0.29214465, 0.36636184, 0.45606998],
               [0.78517596, 0.19967378, 0.51423444, 0.59241457, 0.04645041],
               [0.60754485, 0.17052412, 0.06505159, 0.94888554, 0.96563203],
               [0.80839735, 0.30461377, 0.09767211, 0.68423303, 0.44015249],
               [0.12203823, 0.49517691, 0.03438852, 0.9093204 , 0.25877998],
               [0.66252228, 0.31171108, 0.52006802, 0.54671028, 0.18485446]])
```

```
In [3]: y = np.array(['m','m','f','f','m','f','m','m','f','f'])
        y
```

```
Out[3]: array(['m', 'm', 'f', 'f', 'm', 'f', 'm', 'm', 'f', 'f'], dtype='<U1')
```

```
In [4]: x [x<0.7] = 0
        x
```

```
Out[4]: array([[0.        , 0.95071431, 0.73199394, 0.        , 0.        ],
               [0.        , 0.        , 0.86617615, 0.        , 0.70807258],
               [0.        , 0.96990985, 0.83244264, 0.        , 0.        ],
               [0.        , 0.        , 0.        , 0.        , 0.        ],
               [0.        , 0.        , 0.        , 0.        , 0.        ],
               [0.78517596, 0.        , 0.        , 0.        , 0.        ],
               [0.        , 0.        , 0.        , 0.94888554, 0.96563203],
               [0.80839735, 0.        , 0.        , 0.        , 0.        ],
               [0.        , 0.        , 0.        , 0.9093204 , 0.        ],
               [0.        , 0.        , 0.        , 0.        , 0.        ]])
```

```
In [5]: #data split
        x_train, x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=0)
```

```
In [16]: print("x Training data shape:", x_train.shape)
         print("x Testing data shape:", x_test.shape)
         print("y Training data shape:", y_train.shape)
         print("y Testing data shape:", y_test.shape)

         x Training data shape: (7, 5)
         x Testing data shape: (3, 5)
         y Training data shape: (7,)
         y Testing data shape: (3,)
```

In [7]:
```
#Data preprocessing
#1. Standardization
# here we want to impport a lib so add it above for simplicity
# from sklearn.preprocessing import StandardScaler
b = np.array([(1.5,2,3),(4,5,6)], dtype=float)
print(b)
print(b.mean())
print(b.std())
```

```
[[1.5 2.  3. ]
 [4.  5.  6. ]]
3.5833333333333335
1.5920810978785667
```

In [8]:
```
scaler = StandardScaler()
b_std = scaler.fit_transform(b)
print (b_std)
```

```
[[-1. -1. -1.]
 [ 1.  1.  1.]]
```

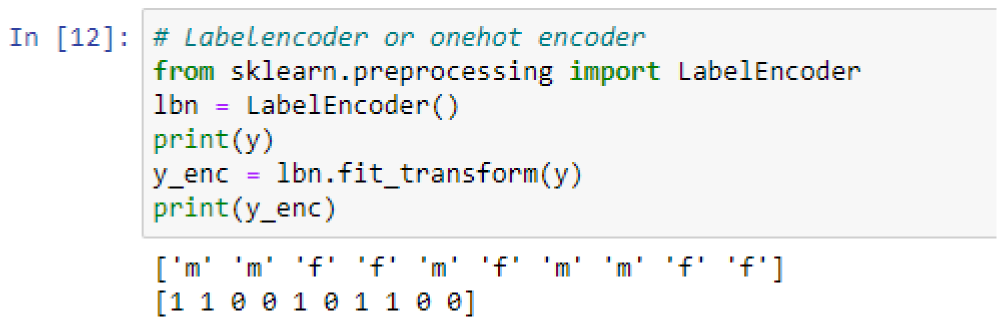
In [9]:
```
print(b_std.mean())
print(b_std.std())
```

```
0.0
1.0
```

In [10]:
```
std_X = scaler.fit_transform(x_train)
std_xtest = scaler.transform(x_test)
print(std_X)
print(std_xtest)
print(std_X.mean())
```

```
[[-0.63236155 -0.40824829 -0.6293576  -0.40824829 -0.62222525]
 [-0.63236155 -0.40824829  1.75833681 -0.40824829  1.22043232]
 [-0.63236155 -0.40824829 -0.6293576   2.44948974  1.89069395]
 [ 1.61315534 -0.40824829 -0.6293576  -0.40824829 -0.62222525]
 [-0.63236155 -0.40824829 -0.6293576  -0.40824829 -0.62222525]
 [-0.63236155  2.44948974  1.38845119 -0.40824829 -0.62222525]
 [ 1.54865239 -0.40824829 -0.6293576  -0.40824829 -0.62222525]]
[[-0.63236155  2.50718935  1.66534729 -0.40824829 -0.62222525]
 [-0.63236155 -0.40824829 -0.6293576   2.33033228 -0.62222525]
 [-0.63236155 -0.40824829 -0.6293576  -0.40824829 -0.62222525]]
1.586032892321652e-17
```

In [11]:
```
#2. Normalization
#from sklearn.preprocessing import Normalizer
norm = Normalizer()
norm_X = norm.fit_transform(x_train)
norm_xtest = norm.transform(x_test)
```



```
In [12]: # Labelencoder or onehot encoder
         from sklearn.preprocessing import LabelEncoder
         lbn = LabelEncoder()
         print(y)
         y_enc = lbn.fit_transform(y)
         print(y_enc)

         ['m' 'm' 'f' 'f' 'm' 'f' 'm' 'm' 'f' 'f']
         [1 1 0 0 1 0 1 1 0 0]
```

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

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.

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

Standardization:
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.

Program and Outputs:

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]: df = pd.read_csv('diabetes.csv')
```

```
In [3]: df.head()
```

Out[3]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [4]: df.tail()
```

Out[4]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1

```
In [5]: df.describe()     #statistical data
```

Out[5]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
In [6]: df.count()  #record of each attribute

Out[6]: pregnencies                 768
        glucose                     768
        bloodpressure               768
        skinthickness               768
        insulin                     768
        bmi                         768
        diabetespedigreefunction    768
        age                         768
        outcome                     768
        dtype: int64
```

```
In [7]: df.isna().sum()     #shows the count of null values

Out[7]: pregnencies                 0
        glucose                     0
        bloodpressure               0
        skinthickness               0
        insulin                     0
        bmi                         0
        diabetespedigreefunction    0
        age                         0
        outcome                     0
        dtype: int64
```

```
In [9]: df.corr()    #correlation between the columns
```

Out[9]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
pregnencies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
bloodpressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
skinthickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
diabetespedigreefunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
In [10]: df.iloc[1]  #i locator, what values are there in the dataset
```

```
Out[10]: pregnencies                  1.000
         glucose                     85.000
         bloodpressure               66.000
         skinthickness               29.000
         insulin                      0.000
         bmi                         26.600
         diabetespedigreefunction     0.351
         age                         31.000
         outcome                      0.000
         Name: 1, dtype: float64
```

```
In [11]: df.sort_values('age')
```

Out[11]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
255	1	113	64	35	0	33.6	0.543	21	1
60	2	84	0	0	0	0.0	0.304	21	0
102	0	125	96	0	0	22.5	0.262	21	0
182	1	0	74	20	23	27.7	0.299	21	0
623	0	94	70	27	115	43.5	0.347	21	0
...	...	...	...	...	...	...	...	...	...
123	5	132	80	0	0	26.8	0.186	69	0
684	5	136	82	0	0	0.0	0.640	69	0
666	4	145	82	18	0	32.5	0.235	70	1
453	2	119	0	0	0	19.6	0.832	72	0
459	9	134	74	33	60	25.9	0.460	81	0

768 rows × 9 columns

```
In [12]: np.sort(df.age.unique())
```

```
Out[12]: array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
                38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
                55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 72,
                81], dtype=int64)
```

```
In [13]: import matplotlib.pyplot as plt
```

```
In [14]: cols = df.columns
```

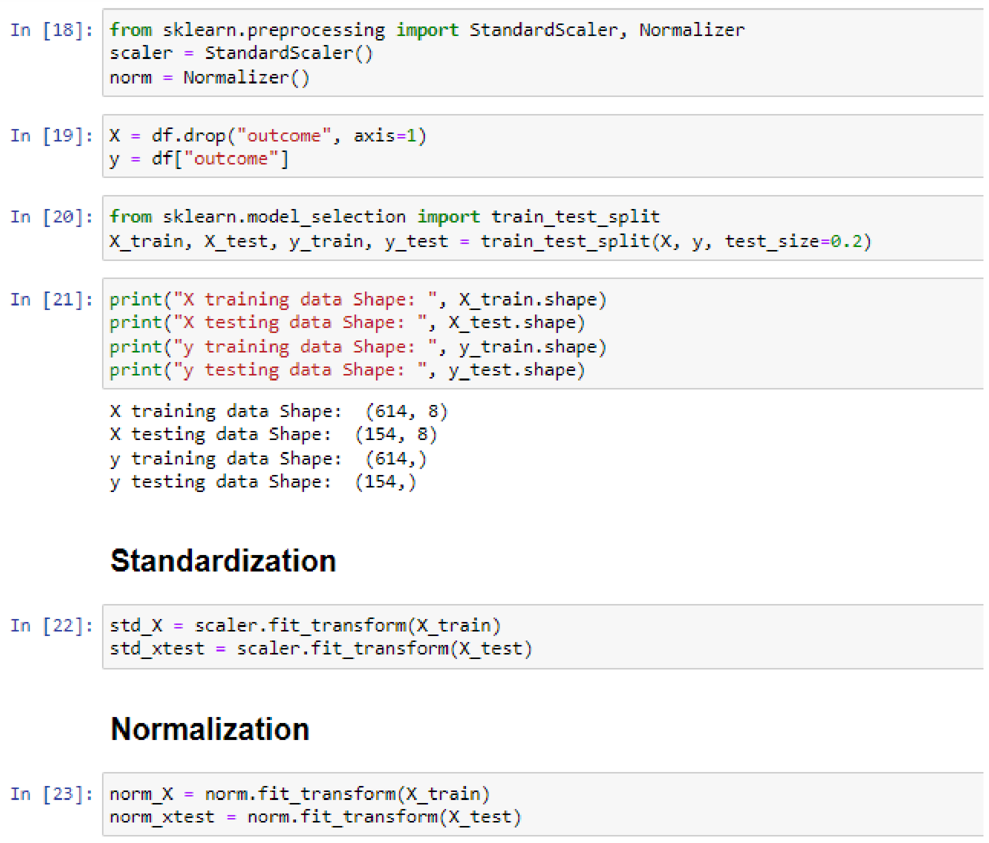
```
In [15]: print(cols)
```

```
Index(['pregnencies', 'glucose', 'bloodpressure', 'skinthickness', 'insulin',
       'bmi', 'diabetespedigreefunction', 'age', 'outcome'],
      dtype='object')
```

```
In [16]: df.corr()
```

Out[16]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
pregnencies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
bloodpressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
skinthickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
diabetespedigreefunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000



```
In [18]: from sklearn.preprocessing import StandardScaler, Normalizer
         scaler = StandardScaler()
         norm = Normalizer()
```

```
In [19]: X = df.drop("outcome", axis=1)
         y = df["outcome"]
```

```
In [20]: 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)
```

```
In [21]: print("X training data Shape: ", X_train.shape)
         print("X testing data Shape: ", X_test.shape)
         print("y training data Shape: ", y_train.shape)
         print("y testing data Shape: ", y_test.shape)
```

```
X training data Shape:  (614, 8)
X testing data Shape:  (154, 8)
y training data Shape:  (614,)
y testing data Shape:  (154,)
```

Standardization

```
In [22]: std_X = scaler.fit_transform(X_train)
         std_xtest = scaler.fit_transform(X_test)
```

Normalization

```
In [23]: norm_X = norm.fit_transform(X_train)
         norm_xtest = norm.fit_transform(X_test)
```

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 multi-layer 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.

Program and Outputs:

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.neural_network import MLPClassifier, MLPRegressor
        from sklearn.preprocessing import StandardScaler, Normalizer
```

MLP Classifier using Diabetes Dataset

```
In [2]: dbt = pd.read_csv("diabetes.csv")
        dbt.head()
```

Out[2]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [3]: dbt.isna().sum()
```

```
Out[3]: pregnencies                 0
        glucose                     0
        bloodpressure               0
        skinthickness               0
        insulin                     0
        bmi                         0
        diabetespedigreefunction    0
        age                         0
        outcome                     0
        dtype: int64
```

```
In [4]: X = dbt.drop("outcome", axis=1)
        Y = dbt["outcome"]
```

```
In [6]: 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, random_state=0)
```

In [7]:
```
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.fit_transform(X_test)
mlpclassifier = MLPClassifier(hidden_layer_sizes=(120), activation='relu'
mlpclassifier.fit(X_train_std, Y_train)
```

```
Iteration 1, loss = 0.69960017
Iteration 2, loss = 0.66805294
Iteration 3, loss = 0.64108380
Iteration 4, loss = 0.61798304
Iteration 5, loss = 0.59926141
Iteration 6, loss = 0.58304939
Iteration 7, loss = 0.57000865
Iteration 8, loss = 0.55911128
Iteration 9, loss = 0.54877880
Iteration 10, loss = 0.54009470
Iteration 11, loss = 0.53261359
Iteration 12, loss = 0.52608915
Iteration 13, loss = 0.52032570
Iteration 14, loss = 0.51500548
Iteration 15, loss = 0.50965405
Iteration 16, loss = 0.50508167
Iteration 17, loss = 0.50175358
Iteration 18, loss = 0.49861808
Iteration 19, loss = 0.49544914
```

In [8]:
```
print("Training Score:", mlpclassifier.score(X_train_std, Y_train)*100)
print("Testing Score:", mlpclassifier.score(X_test_std, Y_test)*100)
```

```
Training Score: 80.78175895765473
Testing Score: 81.16883116883116
```

MLP Regressor using Housing Dataset

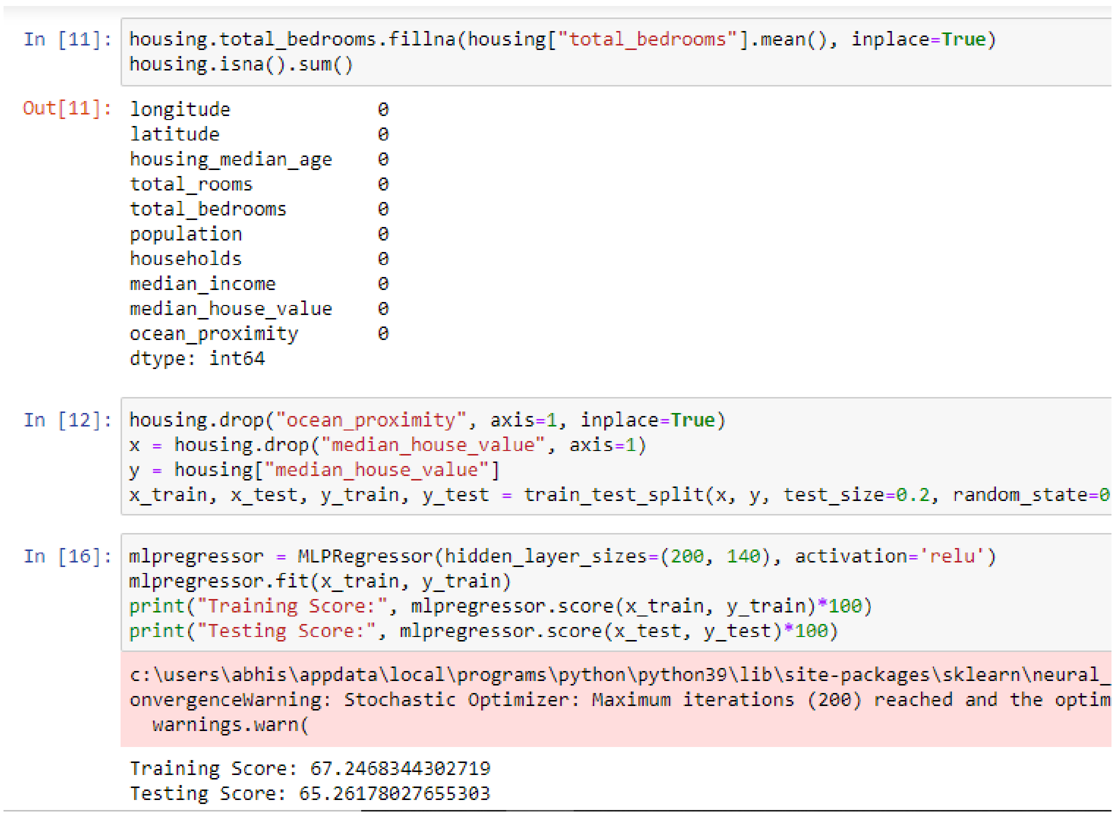
In [9]:
```
housing = pd.read_csv("housing.csv")
housing.head()
```

Out[9]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

In [10]:
```
housing.isna().sum()
```

Out[10]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```



```
In [11]: housing.total_bedrooms.fillna(housing["total_bedrooms"].mean(), inplace=True)
         housing.isna().sum()

Out[11]: longitude             0
         latitude              0
         housing_median_age    0
         total_rooms           0
         total_bedrooms        0
         population            0
         households            0
         median_income         0
         median_house_value    0
         ocean_proximity       0
         dtype: int64

In [12]: housing.drop("ocean_proximity", axis=1, inplace=True)
         x = housing.drop("median_house_value", axis=1)
         y = housing["median_house_value"]
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0

In [16]: mlpregressor = MLPRegressor(hidden_layer_sizes=(200, 140), activation='relu')
         mlpregressor.fit(x_train, y_train)
         print("Training Score:", mlpregressor.score(x_train, y_train)*100)
         print("Testing Score:", mlpregressor.score(x_test, y_test)*100)

         c:\users\abhis\appdata\local\programs\python\python39\lib\site-packages\sklearn\neural_
         onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optim
           warnings.warn(

         Training Score: 67.2468344302719
         Testing Score: 65.26178027655303
```

Conclusion

Thus we have successfully completed the implementation of Multilayer perceptron.

<u>Assignment 4</u>

Aim: Develop a Bayesian classifier for "Diabetes dataset"

Objectives

1. To learn Bayes theorem
2. To implement Bayesian classifier

Theory

<u>Bayes Theorem</u>

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)

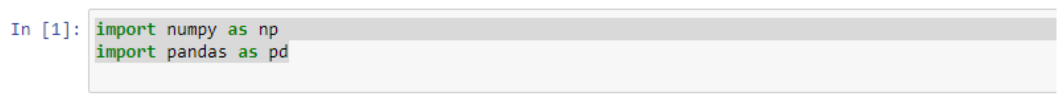
Which tells us: how often A happens *given that B happens:* **P(A|B),**
When we know: how often B happens *given that A happens:* **P(B|A)**
and how likely A is on its own: **P(A)**
and how likely B is on its own: **P(B)**

Bayes Classifier with example

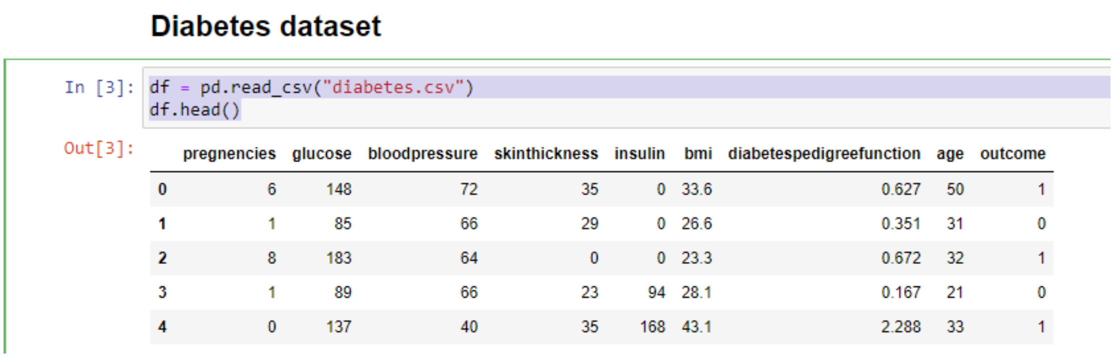
In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models. But they could be coupled with Kernel density estimation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

Program and Output:



```
In [1]: import numpy as np
        import pandas as pd
```

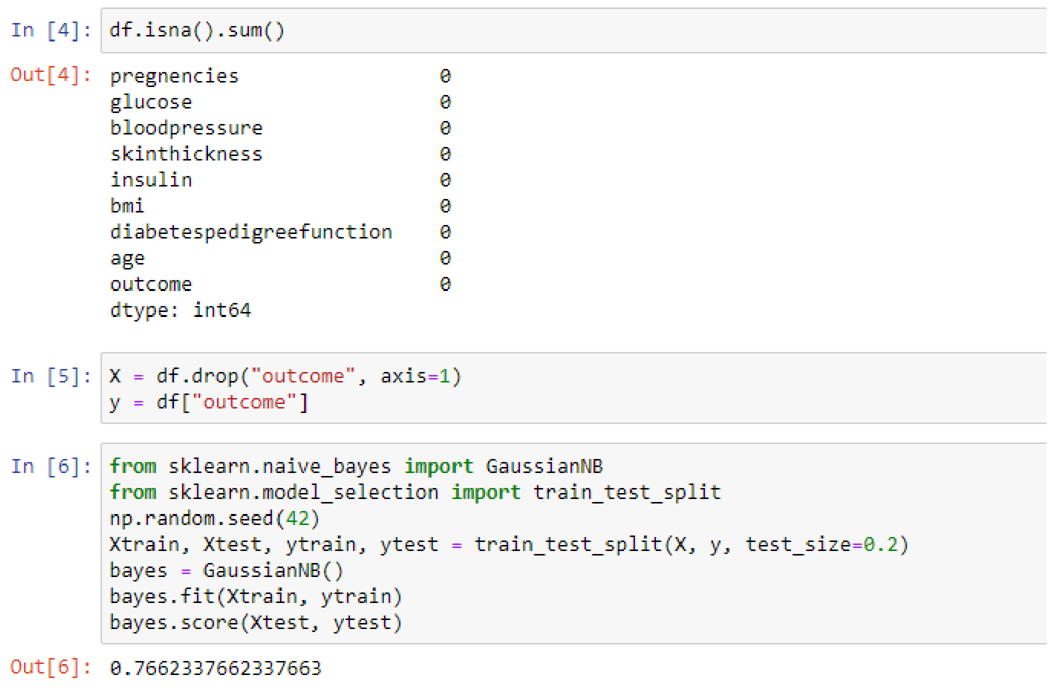


Diabetes dataset

```
In [3]: df = pd.read_csv("diabetes.csv")
        df.head()
```

Out[3]:

	pregnencies	glucose	bloodpressure	skinthickness	insulin	bmi	diabetespedigreefunction	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1



```
In [4]: df.isna().sum()

Out[4]: pregnencies                 0
        glucose                     0
        bloodpressure               0
        skinthickness               0
        insulin                     0
        bmi                         0
        diabetespedigreefunction    0
        age                         0
        outcome                     0
        dtype: int64
```

```
In [5]: X = df.drop("outcome", axis=1)
        y = df["outcome"]
```

```
In [6]: from sklearn.naive_bayes import GaussianNB
        from sklearn.model_selection import train_test_split
        np.random.seed(42)
        Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)
        bayes = GaussianNB()
        bayes.fit(Xtrain, ytrain)
        bayes.score(Xtest, ytest)

Out[6]: 0.7662337662337663
```

Conclusion

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

Assignment 5

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

Objectives:

1. To learn about dimensionality reduction techniques
2. To implement principle component analysis

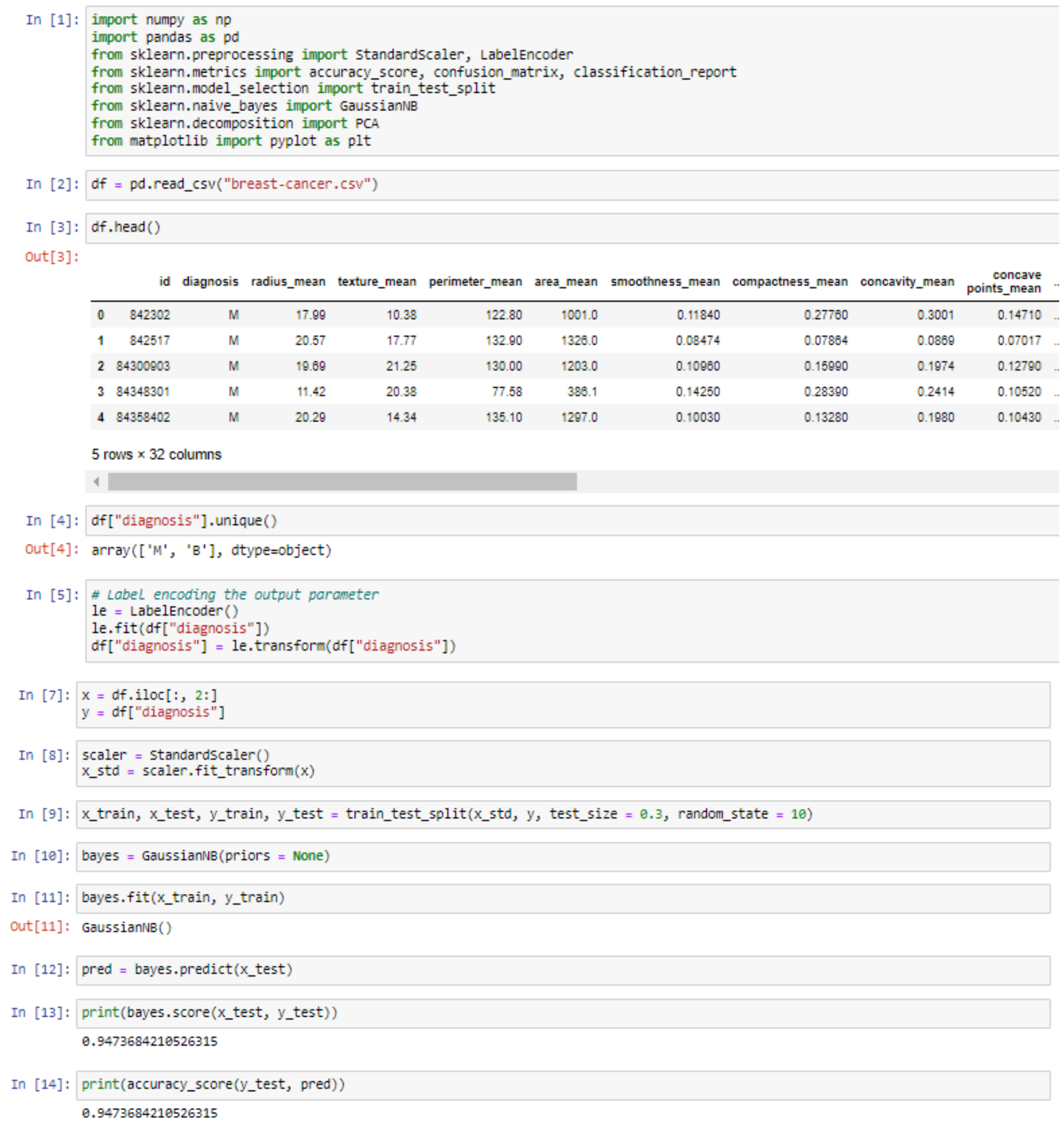
Theory

Given a collection of points in two, three, or higher dimensional space, a "best fitting" line can be defined as one that minimizes the average squared distance from a point to the line. The next best-fitting line can be similarly chosen from directions perpendicular to the first. Repeating this process yields an orthogonal basis in which different individual dimensions of the data are uncorrelated. These basis vectors are called **principal components**, and several related procedures **principal component analysis** (**PCA**).

PCA is mostly used as a tool in exploratory data analysis and for

making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA is either done in the following 2 steps:
1. calculating the data co-variance (or correlation) matrix of the original data
2. performing Eigenvalue decomposition on the co-variance matrix

Program and Outputs



```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.preprocessing import StandardScaler, LabelEncoder
        from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import GaussianNB
        from sklearn.decomposition import PCA
        from matplotlib import pyplot as plt
```

```
In [2]: df = pd.read_csv("breast-cancer.csv")
```

```
In [3]: df.head()
Out[3]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 32 columns

```
In [4]: df["diagnosis"].unique()
Out[4]: array(['M', 'B'], dtype=object)
```

```
In [5]: # Label encoding the output parameter
        le = LabelEncoder()
        le.fit(df["diagnosis"])
        df["diagnosis"] = le.transform(df["diagnosis"])
```

```
In [7]: x = df.iloc[:, 2:]
        y = df["diagnosis"]
```

```
In [8]: scaler = StandardScaler()
        x_std = scaler.fit_transform(x)
```

```
In [9]: x_train, x_test, y_train, y_test = train_test_split(x_std, y, test_size = 0.3, random_state = 10)
```

```
In [10]: bayes = GaussianNB(priors = None)
```

```
In [11]: bayes.fit(x_train, y_train)
Out[11]: GaussianNB()
```

```
In [12]: pred = bayes.predict(x_test)
```

```
In [13]: print(bayes.score(x_test, y_test))
         0.9473684210526315
```

```
In [14]: print(accuracy_score(y_test, pred))
         0.9473684210526315
```

Model Training using PCA

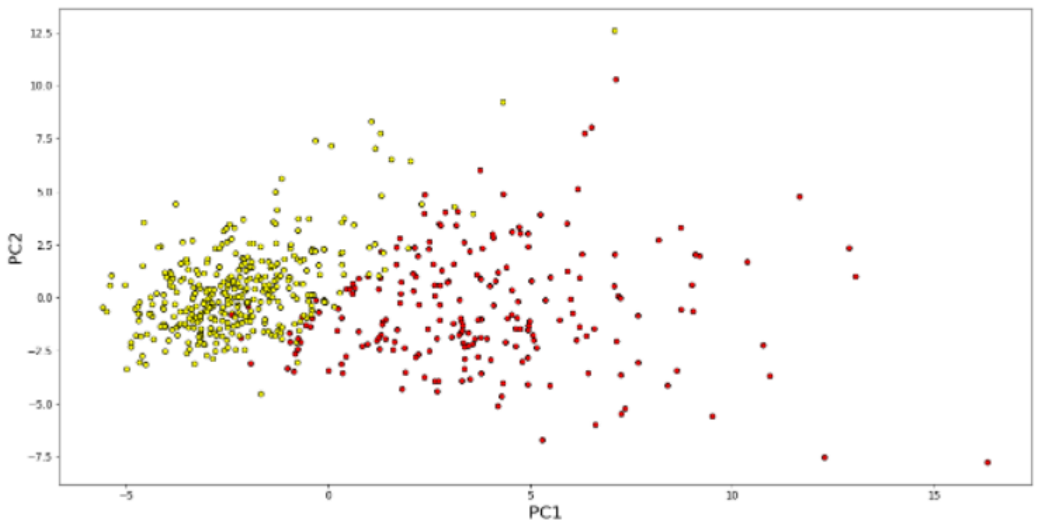
```
In [15]: pca = PCA(n_components = 2)
```

```
In [16]: pc = pca.fit_transform(x_std)
```

```
In [17]: pdf = pd.DataFrame(data = pc, columns = ["pc1", "pc2"])
```

```
In [18]: pdf.head()
Out[18]:
        pc1        pc2
0  9.192837   1.948583
1  2.387802  -3.768172
2  5.733896  -1.075174
3  7.122953  10.275589
4  3.935302  -1.948072
```

```
In [23]: plt.figure(figsize=(20,10))

         colour = ['red' if i == 1 else 'yellow' for i in y]
         plt.scatter(pdf.pc1,pdf.pc2 ,c=colour,edgecolors='#000000')
         plt.ylabel("Glucose",size=20)
         plt.xlabel('Age',size=20)
         plt.yticks(size=12)
         plt.xticks(size=12)
         plt.xlabel('PC1')
         plt.ylabel('PC2')
Out[23]: Text(0, 0.5, 'PC2')
```



Conclusion

Thus we have successfully completed the implementation of the Principle component Analysis Algorithm.

Assignment 6

Aim: Implement decision tree classification/regression technique for given dataset.
Developed model should be able to answer the given queries.

Objectives

1. To learn about decision trees
2. To implement decision trees and compare results

Decision Tree:

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
A tree can be *"learned"* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Different types:
Decision trees used in data mining are of two main types:
• **Classification tree** analysis is when the predicted outcome is the class (discrete) to which the data belongs.
• **Regression tree** analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

Program and Outputs

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: df = pd.read_csv("car.data")
        df.head()
```

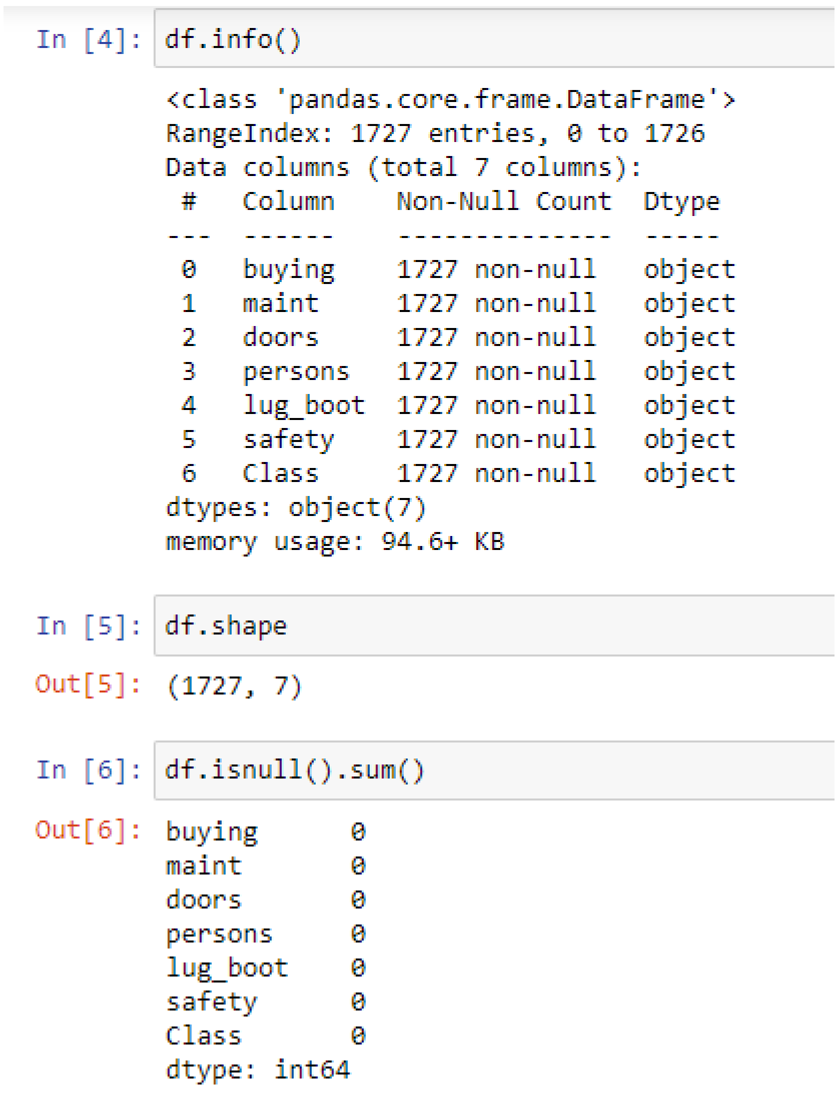
Out[2]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

```
In [3]: df.columns = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'Class']
        df.head()
```

Out[3]:

	buying	maint	doors	persons	lug_boot	safety	Class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc



```
In [4]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1727 non-null   object
 1   maint     1727 non-null   object
 2   doors     1727 non-null   object
 3   persons   1727 non-null   object
 4   lug_boot  1727 non-null   object
 5   safety    1727 non-null   object
 6   Class     1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
In [5]: df.shape

Out[5]: (1727, 7)
```

```
In [6]: df.isnull().sum()

Out[6]: buying      0
        maint       0
        doors       0
        persons     0
        lug_boot    0
        safety      0
        Class       0
        dtype: int64
```

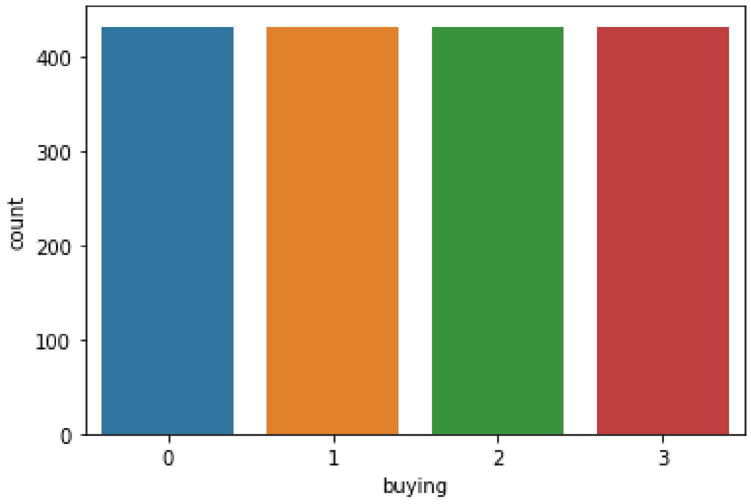
In [7]:
```
df.nunique()
```
Out[7]:
```
buying      4
maint       4
doors       4
persons     3
lug_boot    3
safety      3
Class       4
dtype: int64
```
In [8]:
```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
```
In [20]:
```
df.buying = enc.fit_transform(df.buying)
df.maint = enc.fit_transform(df.maint)
df.lug_boot = enc.fit_transform(df.lug_boot)
df.safety = enc.fit_transform(df.safety)
df.doors = df.doors.replace('5more', 5)
df.persons = df.persons.replace('more', 5)
```
In [21]:
```
df.head()
```
Out[21]:

	buying	maint	doors	persons	lug_boot	safety	Class
0	3	3	2	2	2	2	2
1	3	3	2	2	2	0	2
2	3	3	2	2	1	1	2
3	3	3	2	2	1	2	2

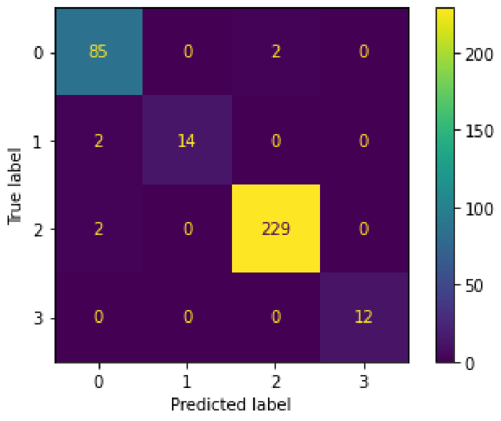
In [22]:
```
df.Class = enc.fit_transform(df.Class)
```
In [23]:
```
df.head()
```
Out[23]:

	buying	maint	doors	persons	lug_boot	safety	Class
0	3	3	2	2	2	2	2
1	3	3	2	2	2	0	2
2	3	3	2	2	1	1	2
3	3	3	2	2	1	2	2
4	3	3	2	2	1	0	2

```
In [24]: sns.countplot(data = df, x="buying")
         vhigh,high,med,low = df["buying"].value_counts()
         print("Very High:",vhigh)
         print("High:",high)
         print("Medium:",med)
         print("Low:",low)
         plt.show()

         Very High: 432
         High: 432
         Medium: 432
         Low: 431
```



```
In [26]: df["buying"].unique()

Out[26]: array([3, 0, 2, 1], dtype=int64)

In [27]: df["maint"].unique()

Out[27]: array([3, 0, 2, 1], dtype=int64)

In [28]: x = df.drop("Class", axis=1)
         y = df["Class"]

In [29]: from sklearn.model_selection import train_test_split
         xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)

In [30]: from sklearn.tree import DecisionTreeClassifier
         model = DecisionTreeClassifier()
         model.fit(xtrain, ytrain)

Out[30]: DecisionTreeClassifier()

In [31]: base_pred = model.predict(xtest)
In [33]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
         print("Accuracy: ",accuracy_score(ytest, base_pred))

         Accuracy:  0.9826589595375722

In [34]: print("Confusion Matrix:\n", confusion_matrix(ytest, base_pred))

         Confusion Matrix:
          [[ 85   0   2   0]
          [  2  14   0   0]
          [  2   0 229   0]
          [  0   0   0  12]]

In [35]: from sklearn.metrics import plot_confusion_matrix
         plot_confusion_matrix(model, xtest, ytest)
         plt.show()

         c:\users\abhis\appdata\local\programs\python\python39\lib\site-packages\sklear
         n plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
         e of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMa
           warnings.warn(msg, category=FutureWarning)
```

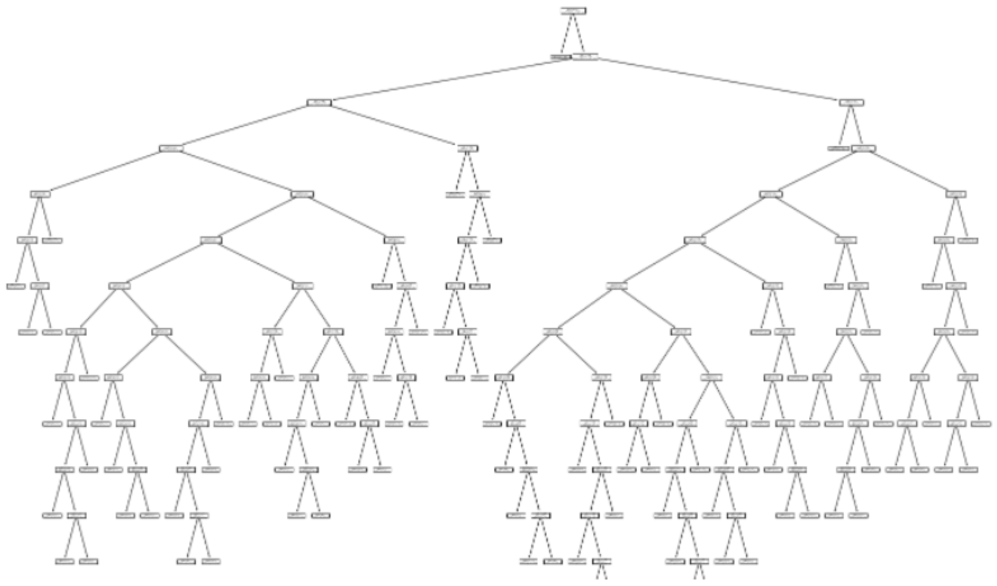


```
In [36]: print("Classification Report:\n", classification_report(ytest, base_pred))

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.96      0.98      0.97        87
                    1       1.00      0.88      0.93        16
                    2       0.99      0.99      0.99       231
                    3       1.00      1.00      1.00        12

             accuracy                           0.98       346
            macro avg       0.99      0.96      0.97       346
         weighted avg       0.98      0.98      0.98       346
```

```
In [38]: pd.DataFrame(index=x.columns, data=model.feature_importances_, columns=["Feature Importances"])
Out[38]:
```

	Feature Importances
buying	0.235046
maint	0.161940
doors	0.052103
persons	0.182273
lug_boot	0.118845
safety	0.249793

```
In [41]: from sklearn.tree import plot_tree
         plt.figure(figsize=(24, 16))
         plot_tree(model)
         plt.show()
```



Conclusion

Thus we have successfully completed the implementation of decision tree classifier.

Assignment 7

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**) 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**". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

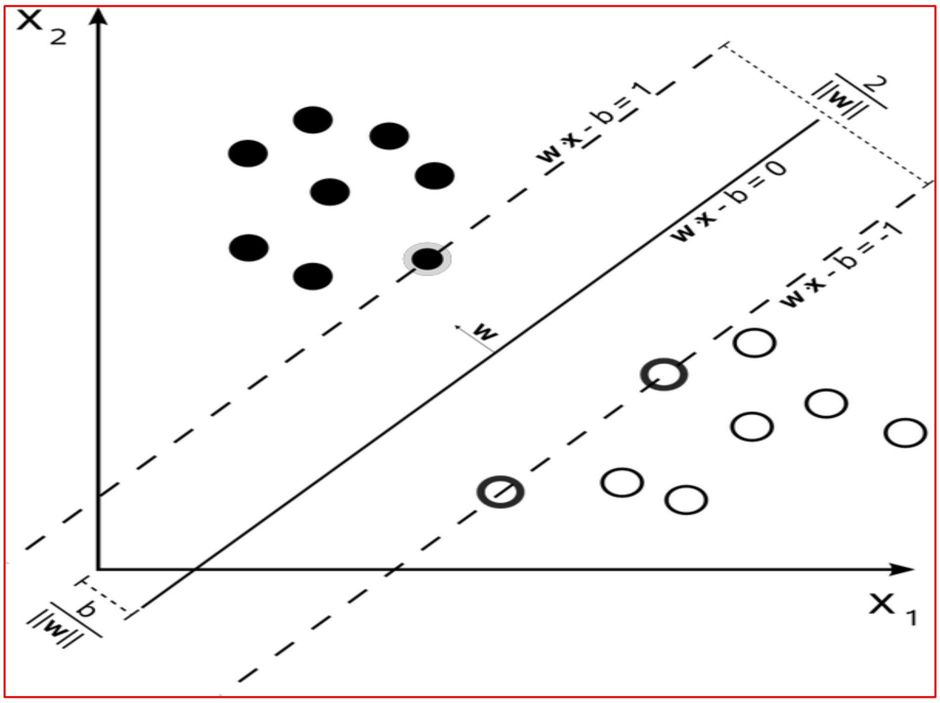
Kernel trick

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 **v** for which a linear transformation T maps to the zero vector — i.e. T(**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.

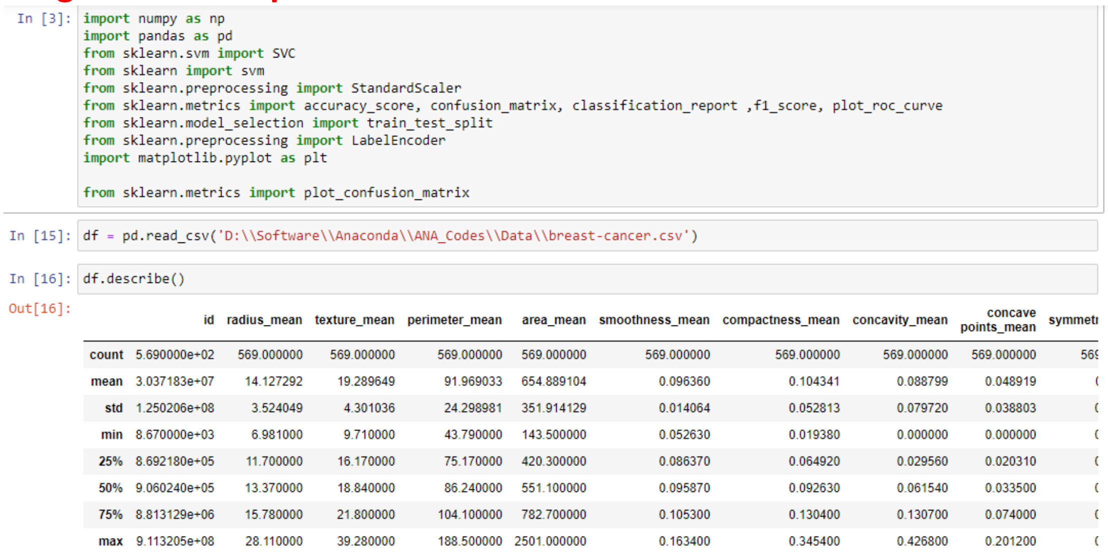
Significance of SVM

It is 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 data points 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.



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.

Program and Outputs:



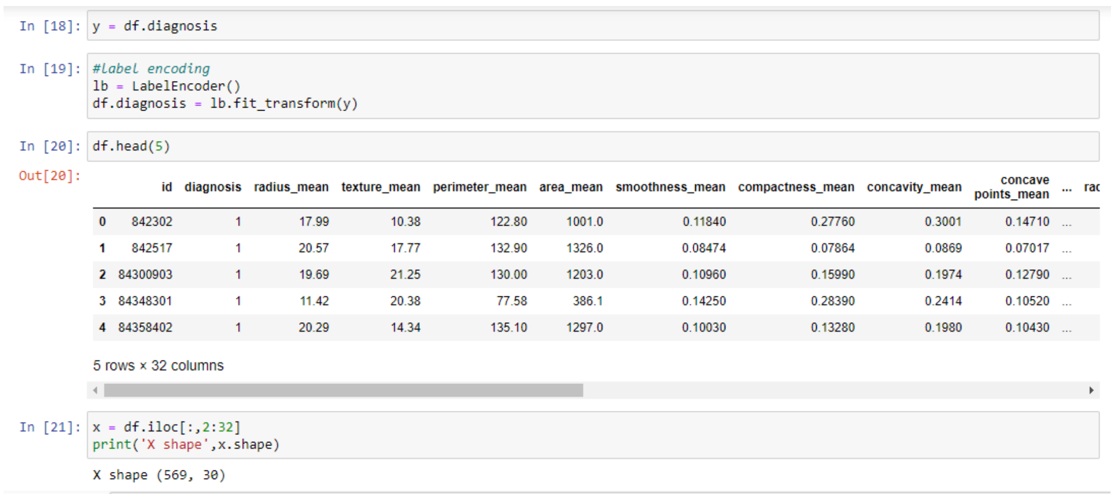
```
In [3]: import numpy as np
        import pandas as pd
        from sklearn.svm import SVC
        from sklearn import svm
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import accuracy_score, confusion_matrix, classification_report ,f1_score, plot_roc_curve
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder
        import matplotlib.pyplot as plt

        from sklearn.metrics import plot_confusion_matrix
```

```
In [15]: df = pd.read_csv('D:\\Software\\Anaconda\\ANA_Codes\\Data\\breast-cancer.csv')
```

```
In [16]: df.describe()
```

Out[16]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	(
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	(
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	(
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	(
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	(
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	(
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	(



```
In [18]: y = df.diagnosis
```

```
In [19]: #Label encoding
         lb = LabelEncoder()
         df.diagnosis = lb.fit_transform(y)
```

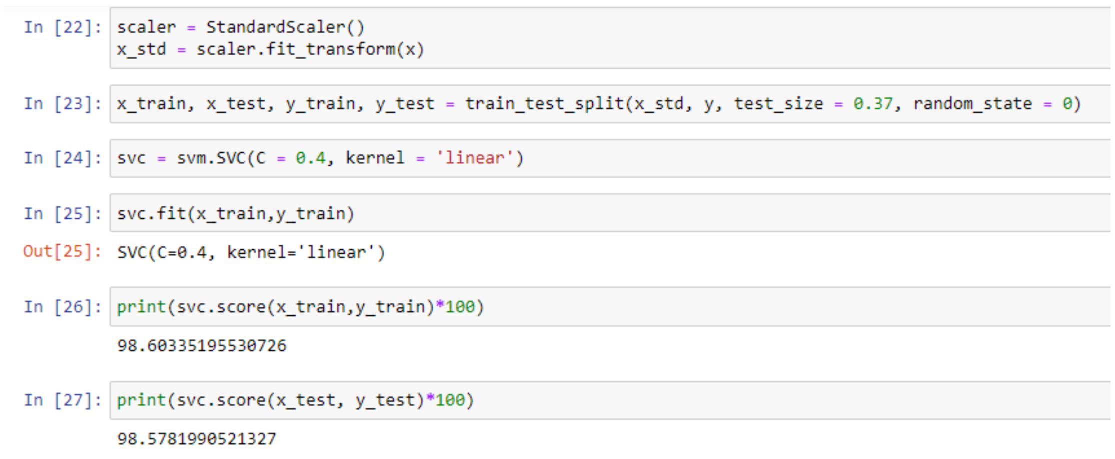
```
In [20]: df.head(5)
```

Out[20]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	rac
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

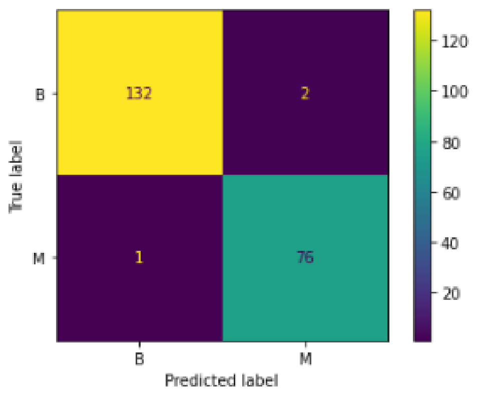
5 rows × 32 columns

```
In [21]: x = df.iloc[:,2:32]
         print('X shape',x.shape)

         X shape (569, 30)
```



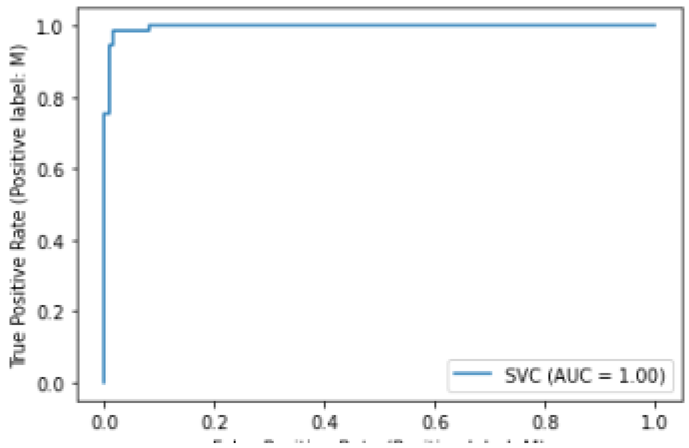
```
In [22]: scaler = StandardScaler()
         x_std = scaler.fit_transform(x)

In [23]: x_train, x_test, y_train, y_test = train_test_split(x_std, y, test_size = 0.37, random_state = 0)

In [24]: svc = svm.SVC(C = 0.4, kernel = 'linear')

In [25]: svc.fit(x_train,y_train)

Out[25]: SVC(C=0.4, kernel='linear')

In [26]: print(svc.score(x_train,y_train)*100)

         98.60335195530726

In [27]: print(svc.score(x_test, y_test)*100)

         98.5781990521327
```

```
In [28]: plot_confusion_matrix (svc, x_test, y_test)

Out[28]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22df046f190>
```



```
In [29]: y_pred = svc.predict(x_test)
         cr = classification_report(y_test, y_pred)
         print(cr)

              precision    recall  f1-score   support

           B       0.99      0.99      0.99       134
           M       0.97      0.99      0.98        77

    accuracy                           0.99       211
   macro avg       0.98      0.99      0.98       211
weighted avg       0.99      0.99      0.99       211
```

```
In [30]: plot_roc_curve(svc, x_test, y_test)

Out[30]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x22df0567100>
```



Conclusion:

Thus we have successfully completed the implementation of Support Vector Machine

Assignment 8

Aim: Implement K means algorithm for multidimensional data for Cars or Wine dataset from UCI repository

Objectives:

1. To learn unsupervised learning
2. To implement K means algorithm

Theory:

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses.
The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Clustering :

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

K-means algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.
The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

Program and Outputs

```
In [9]:  import pandas as pd
         from matplotlib import pyplot as plt
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split
         from sklearn.cluster import KMeans
         from sklearn.metrics import accuracy_score
```

```
In [10]: df = pd.read_csv("wine.csv")
```

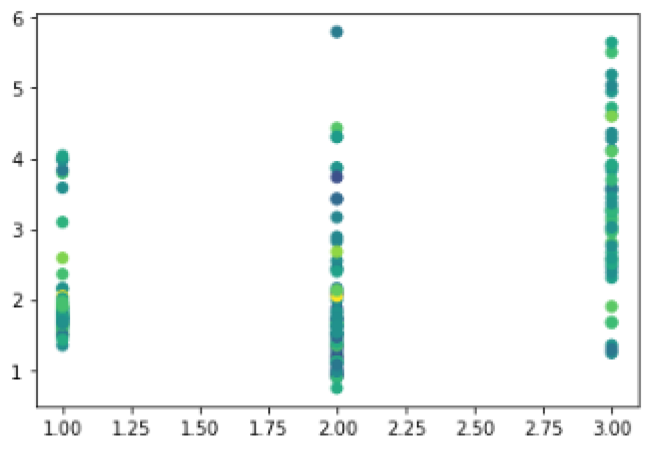
```
In [11]: df.head()
```

Out[11]:

	1	14.23	1.71	2.43	15.6	127	2.8	3.06	.28	2.29	5.64	1.04	3.92	1065
0	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
1	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
2	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
3	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
4	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450

```
In [12]: km = KMeans(n_clusters = 3, max_iter = 400,verbose = True, tol = 0.2)
```

```
In [25]: plt.scatter(df['1'], df['1.71'], c = df['2.43'])

Out[25]: <matplotlib.collections.PathCollection at 0x15e5be107c0>
```



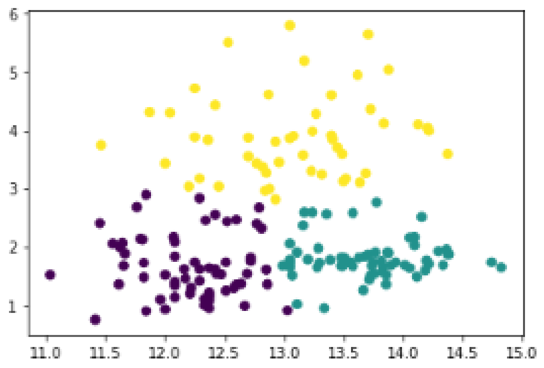
```
In [26]: pred = km.fit_predict(df[['14.23','1.71']])
```

```
Initialization complete
Iteration 0, inertia 136.85850000000005
Iteration 1, inertia 99.78942213885551
Converged at iteration 1: center shift 0.04406827021324147 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 128.23490000000004
Iteration 1, inertia 107.39623502819985
Converged at iteration 1: center shift 0.12470711782093827 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 178.3991000000001
Iteration 1, inertia 107.85058374516473
Converged at iteration 1: center shift 0.12208344688764458 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 106.54260000000002
Converged at iteration 0: center shift 0.10092973728350511 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 142.29470000000006
Iteration 1, inertia 118.24426623767907
Converged at iteration 1: center shift 0.14099100256491467 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 170.42960000000005
Iteration 1, inertia 98.69847742038607
Converged at iteration 1: center shift 0.035418013413237706 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 188.23680000000004
Iteration 1, inertia 152.44243457410363
Converged at iteration 1: center shift 0.11149434703790705 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 158.97380000000007
Iteration 1, inertia 106.32510326464526
Converged at iteration 1: center shift 0.09336524016460139 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 185.9999
Iteration 1, inertia 104.66079174517712
Converged at iteration 1: center shift 0.1435361922175397 within tolerance 0.18962616425675893.
Initialization complete
Iteration 0, inertia 183.13930000000002
Iteration 1, inertia 98.4087771909797
Converged at iteration 1: center shift 0.04079211056738101 within tolerance 0.18962616425675893.
```

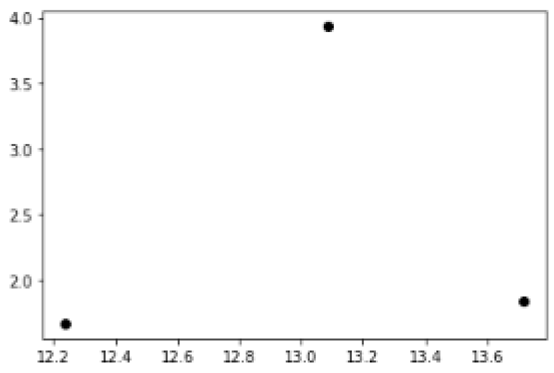
```
In [27]: score = accuracy_score(df['1'], pred)
```

```
In [31]: score

Out[31]: 0.3389830508474576
```

```
In [29]: plt.scatter(df['14.23'], df['1.71'], c = pred)

Out[29]: <matplotlib.collections.PathCollection at 0x15e5c5dceb0>
```



```
In [30]: plt.scatter(km.cluster_centers_[:,0] ,km.cluster_centers_[:,1], color='black')

Out[30]: <matplotlib.collections.PathCollection at 0x15e5ca64d00>
```



```
In [32]: score

Out[32]: 0.3389830508474576
```

Conclusion

Thus we have successfully completed the implementation of K-means algorithm

Assignment 9

Aim: Download the famous dataset of iris and Implement KNN algorithm to predict the class to which these plants belong, Calculate the performance metrics and compare the error rate with K value(K value range).

Objectives:

1. To learn KNN algorithm
2. To implement KNN classifier

Theory:

Lazy learners

Lazy learners simply store the training data and wait until a testing data appear.

When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.

Eager learners

Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to the model construction, eager learners take a long time for train and less time to predict.

Lazy learner:
1. Just store Data set **without** learning from it
2. Start classifying data when it receives **Test data**
3. So, it takes less time learning and more time classifying data

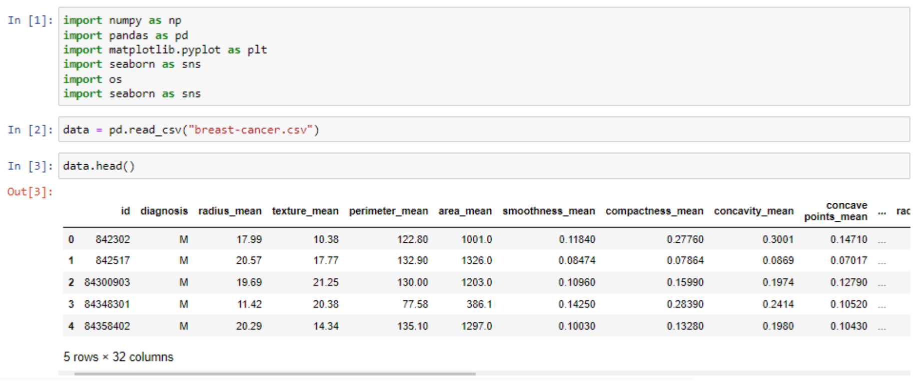
Eager learner:
1. When it receives data set it starts classifying (learning)
2. Then it does not wait for test data to learn3. So, it takes long time learning and less time classifying data

KNN Algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

The KNN Algorithm
1. Load the data
2. Initialize K to your chosen number of neighbours
3. For each example in the data
    - 3.1- Calculate the distance between the query example and the current example from the data.
    - 3.2- Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Program and Outputs



In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import seaborn as sns
```

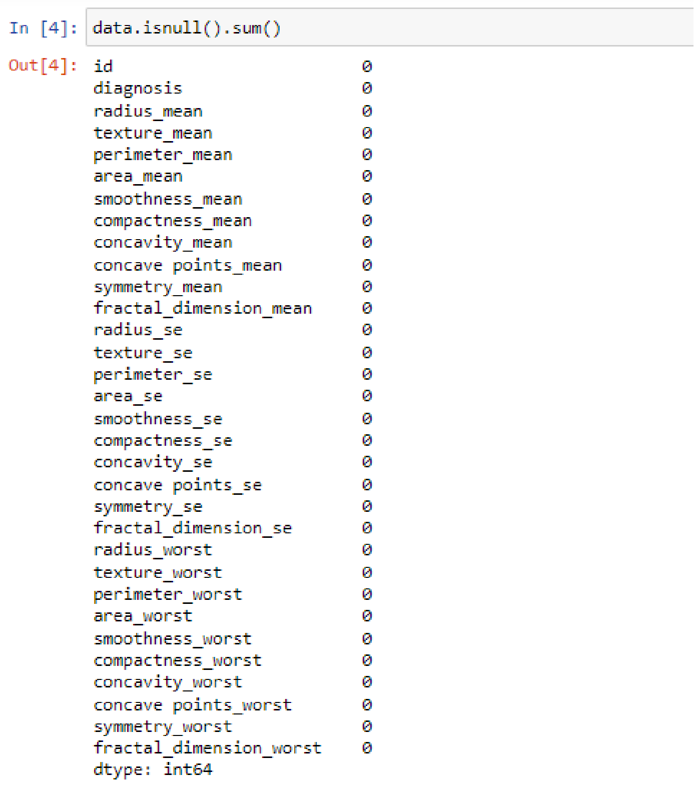
In [2]:
```
data = pd.read_csv("breast-cancer.csv")
```

In [3]:
```
data.head()
```

Out[3]:

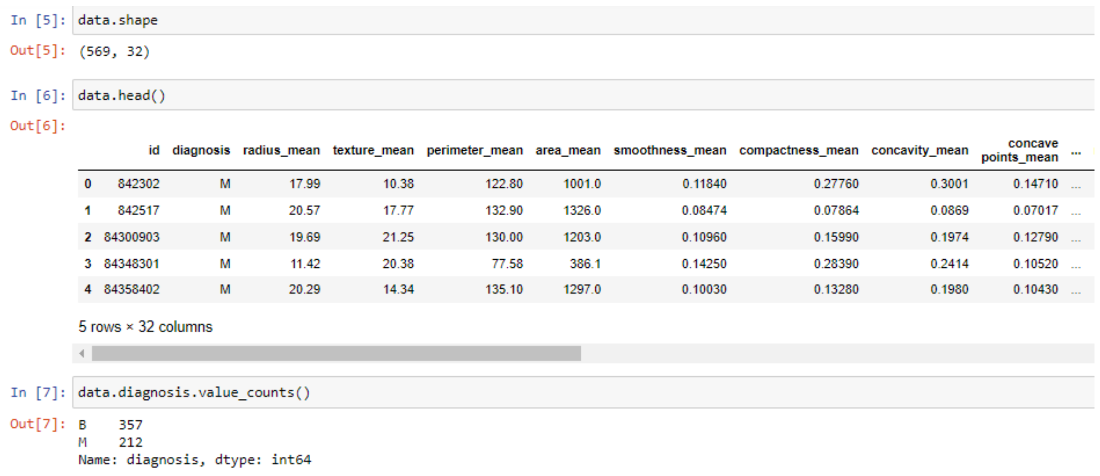
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	rac
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 32 columns



In [4]:
```
data.isnull().sum()
```

Out[4]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```



In [5]:
```
data.shape
```

Out[5]:
```
(569, 32)
```

In [6]:
```
data.head()
```

Out[6]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 32 columns

In [7]:
```
data.diagnosis.value_counts()
```

Out[7]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
In [9]: data.drop(columns = ["id"] , inplace = True)
```

```
In [10]: data["diagnosis"] = np.where(data["diagnosis"] == "M" , 1 , 0)
```

```
In [12]: X = data.drop(columns = ["diagnosis"])
         Y = data["diagnosis"]
```

```
In [14]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test= train_test_split(X , Y, test_size= 0.25, random_state=0)
```

```
In [15]: from sklearn.preprocessing import StandardScaler
         st_x= StandardScaler()
         x_train= st_x.fit_transform(x_train)
         x_test= st_x.transform(x_test)
```

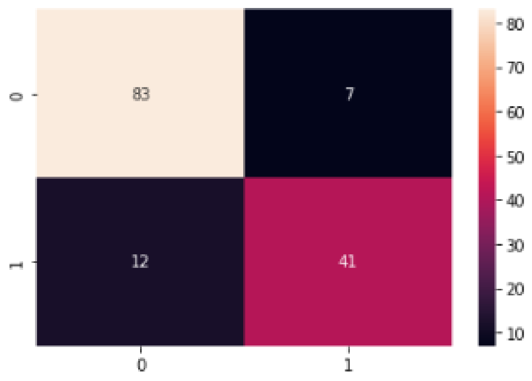
```
In [16]: from sklearn.neighbors import KNeighborsClassifier
         classifier= KNeighborsClassifier(n_neighbors=10)
         classifier.fit(x_train[:, [0 , 1]], y_train)

Out[16]: KNeighborsClassifier(n_neighbors=10)
```

```
In [17]: y_pred = classifier.predict(x_test[:, [0, 1]])
```

```
In [18]: from sklearn.metrics import accuracy_score , confusion_matrix
```

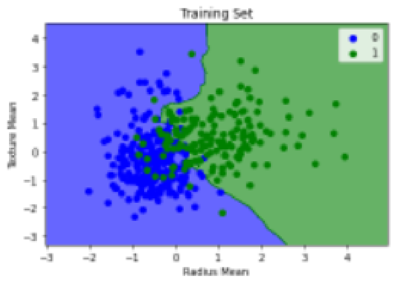
```
In [19]: accuracy_score(y_test , y_pred)

Out[19]: 0.8671328671328671
```

```
In [20]: sns.heatmap(confusion_matrix(y_test , y_pred) , annot = True , fmt = 'd')

Out[20]: <AxesSubplot:>
```



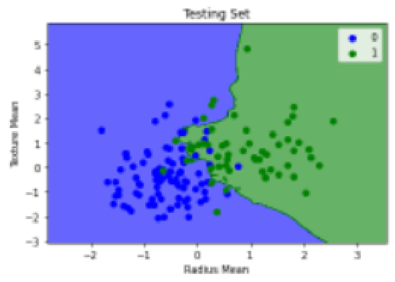
```
In [21]: from matplotlib.colors import ListedColormap
In [22]: # For training set
         x1 , x2 = x_train[:,0] , x_train[:, 1]
         x1_min , x1_max = x1.min() - 1 , x1.max() + 1
         x2_min , x2_max = x2.min() - 1 , x2.max() + 1

         xx1 , xx2 = np.meshgrid(np.arange(x1_min , x1_max , 0.01),
                                 np.arange(x2_min , x2_max , 0.01))
         plt.contourf(xx1 , xx2 ,
                      classifier.predict(np.array([xx1.ravel() , xx2.ravel()]).T).reshape(xx1.shape)
                       , alpha = 0.6 , cmap = ListedColormap(("blue" , "green"))
                      )
         plt.xlim(x1_min  , x1_max)
         plt.ylim(x2_min , x2_max)
         for i, j in enumerate(np.unique(y_train)):
             plt.scatter(x1[y_train == j], x2[y_train == j],
                 c = ListedColormap(('blue', 'green'))(i), label = j)

         plt.title("Training Set")
         plt.xlabel("Radius Mean")
         plt.ylabel("Texture Mean")
         plt.legend()
         plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```



```
In [23]: # For training set
         x1 , x2 = x_test[:,0] , x_test[:, 1]
         x1_min , x1_max = x1.min() - 1 , x1.max() + 1
         x2_min , x2_max = x2.min() - 1 , x2.max() + 1

         xx1 , xx2 = np.meshgrid(np.arange(x1_min , x1_max , 0.01),
                                 np.arange(x2_min , x2_max , 0.01))
         plt.contourf(xx1 , xx2 ,
                      classifier.predict(np.array([xx1.ravel() , xx2.ravel()]).T).reshape(xx1.shape)
                       , alpha = 0.6 , cmap = ListedColormap(("blue" , "green"))
                      )
         plt.xlim(x1_min  , x1_max)
         plt.ylim(x2_min , x2_max)
         for i, j in enumerate(np.unique(y_test)):
             plt.scatter(x1[y_test == j], x2[y_test == j],
                 c = ListedColormap(('blue', 'green'))(i), label = j)

         plt.title("Testing Set")
         plt.xlabel("Radius Mean")
         plt.ylabel("Texture Mean")
         plt.legend()
         plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```



Conclusion

Thus we have successfully completed the implementation of KNN algorithm

Assignment 10

Aim: Implement CNN for MNIST/CIFAR10 dataset using tensorflow

Objectives:

1. To learn basics of deep learning
2. To learn and implement CNN

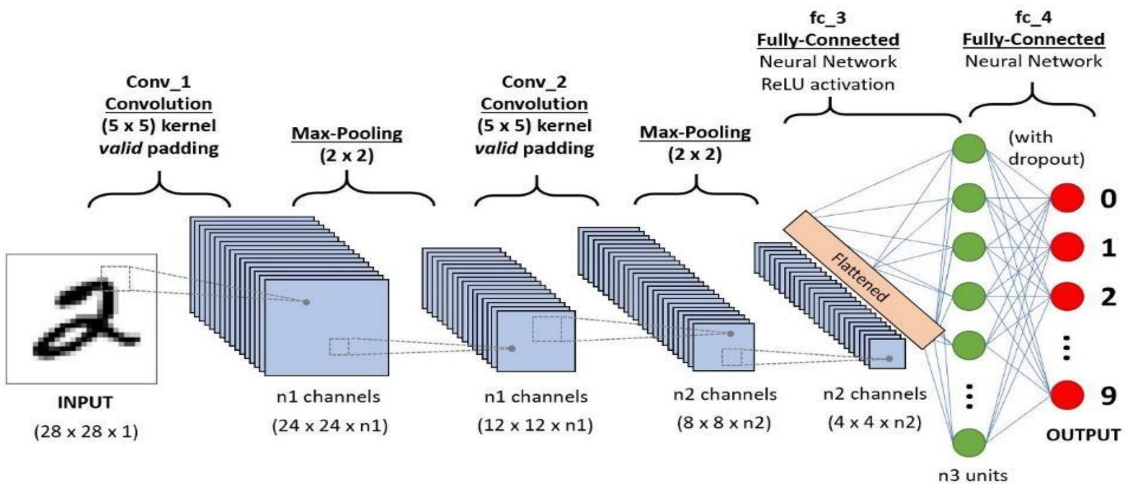
Theory:

Deep Learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.

Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

CNN Architecture



CNN Working

A Convolutional Neural Networks Introduction so to speak.

• Step 1: Convolution Operation

The first building block in our plan of attack is convolution operation. In this step, we will touch on feature detectors, which basically serve as the neural network's filters. We will also discuss feature maps, learning the parameters of such maps, how patterns are detected, the layers of detection, and how the findings are mapped out.

• Step 1(b): ReLU Layer

The second part of this step will involve the Rectified Linear Unit or ReLU. We will cover ReLU layers and explore how linearity functions in the context of Convolutional Neural Networks. Not necessary for understanding CNN's, but there's no harm in a quick lesson to improve your skills.

• Step 2: Pooling

In this part, we'll cover pooling and will get to understand exactly how it generally works. Our nexus here, however, will be a specific type of pooling; max pooling. We'll cover various approaches, though, including mean (or sum) pooling. This part will end with a demonstration made using a visual interactive tool that will definitely sort the whole concept out for you.

• Step 3: Flattening

This will be a brief breakdown of the flattening process and how we move from pooled to flattened layers when working with Convolutional Neural Networks.

• Step 4: Full Connection

In this part, everything that we covered throughout the section will be merged together. By learning this, you'll get to envision a fuller picture of how Convolutional Neural Networks operate and how the "neurons" that are finally produced learn the classification of images.

Program and Outputs

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.image as mpimg
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import load_model
import tensorflow as tf
```

```
[2]  mnist = tf.keras.datasets.mnist
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

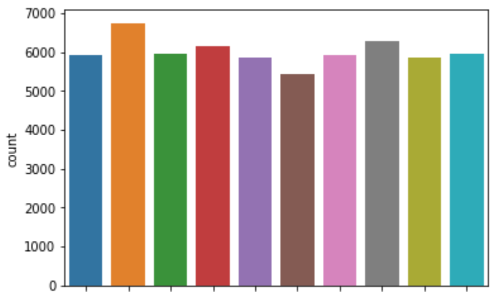
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
print('X Training shape: ',x_train.shape)
print('Y Training shape: ',y_train.shape)
print('X Testing shape: ',x_test.shape)
print('Y Testing shape: ',y_test.shape)
```

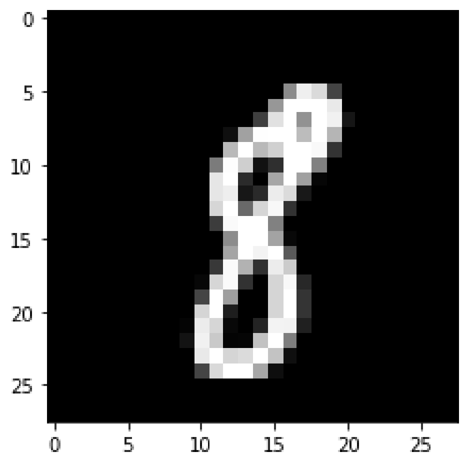
```
X Training shape:  (60000, 28, 28)
Y Training shape:  (60000,)
X Testing shape:  (10000, 28, 28)
Y Testing shape:  (10000,)
```

```
[4]  sns.countplot(y_train)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following varia
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fd80e798e90>
```



```
[5]  plt.imshow(x_train[300], cmap='gray')
     plt.show()
```



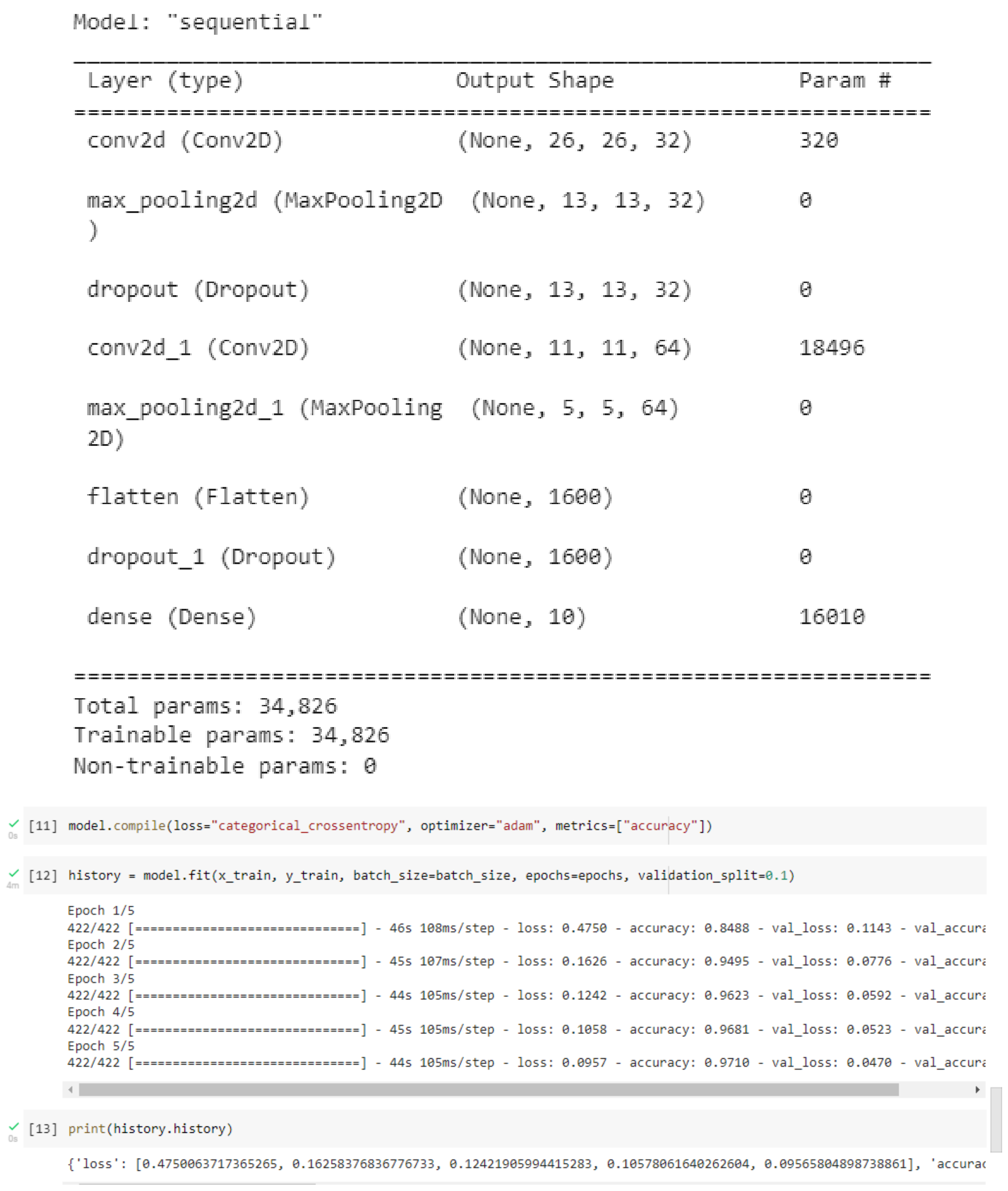
```
[6]  input_shape = (28,28,1)
```

```
[7]  x_train = x_train.astype("float32") / 255
     x_test = x_test.astype("float32") / 255
     x_train = np.expand_dims(x_train,-1)
     x_test = np.expand_dims(x_test,-1)
```

```
[8]  batch_size = 128
     num_classes = 10
     epochs = 5
```

```
[9]  y_train = keras.utils.to_categorical(y_train, num_classes)
     y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
[10] model = keras.Sequential(
         [
             keras.Input(shape=input_shape),
             layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
             layers.MaxPooling2D(pool_size=(2, 2)),
             layers.Dropout(0.5),
             layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
             layers.MaxPooling2D(pool_size=(2, 2)),
             layers.Flatten(),
             layers.Dropout(0.5),
             layers.Dense(num_classes, activation="softmax"),
         ]
     )

     model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout_1 (Dropout)         (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

```
[11] model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

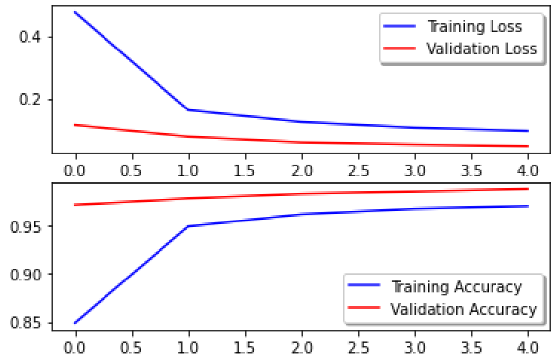
```
[12] history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/5
422/422 [==============================] - 46s 108ms/step - loss: 0.4750 - accuracy: 0.8488 - val_loss: 0.1143 - val_accura
Epoch 2/5
422/422 [==============================] - 45s 107ms/step - loss: 0.1626 - accuracy: 0.9495 - val_loss: 0.0776 - val_accura
Epoch 3/5
422/422 [==============================] - 44s 105ms/step - loss: 0.1242 - accuracy: 0.9623 - val_loss: 0.0592 - val_accura
Epoch 4/5
422/422 [==============================] - 45s 105ms/step - loss: 0.1058 - accuracy: 0.9681 - val_loss: 0.0523 - val_accura
Epoch 5/5
422/422 [==============================] - 44s 105ms/step - loss: 0.0957 - accuracy: 0.9710 - val_loss: 0.0470 - val_accura
```

```
[13] print(history.history)
```

```
{'loss': [0.4750063717365265, 0.16258376836776733, 0.12421905994415283, 0.10578061640262604, 0.09565804898738861], 'accurac
```

```python
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training Loss")
ax[0].plot(history.history['val_loss'], color='r', label="Validation Loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training Accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation Accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



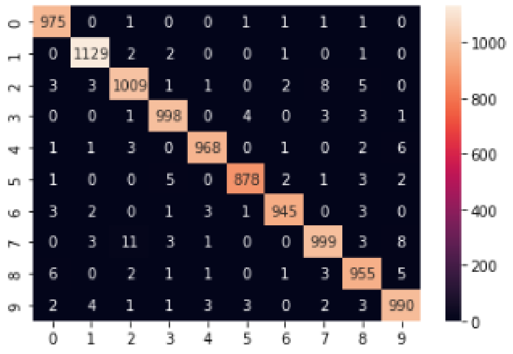
```
[15] test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0464 - accuracy: 0.9846
```

```python
[16] Y_pred = model.predict(x_test)
     # Convert predictions classes to one hot vectors
     Y_pred_classes = np.argmax(Y_pred,axis = 1)
     # Convert testing observations to one hot vectors
     Y_true = np.argmax(y_test,axis = 1)
     # compute the confusion matrix
     confusion_mtx = tf.math.confusion_matrix(Y_true, Y_pred_classes)
```

```python
sns.heatmap(confusion_mtx, annot=True, fmt='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd809a38a50>
```



	0	1	2	3	4	5	6	7	8	9
0	975	0	1	0	0	1	1	1	1	0
1	0	1129	2	2	0	0	1	0	1	0
2	3	3	1009	1	1	0	2	8	5	0
3	0	0	1	998	0	4	0	3	3	1
4	1	1	3	0	968	0	1	0	2	6
5	1	0	0	5	0	878	2	1	3	2
6	3	2	0	1	3	1	945	0	3	0
7	0	3	11	3	1	0	0	999	3	8
8	6	0	2	1	1	0	1	3	955	5
9	2	4	1	1	3	3	0	2	3	990

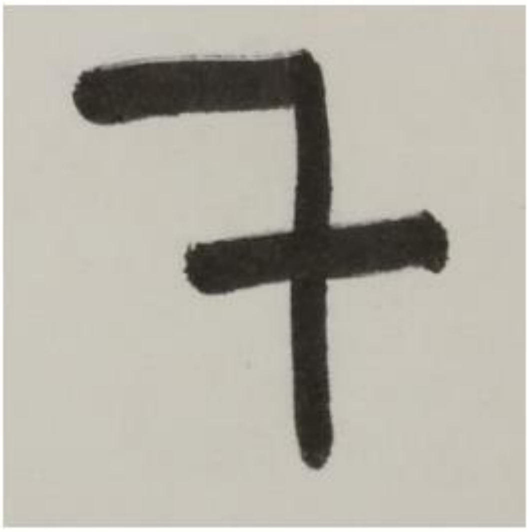
```
[21] from google.colab import drive
     drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", fo
```

```
[25] import cv2 as cv
     from google.colab.patches import cv2_imshow
```

```
[26] img = cv.imread("gdrive/MyDrive/7.jpg")
```

```
cv2_imshow(img)
```



```
[28] img.shape
```

```
(290, 290, 3)
```

```
[29] gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

```
gray.shape
```

```
(290, 290)
```

```
[31] img_rs = cv.resize(gray, (28, 28))
     img_rs.shape
```

```
(28, 28)
```

```
[32] cv2_imshow(img_rs)
     img_rs = np.expand_dims(img_rs,0)
     img_rs.shape
```

```
(1, 28, 28)
```

```
[33] img_rs = np.expand_dims(img_rs,-1)
     img_rs.shape
```

```
(1, 28, 28, 1)
```

```
[34] num = model.predict(img_rs)
     num
```

```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

```
[35] rs = [0,1,2,3,4,5,6,7,8,9]
```

```
[36] from numpy.core.fromnumeric import argmax
     result = rs[argmax(num)]
```

```
[37] result
```

```
0
```

Conclusion

Thus we have successfully completed the implementation of CNN algorithm using tensorflow

Assignment 11

Aim: Perform basic image processing using OpenCV

Objectives:

1. To learn about images processing
2. To perform various operations on images

Theory

OpenCV

OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision. [1] Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers. It promotes OpenVisionCapsules, which is a portable format, compatible with all other formats.

Image processing operations

Digital image processing is the use of a digital computer to *process* digital images through an *algorithm*. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modeled in the form of multidimensional systems. The generation and development of digital image processing are mainly affected by three factors:

first, the development of computers;

second, the development of mathematics (especially the creation and improvement of discrete mathematics theory);

third, the demand for a wide range of applications in environment, agriculture, military, industry and medical science has increased.Digital image processing allows the use of much more complex algorithms, and hence, can offer both more sophisticated performance at simple tasks, and the implementation of methods which would be impossible by analogue means.

In particular, digital image processing is a concrete application of, and a practical technology based on:

• Classification
• Feature extraction
• Multi-scale signal analysis
• Pattern recognition
• Projection

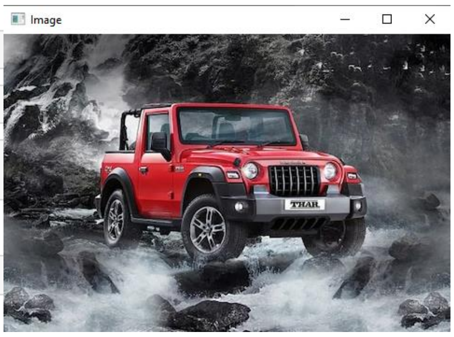
Some techniques which are used in digital image processing include:
• AnIsotropic diffusion
• Hidden Markov models
• Image editing
• Image restoration
• Independent component analysis
• Linear filtering
• Neural networks
• Partial differential equations
• Pixelation
• Point feature matching
• Principal components analysis
• Self-organizing maps
• Wavelets

OpenCV methods
Image Acquisition
OpenCV gives the flexibility to capture image directly from a pre-recorded video stream, camera input feed, or a directory path.
#Taking input from a directory path
img = cv2.imread('C:\Users\USER\Desktop\image.jpg',0)#Capturing input from a
video stream
cap = cv2.VideoCapture(0)
Histogram Equalization
Representation of intensity distribution vs no. of pixels of an image is termed as the histogram.
Equalization stretches out the intensity range in order to suit contrast levels appropriately.
Erosion and Dilation
Erosion and Dilation belong to the group of morphological transformations and widely used together for the treatment of noise or detection of intensity bumps.
Image De-noising

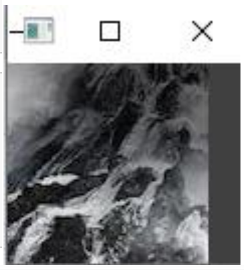
Noise has a very peculiar property that its mean is zero, and this is what helps in its removal by averaging it out.
OpenCV provides four variations of this technique.
1. **cv2.fastNlMeansDenoising()** — works with a single grayscale images
2. **cv2.fastNlMeansDenoisingColored()** — works with a color image.
3. **cv2.fastNlMeansDenoisingMulti()** — works with image sequence captured in short period of time (grayscale images)
4. **cv2.fastNlMeansDenoisingColoredMulti()** — same as above, but for color images.

Program and Outputs

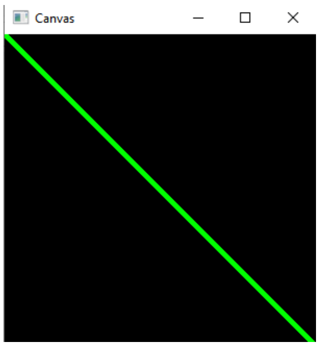
```
In [1]: import cv2 as cv

In [2]: print(cv.__version__)

        4.5.3

In [3]: import numpy as np

In [4]: image = cv.imread("car.jpg")

In [5]: cv.imshow("Image", image)
        cv.waitKey(0)

Out[5]: -1

In [6]: image.shape

Out[6]: (320, 480, 3)
```



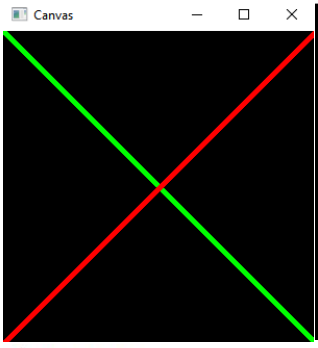
```
In [*]: corner = image[0:100, 0:100]
        cv.imshow("Corner", corner)
        cv.waitKey(0)
```

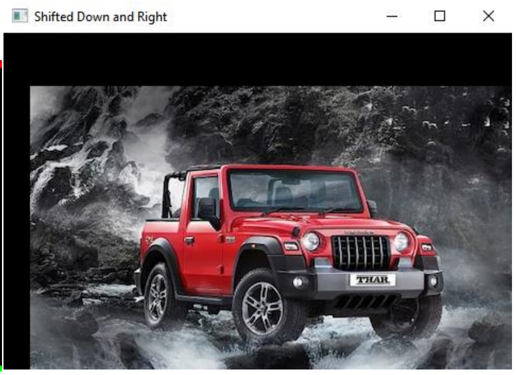


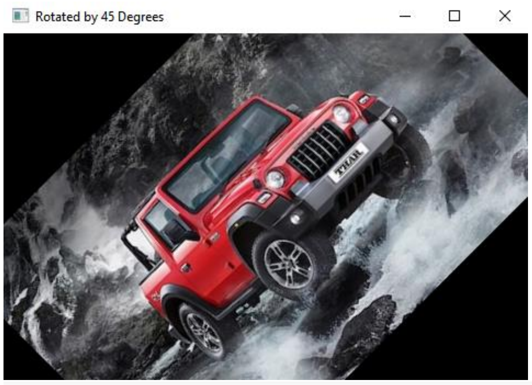
```
In [*]: canvas = np.zeros((300, 300, 3), dtype = "uint8")
        #RGB(255,255,255)
        green = (0, 255, 0)
        cv.line(canvas, (0, 0), (300, 300), green,4)
        cv.imshow("Canvas", canvas)
        cv.waitKey(0)
```



```
In [*]: red = (0, 0, 255)
        cv.line(canvas, (300, 0), (0, 300), red, 3)
        cv.imshow("Canvas", canvas)
        cv.waitKey(0)

In [*]: M = np.float32([[1, 0, 25], [0, 1, 50]])
        print(image.shape[0])
        print(image.shape[1])
        shifted = cv.warpAffine(image, M, (image.shape[1], image.shape[0]))
        cv.imshow("Shifted Down and Right", shifted)
        cv.waitKey(0)

In [*]: cv.imshow("Original", image)
        (h, w) = image.shape[:2]
        #h=image.shape[0]
        #w=image.shape[1]

        center = (w // 2, h // 2)
        M = cv.getRotationMatrix2D(center, 45, 1.0)
        rotated = cv.warpAffine(image, M, (w, h))
        cv.imshow("Rotated by 45 Degrees", rotated)
        cv.waitKey(0)
```







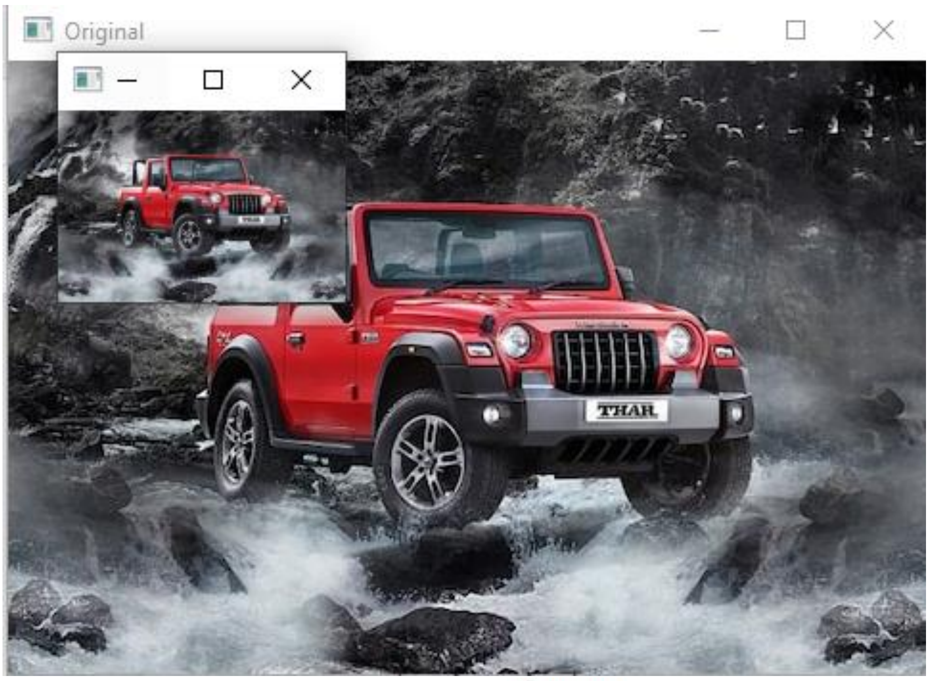
```
In [*]:
cv.imshow("Original", image)
print(image.shape)
r = 150.0 / image.shape[1]
dim = (150, int(image.shape[0] * r))
resized = cv.resize(image, dim, interpolation = cv.INTER_AREA)
print(resized.shape)
cv.imshow("Resized (Width)", resized)
cv.waitKey(0)
```

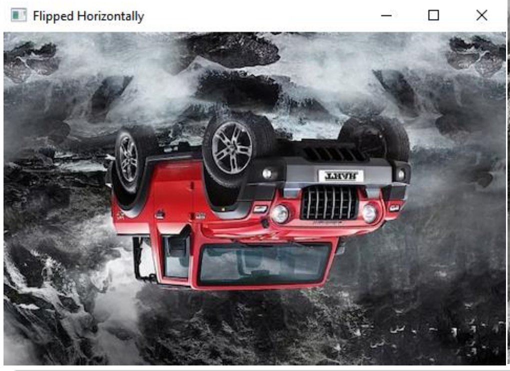
```
(320, 480, 3)
(100, 150, 3)
```

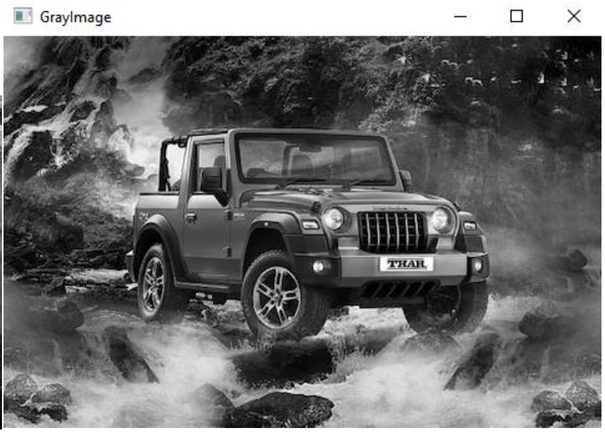
```
In [*]:
cv.imshow("Original", image)
flipped = cv.flip(image, 0)
cv.imshow("Flipped Horizontally", flipped)
cv.waitKey(0)
```

```
In [*]: #Turn image black and white
i_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
cv.imshow("GrayImage", i_gray)
cv.waitKey(0)
```

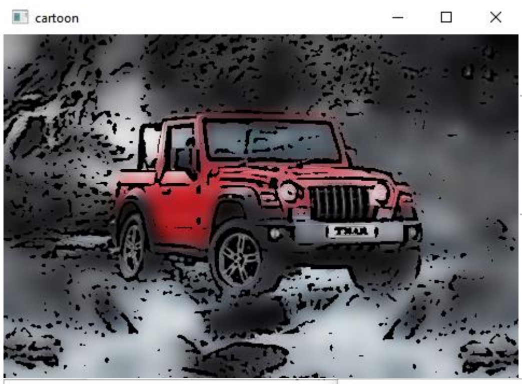
```
In [*]: i_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)
cv.imshow("RGB", i_rgb)
cv.waitKey(0)
```

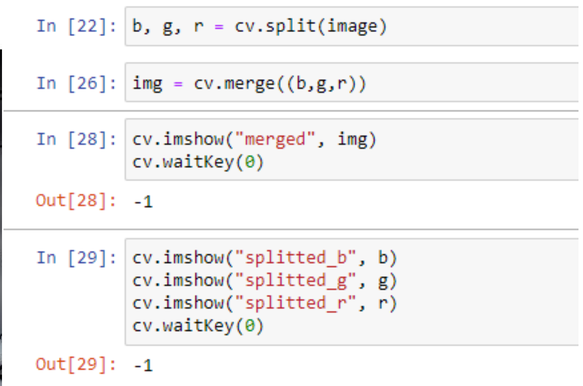




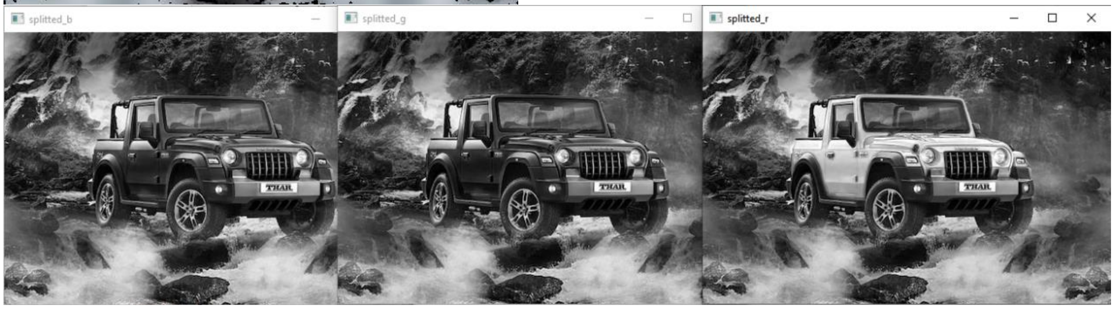


```
#Cartoonify
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
gray = cv.GaussianBlur(gray,(5,5),-1)
edges = cv.adaptiveThreshold(gray, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,9,10)
color = cv.bilateralFilter(image, 20, 245, 245)
cartoon = cv.bitwise_and(color, color, mask=edges)
cv.imshow("cartoon",cartoon)
cv.waitKey(0)
```





```
In [22]: b, g, r = cv.split(image)

In [26]: img = cv.merge((b,g,r))

In [28]: cv.imshow("merged", img)
         cv.waitKey(0)

Out[28]: -1

In [29]: cv.imshow("splitted_b", b)
         cv.imshow("splitted_g", g)
         cv.imshow("splitted_r", r)
         cv.waitKey(0)

Out[29]: -1
```



Conclusion

Thus we have successfully learned the usage of OpenCV and it’s functions