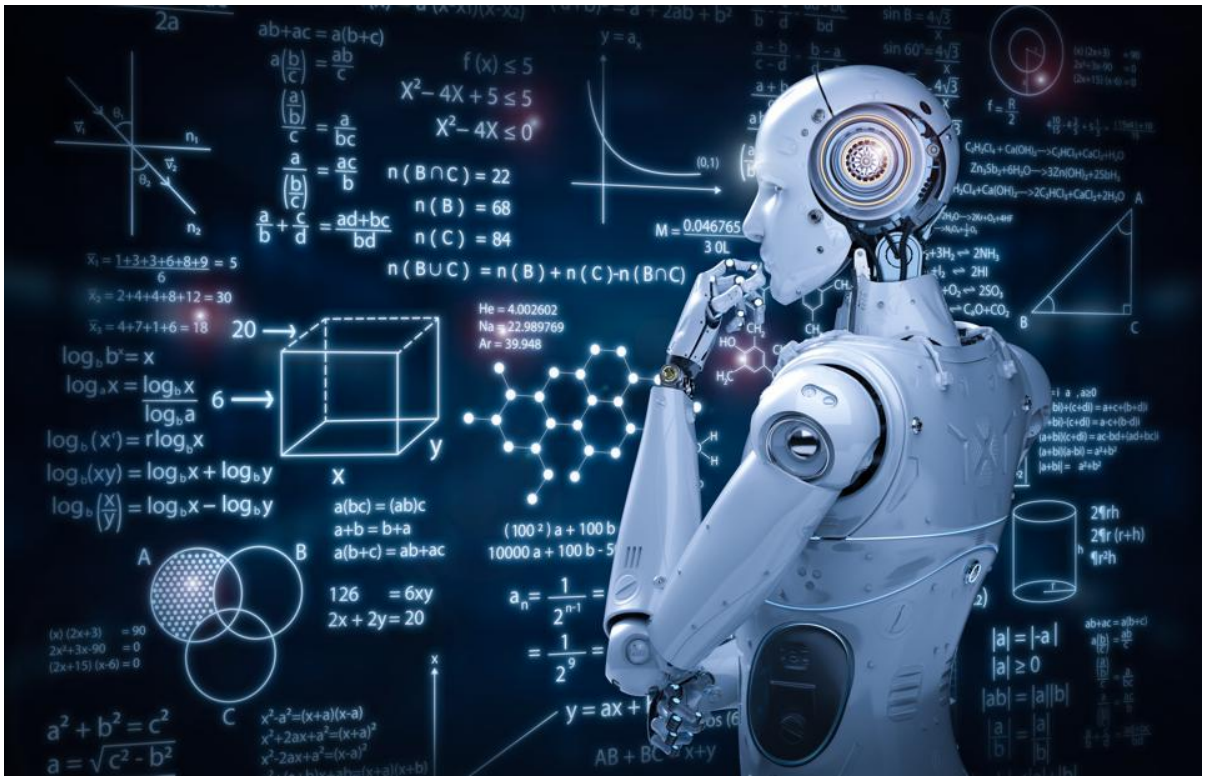
PROGRAMMING LANGUAGE II_ PYTHON

DATASCIENCE PROJECT



SUBMITTED TO: SIR. MUBASHIR

DEPARTMENT: Mathematics (CMAT)

DATE: 22-12-2022

By: MAHEEN KHAN

Table of contents(ToC)

This section contains a list of topics sub-topics, charts, graph and other pictorial information used in the project, which are as below:

- Importing Libraries

- Reading the dataset into a dataframe
- Droping some columns which is not relevant to analysis (ie those which are not numeric)
- Check for null values
- replacing missing values with interpolated values
- Dropping rows with missing Values
- Converting Categorial Columns to binary
- Splitting Dataset
- Building Decision Tree Classifier
- Building a Random Forest classifer
- Building Nave Bayes Classifier
- Building KNN classifier
- Building Logistic Regression Classifier
- Building Support Vector Machine
- Predicting Probabilities
- Probabilities for the positive outcome is kept
- Computing the AUROC Values
- Displaying the AUROC Scores
- Calculating The Roc Curve
- Plot the ROC Curve
- Computing Conclusions and Results

ABSTRACT

Data science is the practice of mining large data sets of raw data, both structured and unstructured, to identify patterns and extract actionable insight from them. This is an interdisciplinary field, and the foundations of data science include statistics, inference, computer science, predictive analytics, machine learning algorithm development, and new technologies to gain insights from big data.

In this project, we have analysed a data of titanic to check the rate of survival. Along with predictions and sorting and analysing, we have applied 7 different classifications in oyr project. However, the main working along with classifications are:

importing csv data

Converting categorical data to binary

Performing Classification using Decision Tree Classifier

Using Random Forest Classifier

Using Gradient Boosting Classifier

Examining the Confusion Matrix

Using Nave Bayes

Using Logistic Regression

Using KNN

Using SVM

Using ROC graph

DATA SELECTION

The data contains liste of passengers along with other attribute like sex, seat no, names, cabins, etc, who were in the titanic ship. The data uncovers the survival and unsurvival of each passenger which is large to count and put calculations easily. With the help of machine learning algorithms we are able to uncover the rate of survivals for the large data.

DATASET (Division and Methodology):

The data has been split into two groups:

1. training set (train.csv)
2. test set (test.csv)

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the “ground truth”) for each passenger. Your model will be based on “features” like passengers’ gender and class. You can also use feature engineering to create new features.

The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

PROBLEM STATEMENT:

Using titanic dataset to analyze the rate of survival of the passengers who were in the ship.

ANALYSIS SECTION:

In this section we have explained what and the procedure of our analyzation. Including codes, results and charts...

Importing libraries:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets.

Seaborn: seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
import pandas as pd
import numpy as np
import seaborn as sb
```

Read the dataset into a dataframe

We are working on the dataset of Titanic So first we load the dataset .

```
# Read the dataset into a dataframe
df = pd.read_csv('/content/titanic1.csv', sep='\t', engine='python')
```

```
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

The first 10 columns of the dataset.

Drop some columns which is not relevant to analysis (they are not numeric)

(Lily May Peel)

```
cols_to_drop = ['Name', 'Ticket', 'Cabin']
df = df.drop(cols_to_drop, axis=1)
```

Drop the columns which have null values such as in ticket and as well as the cabin as well the cabin we are dropping in because there are a lot of missing values there and we cannot use it because the missing values are just too much that if we drop all the missing value our rows or how there are little data so the best thing to do is drop the columns which are not numeric.

```
df.head(3)
```

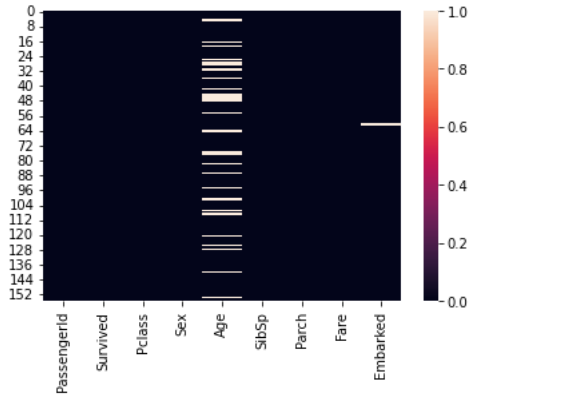
	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S

Check for null values

isnull function: isnull() function detect missing values in the given series object. It return a boolean same-sized object indicating if the values are NA. Missing values gets mapped to True and non-missing value gets mapped to False

```
sb.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f11dc951940>
```

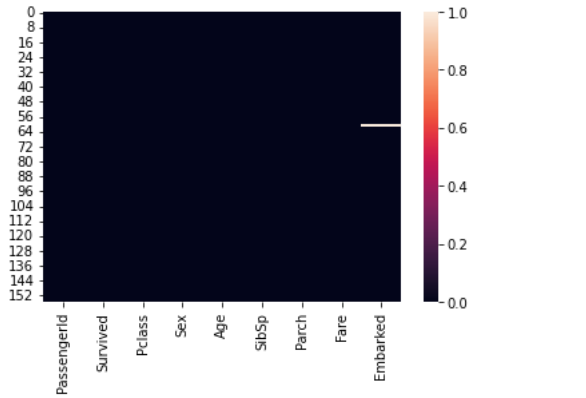


Now so to check whether there are no values because no value is actually a problem in analysis so we check whether there are no values we are now going to use sea borne and see in a graphical form where there are no values I'm gonna run this . so it tells us that the age contains several told Values .

To replace missing values with interpolated values

```
df['Age'] = df['Age'].interpolate()
sb.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f11da008d00>
```



Now we are going to replace this null values by using the method call Interpolation so basically interpolation is saying that if there are two ages that is let's say 15 and 20 it means if there is a by a a row that is in between therefore that would be between 15 and 20 so in that way im going to fill up all the missing values using interpolation.

Drop rows with missing Values

dropna function: The dropna() method removes the rows that contains NULL values. The dropna() method returns a new DataFrame object unless the inplace parameter is set to True , in that case the dropna() method does the removing in the original DataFrame instead.

```
df = df.dropna()
```

```
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

if we run the heatmap again using Seaborn you can see is all the missing values has being interpolated and now we have one single missing value in impact so we can actually just simply drop it by using the dropna to simply drop it So we're just using it.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 155 entries, 0 to 155
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  155 non-null    int64  
 1   Survived     155 non-null    int64  
 2   Pclass       155 non-null    int64  
 3   Sex          155 non-null    object 
 4   Age          155 non-null    float64
 5   SibSp        155 non-null    int64  
 6   Parch        155 non-null    int64  
 7   Fare         155 non-null    float64
 8   Embarked     155 non-null    object 
dtypes: float64(2), int64(5), object(2)
memory usage: 12.1+ KB
```

Some information of the columns of Dataset.Here using head and df.info function we have extracted some information as well as review our data after removing null values.

Convert Categorial Columns to binary:

Now we have categorical values in our data set which is a problem again. Therefore, we convert categorical values to binary.

To do that, create dummycolumns for the columns you want to convert, concentrate it with the dataframe, then draw consistant columns.

```
# First, create dummy columns from the Embarked and Sex columns
EmbarkedColumnDummy = pd.get_dummies(df['Embarked'])
SexColumnDummy = pd.get_dummies(df['Sex'])
```

```
df.head(10)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S
5	6	0	3	male	44.5	0	0	8.4583	Q
6	7	0	1	male	54.0	0	0	51.8625	S
7	8	0	3	male	2.0	3	1	21.0750	S
8	9	1	3	female	27.0	0	2	11.1333	S
9	10	1	2	female	14.0	1	0	30.0708	C

```
df = pd.concat((df, EmbarkedColumnDummy, SexColumnDummy), axis=1)
```

let me trace dummy columns so if you create Dummy columns. Dummy columns are created To simply do the dummy columns I'm going to kind of insert this cell below and want to show you how Tommy column work. So in this we have created a column for male and female.

```
#Check that the columns were concentrated
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	C	Q	S	female	male
0	1	0	3	male	22.0	1	0	7.2500	S	0	0	1	0	1
1	2	1	1	female	38.0	1	0	71.2833	C	1	0	0	1	0
2	3	1	3	female	26.0	0	0	7.9250	S	0	0	1	1	0
3	4	1	1	female	35.0	1	0	53.1000	S	0	0	1	1	0
4	5	0	3	male	35.0	0	0	8.0500	S	0	0	1	0	1

```
# Drop the redundant columns thus converted
df = df.drop(['Sex','Embarked'],axis=1)
```

Separate the data from x and y. So survived is actually the target variable what we called is dependent variable. We are trying to check the Y and pessenger ID . People's added others are x1,x2 and so on .

```
# Seperate the dataframe into X and y data
X = df.values
y = df['Survived'].values

# Delete the Survived column from X
X = np.delete(X,1,axis=1)
```

Split Dataset

Now we want to split our data into training and testing So we are splitting it 70% to the training dataset and remaining 30% to the test dataset.

```
# Split the dataset into 70% Training and 30% Test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Build Decision Tree Classifier

So let's start classifications, First we will be applying decision tree classifier which has been imported from sklearn library. dt_clf is the name of our classifier. Following is the code for fitting and predicting the decision tree classifier. At the last of our code we have checked the scores of the classifiers which is obtained as 1.0% which isn't that bad.

```
# Using simple Decision Tree classifier
from sklearn import tree
dt_clf = tree.DecisionTreeClassifier(max_depth=5)
dt_clf.fit(X_train, y_train)
dt_clf.score(X_test, y_test)

y_pred = dt_clf.predict(X_test)
dt_clf.score(X_test, y_pred)
```

```
1.0
```

```
y_pred = dt_clf.predict(X_test)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

```
array([[30,  5],
       [ 5,  7]])
```

Build a Random Forest classifer

So let's moving forward we have applied Random Forest Classifier, which again has been imported from Sklearn library. rf_clf is the new name of our new Classifier and again running code for testing, training, predicting along with the scores we get 0.787% which is very close to our previous classifier.

```
from sklearn import ensemble
rf_clf = ensemble.RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)
rf_clf.score(X_test, y_test)
```

```
0.7872340425531915
```

Build Nave Bayes Classifier

Another classifier we use is Naive Bayes . So we are using guassian Naive Bayes. As we apply training and testing in naive Bayes we see that there are 76% accuracy which is almost near to random forest which is 0.765%.

```
from sklearn.naive_bayes import GaussianNB
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
nb_clf.score(X_test, y_test)
```

```
0.7659574468085106
```

Build KNN clasifier

K-NN algorithm stores all the available data and classifies a new data point based on the similarity Now we apply KNN, so we see that there are only 57.4% which is very low. So we check more algorithm.

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(X_train, y_train)
knn_clf.score(X_test, y_test)
```

```
0.574468085106383
```

Build Logistic Regression Classifier

It is used to calculate or predict the probability of a binary (yes/no) event occurring. Then we apply logistic regression And we see that there are 80.8% which fits well than other dataset.

```
from sklearn.linear_model import LogisticRegression
lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_clf.score(X_test, y_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.8085106382978723
```

Build Support Vector Machine

Another classifier is SVM (Support Vector Machine) Classifier. We have imported this classifier from sklearn.svm. After going through the similar codes for this classifier as well. we get 0.89 score which is better. So far, SVM is better Classifier comparatively. In addition, In the first line of the code kennel as Linear can also be used as default.

```
from sklearn.svm import SVC
sv_clf = SVC(probability=True, kernel='linear')
sv_clf.fit(X_test, y_test)
sv_clf.score(X_test, y_test)
```

```
0.8936170212765957
```

Prediction Probabilities:

The below code show the better predicted probabilities of all the Classifiers.

```
r_probs = [0 for _ in range(len(y_test))]
rf_probs = rf_clf.predict_proba(X_test)
nb_probs = nb_clf.predict_proba(X_test)
dt_probs = dt_clf.predict_proba(X_test)
knn_probs = knn_clf.predict_proba(X_test)
lr_probs = lr_clf.predict_proba(X_test)
sv_probs = sv_clf.predict_proba(X_test)
```

The sklearn library has the predict_proba() command that can be used to generate a two column array, the first column being the probability that the outcome will be 0 and the second being the probability that the outcome will be 1. The sum of each row of the two columns should also equal one.

Probabilities for the positive outcome is kept.

```
rf_probs = rf_probs[:, 1]
nb_probs = nb_probs[:, 1]
dt_probs = dt_probs[:, 1]
knn_probs = knn_probs[:, 1]
lr_probs = lr_probs[:, 1]
sv_probs = sv_probs[:, 1]
```

Compute the AUROC Values

Now we compute the AUROC values to display the AUROC scores. AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). Figure 5. AUC (Area under the ROC Curve). AUC provides an aggregate measure of performance across all possible classification thresholds.

```
from sklearn.metrics import roc_curve, roc_auc_score
r_auc = roc_auc_score(y_test, r_probs)
rf_auc = roc_auc_score(y_test, rf_probs)
nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)
knn_auc = roc_auc_score(y_test, knn_probs)
```

```
lr_auc = roc_auc_score(y_test, lr_probs)
sv_auc = roc_auc_score(y_test, sv_probs)
```

Display the AUROC Scores

```
print("Random Prediction: AUROC = %.3f" %(r_auc))
print("Random Forest: AUROC = %.3f" %(rf_auc))
print("Nave Bayes: AUROC = %.3f" %(nb_auc))
print("Decision Trees: AUROC = %.3f" %(dt_auc))
print("K Nearest Neighbors: AUROC = %.3f" %(knn_auc))
print("Logistic Regression: AUROC = %.3f" %(lr_auc))
print("Support Vector Machine: AUROC = %.3f" %(sv_auc))
```

```
Random Prediction: AUROC = 0.500
Random Forest: AUROC = 0.757
Nave Bayes: AUROC = 0.714
Decision Trees: AUROC = 0.615
K Nearest Neighbors: AUROC = 0.425
Logistic Regression: AUROC = 0.769
Support Vector Machine: AUROC = 0.910
```

This Will be the AUROC scores of all th classifiers.

Calculate The Roc Curve

At the last we have done an extra coding for ROC. ROC (Receiver Operating Characteristic) basically shows the performance of each classifiers through a graph. The code and results are below:

```
r_fpr, r_tpr, _ = roc_curve(y_test, r_probs)
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)
nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)
knn_fpr, knn_tpr, _ = roc_curve(y_test, knn_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
sv_fpr, sv_tpr, _ = roc_curve(y_test, sv_probs)
```

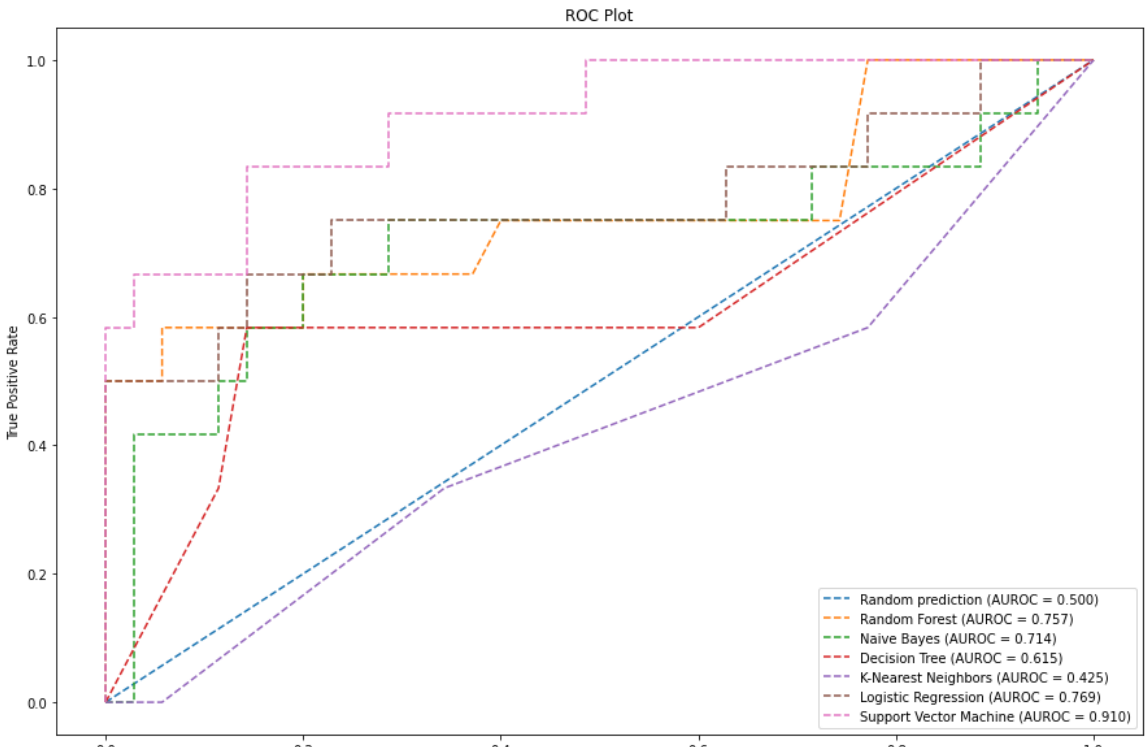
Plot the ROC Curve

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.plot(r_fpr, r_tpr, linestyle='--', label='Random prediction (AUROC = %0.3f)' %r_auc)
plt.plot(rf_fpr, rf_tpr, linestyle='--', label='Random Forest (AUROC = %0.3f)' %rf_auc)
plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' %nb_auc)
plt.plot(dt_fpr, dt_tpr, linestyle='--', label='Decision Tree (AUROC = %0.3f)' %dt_auc)
plt.plot(knn_fpr, knn_tpr, linestyle='--', label='K-Nearest Neighbors (AUROC = %0.3f)' %knn_auc)
plt.plot(lr_fpr, lr_tpr, linestyle='--', label='Logistic Regression (AUROC = %0.3f)' %lr_auc)
plt.plot(sv_fpr, sv_tpr, linestyle='--', label='Support Vector Machine (AUROC = %0.3f)' %sv_auc)

#Title
plt.title('ROC Plot')

#Axis Labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

#Show Legend
plt.legend()
plt.show()
```



CONCLUSION AND RESULT

In the above ROC graph the blue line indicates random classification which basically divides the graph into two. According to the graph the poor classification is of KNN, represented in brown line which lies the below of all the lines and the best classification which is SVM, line lies top of all the other graph lines.

✓ 0s completed at 9:55 PM