ML CAT 3 REPORT

DATASET : - TITANIC

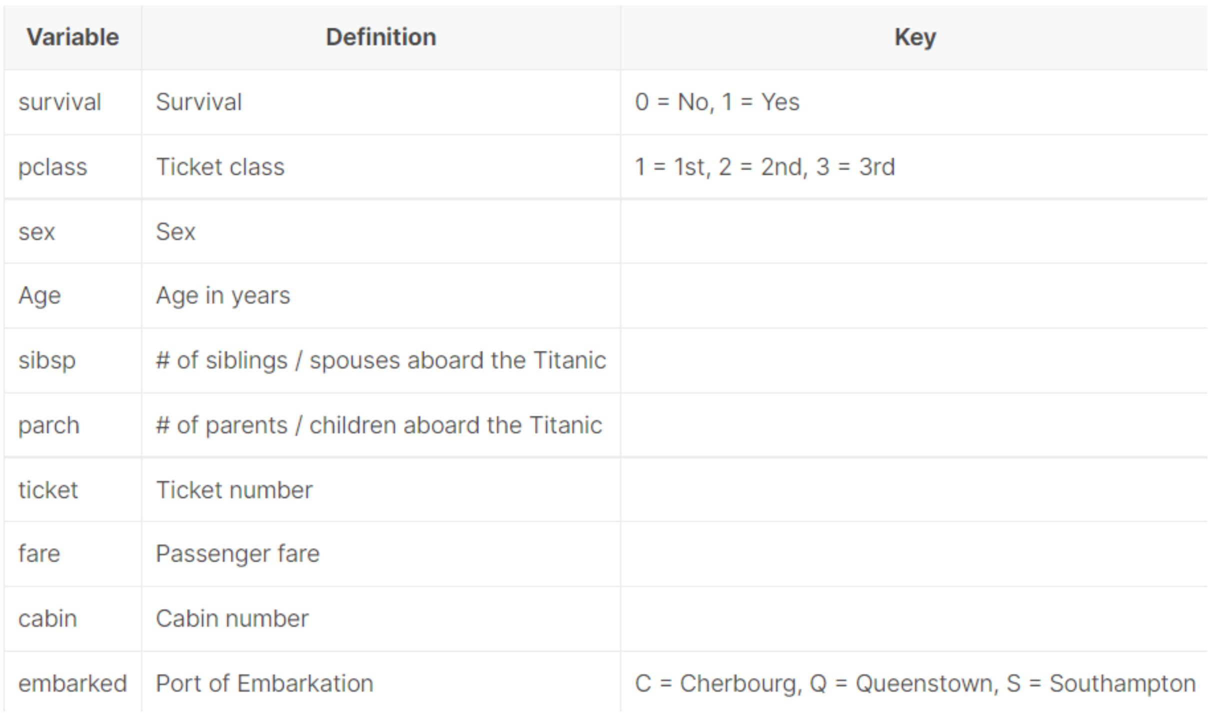
GITHUB LINK:-

https://github.com/2147229MCA/MACHINE-LEARNING-SEM-4-LAB-PROGRAMS/tree/main/ML%20LAB%20FILES

ALTERNATE GITHUB LINK:-

https://github.com/2147229MCA/MACHINE-LEARNING-SEM-4-LAB-PROGRAMS

DATASET MEMBERS:-



Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

INFORMATION:-

pclass: A proxy for socio-economic status (SES)
1st = Upper
2nd = Middle
3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

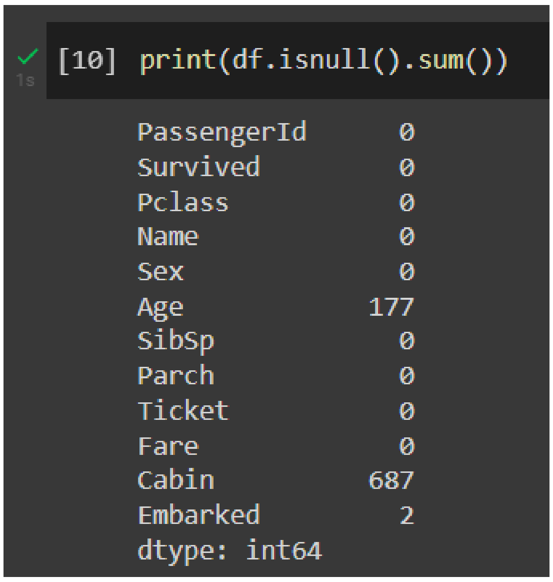
sibsp: The dataset defines family relations in this way...

Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...
Parent = mother, father
Child = daughter, son, stepdaughter, stepson
Some children travelled only with a nanny, therefore parch=0 for them.

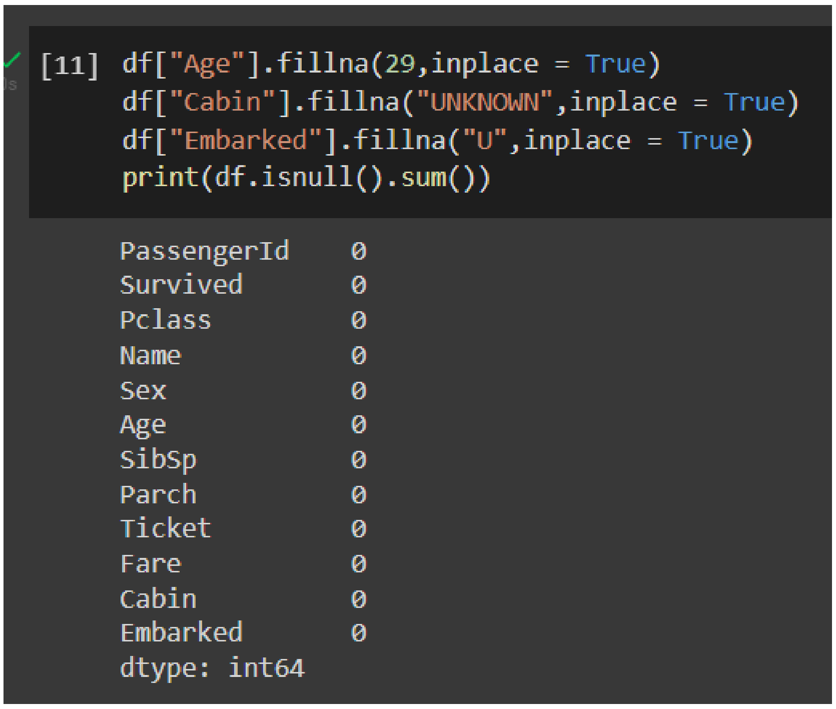
SOURCE:- https://www.kaggle.com/competitions/titanic/data

PREPROCESSING TECHNIQUES USED:-



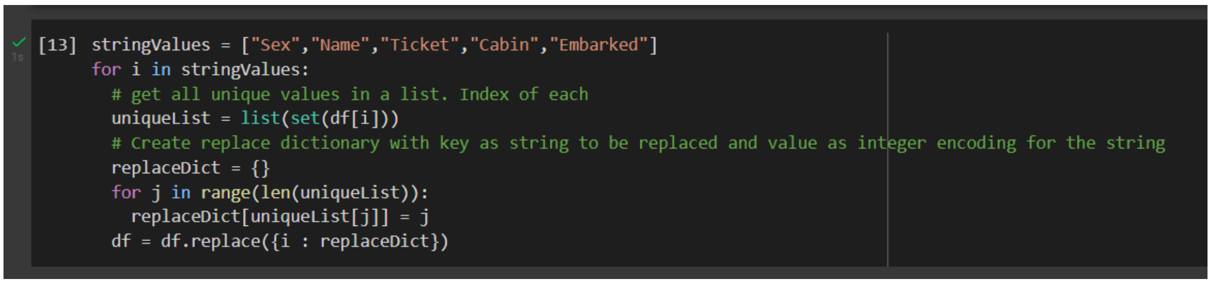
```
[10] print(df.isnull().sum())

PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

TO FIND NULL VALUES AND REPLACE THEM



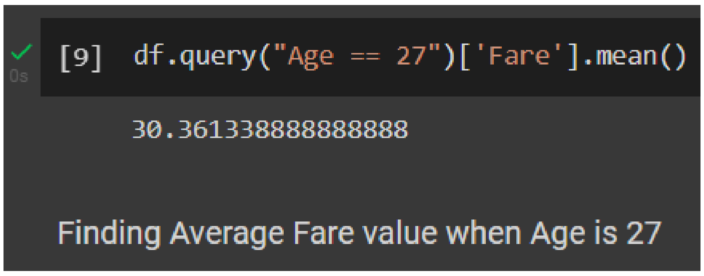
```
[11] df["Age"].fillna(29,inplace = True)
     df["Cabin"].fillna("UNKNOWN",inplace = True)
     df["Embarked"].fillna("U",inplace = True)
     print(df.isnull().sum())

PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

CONVERTING CATEGORICAL TO NUMERICAL VALUES



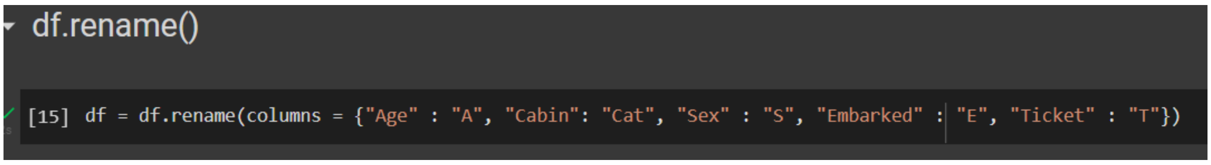
```
[13] stringValues = ["Sex","Name","Ticket","Cabin","Embarked"]
     for i in stringValues:
       # get all unique values in a list. Index of each
       uniqueList = list(set(df[i]))
       # Create replace dictionary with key as string to be replaced and value as integer encoding for the string
       replaceDict = {}
       for j in range(len(uniqueList)):
         replaceDict[uniqueList[j]] = j
       df = df.replace({i : replaceDict})
```

EDA PERFORMED:-



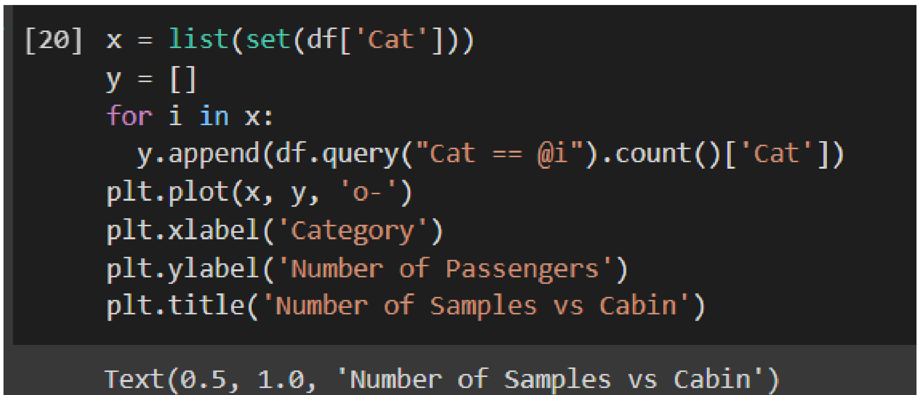
```
[9]  df.query("Age == 27")['Fare'].mean()

     30.361338888888888
```

Finding Average Fare value when Age is 27

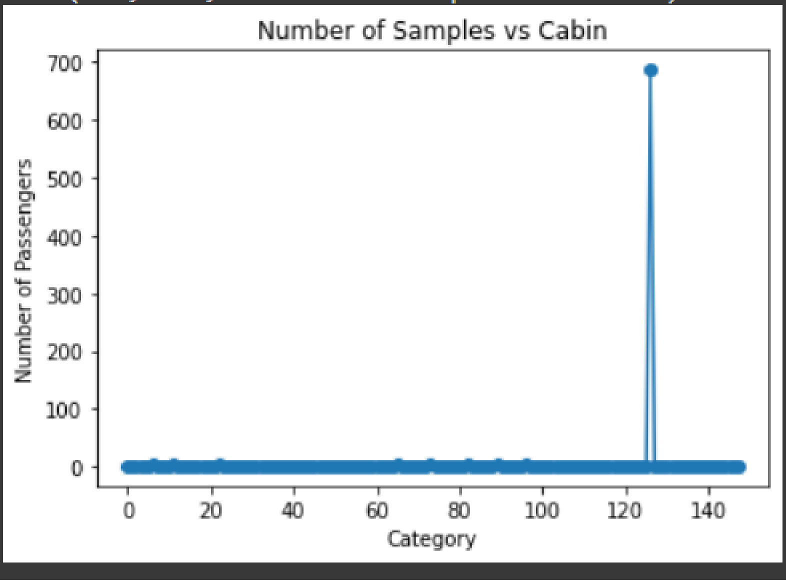


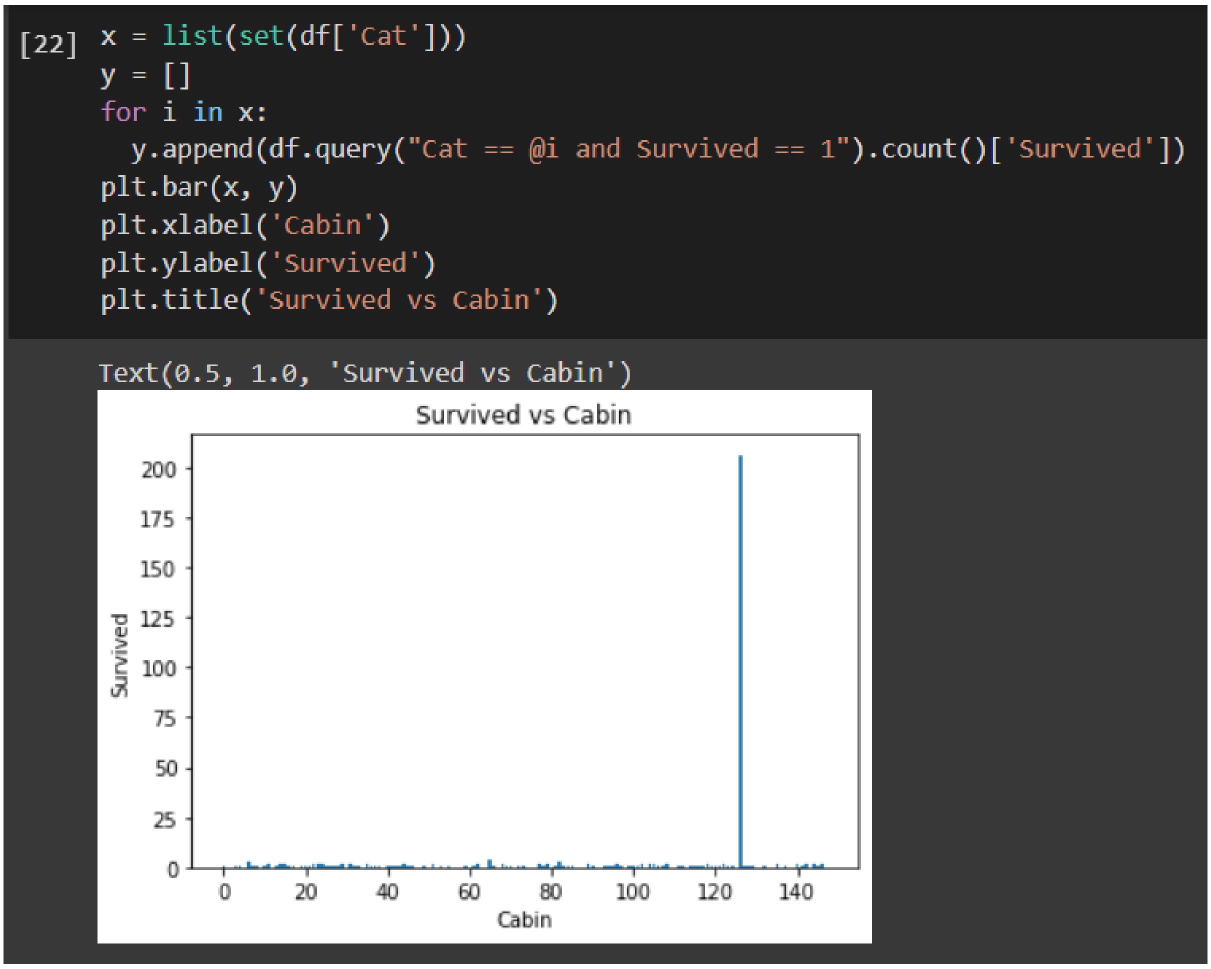
df.rename()

```
[15] df = df.rename(columns = {"Age" : "A", "Cabin": "Cat", "Sex" : "S", "Embarked" : "E", "Ticket" : "T"})
```



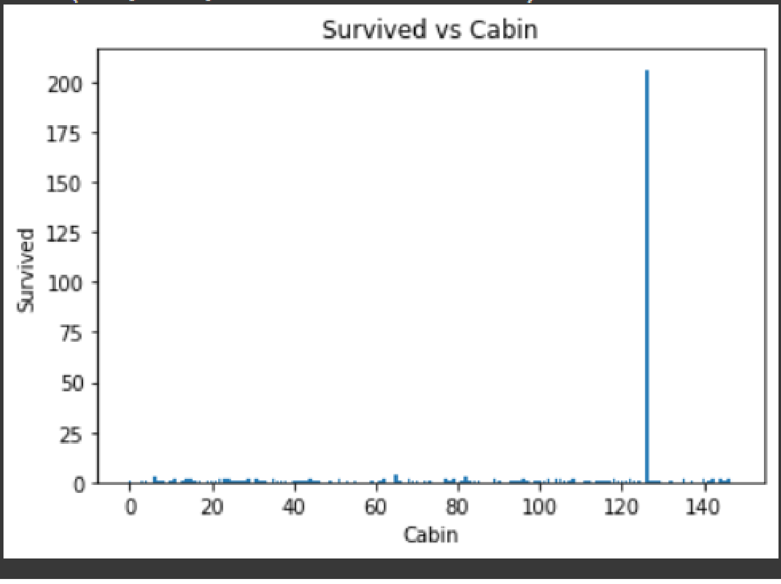
```
[20] x = list(set(df['Cat']))
     y = []
     for i in x:
       y.append(df.query("Cat == @i").count()['Cat'])
     plt.plot(x, y, 'o-')
     plt.xlabel('Category')
     plt.ylabel('Number of Passengers')
     plt.title('Number of Samples vs Cabin')

     Text(0.5, 1.0, 'Number of Samples vs Cabin')
```

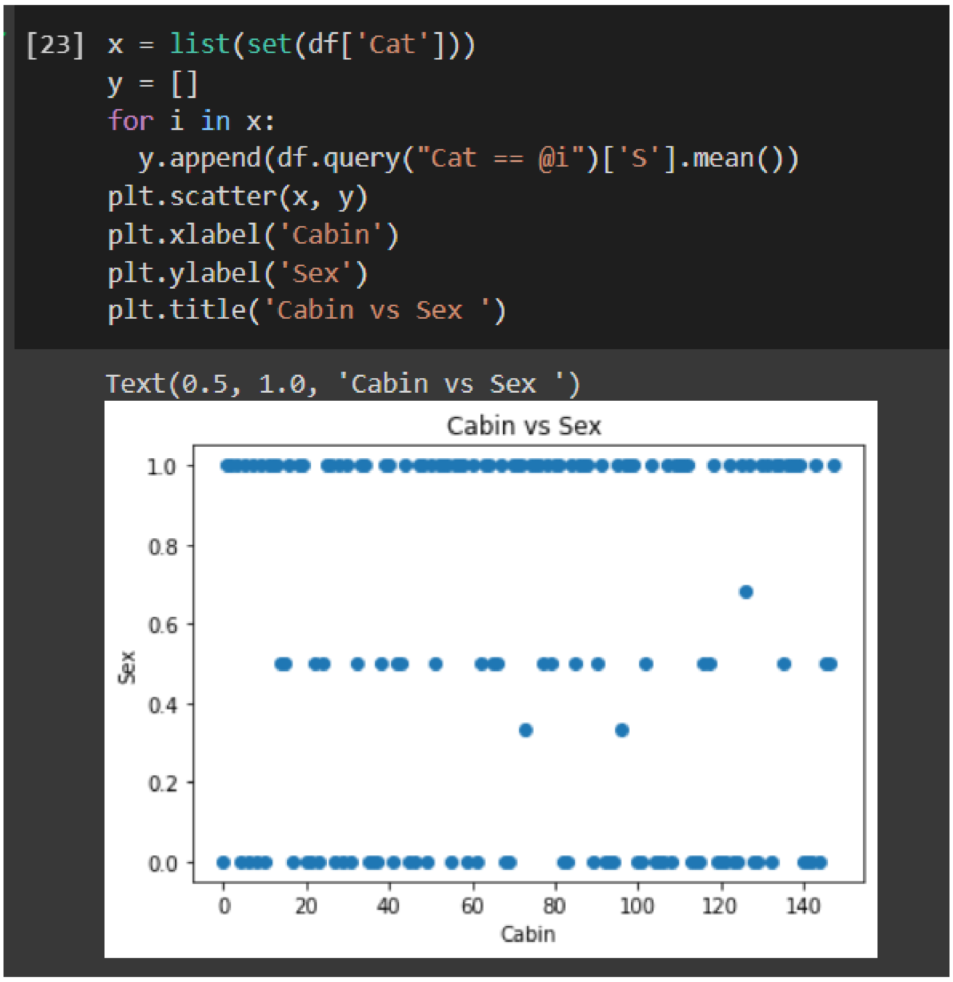




```
[22] x = list(set(df['Cat']))
     y = []
     for i in x:
       y.append(df.query("Cat == @i and Survived == 1").count()['Survived'])
     plt.bar(x, y)
     plt.xlabel('Cabin')
     plt.ylabel('Survived')
     plt.title('Survived vs Cabin')
```

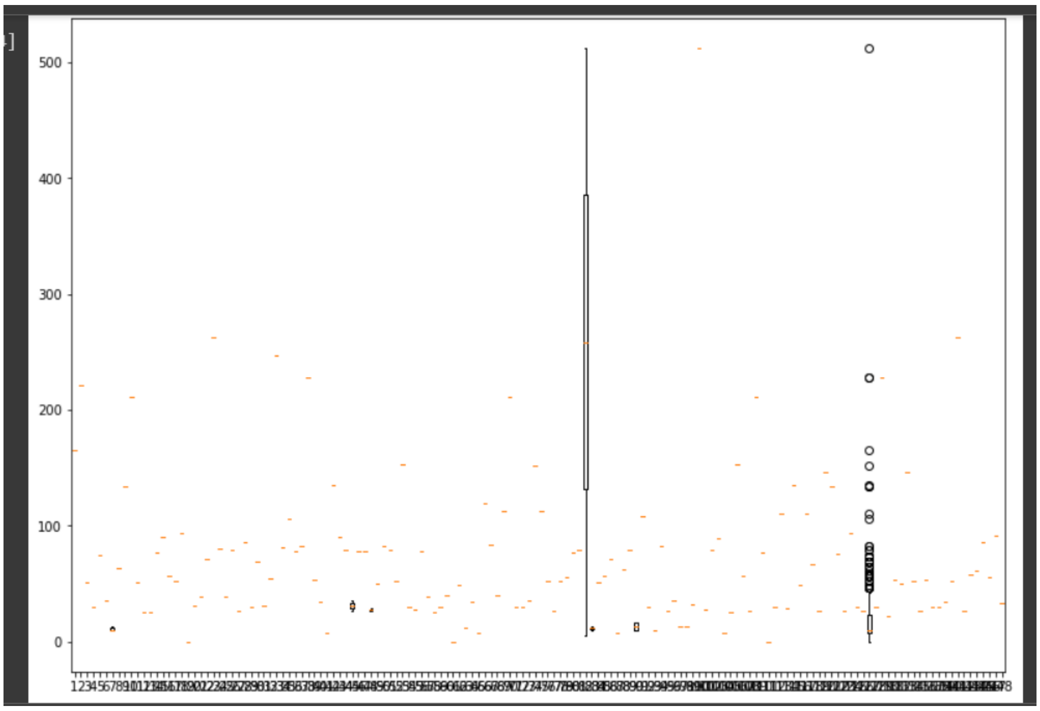
```
Text(0.5, 1.0, 'Survived vs Cabin')
```



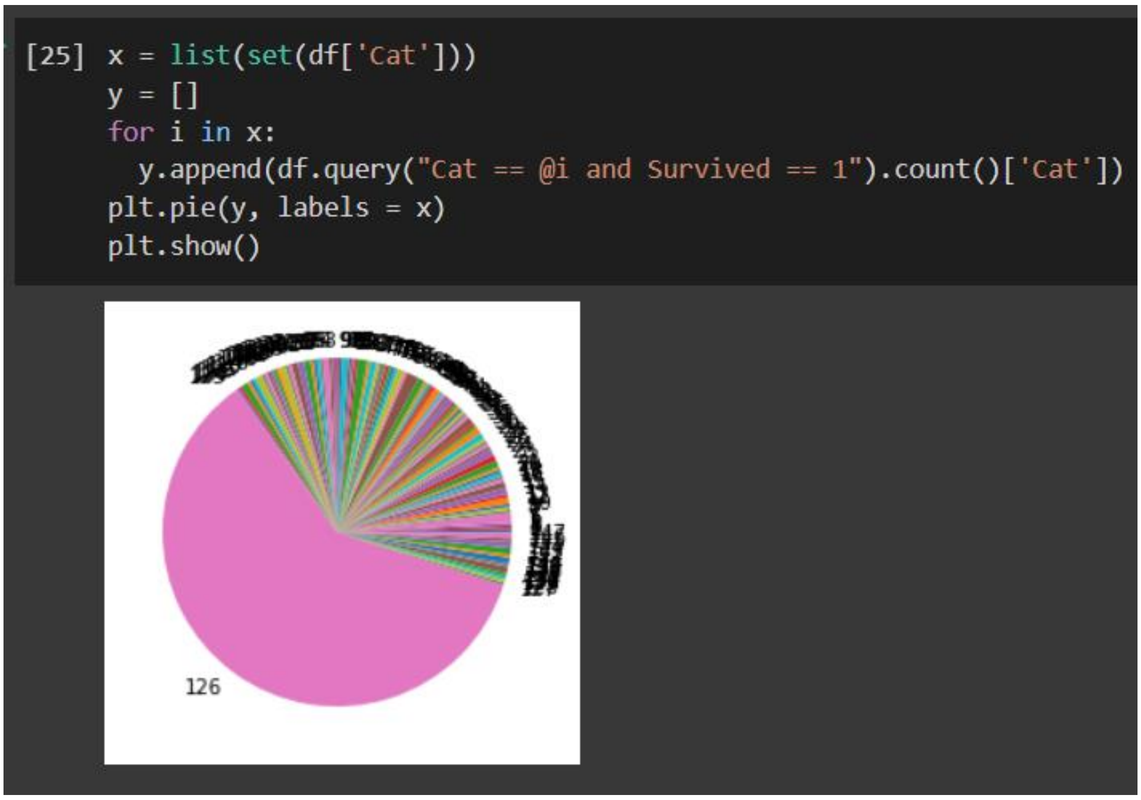


```
[23] x = list(set(df['Cat']))
     y = []
     for i in x:
       y.append(df.query("Cat == @i")['S'].mean())
     plt.scatter(x, y)
     plt.xlabel('Cabin')
     plt.ylabel('Sex')
     plt.title('Cabin vs Sex ')

     Text(0.5, 1.0, 'Cabin vs Sex ')
```

BOX PLOT

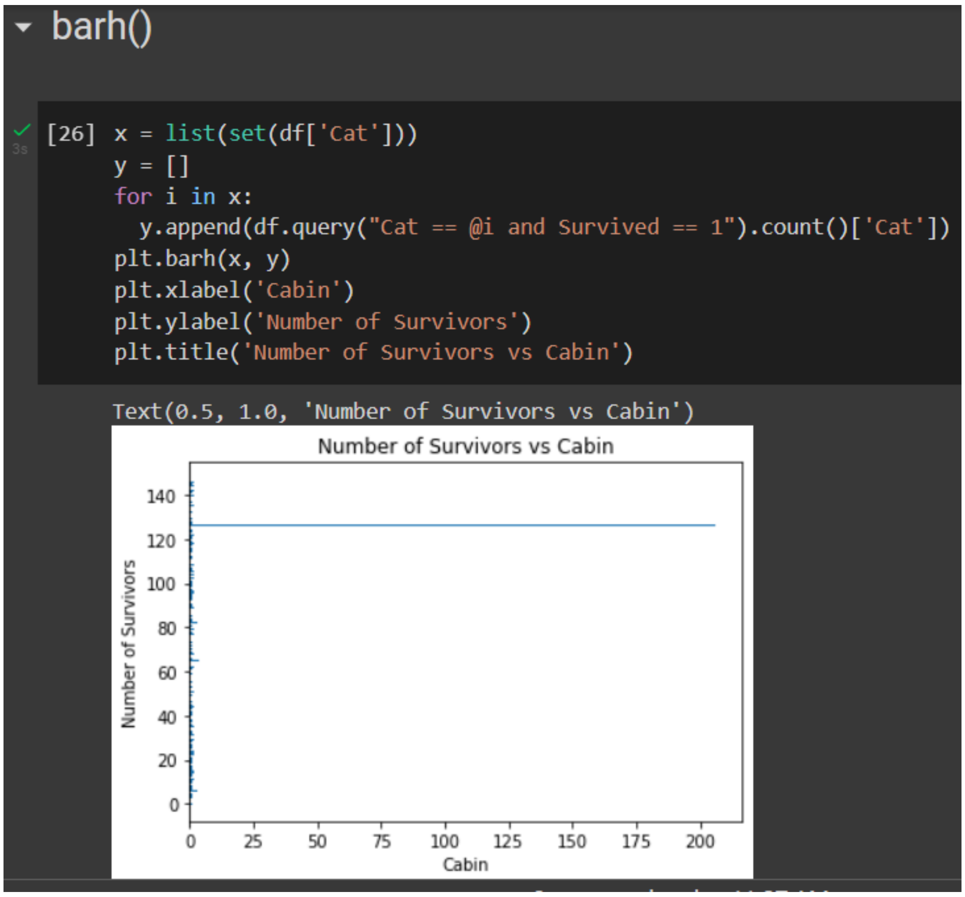


PIE CHART



```
[25] x = list(set(df['Cat']))
     y = []
     for i in x:
       y.append(df.query("Cat == @i and Survived == 1").count()['Cat'])
     plt.pie(y, labels = x)
     plt.show()
```

BARH()

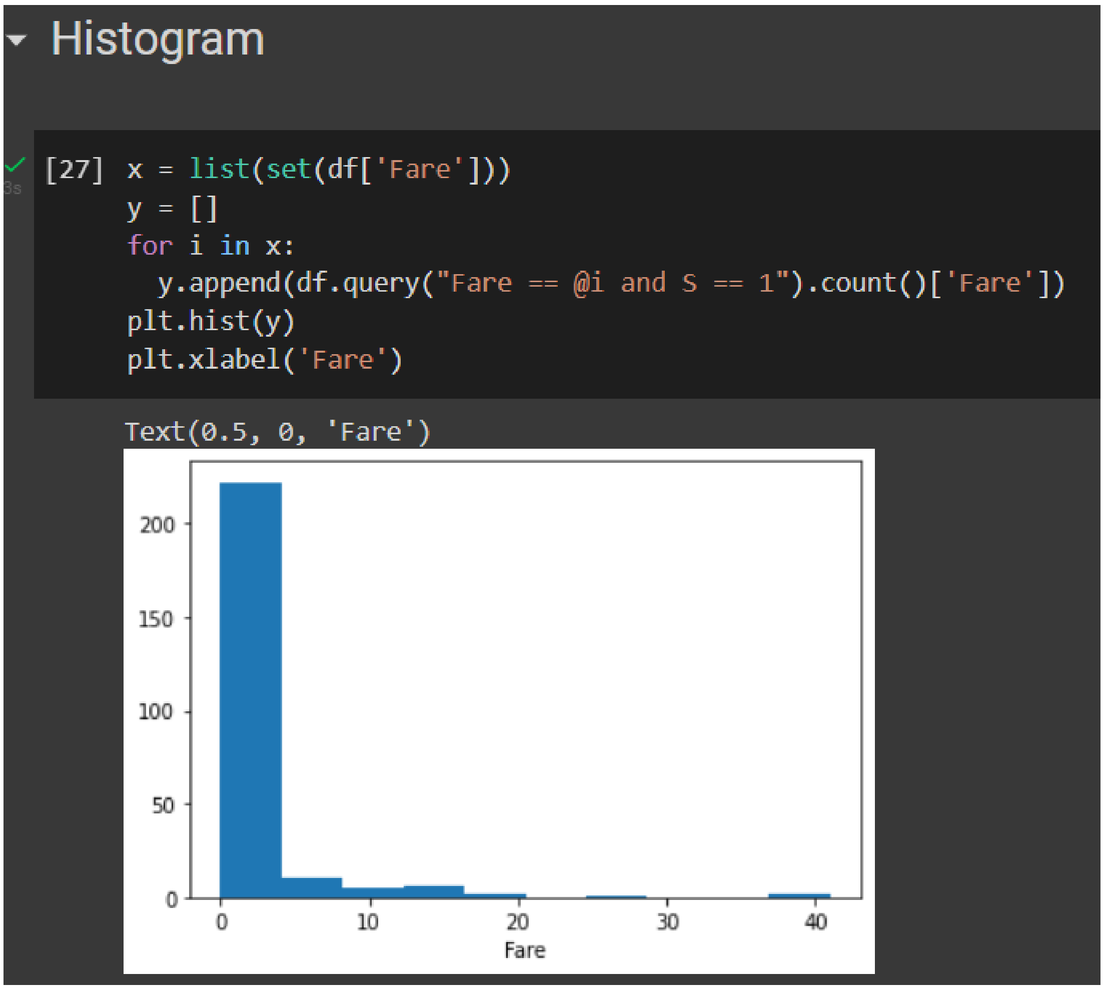


barh()

```
[26] x = list(set(df['Cat']))
     y = []
     for i in x:
       y.append(df.query("Cat == @i and Survived == 1").count()['Cat'])
     plt.barh(x, y)
     plt.xlabel('Cabin')
     plt.ylabel('Number of Survivors')
     plt.title('Number of Survivors vs Cabin')
```

```
Text(0.5, 1.0, 'Number of Survivors vs Cabin')
```

HISTOGRAM

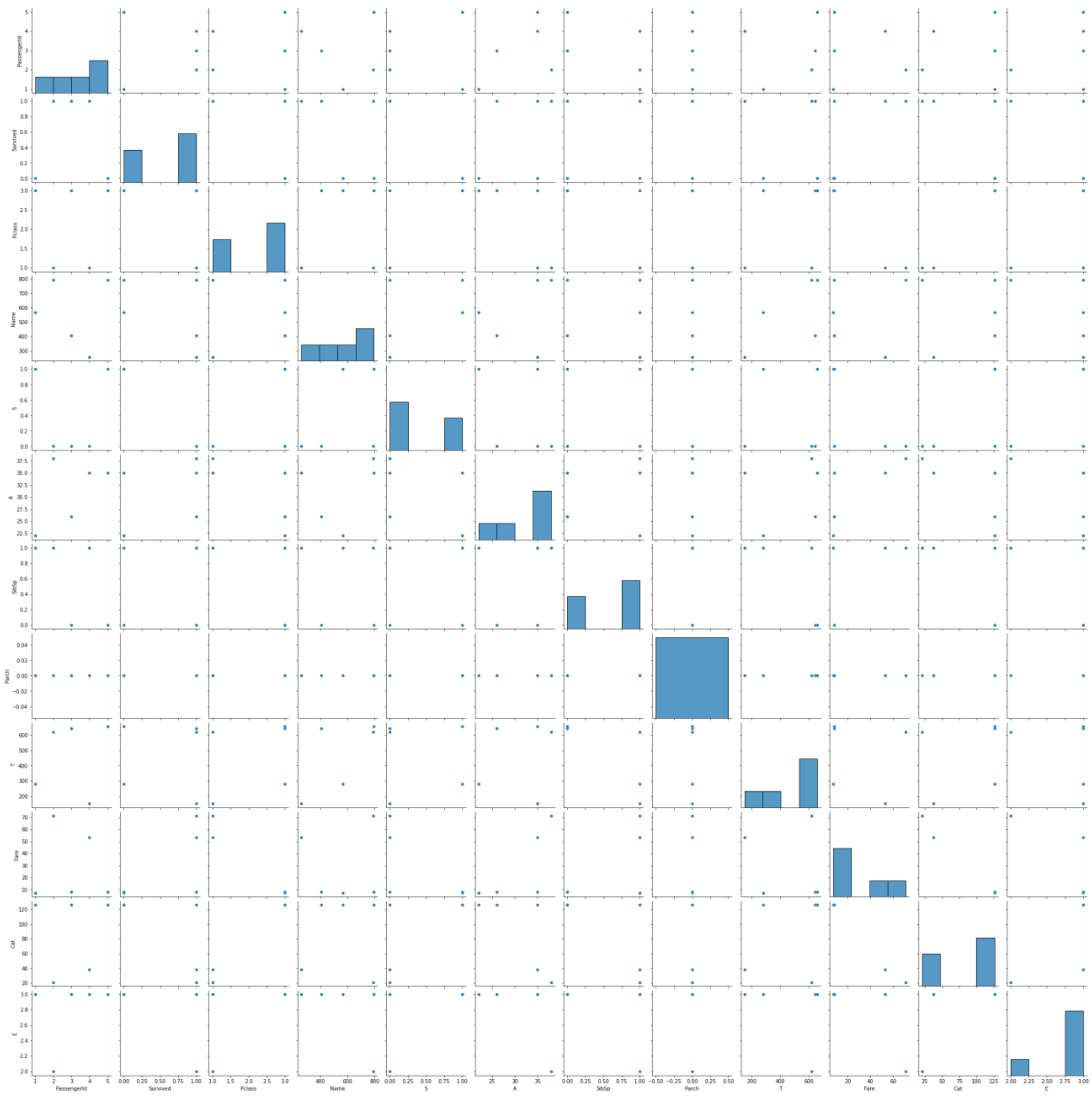


Histogram

```
[27] x = list(set(df['Fare']))
     y = []
     for i in x:
       y.append(df.query("Fare == @i and S == 1").count()['Fare'])
     plt.hist(y)
     plt.xlabel('Fare')
```

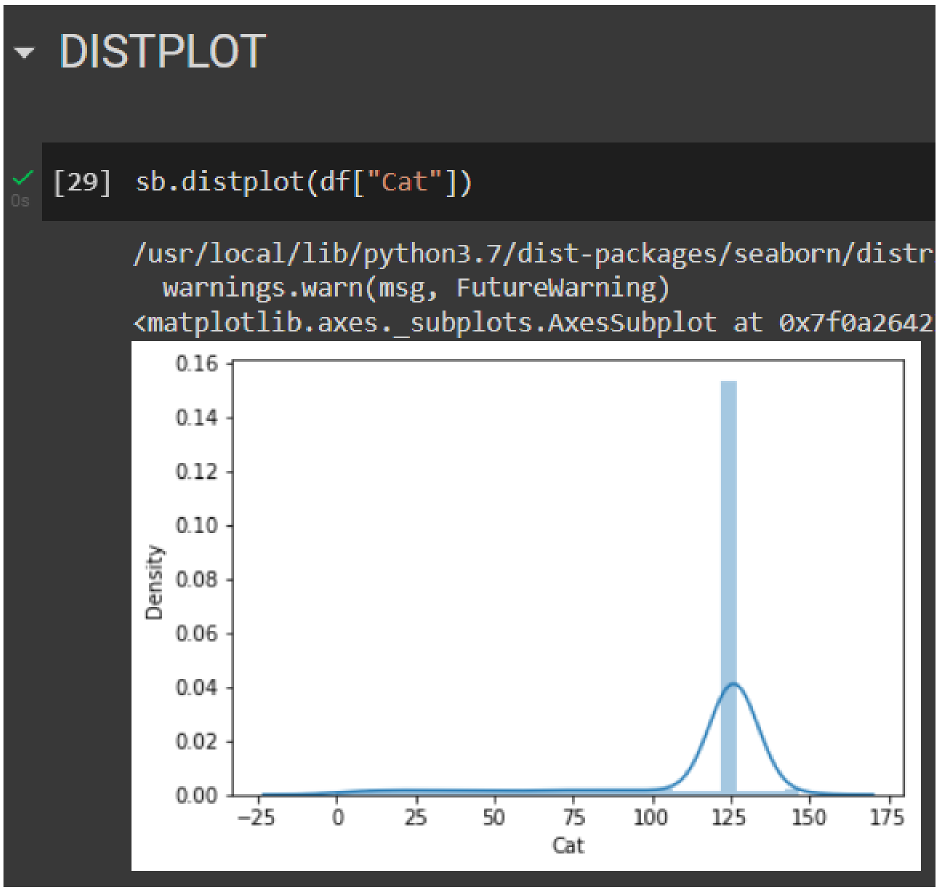
```
Text(0.5, 0, 'Fare')
```

PAIRPLOT



PassengerId
Survived
Pclass
Name
S
A
SibSp
Parch
T
Fare
Cat
E

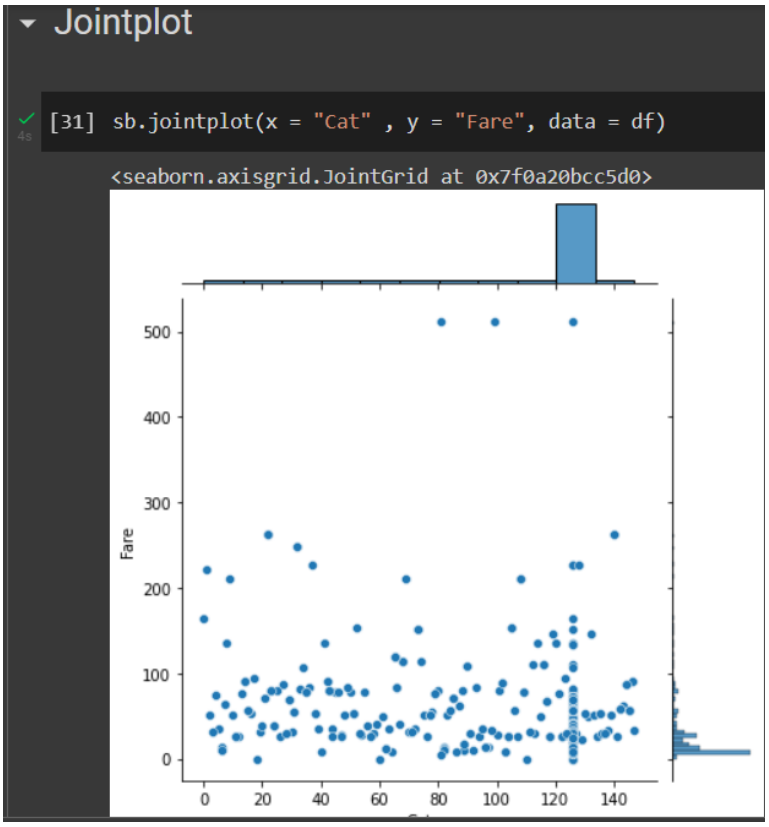
DISTPLOT:-



DISTPLOT

```
[29] sb.distplot(df["Cat"])

     /usr/local/lib/python3.7/dist-packages/seaborn/distr
       warnings.warn(msg, FutureWarning)
     <matplotlib.axes._subplots.AxesSubplot at 0x7f0a2642
```

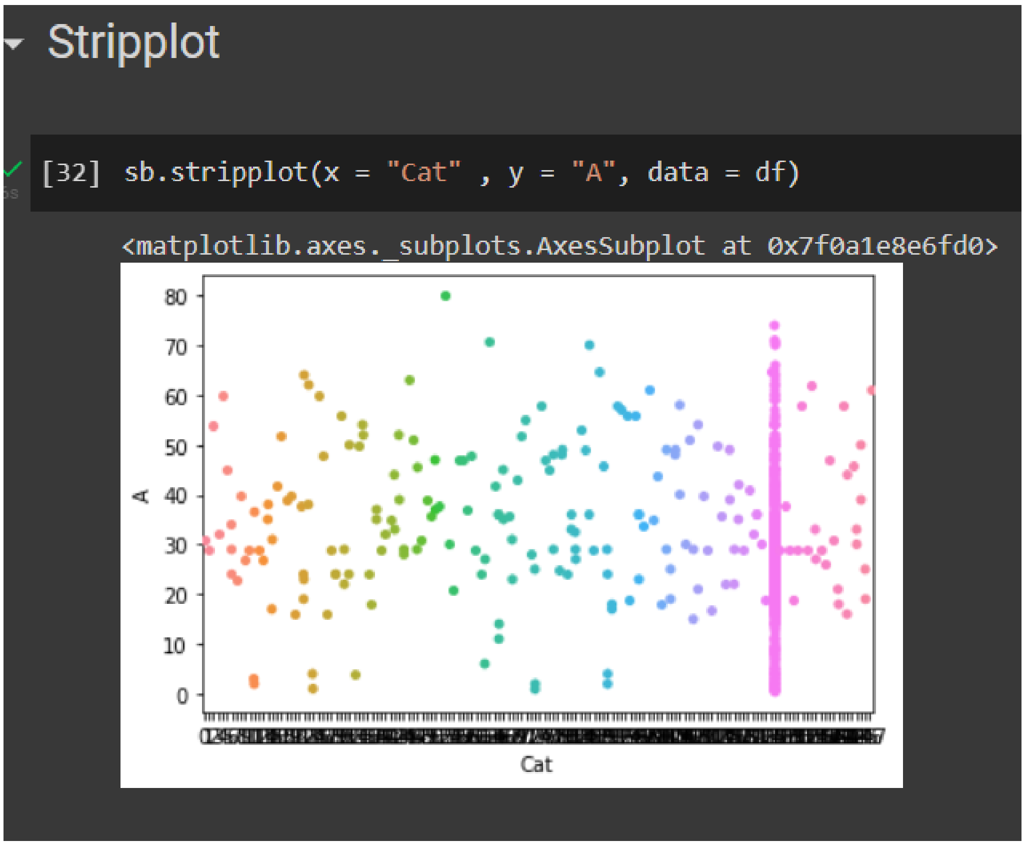
JOINTPLOT:-



Jointplot

```
[31] sb.jointplot(x = "Cat" , y = "Fare", data = df)

     <seaborn.axisgrid.JointGrid at 0x7f0a20bcc5d0>
```

STRIPPLOT

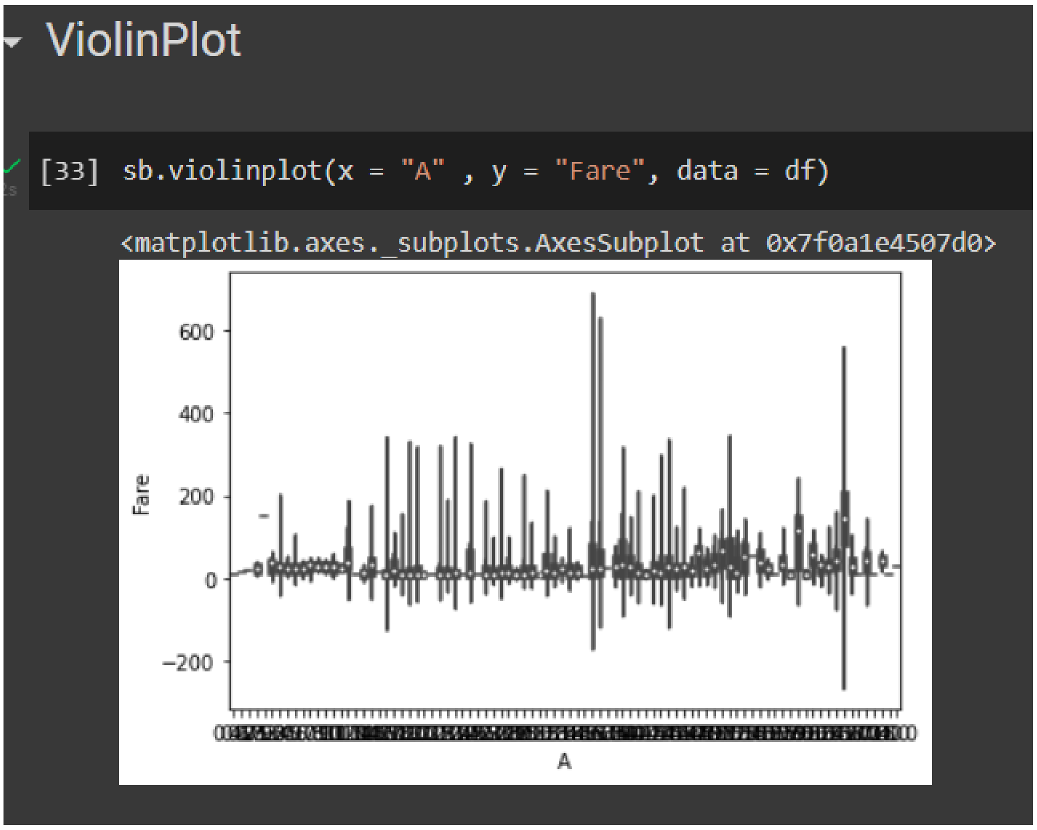


Stripplot

```
[32] sb.stripplot(x = "Cat" , y = "A", data = df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a1e8e6fd0>
```

VIOLINPLOT



ViolinPlot

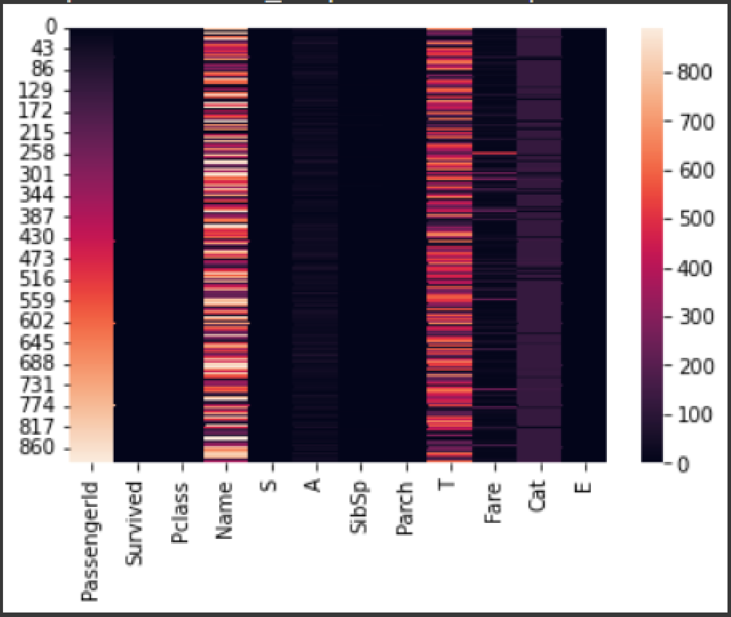
```
[33] sb.violinplot(x = "A" , y = "Fare", data = df)
```

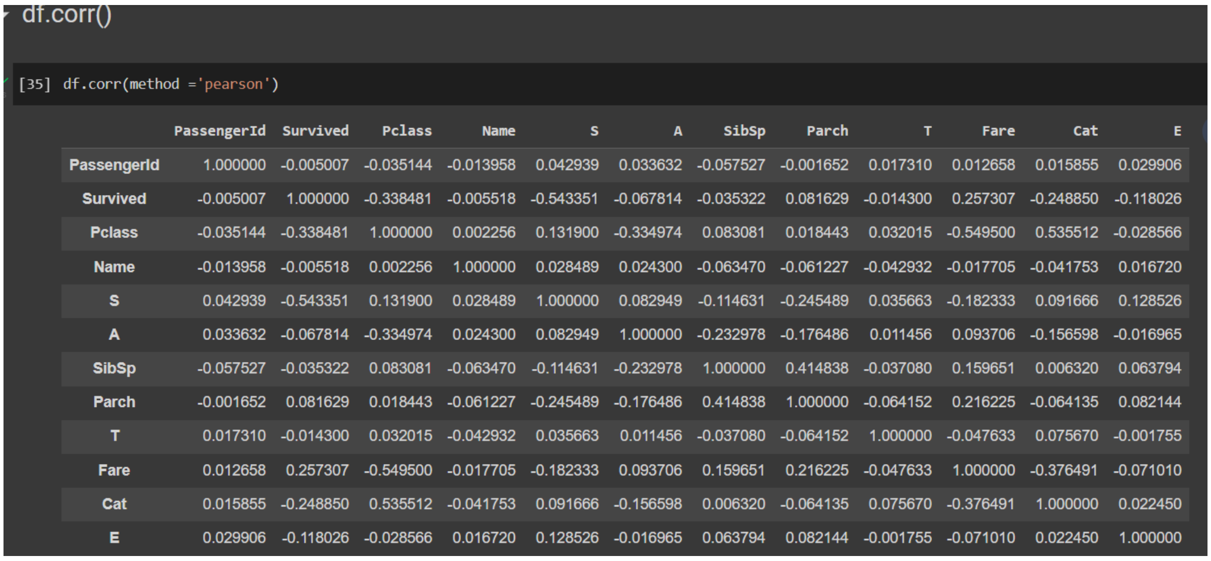
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a1e4507d0>
```

HEATMAP

HeatMap

```
[34] sb.heatmap(data = df)

<matplotlib.axes._subplots.AxesSubplot at 0x7f0a1e47f6d0>
```

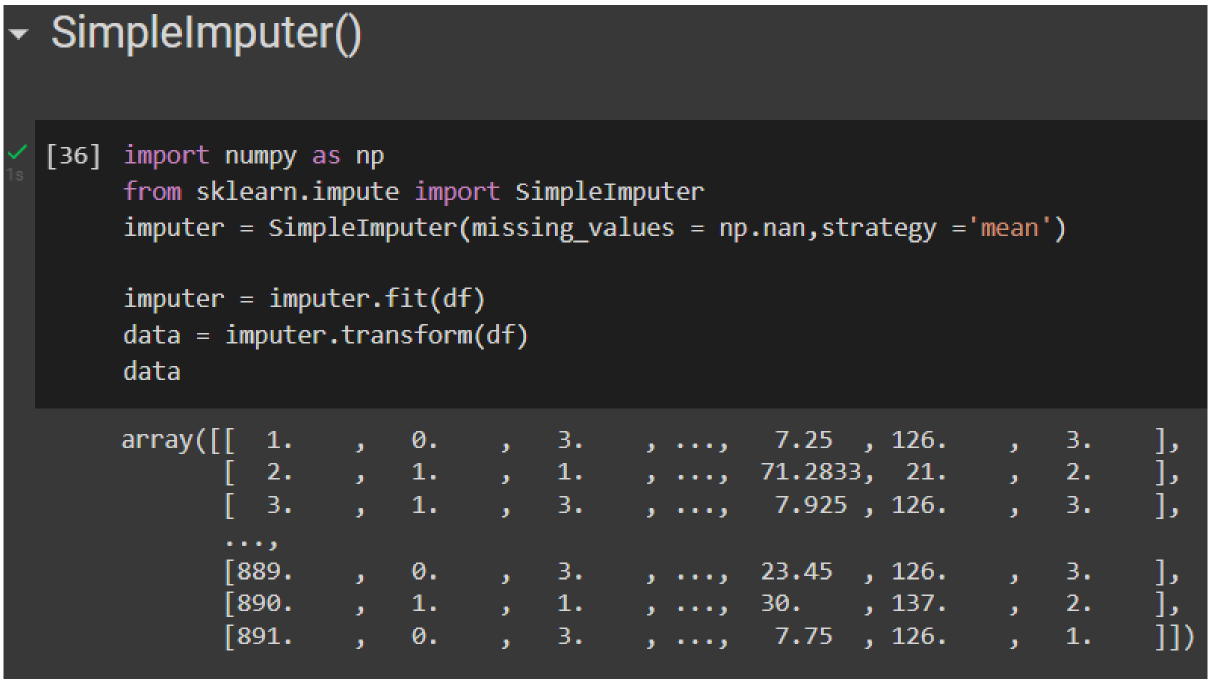




df.corr()

```
[35] df.corr(method ='pearson')
```

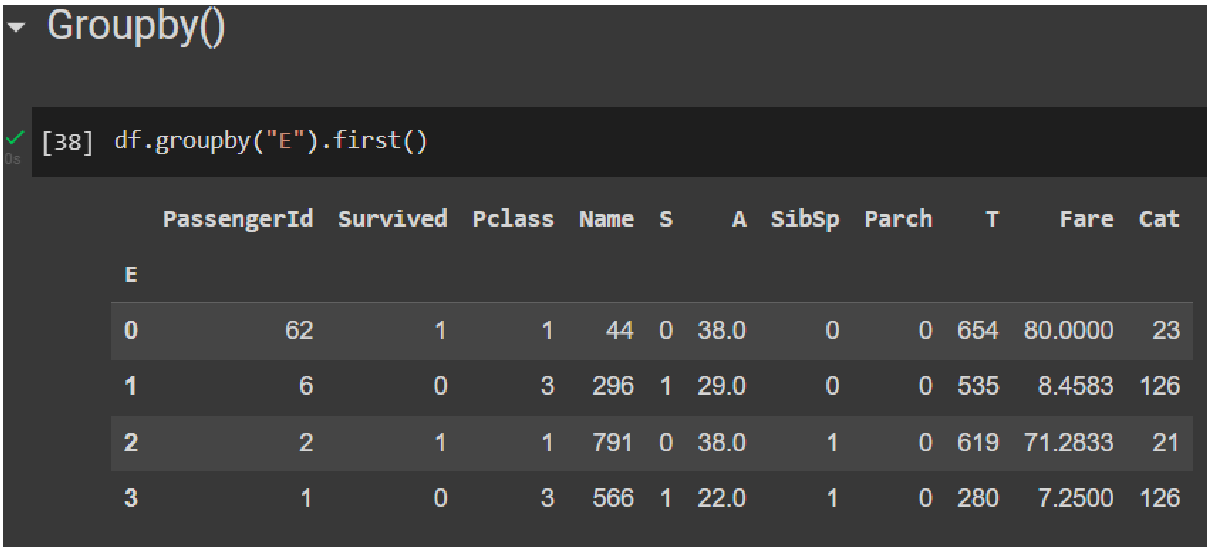
	PassengerId	Survived	Pclass	Name	S	A	SibSp	Parch	T	Fare	Cat	E
PassengerId	1.000000	-0.005007	-0.035144	-0.013958	0.042939	0.033632	-0.057527	-0.001652	0.017310	0.012658	0.015855	0.029906
Survived	-0.005007	1.000000	-0.338481	-0.005518	-0.543351	-0.067814	-0.035322	0.081629	-0.014300	0.257307	-0.248850	-0.118026
Pclass	-0.035144	-0.338481	1.000000	0.002256	0.131900	-0.334974	0.083081	0.018443	0.032015	-0.549500	0.535512	-0.028566
Name	-0.013958	-0.005518	0.002256	1.000000	0.028489	0.024300	-0.063470	-0.061227	-0.042932	-0.017705	-0.041753	0.016720
S	0.042939	-0.543351	0.131900	0.028489	1.000000	0.082949	-0.114631	-0.245489	0.035663	-0.182333	0.091666	0.128526
A	0.033632	-0.067814	-0.334974	0.024300	0.082949	1.000000	-0.232978	-0.176486	0.011456	0.093706	-0.156598	-0.016965
SibSp	-0.057527	-0.035322	0.083081	-0.063470	-0.114631	-0.232978	1.000000	0.414838	-0.037080	0.159651	0.006320	0.063794
Parch	-0.001652	0.081629	0.018443	-0.061227	-0.245489	-0.176486	0.414838	1.000000	-0.064152	0.216225	-0.064135	0.082144
T	0.017310	-0.014300	0.032015	-0.042932	0.035663	0.011456	-0.037080	-0.064152	1.000000	-0.047633	0.075670	-0.001755
Fare	0.012658	0.257307	-0.549500	-0.017705	-0.182333	0.093706	0.159651	0.216225	-0.047633	1.000000	-0.376491	-0.071010
Cat	0.015855	-0.248850	0.535512	-0.041753	0.091666	-0.156598	0.006320	-0.064135	0.075670	-0.376491	1.000000	0.022450
E	0.029906	-0.118026	-0.028566	0.016720	0.128526	-0.016965	0.063794	0.082144	-0.001755	-0.071010	0.022450	1.000000



SimpleImputer()

```
[36] import numpy as np
     from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(missing_values = np.nan,strategy ='mean')

     imputer = imputer.fit(df)
     data = imputer.transform(df)
     data
```

```
array([[  1.    ,   0.    ,   3.    , ...,   7.25  , 126.    ,   3.    ],
       [  2.    ,   1.    ,   1.    , ...,  71.2833,  21.    ,   2.    ],
       [  3.    ,   1.    ,   3.    , ...,   7.925 , 126.    ,   3.    ],
       ...,
       [889.    ,   0.    ,   3.    , ...,  23.45  , 126.    ,   3.    ],
       [890.    ,   1.    ,   1.    , ...,  30.    , 137.    ,   2.    ],
       [891.    ,   0.    ,   3.    , ...,   7.75  , 126.    ,   1.    ]])
```



Groupby()

```
[38] df.groupby("E").first()
```

	PassengerId	Survived	Pclass	Name	S	A	SibSp	Parch	T	Fare	Cat
E											
0	62	1	1	44	0	38.0	0	0	654	80.0000	23
1	6	0	3	296	1	29.0	0	0	535	8.4583	126
2	2	1	1	791	0	38.0	1	0	619	71.2833	21
3	1	0	3	566	1	22.0	1	0	280	7.2500	126

ALGORITHMS:-

1)KNN CLASSIFIER ALGORITHM:-

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

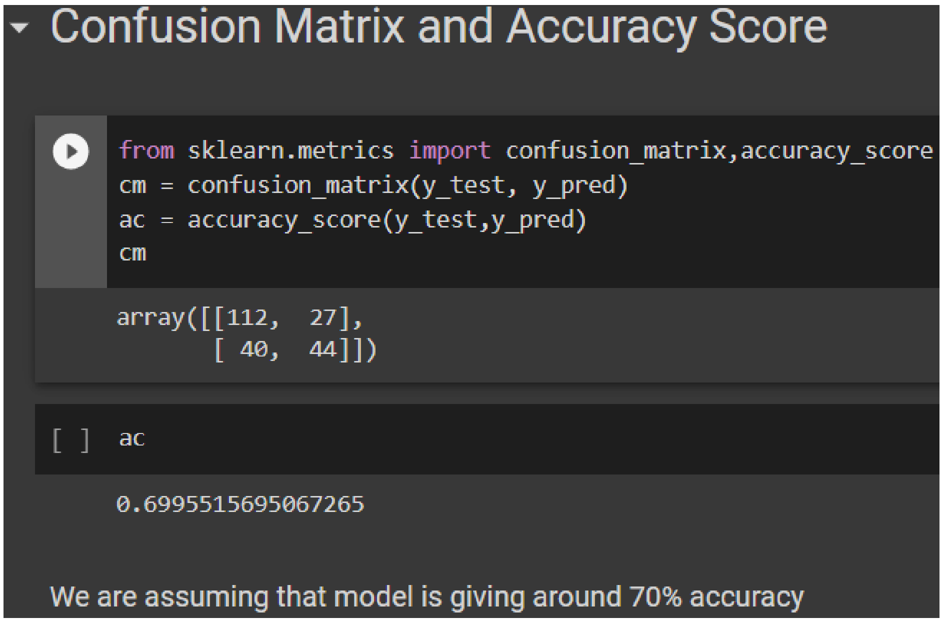
How Does it Work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Values Derived From Our Dataset:-

Confusion Matrix and Accuracy Score



Confusion Matrix and Accuracy Score

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
cm
```

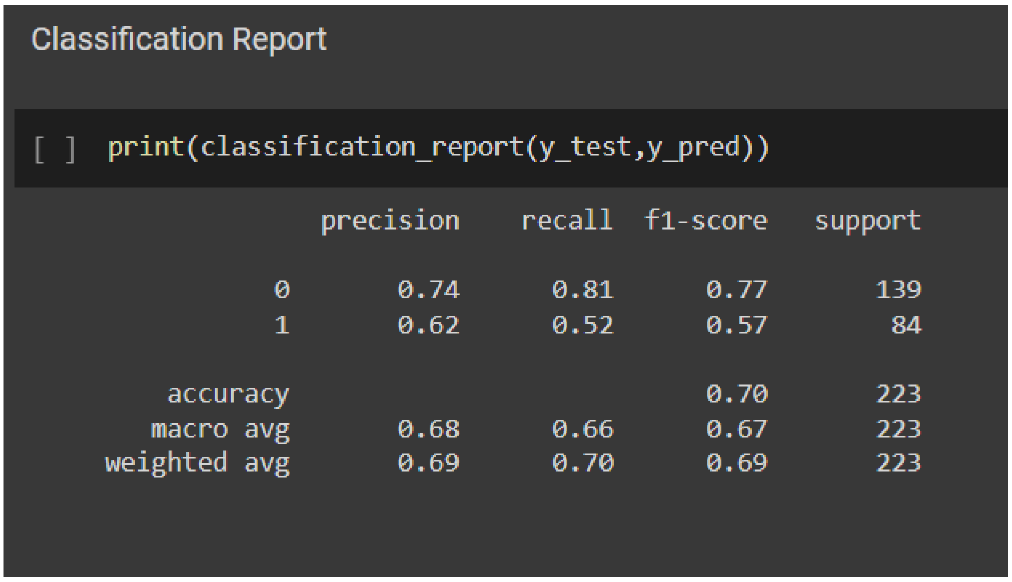
```
array([[112,  27],
       [ 40,  44]])
```

```
[ ] ac
```

```
0.6995515695067265
```

We are assuming that model is giving around 70% accuracy

We also performed classification Report to check for different Evaluation Measures

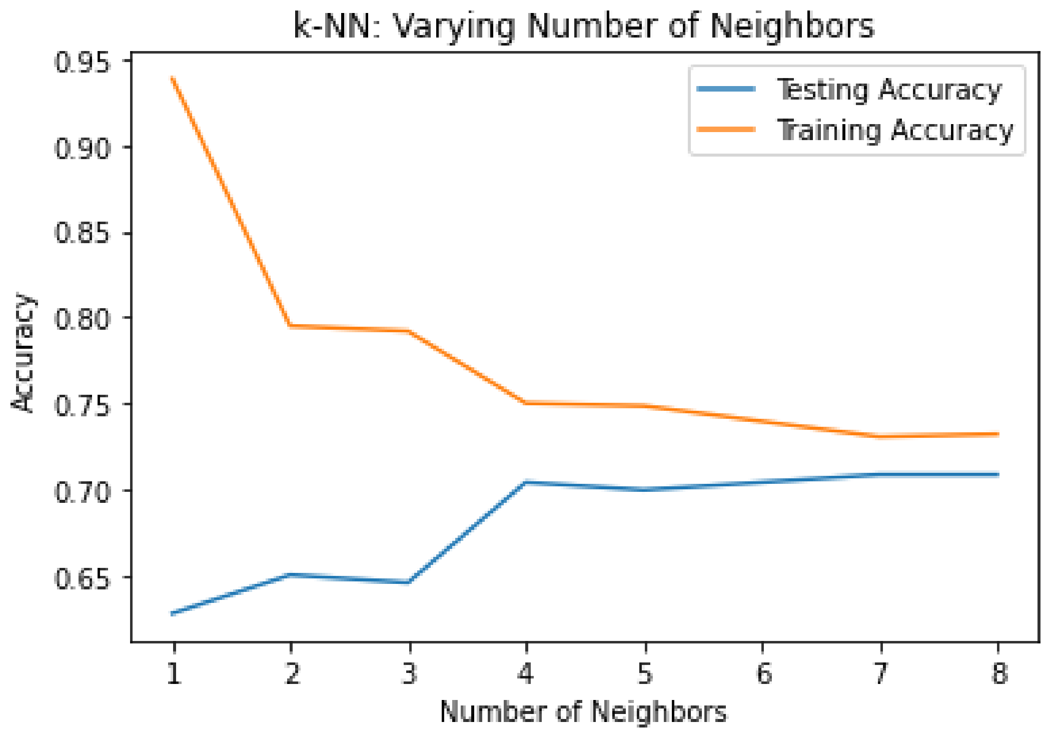


Classification Report

```
[ ] print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.81      0.77       139
           1       0.62      0.52      0.57        84

    accuracy                           0.70       223
   macro avg       0.68      0.66      0.67       223
weighted avg       0.69      0.70      0.69       223
```

RELATION BETWEEN K VALUES



INFERENCE:-

KNN CLASSIFICATION PERFORMS RELATIVELY WELL FOR THE DATASET GIVEN, BOTH TEST AND TRAIN LINES SEEM TO GIVE AN ACCURACY VALUE WITHIN THE RANGE 0.7-0.79

WE CAN INFER THAT AT LEAST 70% OF OUR PREDICTIONS FOR PASSENGER SURVIVAL WERE ACCURATE.

2)NAÏVE BAYES CLASSIFIER ALGORITHM:-

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and P(B) ≠ 0.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

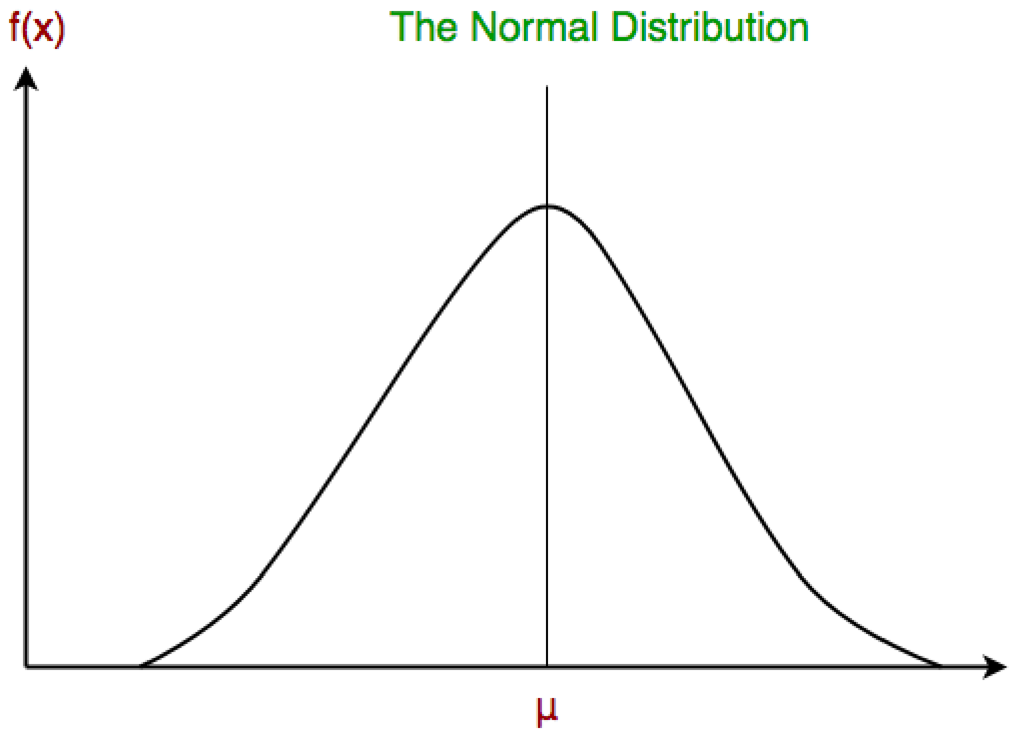
Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector (of size *n*) where:

$$X = (x_1, x_2, x_3, ..., x_n)$$

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:



The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by: $P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$

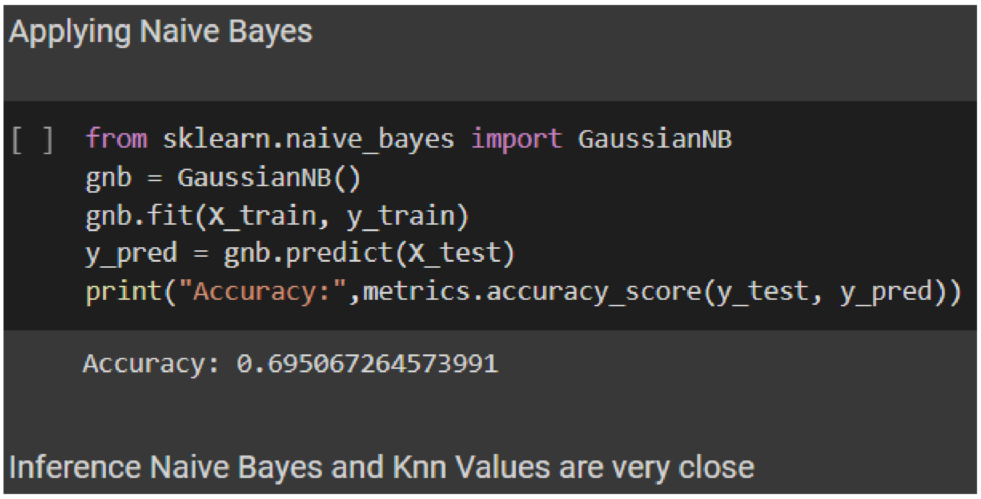
Other popular Naive Bayes classifiers are:

- **Multinomial Naive Bayes**: Feature vectors represent the frequencies with which certain events have been generated by a **multinomial distribution**. This is the event model typically used for document classification.
- **Bernoulli Naive Bayes**: In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence(i.e. a word occurs in a document or not) features are used rather than term frequencies(i.e. frequency of a word in the document).

NOTE:-

- In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.
- Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

VALUES DERIVED FROM OUR DATASET:-



Applying Naive Bayes

```
[ ]  from sklearn.naive_bayes import GaussianNB
     gnb = GaussianNB()
     gnb.fit(X_train, y_train)
     y_pred = gnb.predict(X_test)
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
     Accuracy: 0.695067264573991
```

Inference Naive Bayes and Knn Values are very close

INFERENCE: - NAÏVE BAYES GIVES A SIMILAR RESULT TO THAT OF KNN AND WE CAN INFER THAT ALMOST 69.5% OF OUR PREDICTIONS WERE ACCURATE WITH NAIVE BAYES CLASSIFIER.

3) K MEANS CLUSTERING ALGORITHM:-

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

OPTIMAL K-VALUE

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2 +\sum_{P_i \text{ in Cluster2}} distance(P_i\ C_2)^2+\sum_{P_i \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

$\sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

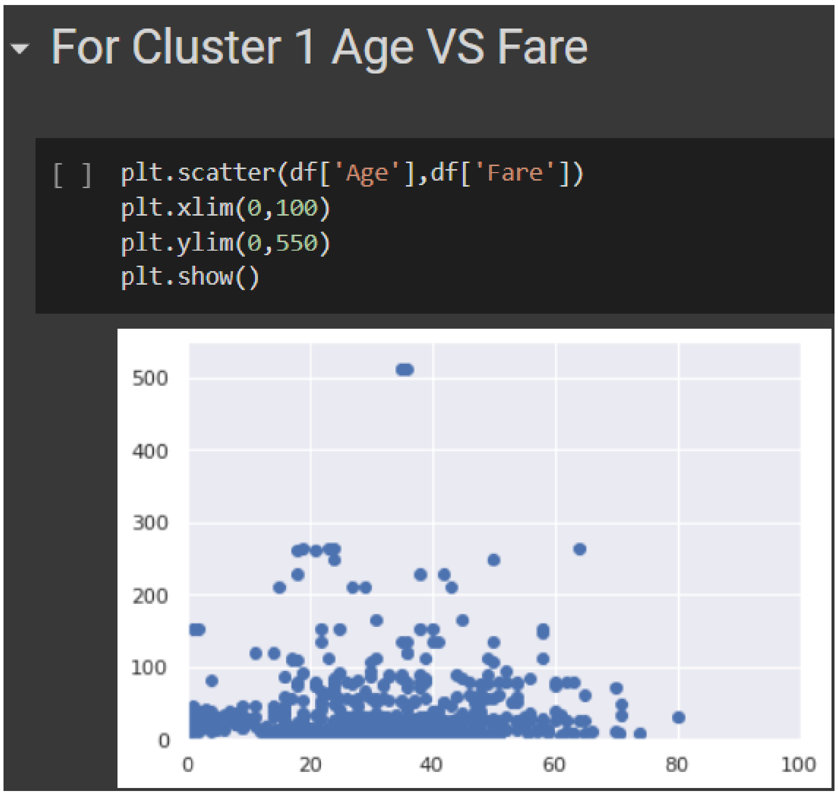
To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

CLUSTERING DONE WITH OUR DATA SET:-

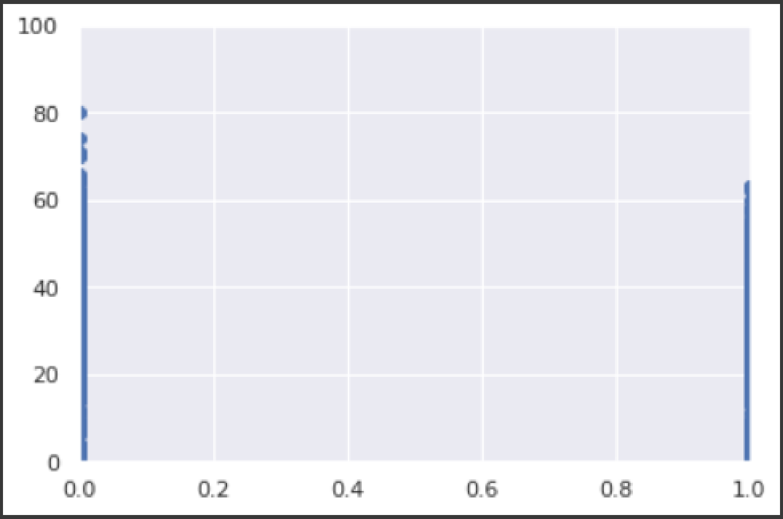
BELOW 5 CLUSTERS ARE FORMED TO EXPLORE DIFFERENT COMBINATIONS WITHIN THE CLASS



For Cluster 1 Age VS Fare

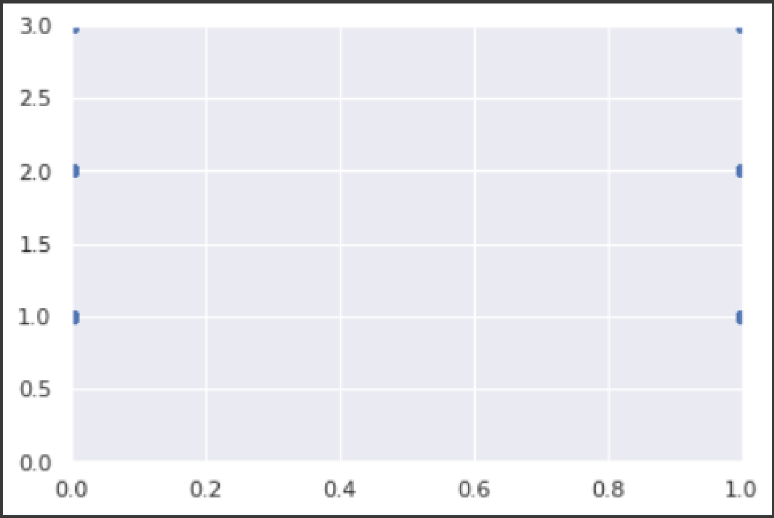
```
[ ]  plt.scatter(df['Age'],df['Fare'])
     plt.xlim(0,100)
     plt.ylim(0,550)
     plt.show()
```

```
[ ]  plt.scatter(df['Sex'],df['Age'])
     plt.xlim(0,1)
     plt.ylim(0,100)
     plt.show()
```



For Cluster 3 Passenger Class vs Survived

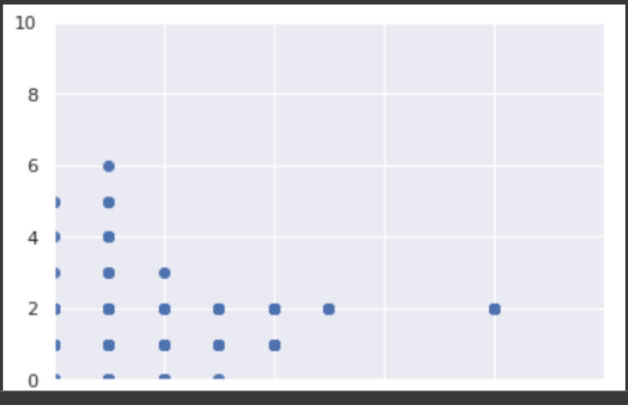
```
[ ]  plt.scatter(df['Survived'],df['Pclass'])
     plt.xlim(0,1)
     plt.ylim(0,3)
     plt.show()
```



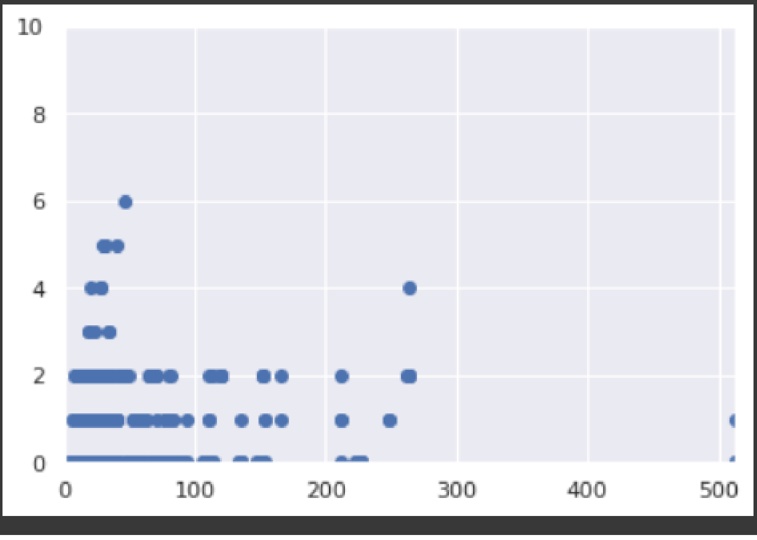
For 4th Cluster Combination Sibling/Spouse Vs Parent/Child

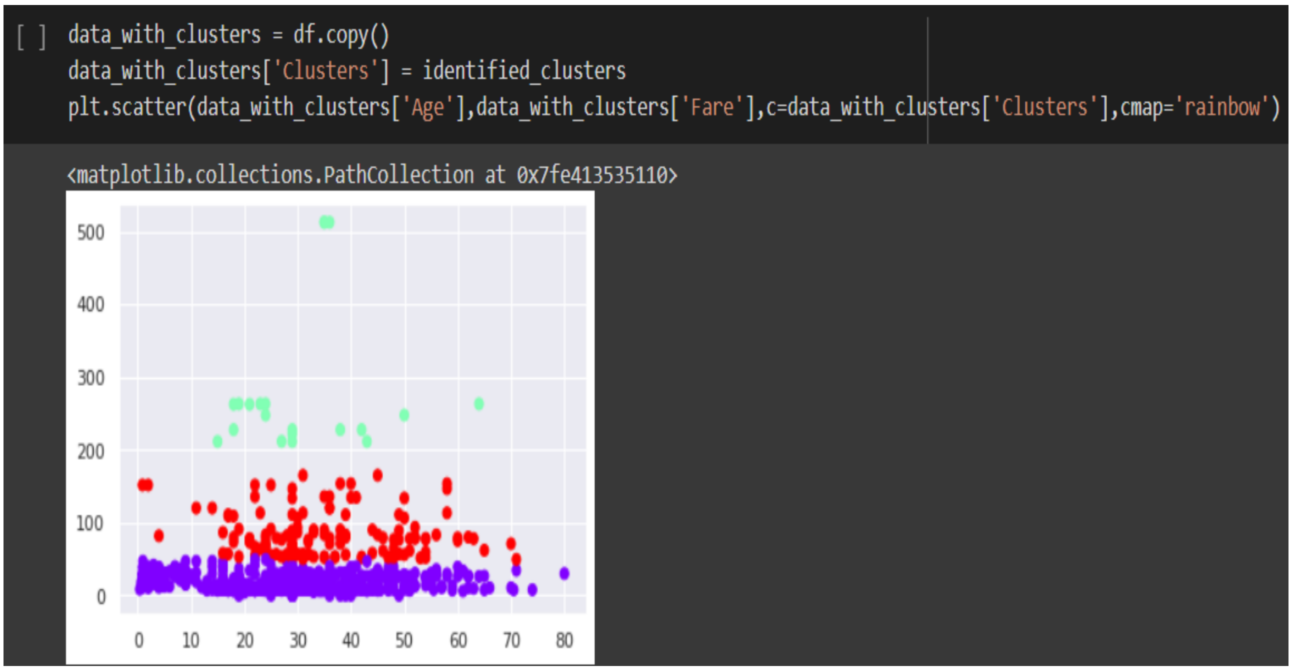
```
[ ]  df['Embarked'].unique()

     array(['S', 'C', 'Q', nan], dtype=object)
```

```
[ ]  plt.scatter(df['SibSp'],df['Parch'])
     plt.xlim(0,10)
     plt.ylim(0,10)
     plt.show()
```

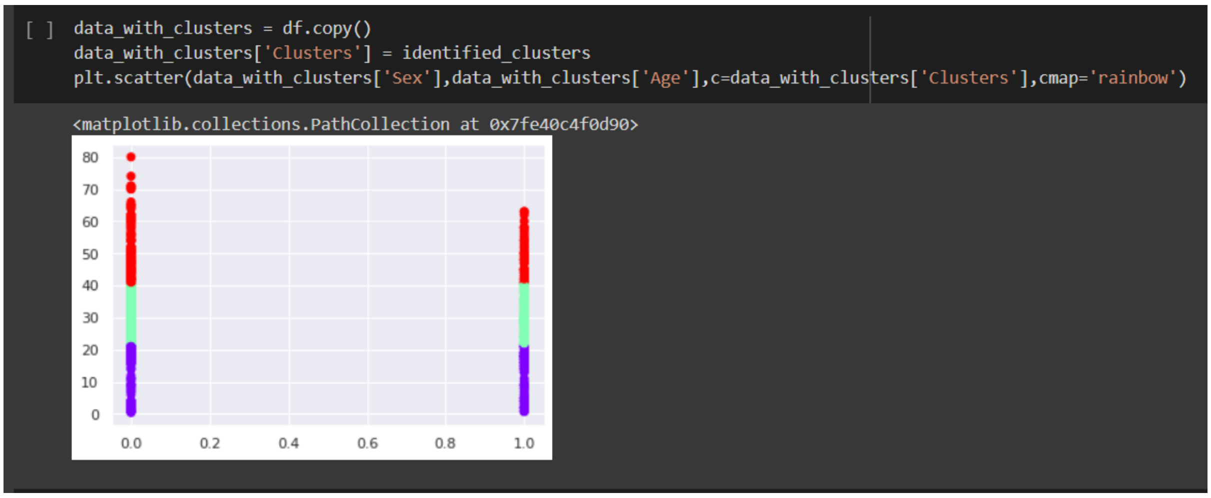


```
[ ]  plt.scatter(df['Fare'],df['Parch'])
     plt.xlim(0,512)
     plt.ylim(0,10)
     plt.show()
```

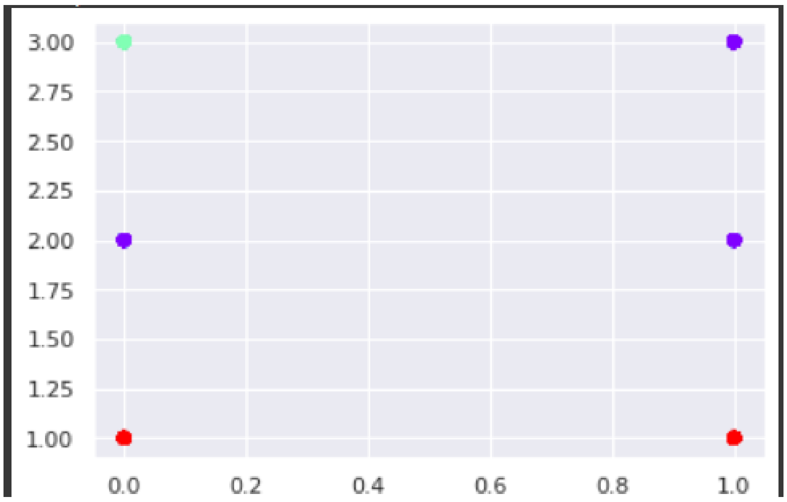




```
[ ] data_with_clusters = df.copy()
    data_with_clusters['Clusters'] = identified_clusters
    plt.scatter(data_with_clusters['Age'],data_with_clusters['Fare'],c=data_with_clusters['Clusters'],cmap='rainbow')

    <matplotlib.collections.PathCollection at 0x7fe413535110>
```

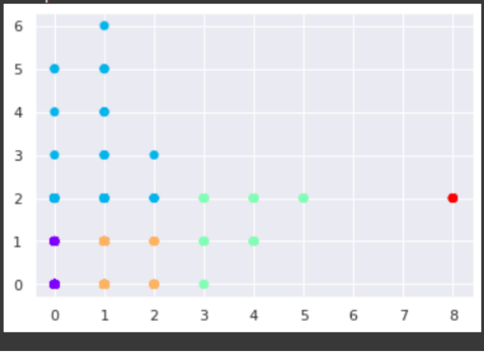


```
[ ] data_with_clusters = df.copy()
    data_with_clusters['Clusters'] = identified_clusters
    plt.scatter(data_with_clusters['Sex'],data_with_clusters['Age'],c=data_with_clusters['Clusters'],cmap='rainbow')

    <matplotlib.collections.PathCollection at 0x7fe40c4f0d90>
```



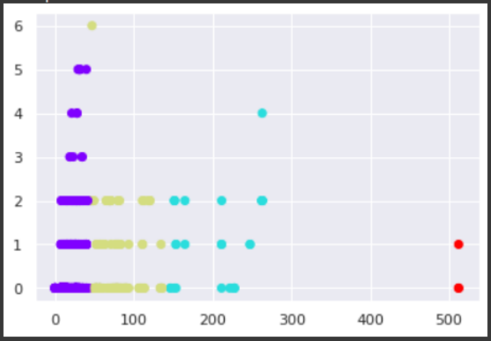
```
] data_with_clusters = df.copy()
  data_with_clusters['Clusters'] = identified_clusters
  plt.scatter(data_with_clusters['SibSp'],data_with_clusters['Parch'],c=data_with_clusters['Clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7fe40c6b6810>
```



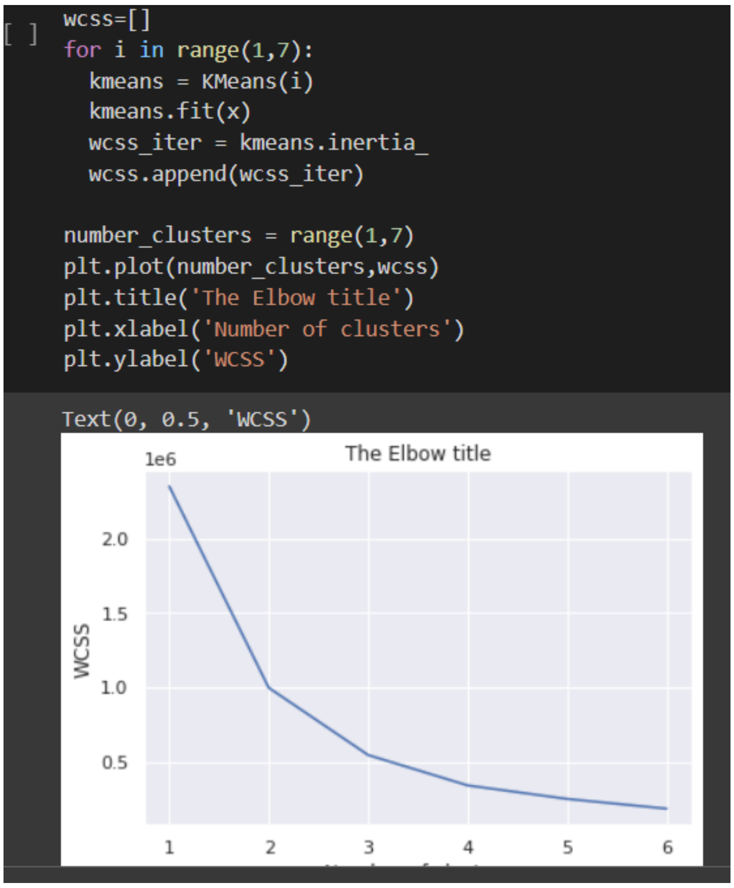
```
data_with_clusters = df.copy()
data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['Fare'],data_with_clusters['Parch'],c=data_with_clusters['Clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7fe40c27b390>
```



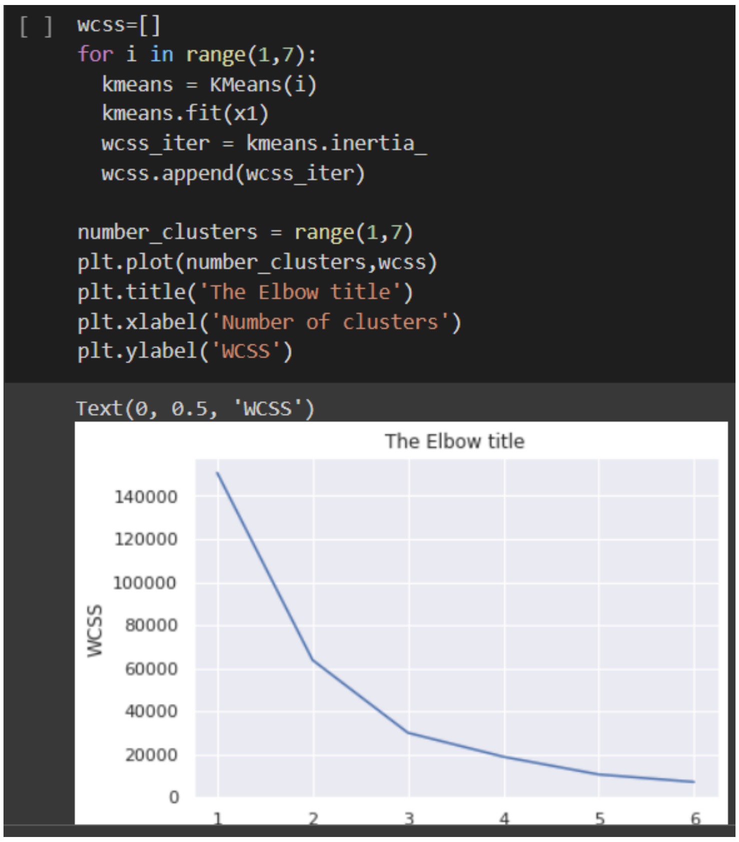
USING COLOR CODING FOR EACH CLUSTER IN ALL OF THE 5 GRAPHS

ELBOW METHOD USED FOR ALL CLUSTERS



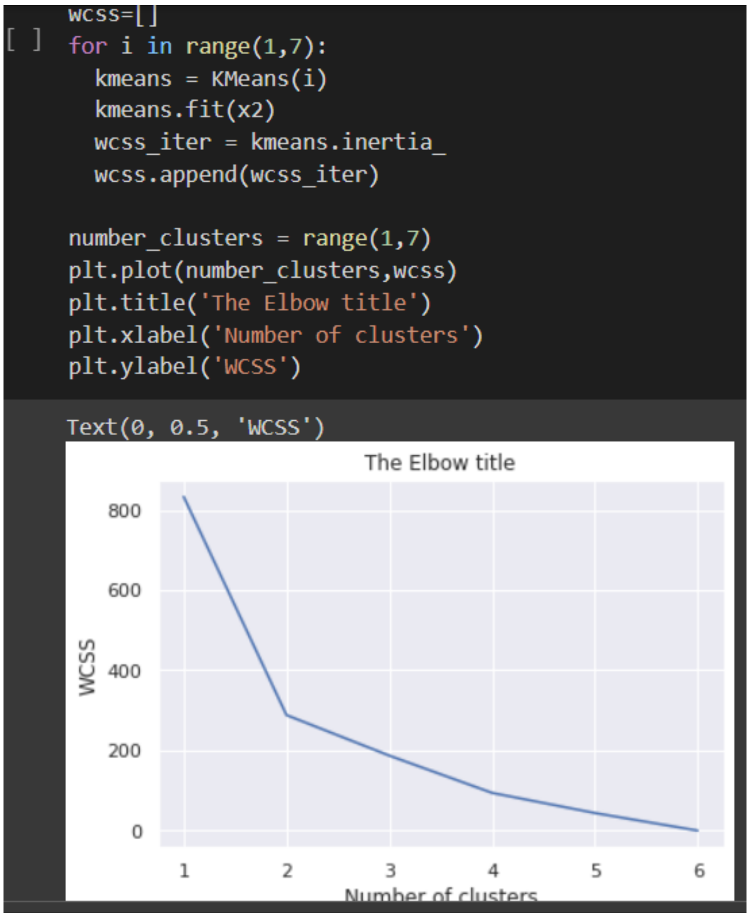
```
[ ]  wcss=[]
     for i in range(1,7):
       kmeans = KMeans(i)
       kmeans.fit(x)
       wcss_iter = kmeans.inertia_
       wcss.append(wcss_iter)

     number_clusters = range(1,7)
     plt.plot(number_clusters,wcss)
     plt.title('The Elbow title')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```



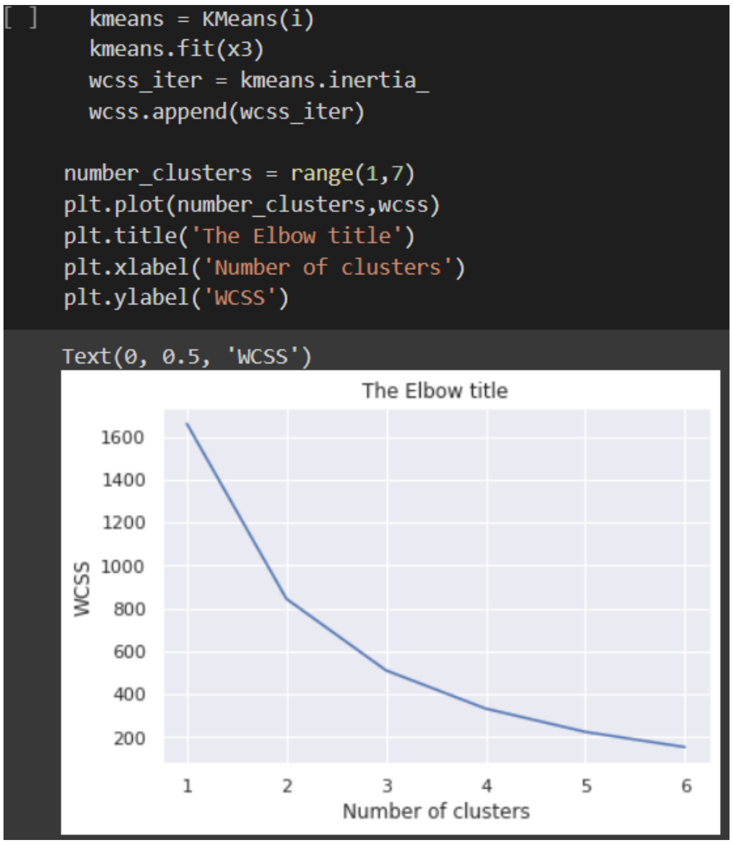
```
[ ]  wcss=[]
     for i in range(1,7):
       kmeans = KMeans(i)
       kmeans.fit(x1)
       wcss_iter = kmeans.inertia_
       wcss.append(wcss_iter)

     number_clusters = range(1,7)
     plt.plot(number_clusters,wcss)
     plt.title('The Elbow title')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```



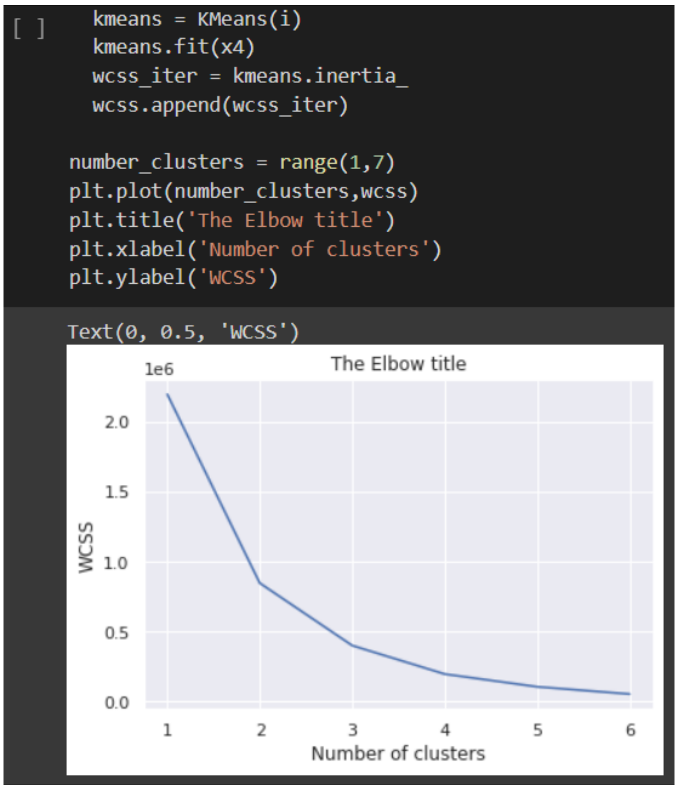
```
wcss=[]
[ ] for i in range(1,7):
      kmeans = KMeans(i)
      kmeans.fit(x2)
      wcss_iter = kmeans.inertia_
      wcss.append(wcss_iter)

    number_clusters = range(1,7)
    plt.plot(number_clusters,wcss)
    plt.title('The Elbow title')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```



```
[ ]   kmeans = KMeans(i)
      kmeans.fit(x3)
      wcss_iter = kmeans.inertia_
      wcss.append(wcss_iter)

    number_clusters = range(1,7)
    plt.plot(number_clusters,wcss)
    plt.title('The Elbow title')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```



```
[ ]     kmeans = KMeans(i)
        kmeans.fit(x4)
        wcss_iter = kmeans.inertia_
        wcss.append(wcss_iter)

      number_clusters = range(1,7)
      plt.plot(number_clusters,wcss)
      plt.title('The Elbow title')
      plt.xlabel('Number of clusters')
      plt.ylabel('WCSS')

      Text(0, 0.5, 'WCSS')
```

THE ABOVE ELBOW METHOD GRAPHS ARE FORMED FOR EACH OF THE 5 CLUSTERS

INFERENCE:-

WITH THE VARIOUS COMBINATIONS USED TO FORM THE CLUSTERS WE ARE ABLE TO INFER DIFFERENT FORMATIONS OF CLUSTERS IN OUR DATASET. PLUS WE ALSO OBSERVE VARIYING RESULTS WITH THE ELBOW METHOD USED FOR THE FIVE DIFFERENT CLUSTERING COMBINATIONS.

WE ARE ABLE TO FIND NEW WAYS TO UNDERSTAND OUR DATA AND THE RELATIONSHIP BETWEEN THE DIFFERENT CLASS LABELS EVEN MORE WITH K MEANS CLUSTERING.

4) HIERARCHICAL CLUSTERING ALGORITHM:-

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the bottom-up approach. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

The **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**.

Single Linkage: It is the Shortest Distance between the closest points of the clusters

Complete Linkage: It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.

Average Linkage: It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

Centroid Linkage: It is the linkage method in which the distance between the centroid of the clusters is calculated.

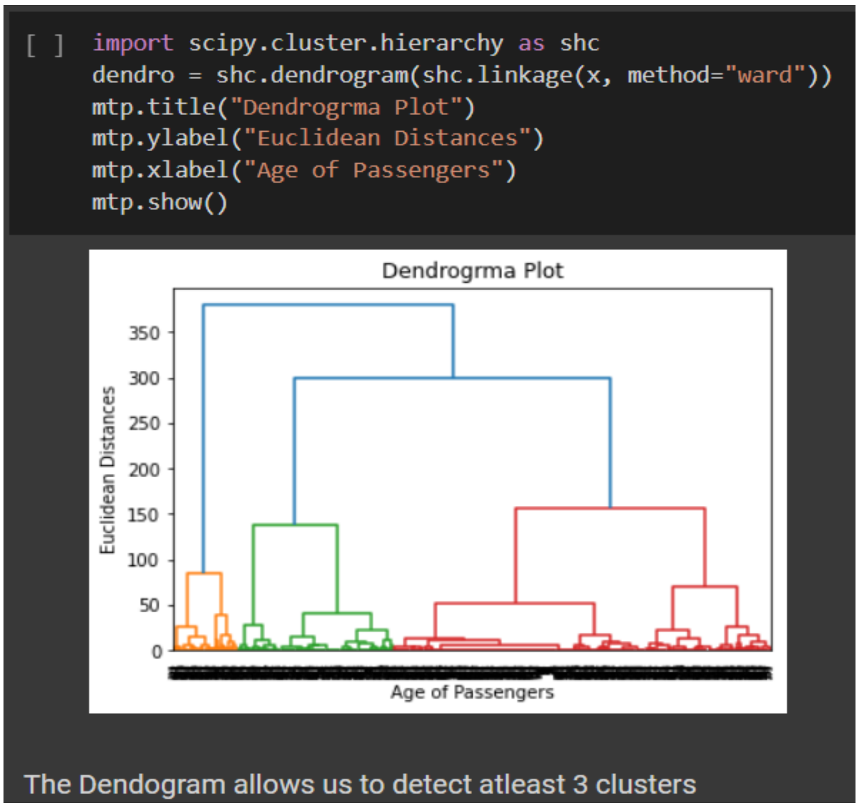
The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

We can obtain the optimal number of clusters from the model itself, human intervention not required. Dendrograms help us in clear visualization, which is practical and easy to understand. Disadvantages of Hierarchical Clustering:

Not suitable for large datasets due to high time and space complexity. There is no mathematical objective for Hierarchical clustering. All the approaches to calculate the similarity between clusters has their own disadvantages.

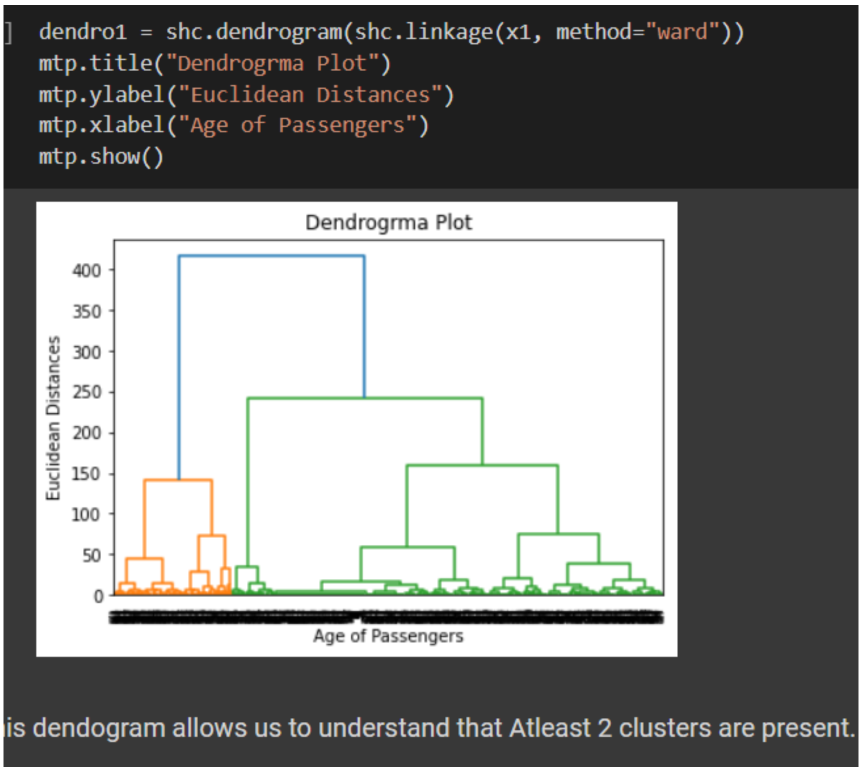
HIERARCHICAL CLUSTERING USING OUR DATASET

DENDOGRAMS OBTAINED:-



```
[ ] import scipy.cluster.hierarchy as shc
    dendro = shc.dendrogram(shc.linkage(x, method="ward"))
    mtp.title("Dendrogrma Plot")
    mtp.ylabel("Euclidean Distances")
    mtp.xlabel("Age of Passengers")
    mtp.show()
```

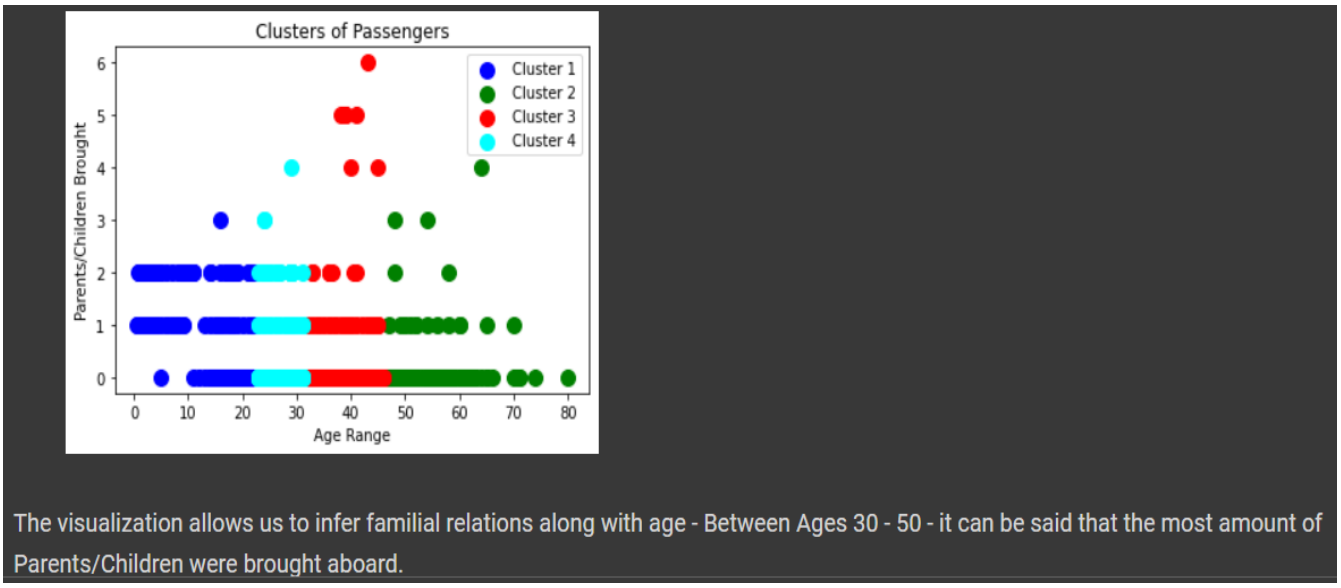
The Dendogram allows us to detect atleast 3 clusters



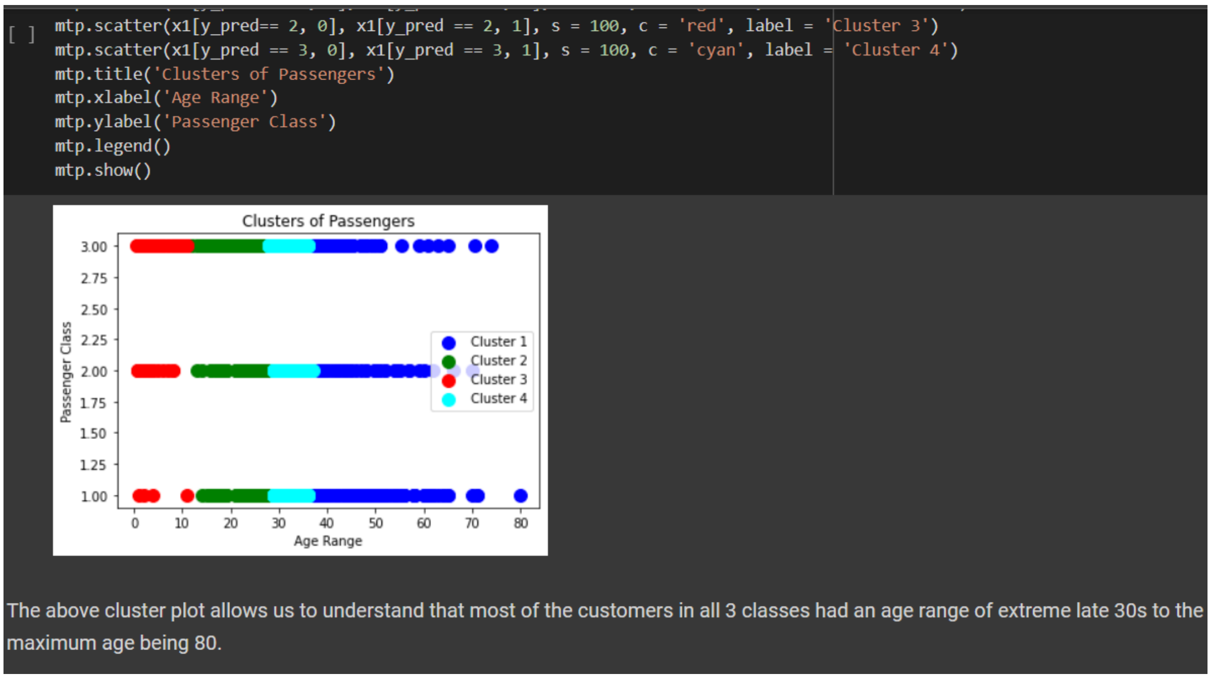
```
] dendro1 = shc.dendrogram(shc.linkage(x1, method="ward"))
  mtp.title("Dendrogrma Plot")
  mtp.ylabel("Euclidean Distances")
  mtp.xlabel("Age of Passengers")
  mtp.show()
```

is dendogram allows us to understand that Atleast 2 clusters are present.

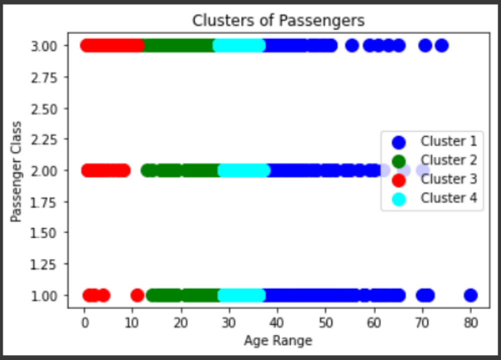
CLUSTERS FORMED:-



The visualization allows us to infer familial relations along with age - Between Ages 30 - 50 - it can be said that the most amount of Parents/Children were brought aboard.



```
[ ]  mtp.scatter(x1[y_pred== 2, 0], x1[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
     mtp.scatter(x1[y_pred == 3, 0], x1[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
     mtp.title('Clusters of Passengers')
     mtp.xlabel('Age Range')
     mtp.ylabel('Passenger Class')
     mtp.legend()
     mtp.show()
```



The above cluster plot allows us to understand that most of the customers in all 3 classes had an age range of extreme late 30s to the maximum age being 80.

INFERENCE:-

WE ARE ABLE TO FORM INTERESTING OBSERVATIONS WITH THE AGE ATTRIBUTE AND OTHER ATTRIBUTES TO IDENTIFY UNIQUE RELATIONS WITH VARYING AGE RANGES AND ARE ABLE TO FORM STRUCTURED CORRELATIONS BETWEEN DIFFERENT ATTRIBUTES IN A HIERARCHICAL MANNER.

Kmeans Vs Heirarchical

k-means, using a pre-specified number of clusters, the method assigns records to each cluster to find the mutually exclusive cluster of spherical shape based on distance. Hierarchical methods can be either divisive or agglomerative.

K Means clustering needed advance knowledge of K i.e. no. of clusters one want to divide your data. In hierarchical clustering one can stop at any number of clusters, one find appropriate by interpreting the dendrogram.

One can use median or mean as a cluster centre to represent each cluster. Agglomerative methods begin with ‘n’ clusters and sequentially combine similar clusters until only one cluster is obtained.

KMEANS Methods used are normally less computationally intensive and are suited with very large datasets. Divisive methods work in the opposite direction, beginning with one cluster that includes all the records and Hierarchical methods are especially useful when the target is to arrange the clusters into a natural hierarchy.

In K Means clustering, since one start with random choice of clusters, the results produced by running the algorithm many times may differ. In Hierarchical Clustering, results are reproducible in Hierarchical clustering

K- means clustering a simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset). A hierarchical clustering is a set of nested clusters that are arranged as a tree.

K Means clustering is found to work well when the structure of the clusters is hyper spherical (like circle in 2D, sphere in 3D). Hierarchical clustering don't work as well as, k means when the shape of the clusters is hyper spherical.

KMEANSAdvantages: 1. Convergence is guaranteed. 2. Specialized to clusters of different sizes and shapes.
HEIRARAdvantages: 1 .Ease of handling of any forms of similarity or distance. 2. Consequently, applicability to any attributes types.

KMEANSDisadvantages: 1. K-Value is difficult to predict 2. Didn't work well with global cluster.
HEIRARDisadvantage: 1. Hierarchical clustering requires the computation and storage of an n×n distance matrix. For very large datasets, this can be expensive and slow

5)DECISION TREES CLASSIFIER ALGORITHM:-

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**

- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

TERMINOLOGIES:-

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**

- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.

- It calculates how much information a feature provides us about a class.

- According to the value of information gain, we split the node and build the decision tree.

- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

Gini Index= 1- $\sum_j P_j^2$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.

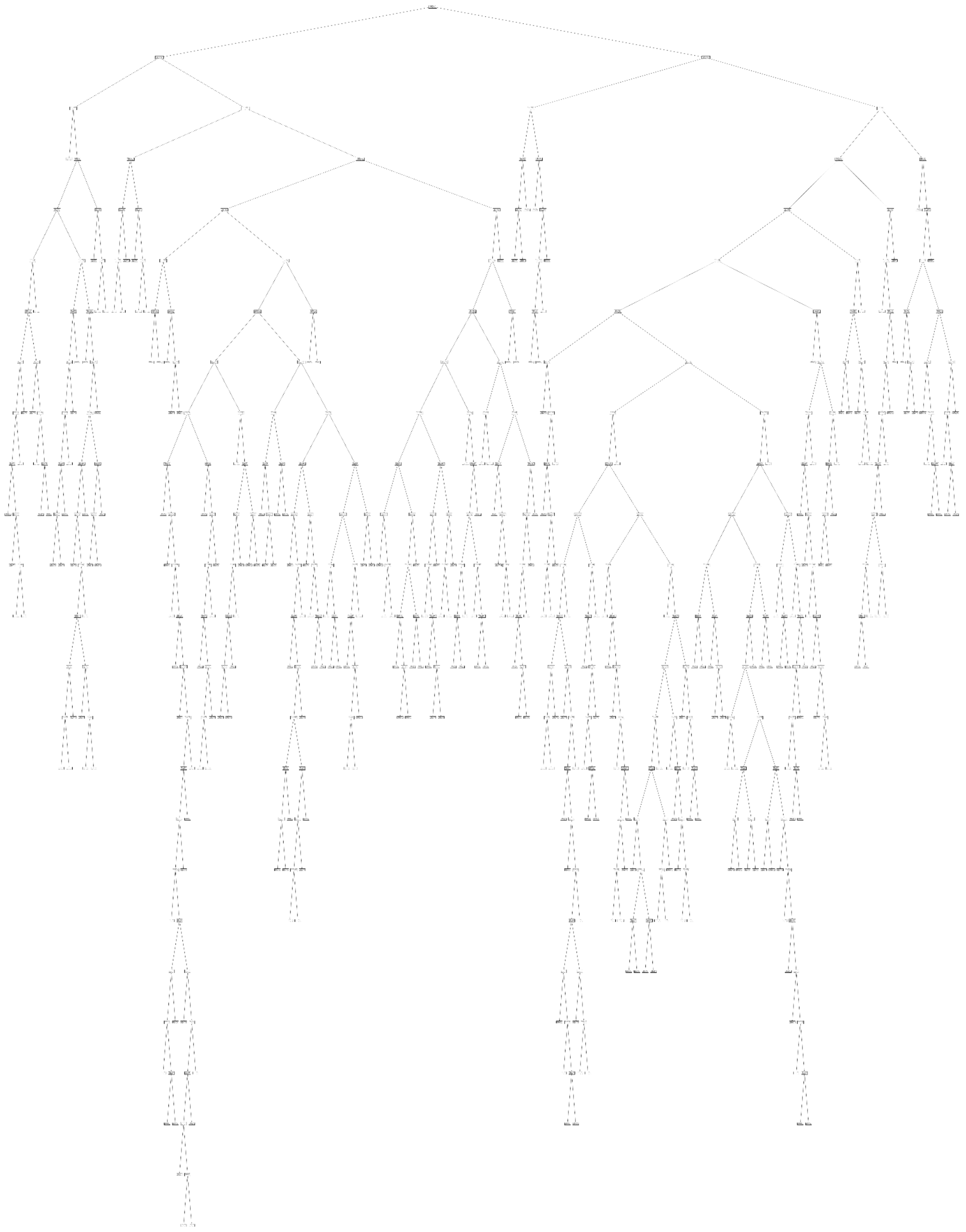
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

DECISION TREES FORMED USING OUR DATASET

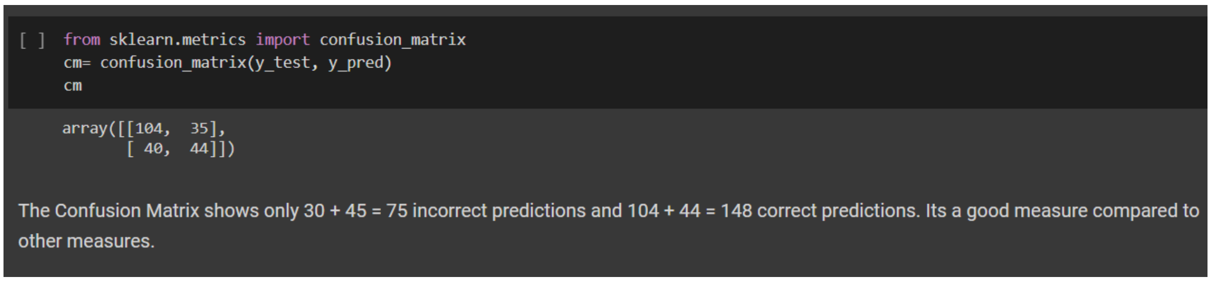
DECISION TREE-



ROOT NODE

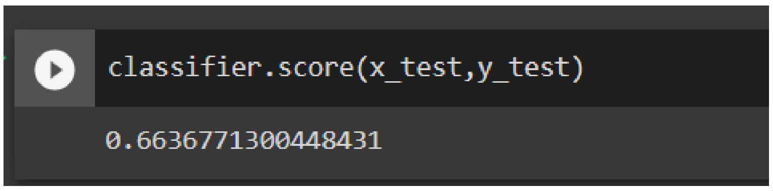
DECISION TREE FORMED USING PCLASS AS ROOT AND AGE AND FARE AS SUB TREES.

CONFUSION MATRIX AND ACCURACY SCORE:-



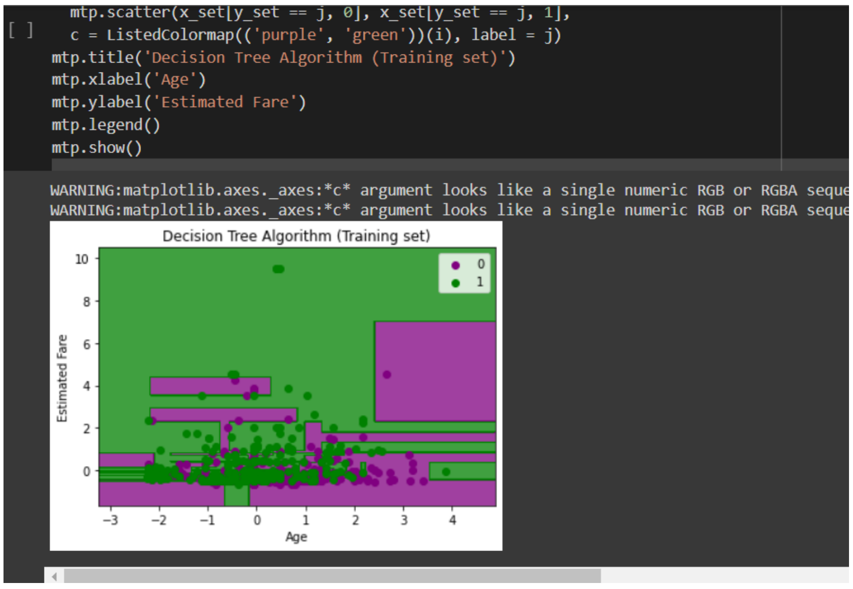
```
[ ] from sklearn.metrics import confusion_matrix
    cm= confusion_matrix(y_test, y_pred)
    cm

    array([[104,  35],
           [ 40,  44]])
```

The Confusion Matrix shows only 30 + 45 = 75 incorrect predictions and 104 + 44 = 148 correct predictions. Its a good measure compared to other measures.



```
classifier.score(x_test,y_test)

0.6636771300448431
```

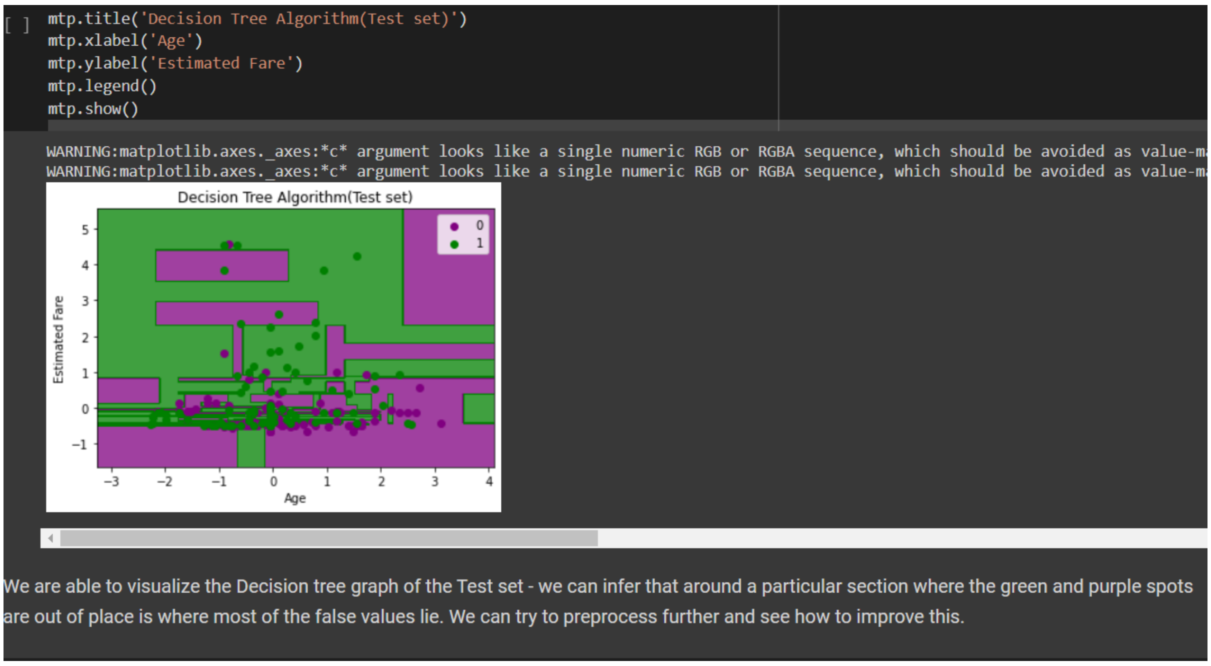
FURTHER VISUALIZATION:-



```
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
[ ]     c = ListedColormap(('purple', 'green'))(i), label = j)
    mtp.title('Decision Tree Algorithm (Training set)')
    mtp.xlabel('Age')
    mtp.ylabel('Estimated Fare')
    mtp.legend()
    mtp.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA seque
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA seque
```

EACH OF THESE GRAPHS REPRESENT HOW THE ATTRIBUTES BLEED INTO EACH OTHER AT SPECIFIC SPOTS



```
[ ] mtp.title('Decision Tree Algorithm(Test set)')
    mtp.xlabel('Age')
    mtp.ylabel('Estimated Fare')
    mtp.legend()
    mtp.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-ma
```

We are able to visualize the Decision tree graph of the Test set - we can infer that around a particular section where the green and purple spots are out of place is where most of the false values lie. We can try to preprocess further and see how to improve this.

INFERENCE:- THE DECISION TREES CLASSIFIER HAS A LOWER ACCURACY THAN THAT OF KNN AND NAÏVE BAYES BUT STILL MANAGES TO PROVIDE FURTHER INSIGHT ONTO HOW THE PCLASS ATTRIBUTE CAN BE USED FOR FURTHER ANALYSIS IN FUTURE MODELS.

6)LINEAR REGRESSION ALGORITHM:-

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables.

Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

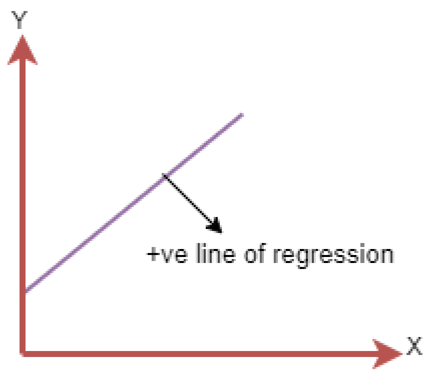
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

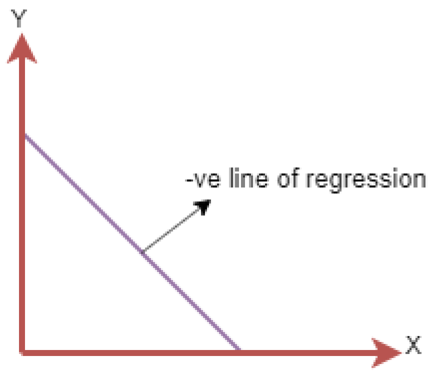
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $\mathbf{Y= a_0+a_1x}$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $\mathbf{Y= -a_0+a_1x}$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.

- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$\text{MSE}= 1\frac{1}{N}\sum_{i=1}^{n}(y_i \;- (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
($a1x_i$+a_0)= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.

- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

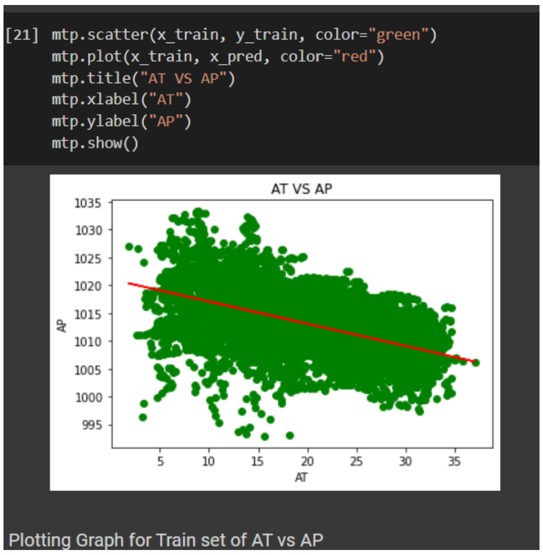
Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
  Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.
- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

FOR THIS ALGORITHM ANOTHER DATASET ABOUT WINES WAS USED:-

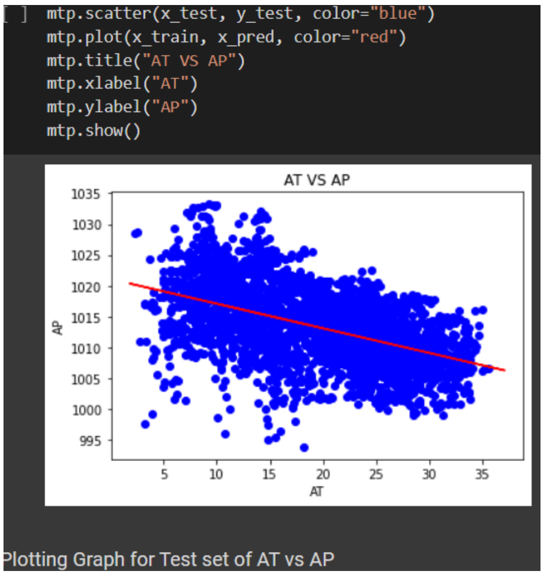
TRAIN SET:-



```
[21] mtp.scatter(x_train, y_train, color="green")
     mtp.plot(x_train, x_pred, color="red")
     mtp.title("AT VS AP")
     mtp.xlabel("AT")
     mtp.ylabel("AP")
     mtp.show()
```

Plotting Graph for Train set of AT vs AP

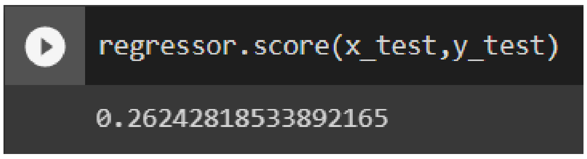
TEST SET:-



```
[ ] mtp.scatter(x_test, y_test, color="blue")
    mtp.plot(x_train, x_pred, color="red")
    mtp.title("AT VS AP")
    mtp.xlabel("AT")
    mtp.ylabel("AP")
    mtp.show()
```

Plotting Graph for Test set of AT vs AP

ACCURACY:-



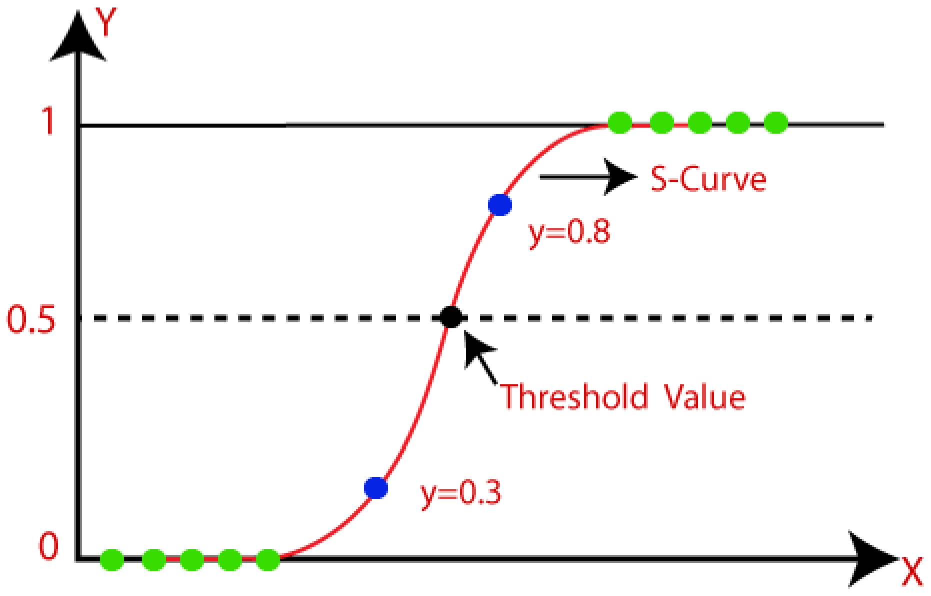
```
regressor.score(x_test,y_test)

0.26242818533892165
```

INFERENCE:-

THE ALGORITHM IS NOT VERY SUITABLE FOR EITHER OF THE DATASETS USED. IT DOES NOT HAVE A HIGH ACCURACY AND THE GRAPHS FORMED ARE FAR TOO CONVOLUTED TO GIVE A CLEAR OBSERVATION. IT DOES HOWEVER HELP US OBSERVE CERTAIN FACETS OF THE EQUATION USED TO DERIVE THE GRAPH.

7)LOGISTIC REGRESSION ALGORITHM:-

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.



Note: Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}\text{; 0 for y= 0, and infinity for y=1}$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

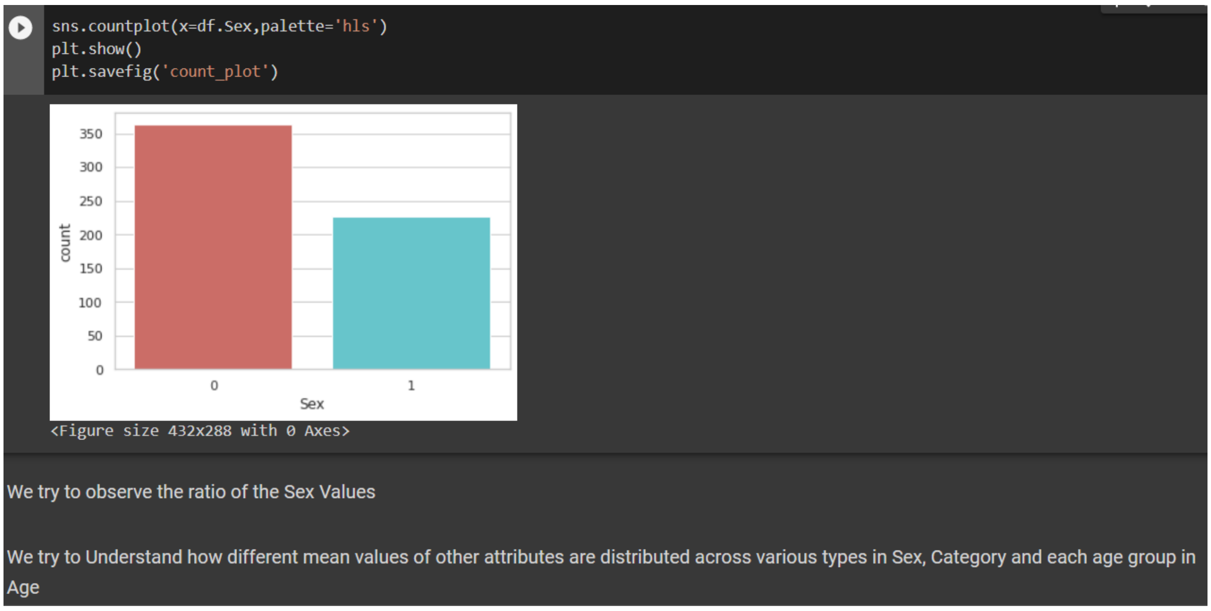
Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High"

RESULTS OBTAINED USING LOGISTIC REGRESSION ON THE GIVEN DATASET hcv0.csv WHICH IS A COLLECTION OF BLOOD TEST REPORTS.

WE FIRST TRY TO SEE WHICH WOULD BE A GOOD PREDICTOR VARIABLE

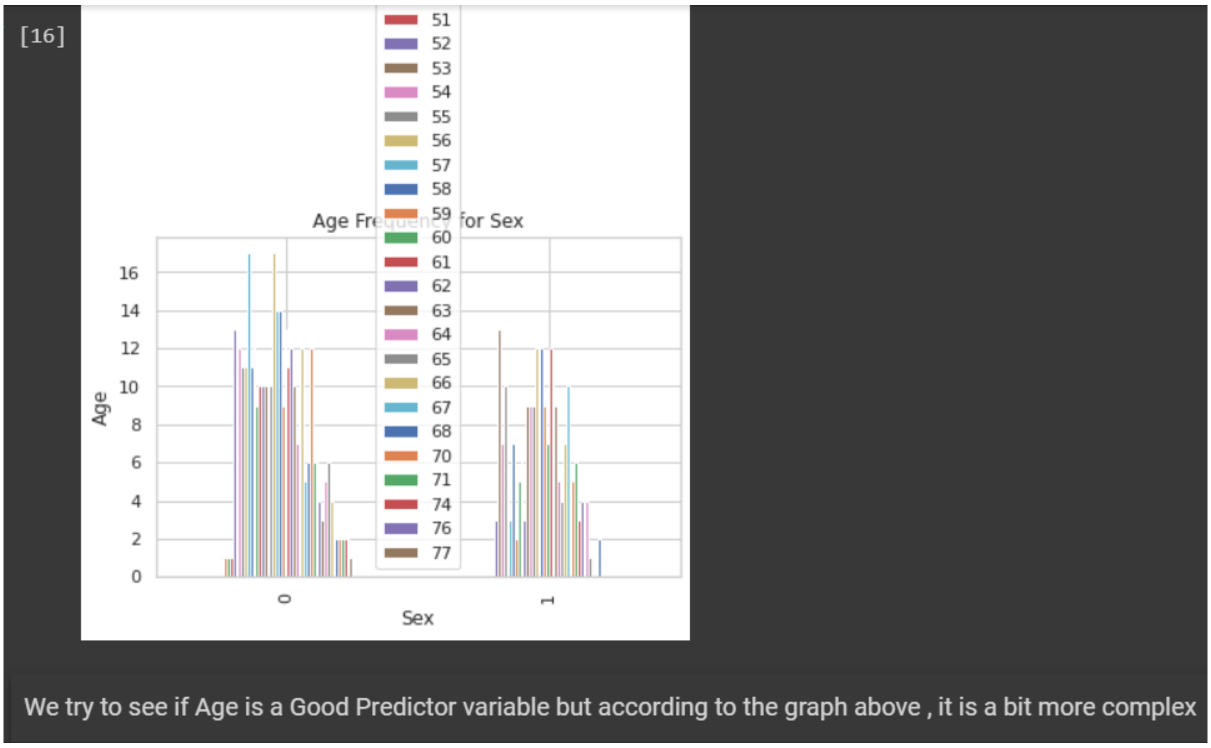


```
sns.countplot(x=df.Sex,palette='hls')
plt.show()
plt.savefig('count_plot')
```

```
<Figure size 432x288 with 0 Axes>
```

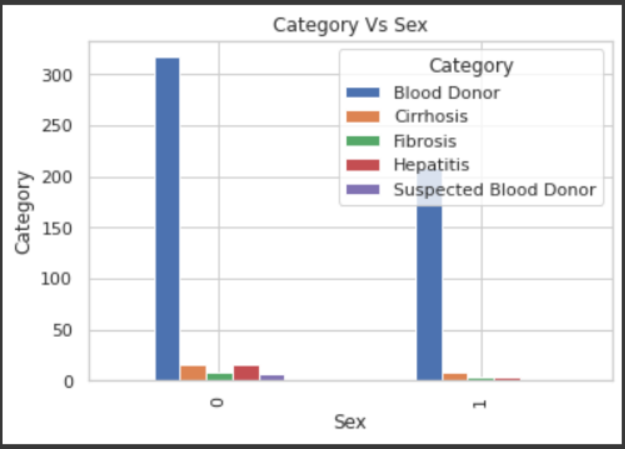
We try to observe the ratio of the Sex Values

We try to Understand how different mean values of other attributes are distributed across various types in Sex, Category and each age group in Age



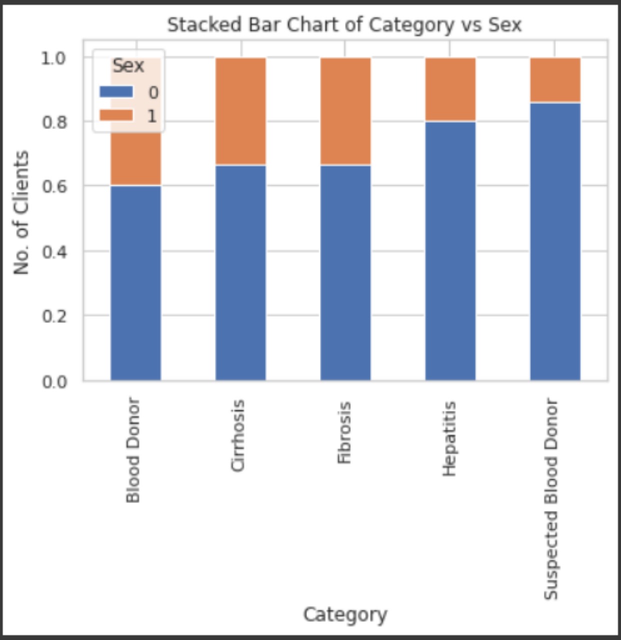
We try to see if Age is a Good Predictor variable but according to the graph above , it is a bit more complex

```
17] %matplotlib inline
    pd.crosstab(df.Sex,df.Category).plot(kind='bar')
    plt.title('Category Vs Sex')
    plt.xlabel('Sex')
    plt.ylabel('Category')
    plt.savefig('svcat')
```



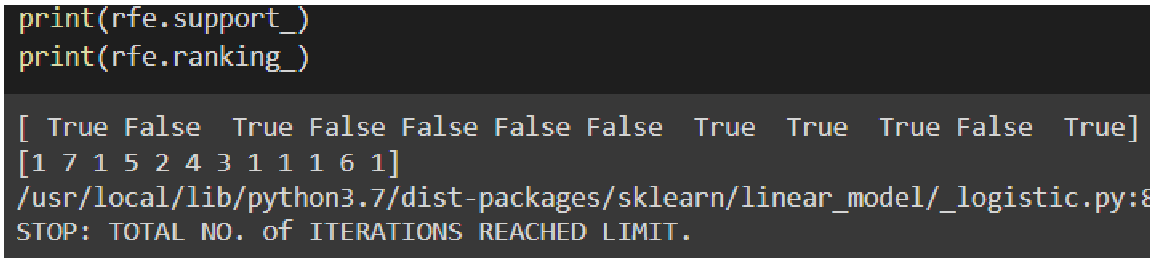
We check distribution of various categories in Sex

```
     table.div(table.sum(1).astype(float), axis=0).plot(k
[18] plt.title('Stacked Bar Chart of Category vs Sex')
     plt.xlabel('Category')
     plt.ylabel('No. of Clients')
     plt.savefig('categvsex')
```



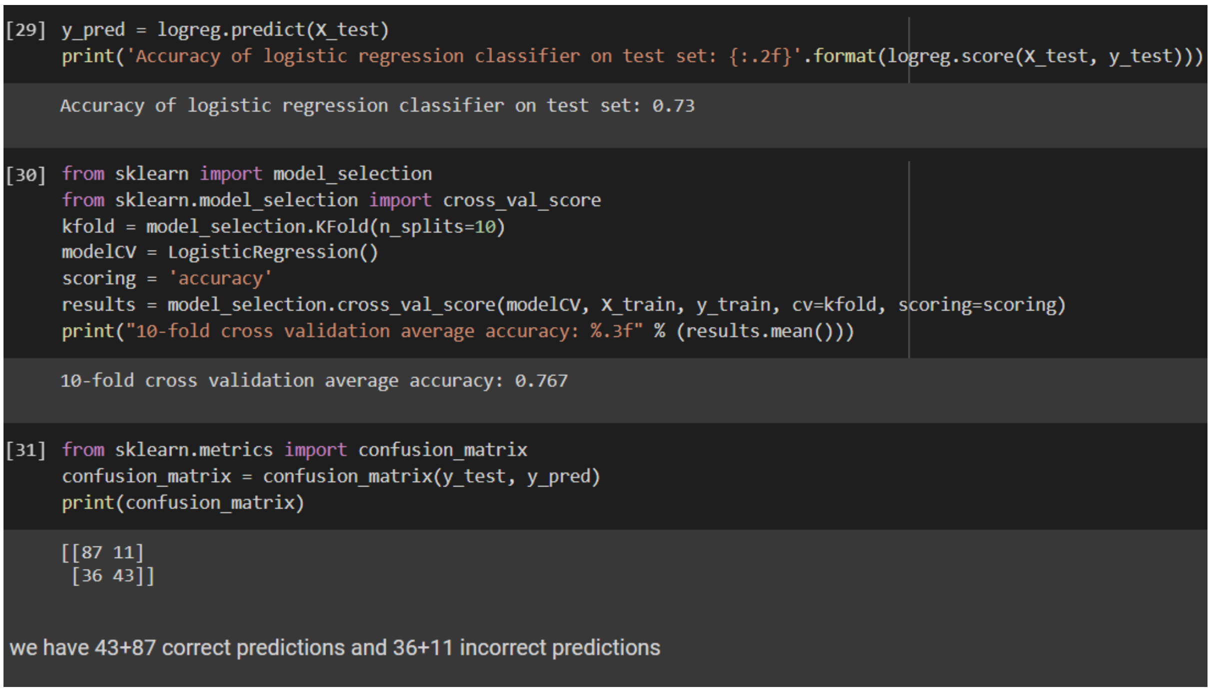
It seems that Category can be a good predictor variable

FEATURE SELECTION



```
print(rfe.support_)
print(rfe.ranking_)

[ True False  True False False False False  True  True  True False  True]
[1 7 1 5 2 4 3 1 1 1 6 1]
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:8
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

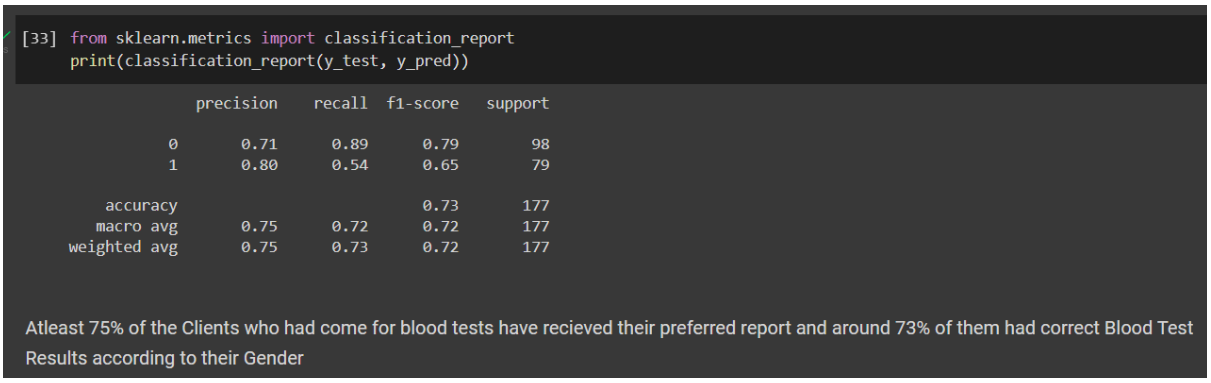
ACCURACIES AND CONFUSION MATRIX:-



```
[29] y_pred = logreg.predict(X_test)
     print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

     Accuracy of logistic regression classifier on test set: 0.73

[30] from sklearn import model_selection
     from sklearn.model_selection import cross_val_score
     kfold = model_selection.KFold(n_splits=10)
     modelCV = LogisticRegression()
     scoring = 'accuracy'
     results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scoring)
     print("10-fold cross validation average accuracy: %.3f" % (results.mean()))

     10-fold cross validation average accuracy: 0.767

[31] from sklearn.metrics import confusion_matrix
     confusion_matrix = confusion_matrix(y_test, y_pred)
     print(confusion_matrix)

     [[87 11]
      [36 43]]
```

we have 43+87 correct predictions and 36+11 incorrect predictions

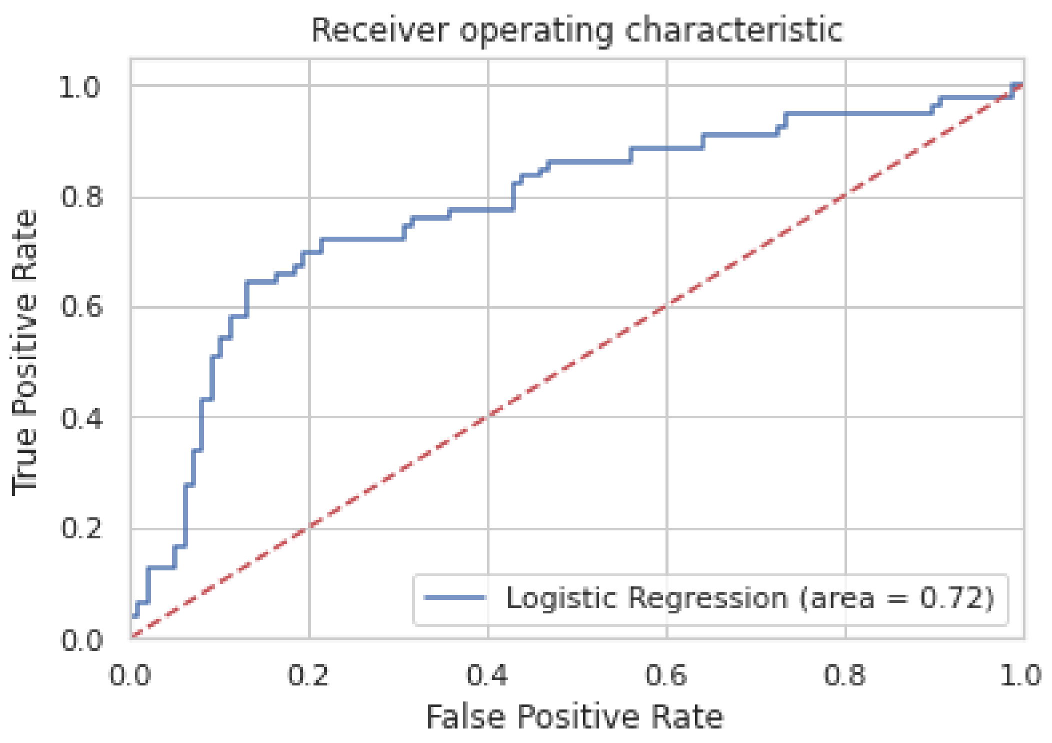
CLASSIFICATION REPORT



```
[33] from sklearn.metrics import classification_report
     print(classification_report(y_test, y_pred))

                  precision    recall  f1-score   support

               0       0.71      0.89      0.79        98
               1       0.80      0.54      0.65        79

        accuracy                           0.73       177
       macro avg       0.75      0.72      0.72       177
    weighted avg       0.75      0.73      0.72       177
```

Atleast 75% of the Clients who had come for blood tests have recieved their preferred report and around 73% of them had correct Blood Test Results according to their Gender

GRAPH BETWEEN TRUE AND FALSE POSITIVITY:-



THE FALSE POSITIVITY INCREASE DRASTICALLY AFTER T.P GOES PAST 0.6, SHOWING SOME INCONSISTENCIES MAY EXIST BEYOND A SPECIFIC VALUE.

INFERENCE:-

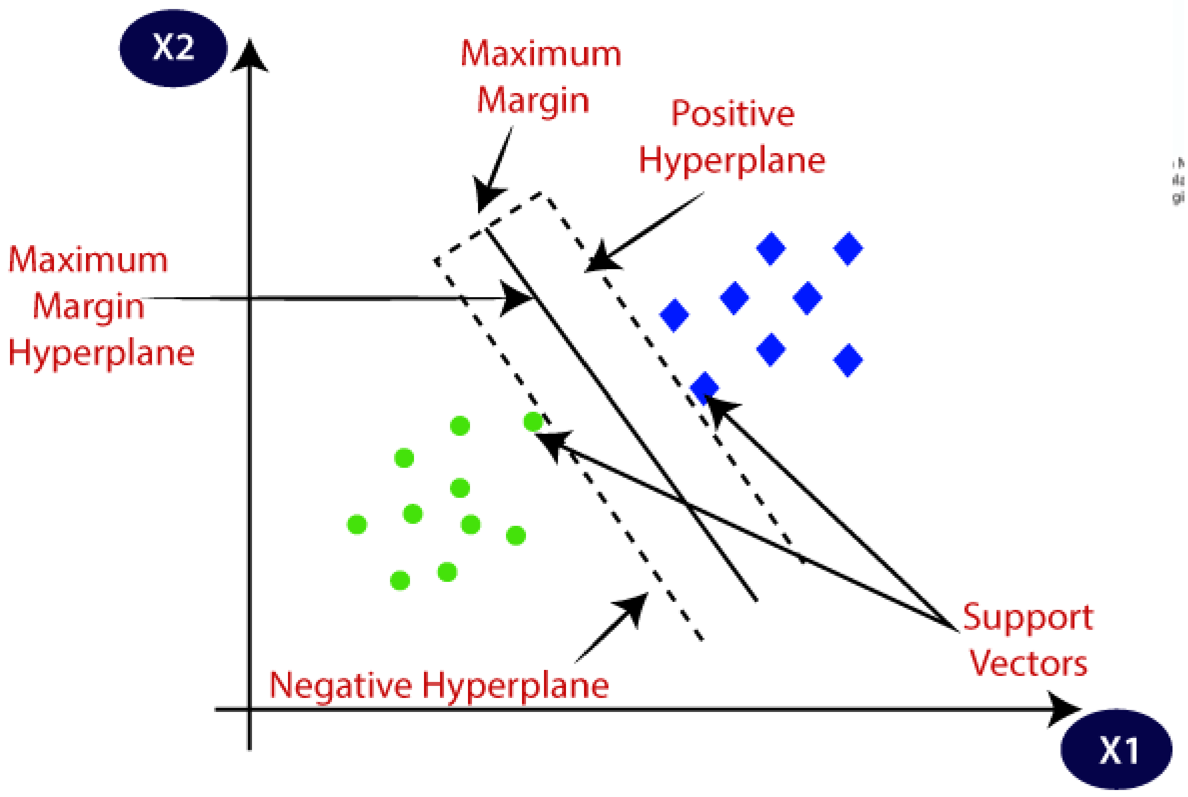
LOGISTIC REGRESSION SHOWS A HIGH ACCURACY VALUE FOR THE GIVEN DATASET AND ALLOWS US TO IDENTIFY AREAS WHERE POTENTIAL FALSE POSITIVES MAY EXIST.

8)SVM CLASSIFIER AND REGRESSOR ALGORITHM:-

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

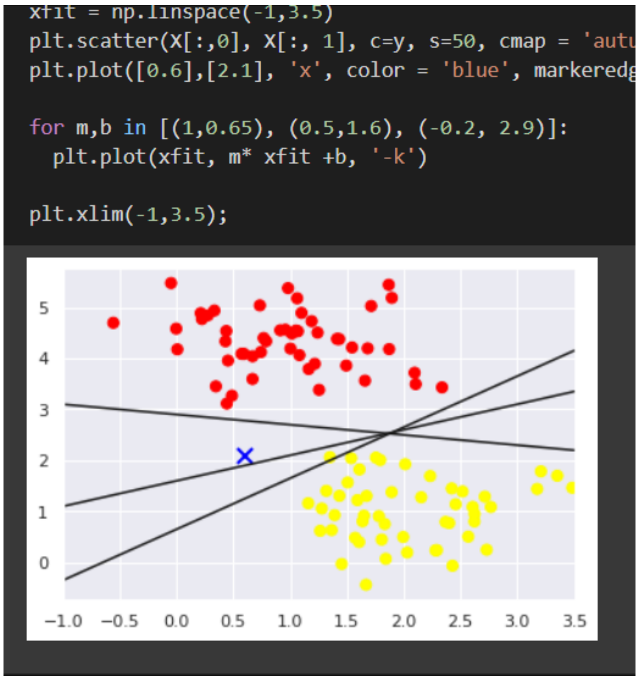
The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

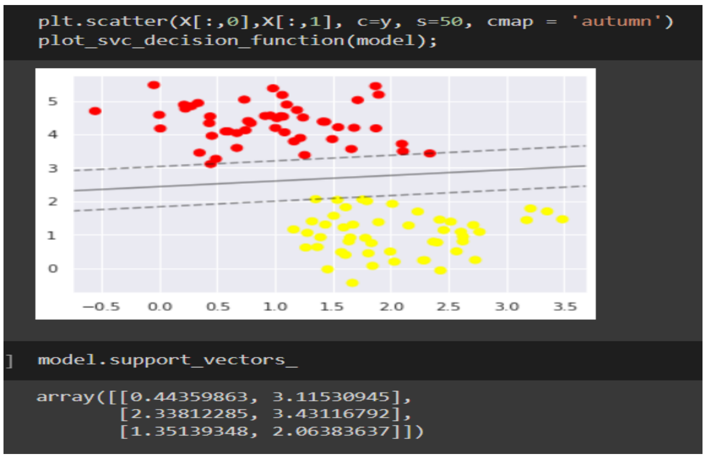
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

BOTH SYNTHETIC AND CURRENT DATASETS ARE USED:=

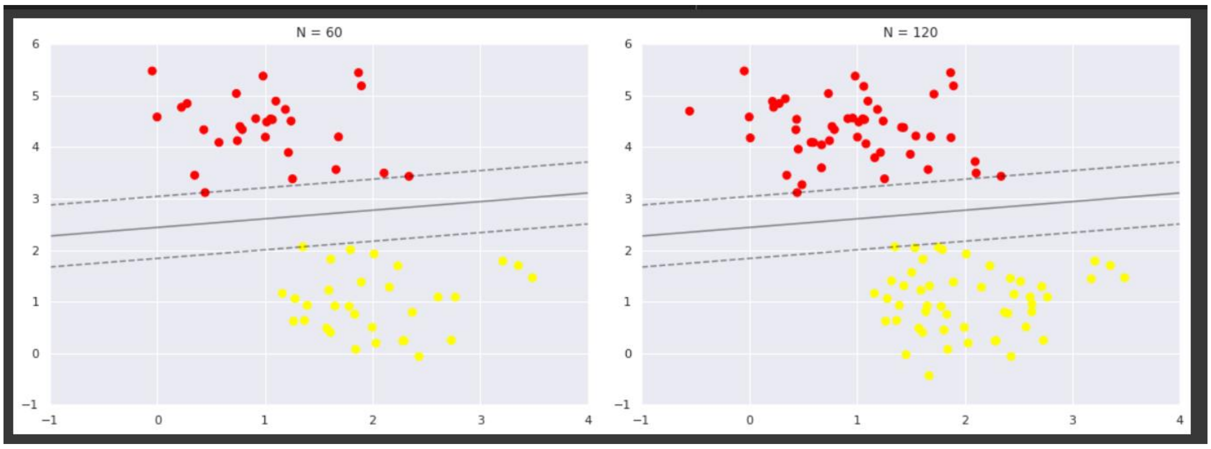


```
xfit = np.linspace(-1,3.5)
plt.scatter(X[:,0], X[:, 1], c=y, s=50, cmap = 'autu
plt.plot([0.6],[2.1], 'x', color = 'blue', markeredg

for m,b in [(1,0.65), (0.5,1.6), (-0.2, 2.9)]:
  plt.plot(xfit, m* xfit +b, '-k')

plt.xlim(-1,3.5);
```

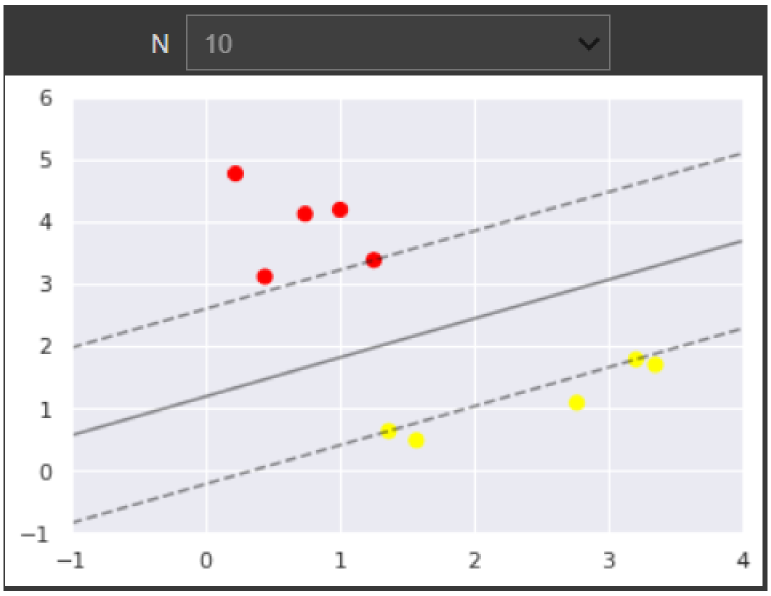


```
plt.scatter(X[:,0],X[:,1], c=y, s=50, cmap = 'autumn')
plot_svc_decision_function(model);
```

```
] model.support_vectors_

array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [1.35139348, 2.06383637]])
```

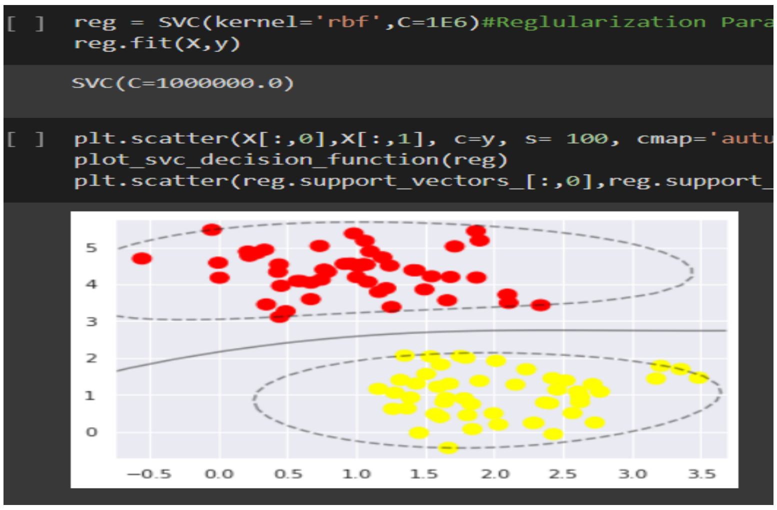


WE CAN OBSERVE THAT THE DEFAULT LINEAR KERNEL SPLITS THE DATA POINTS EFFECTIVELY

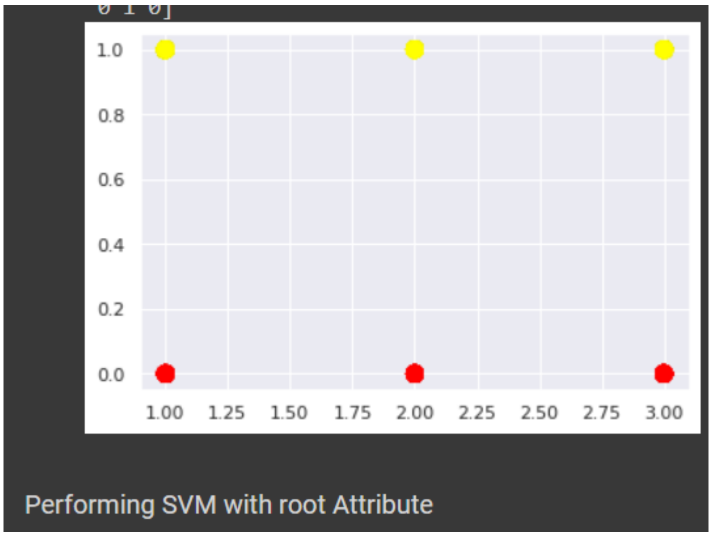


ANALYSED FROM VARYING RANGES OF DATAPOINTS

USING RBF KERNEL

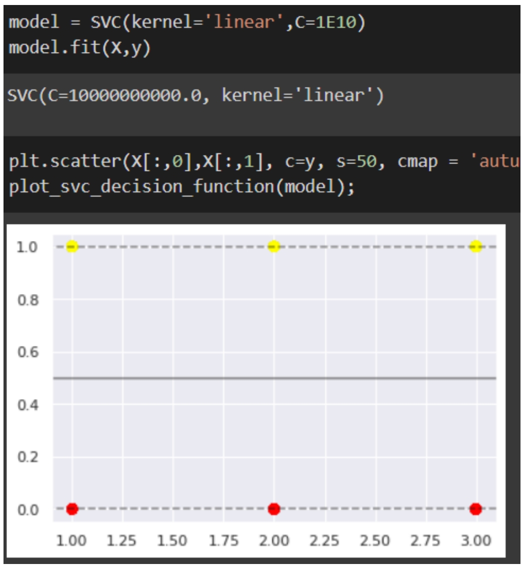


```
[ ]  reg = SVC(kernel='rbf',C=1E6)#Reglularization Para
     reg.fit(X,y)

     SVC(C=1000000.0)

[ ]  plt.scatter(X[:,0],X[:,1], c=y, s= 100, cmap='autu
     plot_svc_decision_function(reg)
     plt.scatter(reg.support_vectors_[:,0],reg.support_
```



Performing SVM with root Attribute

FOR DATASET TITANIC

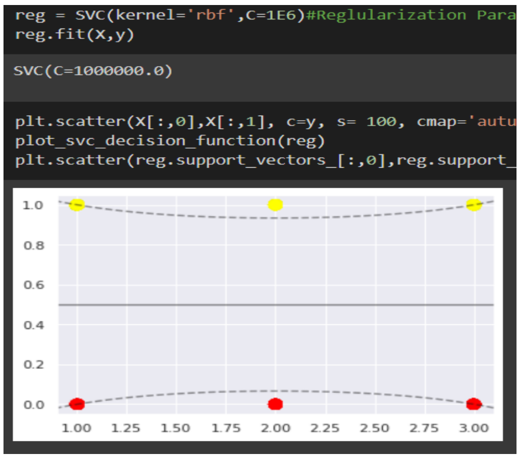


```
model = SVC(kernel='linear',C=1E10)
model.fit(X,y)
```

```
SVC(C=10000000000.0, kernel='linear')
```

```
plt.scatter(X[:,0],X[:,1], c=y, s=50, cmap = 'autu
plot_svc_decision_function(model);
```

LINEAR BARELY DIVIDES THE DATA POINTS



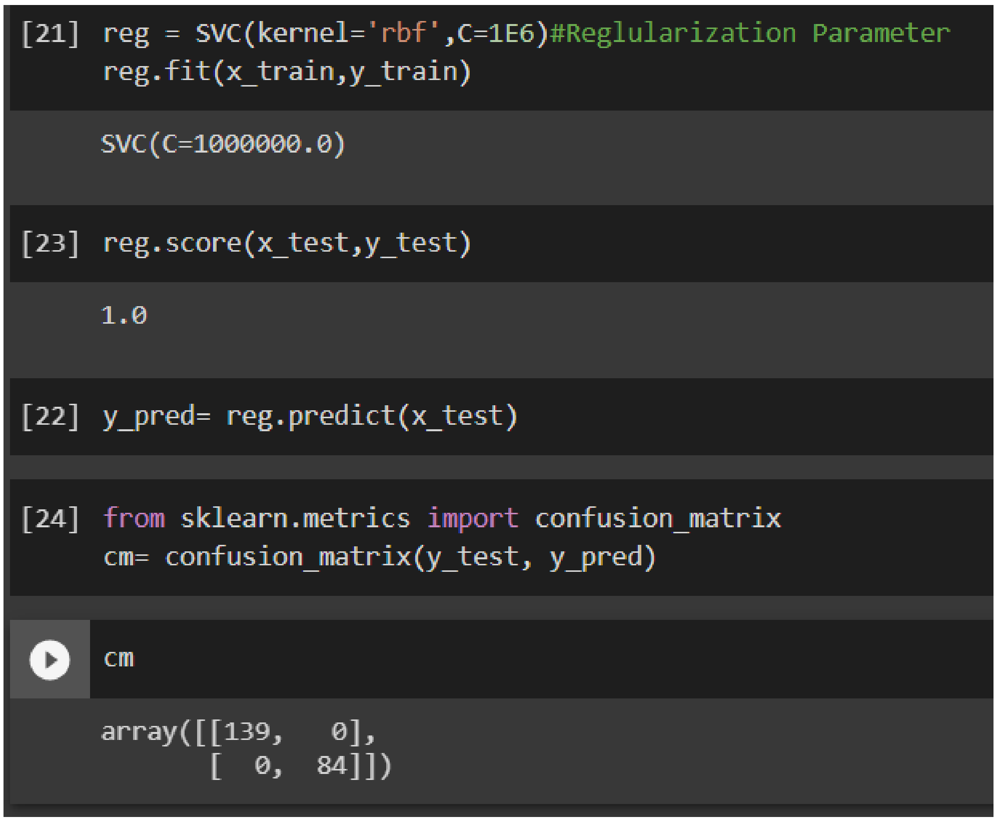
```
reg = SVC(kernel='rbf',C=1E6)#Reglularization Para
reg.fit(X,y)
```

```
SVC(C=1000000.0)
```

```
plt.scatter(X[:,0],X[:,1], c=y, s= 100, cmap='autu
plot_svc_decision_function(reg)
plt.scatter(reg.support_vectors_[:,0],reg.support_
```

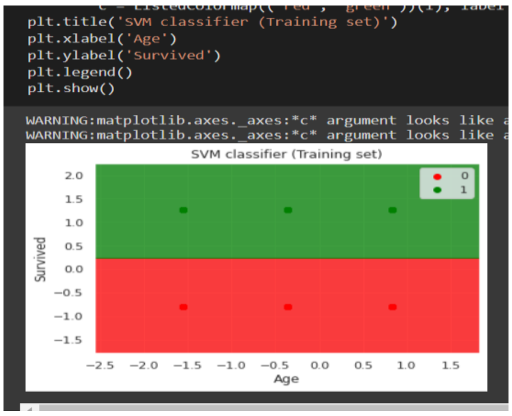
RBF IMPROVES THJE DIVIDE BETWEEN THE DATA POINTS AND IS CONSIDERED FOR FURTHER USE.

ACCURACY SCORE AND CONFUSION MATRIX:-

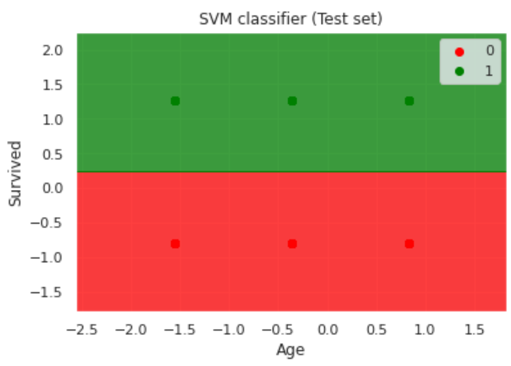


```
[21] reg = SVC(kernel='rbf',C=1E6)#Reglularization Parameter
     reg.fit(x_train,y_train)

     SVC(C=1000000.0)

[23] reg.score(x_test,y_test)

     1.0

[22] y_pred= reg.predict(x_test)

[24] from sklearn.metrics import confusion_matrix
     cm= confusion_matrix(y_test, y_pred)

     cm

     array([[139,   0],
            [  0,  84]])
```

ACCURACY IS 100%BFOR DATASET TITANIC AND IS RECOMMENDED FOR FURTHER ANALYSIS.

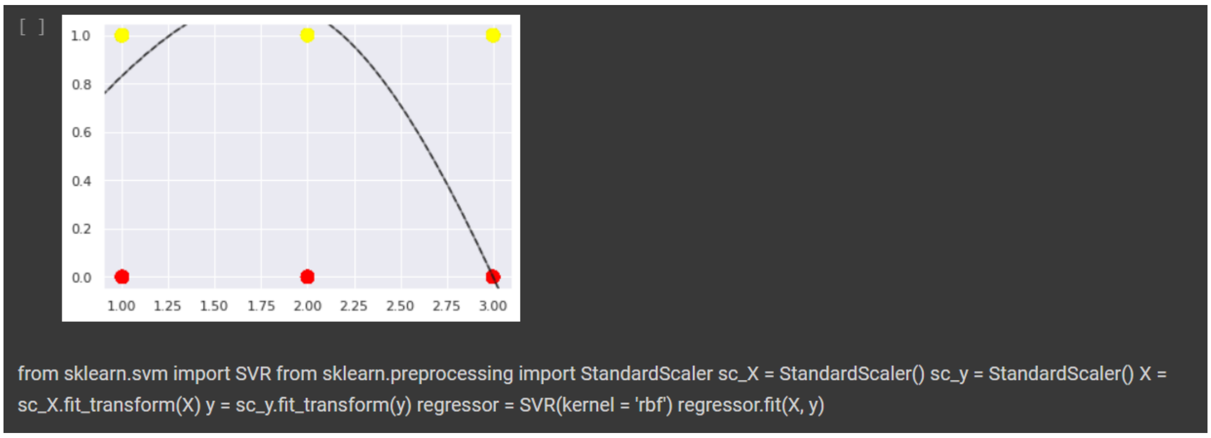


```
        c = ListedColormap(('red', 'green'))(i), label
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Survived')
plt.legend()
plt.show()

WARNING:matplotlib.axes._axes:*c* argument looks like a
WARNING:matplotlib.axes._axes:*c* argument looks like a
```

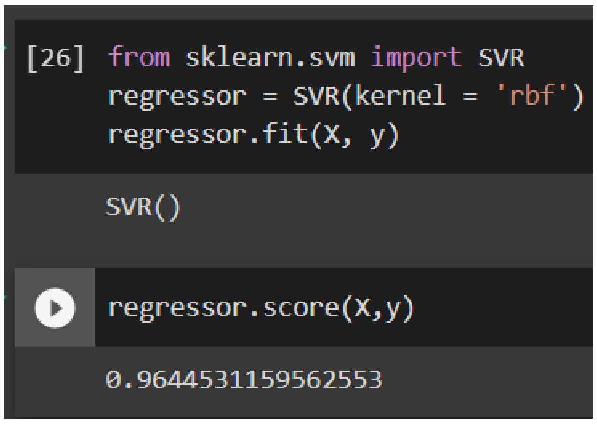


THE ABOVE GRAPHS REPRESENT THE TEST AND TRAINING SET USED FOR THE CLASSIFICATION

FOR SVM REGRESSION THE RESULTS ARE QUITE SIMILAR.



from sklearn.svm import SVR from sklearn.preprocessing import StandardScaler sc_X = StandardScaler() sc_y = StandardScaler() X = sc_X.fit_transform(X) y = sc_y.fit_transform(y) regressor = SVR(kernel = 'rbf') regressor.fit(X, y)



```
[26] from sklearn.svm import SVR
     regressor = SVR(kernel = 'rbf')
     regressor.fit(X, y)

     SVR()

     regressor.score(X,y)

     0.9644531159562553
```

INFERENCE:- BOTH SVM CLASSIFICATION AND REGRESSION DISPLAY HIGH ACCURACIES FOR THE DATASET TITANIC. THESE ALGORITHMS ARE HIGHLY SUITABLE FOR THE DATASET TITANIC.CSV .

9)MLP FEED FORWARD NEURAL NETWORK ALGORITHM:-

In Machine Learning and Artificial Intelligence, Perceptron is the most commonly used term for all folks. It is the primary step to learn Machine Learning and Deep Learning technologies, which consists of a set of weights, input values or scores, and a threshold. ***Perceptron is a building block of an Artificial Neural Network***. Initially, in the mid of 19th century, **Mr. Frank Rosenblatt** invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence. Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers. This algorithm enables neurons to learn elements and processes them one by one during preparation.

What is the Perceptron model in Machine Learning?

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, ***Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence***.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., **input values, weights and Bias, net sum, and an activation function.**

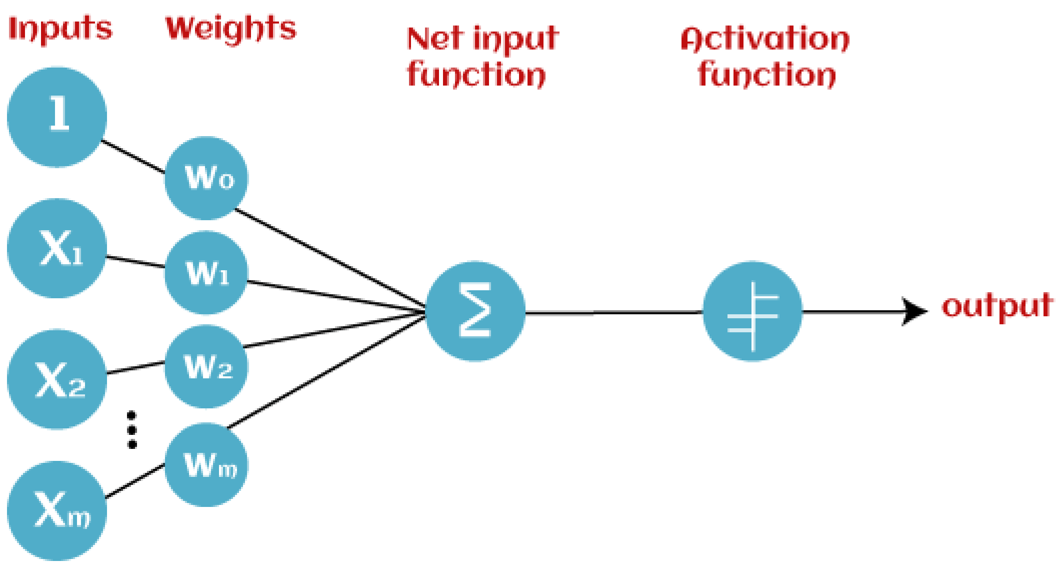
What is Binary classifier in Machine Learning?

In Machine Learning, binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class.

Binary classifiers can be considered as linear classifiers. In simple words, we can understand it as a ***classification algorithm that can predict linear predictor function in terms of weight and feature vectors.***

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

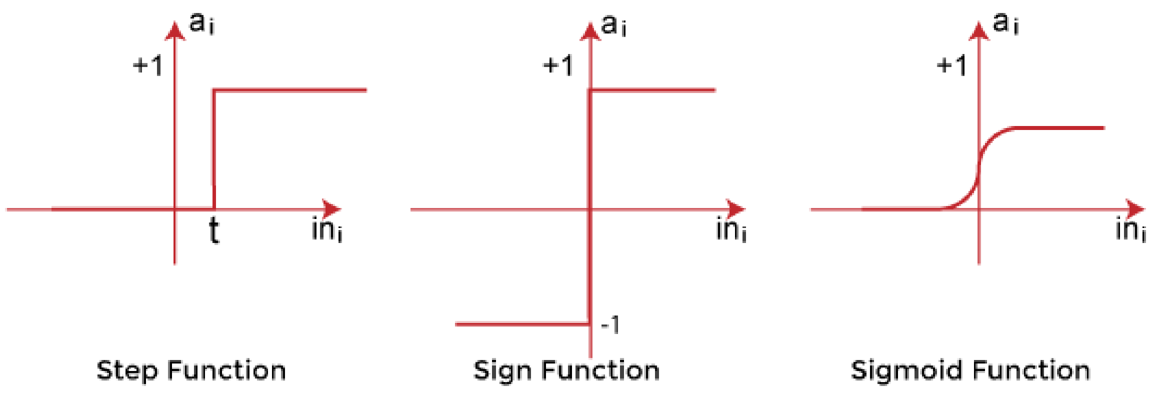
Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function



The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

f(x)=1; if w.x+b>0

otherwise, f(x)=0

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

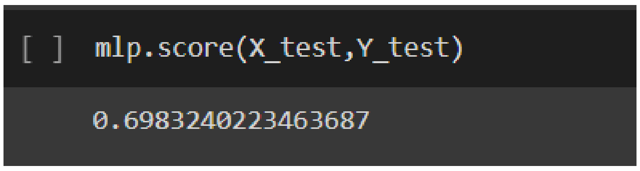
The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.

6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

RESULTS OBTAINED AFTER IMPLEMENTING MLP ON DATASET TITANIC:-

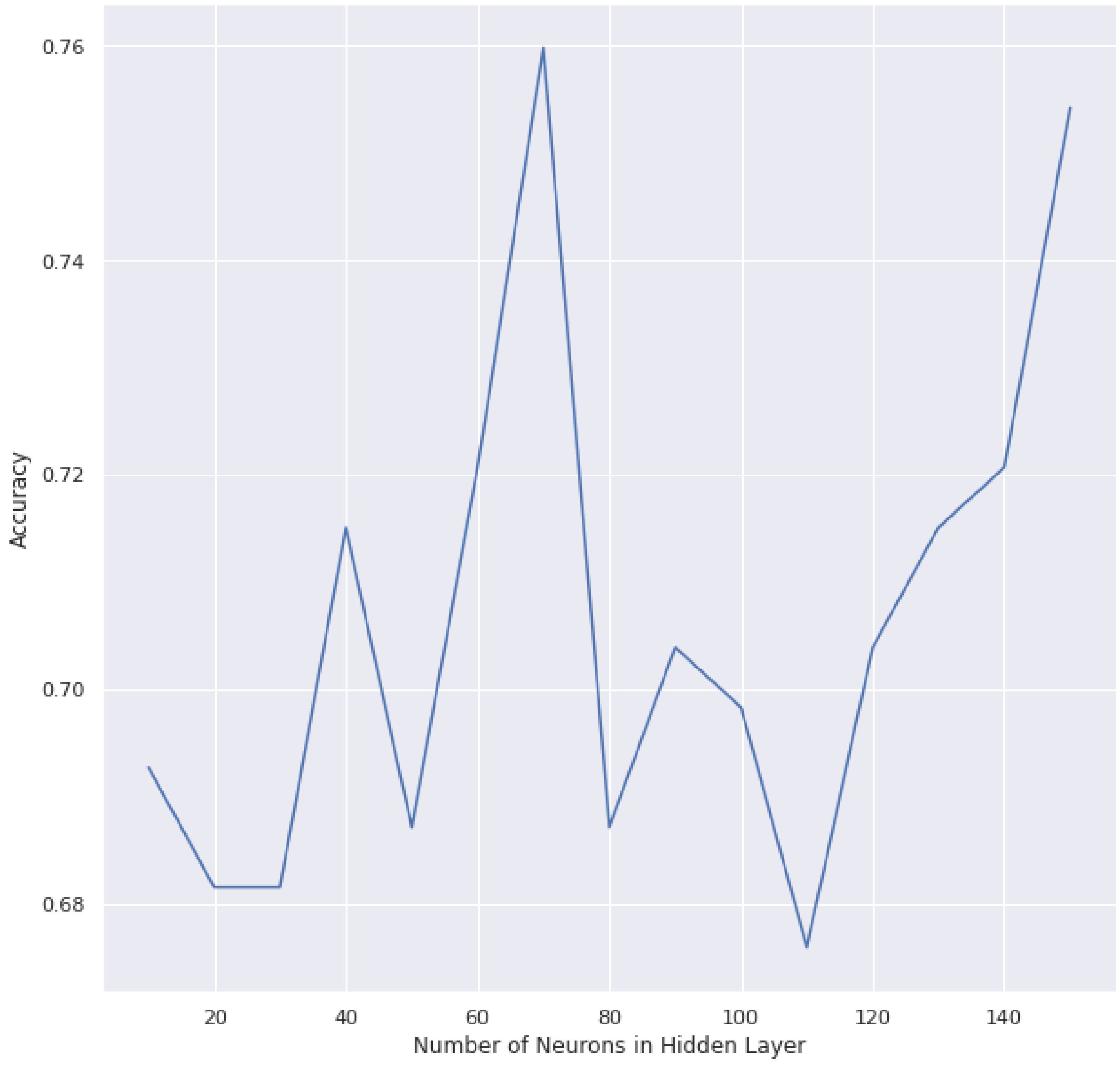
ACCURACY WITHOUT USING HIDDEN LAYER



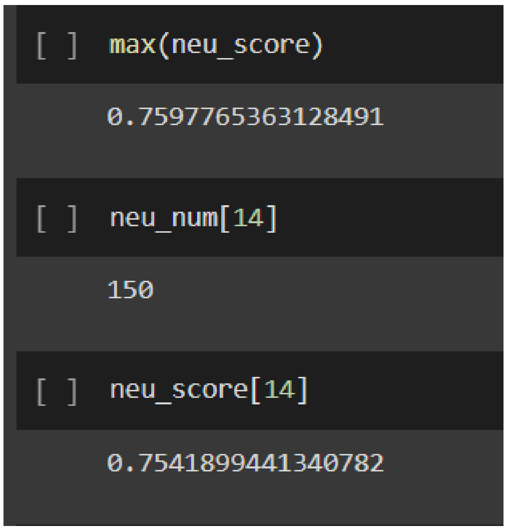
```
[ ]  mlp.score(X_test,Y_test)

     0.6983240223463687
```

HAS THE WORST ACCURACY

ACCURACY USING HIDDEN LAYER



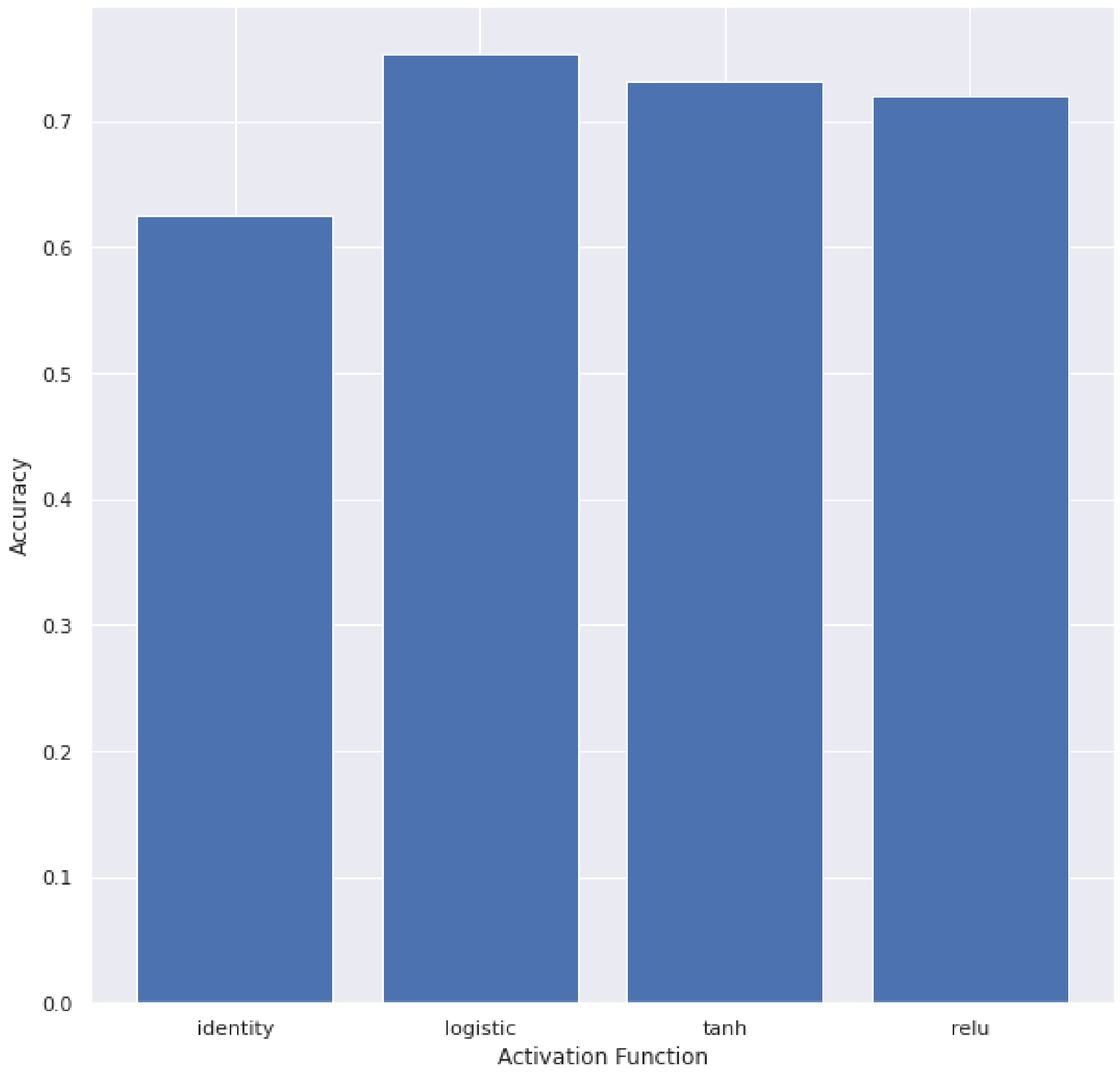
GRADUAL IMPROVEMENT IS SEEN AND PERFORMS BETTER WITH HIDDEN LAYER



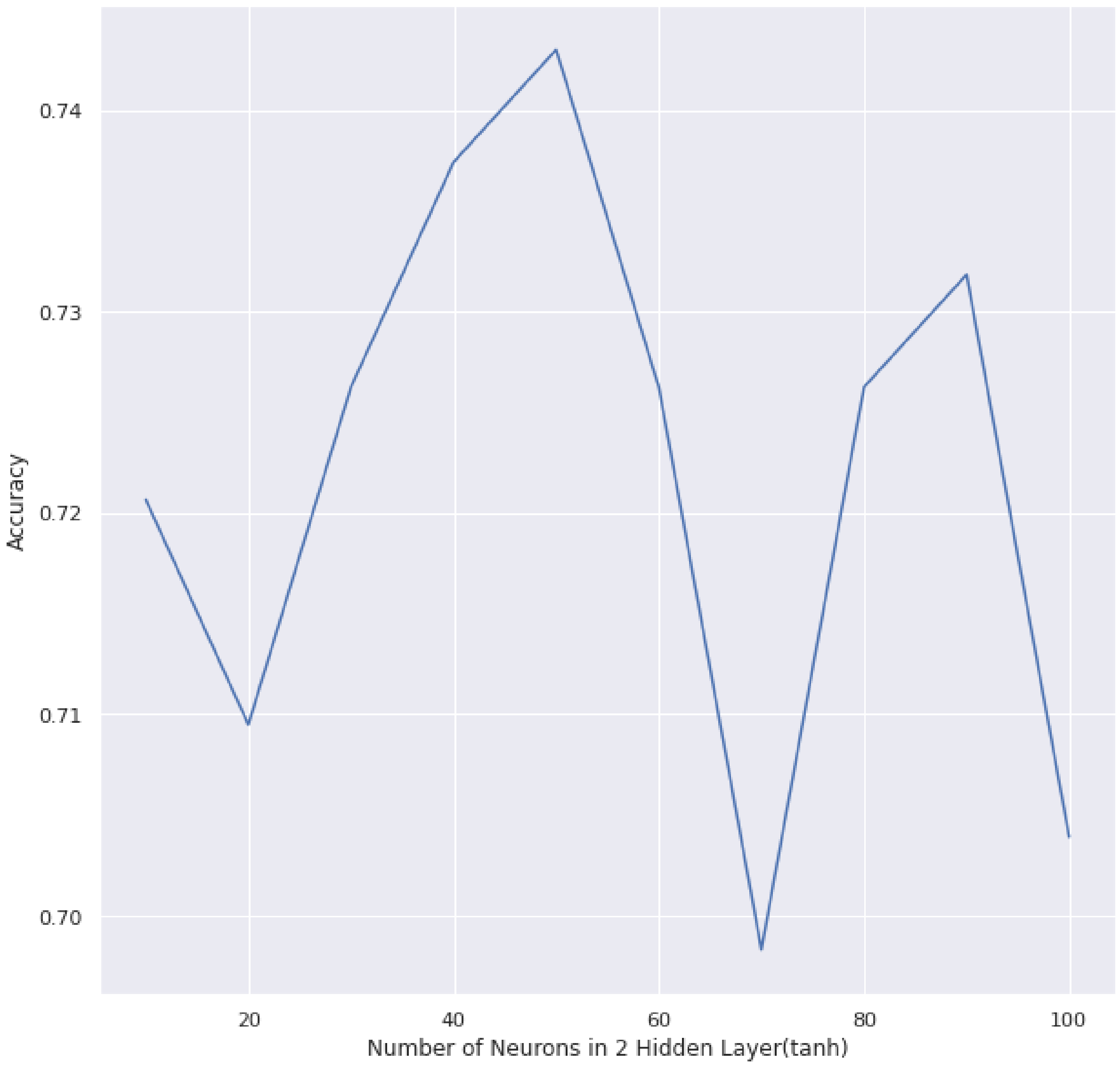
```
[ ]  max(neu_score)

     0.7597765363128491

[ ]  neu_num[14]

     150

[ ]  neu_score[14]

     0.7541899441340782
```

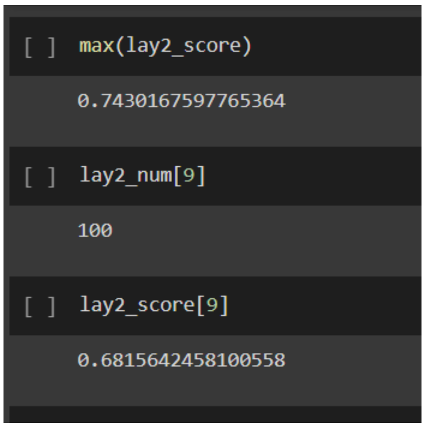
THE SCORES ARE HIGHER

ACCURACY USING DIFFERENT ACTIVATION FUNCTIONS IN THE HIDDEN LAYER



ACCURACY USING 2 HIDDEN LAYERS





```
[ ] max(lay2_score)

    0.7430167597765364

[ ] lay2_num[9]

    100

[ ] lay2_score[9]

    0.6815642458100558
```

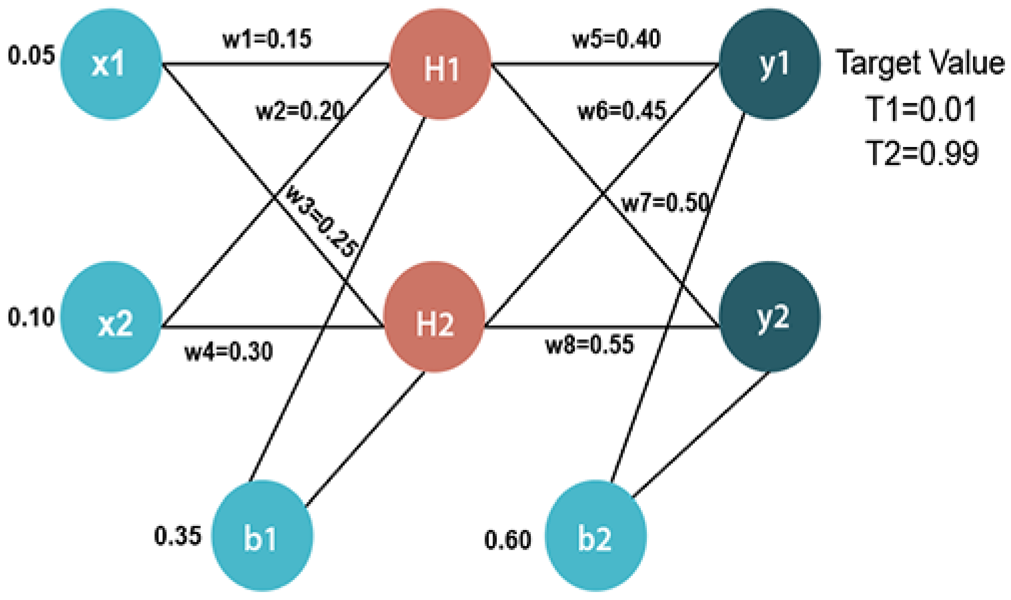
INFERENCE:- DESPITE THE SLOPES MLP CLASSIFIER HAS HIGHER ACCURACY VALUES FOR THE GIVEN DATASET AND IT IS INFERRED THAT THE NO. LAYERS MAY INCREASE THE ACCURACY. THE DEFAULT(IDENTITY) ACCURACY IS HIGHER FOR THE PREPROCESSED DATA SET THAN THE UNPROCESSED ONES IT ALSO SHOWS HIGHER ACCURACY FOR THE HIDDEN LAYERS THAN THE BASE DATA SET AND HAS HIGHER VALUES FOR LOGISTIC AND TANH ACTIVATION FUNCTIONS. THUS THE PREPROCESSING TECHNIQUES HELPS IMPROVE CLASSSIFICATION ACCURACY FOR THE HIDDEN LAYER AND SPECIFIC ACTIVATION FUNCTIONS-LOGISTIC,TANH WHILE GIVING LOWER SCORES FOR IDENTITY AND RELU

10)BPN ALGORITHM:-

Backpropagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter. Similarly here we also use gradient descent algorithm using Backpropagation.

For a single training example, **Backpropagation** algorithm calculates the gradient of the **error function**. Backpropagation can be written as a function of the neural network. Backpropagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.

The main features of Backpropagation are the iterative, recursive and efficient method through which it calculates the updated weight to improve the network until it is not able to perform the task for which it is being trained. Derivatives of the activation function to be known at network design time is required to Backpropagation.



Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Two Types of Backpropagation Networks are:

- Static Back-propagation
- Recurrent Backpropagation

Static back-propagation:

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Backpropagation:

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is nonstatic in recurrent backpropagation.

History of Backpropagation

- In 1961, the basics concept of continuous backpropagation were derived in the context of control theory by J. Kelly, Henry Arthur, and E. Bryson.
- In 1969, Bryson and Ho gave a multi-stage dynamic system optimization method.
- In 1974, Werbos stated the possibility of applying this principle in an artificial neural network.
- In 1982, Hopfield brought his idea of a neural network.
- In 1986, by the effort of David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, backpropagation gained recognition.
- In 1993, Wan was the first person to win an international pattern recognition contest with the help of the backpropagation method.

Backpropagation Key Points

- Simplifies the network structure by elements weighted links that have the least effect on the trained network
- You need to study a group of input and activation values to develop the relationship between the input and hidden unit layers.
- It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.
- Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
- Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.

Best practice Backpropagation

Backpropagation in neural network can be explained with the help of “Shoe Lace” analogy

Too little tension =

- Not enough constraining and very loose

Too much tension =

- Too much constraint (overtraining)

- Taking too much time (relatively slow process)
- Higher likelihood of breaking

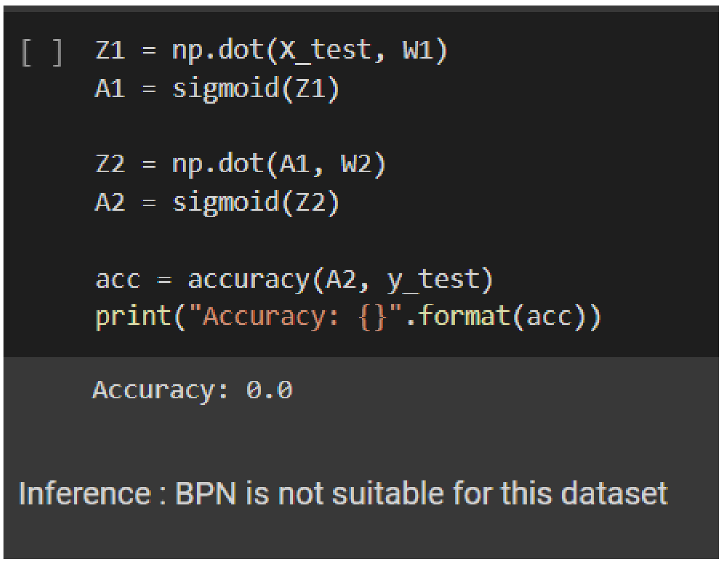
Pulling one lace more than other =

- Discomfort (bias)

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.
- Back propagation algorithm in data mining can be quite sensitive to noisy data
- You need to use the matrix-based approach for backpropagation instead of mini-batch.

RESULTS OBSERVED USING BPN ON OUR DATASET AND MODIFIED DATASET USING IRIS:-

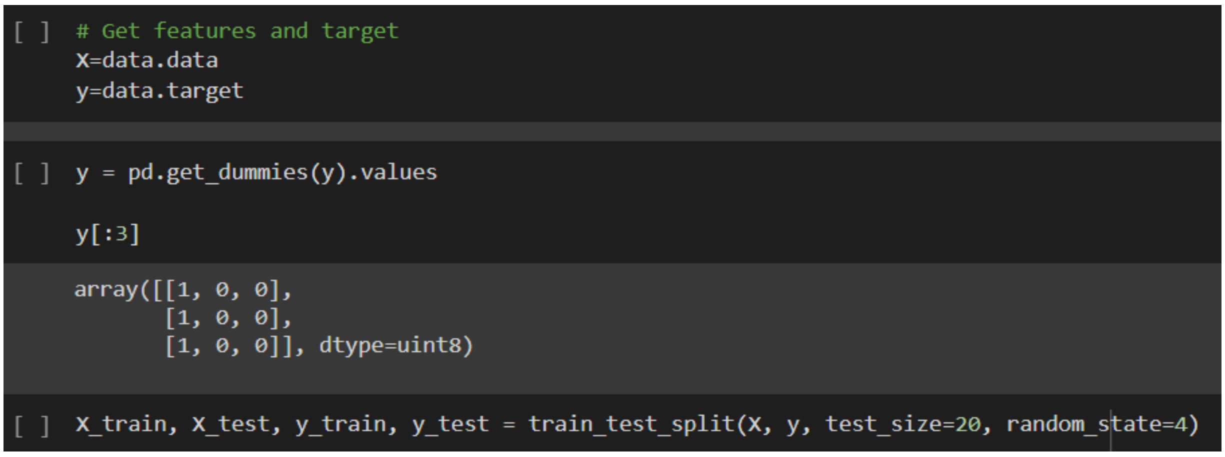


```
[ ]  Z1 = np.dot(X_test, W1)
     A1 = sigmoid(Z1)

     Z2 = np.dot(A1, W2)
     A2 = sigmoid(Z2)

     acc = accuracy(A2, y_test)
     print("Accuracy: {}".format(acc))

     Accuracy: 0.0
```

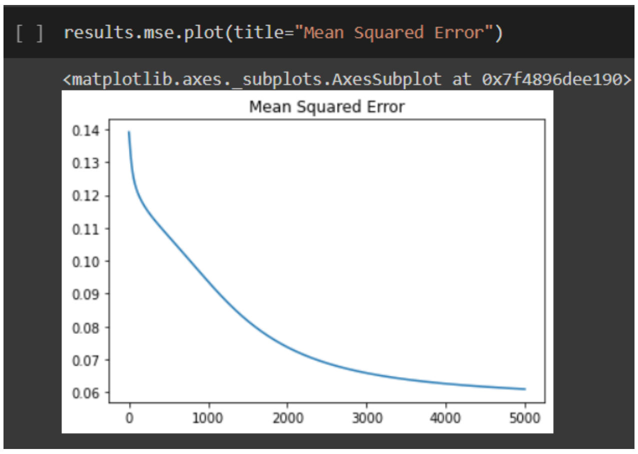
Inference : BPN is not suitable for this dataset

WITHOUT FURTHER FINE TUNING OUR DATASET WAS NOT SUITABLE TO UNDERGO BPN.

WE USED A FRESH DATSET FOLLOWING THE MODEL OF OUR OLD ONE ALONG WITH FEW MODIFICATIONS

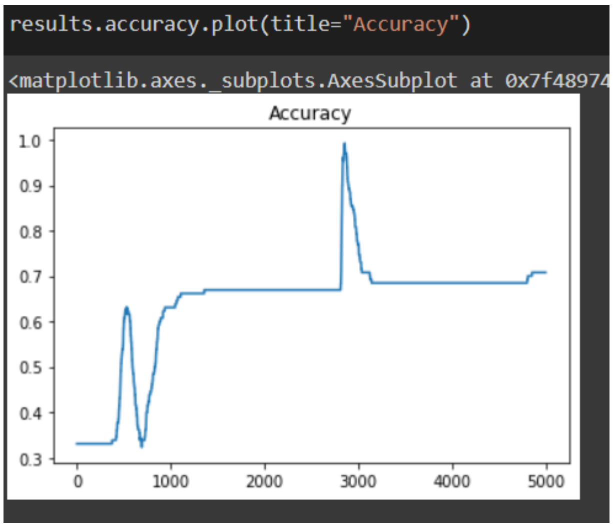


```
[ ]  # Get features and target
     X=data.data
     y=data.target

[ ]  y = pd.get_dummies(y).values

     y[:3]

     array([[1, 0, 0],
            [1, 0, 0],
            [1, 0, 0]], dtype=uint8)

[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
```



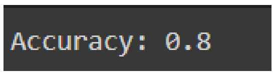
```
[ ] results.mse.plot(title="Mean Squared Error")

<matplotlib.axes._subplots.AxesSubplot at 0x7f4896dee190>
```

WE OBTAINED THE FOLLOWING GRAPH USING MEAN SQUARED ERROR FUNCTION AND DETERMINED THAT THE MSE DROPPED GRADIENTLY IF THE VALUES OF ITERATIONS ON THE X AXIS INCREASED DRASTICALLY. IT DOES HOWEVER FORM THE ELBOW SHAPE WE USUALLY OBSERVE IN K MEANS .



```
results.accuracy.plot(title="Accuracy")

<matplotlib.axes._subplots.AxesSubplot at 0x7f48974
```

WE OBSERVE THE OPPOSITE WITH ACCURACY, AS THE NO. OF ITERATIONS INCREASES THE ACCURACY VALUE TENDS TO PEAK AT CERTAIN SPECIFIC INTERVALS.



```
Accuracy: 0.8
```

THE FINAL ACCURACY VALUE AFTER BPN

INFERENCE:- WE ARE ABLE TO OBSERVE THE DIFFERENT AFFECTS OF BPN ON VARIOUS VERSIONS OF THE DATASET WITH THE WORST CASE OF ACCURACY BEING 0 AND THE BEST CASE OF THE MODIFIED FRESH DATASET BEING 0.8 .

THE BPN ALGORITHM IS OBSERVED TO HAVE MORE SENSITIVITY TO INITIAL PARAMETERS AND REQUIRES CAREFUL TUNING OF HYPER PARAMETERS.

END OF REPORT

THANK YOU