BHARAT INTERN TASK 2 - TITANIC CLASSIFICATION

Description of the dataset

PassengerId - Unique ID of passenger

Survived - 1 if passenger survived the disaster

Pclass - Class of passenger: 1 = 1st/Upper, 2 = 2nd/Middle, 3 = 3rd/Lower)

Name - Name of passenger

Sex - Gender of passenger

Age - Age of passenger

SibSp - Number of siblings/spouses aboard the ship

Parch - Number of parents and/or children

Ticket - Ticket number

Fare - Price of ticket

Cabin - Cabin number

Embarked - Point of embarking the ship: C= Cherbourg ,Q=Queenstown,S=Southampton

Importing necessary libraries

In [1]:

```
# linear algebra
import numpy as np
# data processing
import pandas as pd
# data visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [3]:

```
titanic_df = pd.read_csv('D:/MDA 4th Sem/ML/titanic.csv')
titanic_df.head(5)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [4]:

```
titanic_df.shape
```

Out[4]:

```
(891, 12)
```

The dataset contains 891 rows and 12 columns.

In [5]:

```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

The dataset contains 7 numerical columns and 5 categorical columns. The columns 'Age', 'Cabin', and 'Embarked' are having null values.

In [6]:

```
titanic_df.isnull().sum()
```

Out[6]:

```
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
```

The column 'Age' has 177 null values, 'Cabin' has 687 null values, and 'Embarked' has 2 null values.

In [7]:

```
skewness = titanic_df['Age'].skew()
skewness
```

Out[7]:

```
0.38910778230082704
```

Since the skewness is close to 0, the data is approximately symmetric.

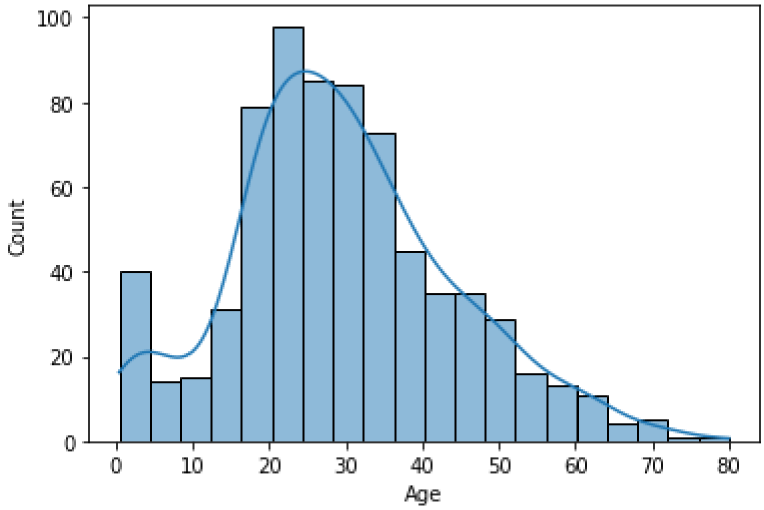
Visualizing the skewness

In [8]:

```
sns.histplot(titanic_df['Age'], kde=True)
```

Out[8]:

```
<AxesSubplot:xlabel='Age', ylabel='Count'>
```



Since the data is not highly skewed and we have less number of missing values in the column 'Age,' we can fill the missing values with mean value.

In [9]:

```
mean_value = titanic_df['Age'].mean()
titanic_df['Age'] = titanic_df['Age'].fillna(mean_value)
```

In [10]:

```
null_values = titanic_df['Age'].isnull().sum()
null_values
```

Out[10]:

```
0
```

Now there are no null values in the column 'Age'.

Treating the column 'Embarked'

In [11]:

```
# the rows containing null values are dropped
titanic_df = titanic_df.dropna(subset=['Embarked'])
```

In [12]:

```
null_values = titanic_df['Embarked'].isnull().sum()
null_values
```

Out[12]:

```
0
```

Since there are only two rows with null values, we are dropping them.

Treating the column 'Cabin'

In [13]:

```
# obtaining the percentage of null values in the column 'Cabin'
titanic_df['Cabin'].isnull().sum()/len(titanic_df['Cabin'])*100
```

Out[13]:

```
77.27784026996626
```

In [14]:

```
titanic_df.drop('Cabin', axis=1, inplace=True)
titanic_df.head()
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

Here we can see in the 'Cabin' feature that there are 77% of null values, which is challenging. Hence we dropped this feature.

Summary Statistics

In [15]:

```
# selecting the categorical variables
categorical_var = titanic_df.select_dtypes(include='object')
# Obtaining summary statistics for the categorical variables
categorical_stat = categorical_var.describe().T
categorical_stat
```

Out[15]:

	count	unique	top	freq
Name	889	889	Braund, Mr. Owen Harris	1
Sex	889	2	male	577
Ticket	889	680	347082	7
Embarked	889	3	S	644

There are 889 passengers on the ship. Among them, 577 are male passengers. Most of the passengers embarked from Southampton.

In [16]:

```
# selecting numerical variables
numerical_var = titanic_df.select_dtypes(exclude='object')
# Obtaining summar statistics for the numerical variables
numerical_stat = numerical_var.describe().T
numerical_stat
```

Out[16]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	889.0	446.000000	256.998173	1.00	224.0000	446.000000	668.0	891.0000
Survived	889.0	0.382452	0.486260	0.00	0.0000	0.000000	1.0	1.0000
Pclass	889.0	2.311586	0.834700	1.00	2.0000	3.000000	3.0	3.0000
Age	889.0	29.653446	12.968366	0.42	22.0000	29.699118	35.0	80.0000
SibSp	889.0	0.524184	1.103705	0.00	0.0000	0.000000	1.0	8.0000
Parch	889.0	0.382452	0.806761	0.00	0.0000	0.000000	0.0	6.0000
Fare	889.0	32.096681	49.697504	0.00	7.8958	14.454200	31.0	512.3292

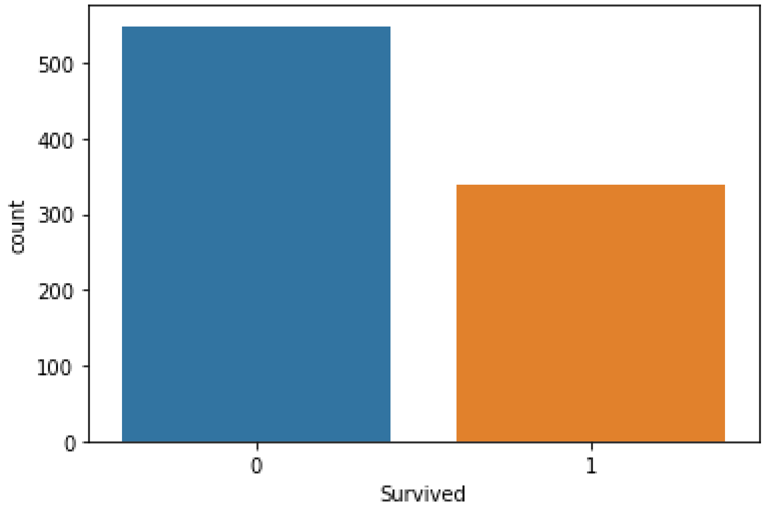
Survived vs Died

In [17]:

```
sns.countplot(x="Survived", data=titanic_df)
```

Out[17]:

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



From the plot, it is clear that most passengers died in the Titanic disaster.

The other critical insight from the plot is that the data is imbalanced. So before going to model the data, we have to balance it.

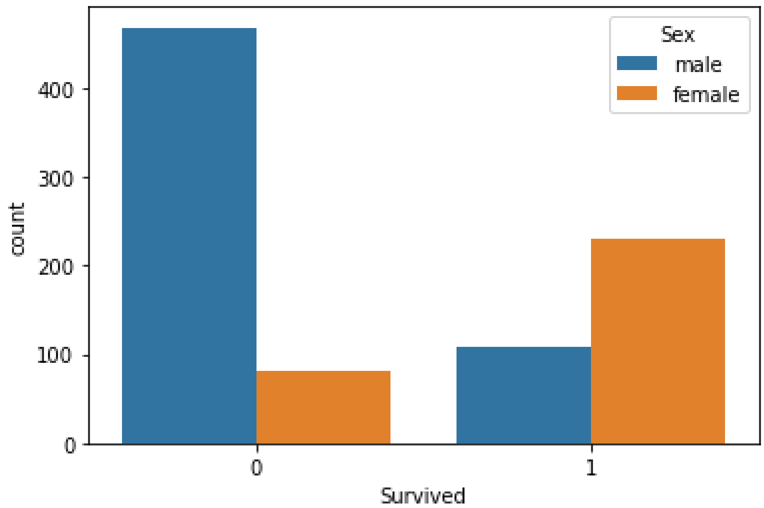
Male vs Female

In [18]:

```
sns.countplot(x='Survived',data=titanic_df,hue='Sex')
```

Out[18]:

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



Female passengers were survived than the male passengers.

Exploring the column 'Age'

In [19]:

```
titanic_df['Age'].min()
```

Out[19]:

```
0.42
```

Youngest Passenger

In [20]:

```
titanic_df['Age'].max()
```

Out[20]:

```
80.0
```

Oldest passenger

In [21]:

```
titanic_df.groupby('Survived')['Age'].value_counts()
```

Out[21]:

```
Survived  Age
0         29.699118    125
          21.000000     19
          28.000000     18
          18.000000     17
          25.000000     17
                       ...
1         47.000000      1
          53.000000      1
          55.000000      1
          62.000000      1
          80.000000      1
Name: Age, Length: 144, dtype: int64
```

Among the people who died, those aged 29 are the most who died.

In [22]:

```
titanic_df['Pclass'].value_counts()
```

Out[22]:

```
3    491
1    214
2    184
Name: Pclass, dtype: int64
```

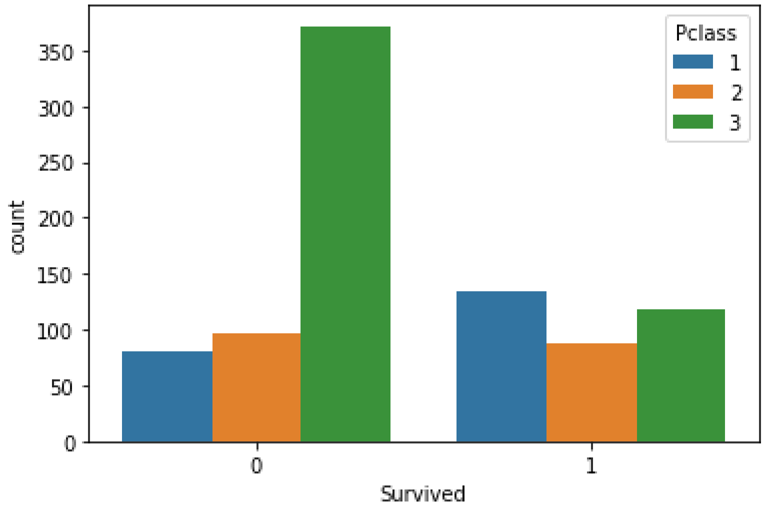
Maximum passengers were in the third class, and the second class contained the least passengers.

In [23]:

```
sns.countplot(x='Survived', data=titanic_df, hue='Pclass')
```

Out[23]:

<AxesSubplot:xlabel='Survived', ylabel='count'>



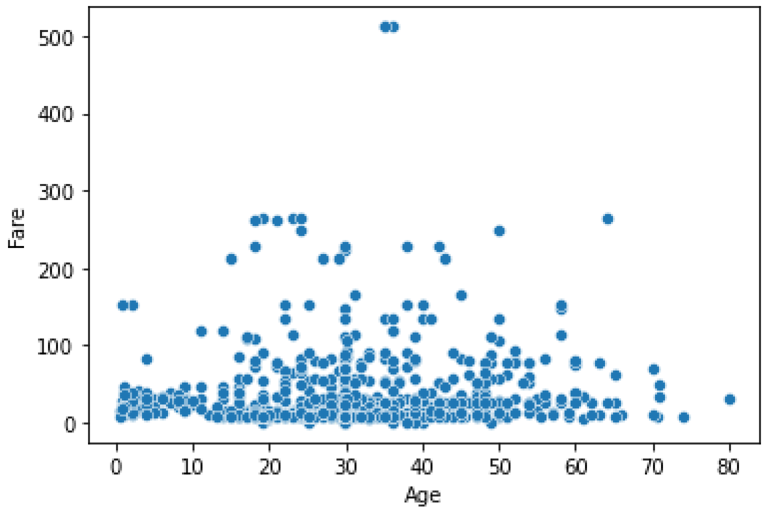
Most of the passengers in the third class died during the Titanic disaster, and the passengers in the first class were the ones who were rescued the most.

In [24]:

```
sns.scatterplot(x=titanic_df['Age'], y=titanic_df['Fare'])
```

Out[24]:

<AxesSubplot:xlabel='Age', ylabel='Fare'>



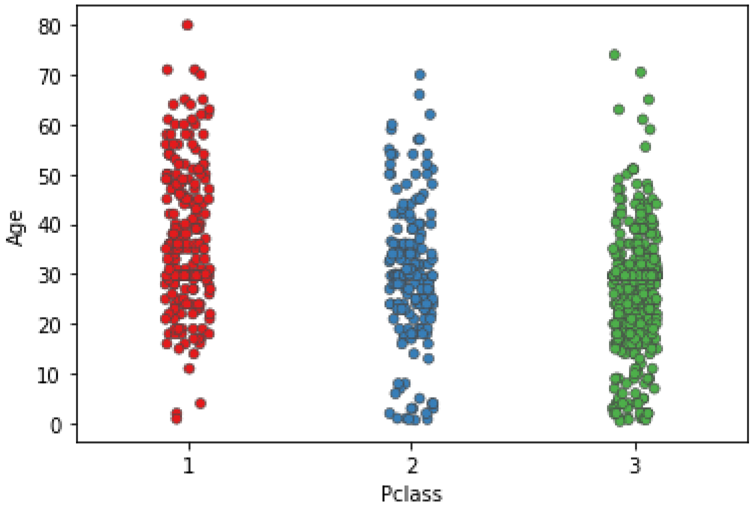
Analyzing the distribution of age within each passenger class

In [25]:

```
sns.stripplot(x=titanic_df['Pclass'], y=titanic_df['Age'], palette='Set1', linewidth=0.6)
```

Out[25]:

```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



In [26]:

```
titanic_df['Embarked'].value_counts()
```

Out[26]:

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [27]:

```
ordinal_mapping = {'Q': 1, 'S': 2,'C':3}
titanic_df['Embarked']= titanic_df['Embarked'].map(ordinal_mapping)
titanic_df.head()
```

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [28]:

```
titanic_df['Sex'].value_counts()
```

Out[28]:

```
male      577
female    312
Name: Sex, dtype: int64
```

In [29]:

```
ordinal_map = {'male': 1, 'female': 2}
titanic_df['Sex']= titanic_df['Sex'].map(ordinal_map)
titanic_df.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Er
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	2	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	2	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	2	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	

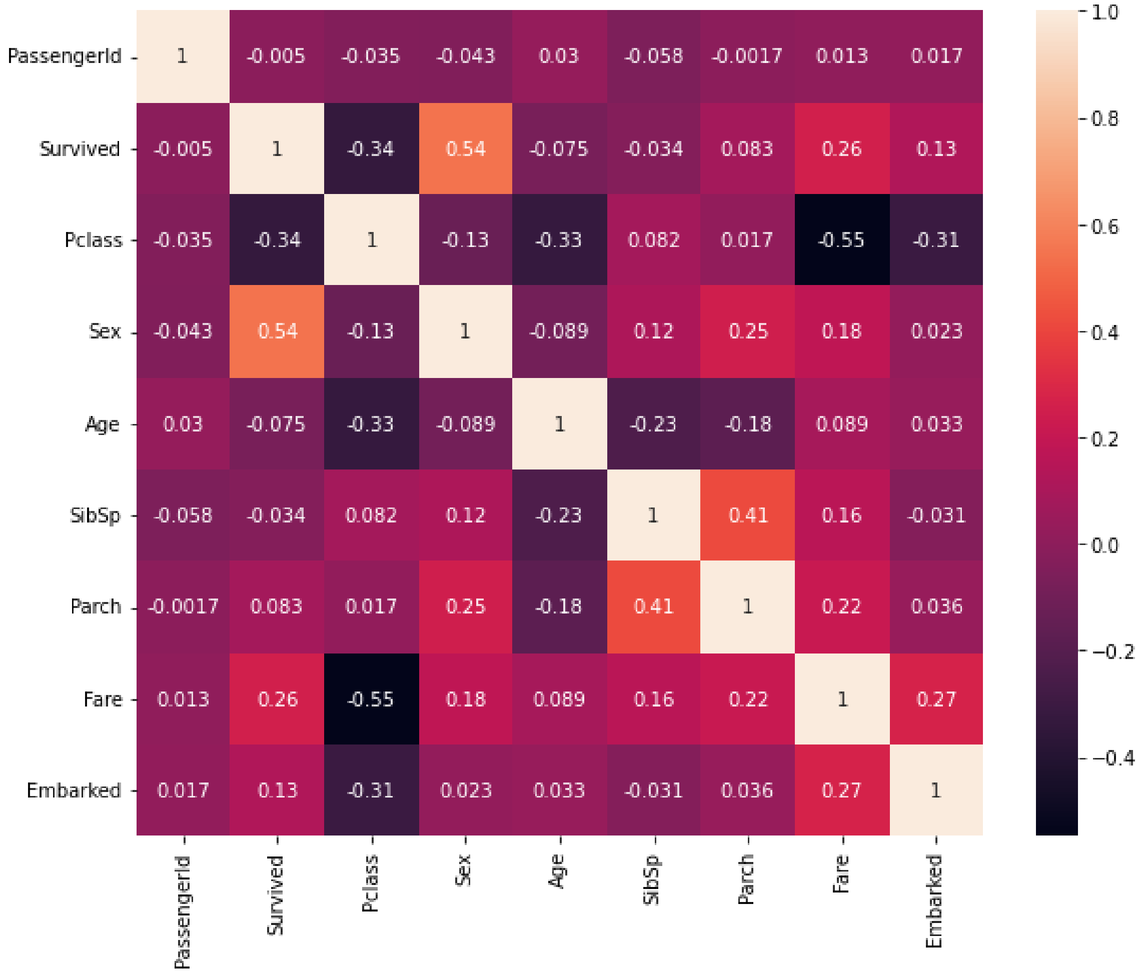
Correlation among variables

In [30]:

```
plt.figure(figsize=(10, 8))
correlation_matrix = titanic_df.corr()
sns.heatmap(correlation_matrix, annot=True)
```

Out[30]:

```
<AxesSubplot:>
```



Among the variables, there is a moderate negetive correlation between Pclass and Fare and there is a positive moderate correlation between Sex and Survived. All the other variables show a weak correlation. So there is no multicollinearity issue.

In [31]:

```
columns_drop = ['PassengerId','Ticket', 'Name']
titanic_df.drop(columns=columns_drop, inplace=True)
```

In [33]:

```
X = titanic_df.drop(['Survived'], axis = 1)
Y = titanic_df['Survived']
```

In [40]:

```
from imblearn.over_sampling import SMOTE

# Create a SMOTE instance
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Apply SMOTE to generate synthetic samples
X, Y = smote.fit_resample(X, Y)
df = pd.concat([pd.DataFrame(X), pd.DataFrame(Y)], axis=1)

# Check the value counts of the target variable after oversampling
print(df['Survived'].value_counts())
```

```
0    549
1    549
Name: Survived, dtype: int64
```

In [41]:

```
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 = 4
```

K-NEAREST NEIGHBOR

In [42]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train,Y_train)
knn_pred = knn.predict(X_test)
```

In [43]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_matrix(Y_test,knn_pred)
```

Out[43]:

```
array([[145,  23],
       [ 74,  88]], dtype=int64)
```

In [44]:

```
accuracy_score(Y_test,knn_pred)
```

Out[44]:

```
0.706060606060606
```

SUPPORT VECTOR MACHINE

In [45]:

```
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(X_train,Y_train)
svm_pred = svm.predict(X_test)
```

In [46]:

```
confusion_matrix(Y_test,svm_pred)
```

Out[46]:

```
array([[135,  33],
       [ 38, 124]], dtype=int64)
```

In [47]:

```
accuracy_score(Y_test,svm_pred)
```

Out[47]:

```
0.7848484848484848
```

RANDOM FOREST CLASSIFIER

In [48]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
rfc_pred = rfc.predict(X_test)
```

In [49]:

```
confusion_matrix(Y_test,rfc_pred)
```

Out[49]:

```
array([[128,  40],
       [ 29, 133]], dtype=int64)
```

In [50]:

```
accuracy_score(Y_test,rfc_pred)
```

Out[50]:

```
0.7909090909090909
```

LOGISTIC REGRESSION

In [53]:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(X_train, Y_train)
LR_pred = LR.predict(X_test)
```

In [54]:

```
confusion_matrix(Y_test,LR_pred)
```

Out[54]:

```
array([[131,  37],
       [ 37, 125]], dtype=int64)
```

In [55]:

```
accuracy_score(Y_test,LR_pred)
```

Out[55]:

```
0.7757575757575758
```

DECISION TREE CLASSSIFIER

In [59]:

```
from sklearn.tree import DecisionTreeClassifier
DT=DecisionTreeClassifier()
DT.fit(X_train,Y_train)
DT_pred = LR.predict(X_test)
```

In [60]:

```
confusion_matrix(Y_test,DT_pred)
```

Out[60]:

```
array([[131,  37],
       [ 37, 125]], dtype=int64)
```

In [66]:

```
accuracy_score(Y_test,DT_pred)
```

Out[66]:

```
0.7757575757575758
```

XGBoosts Classifier

In [68]:

```
from xgboost import XGBClassifier
xgboost = XGBClassifier(n_estimators=1000)
xgboost.fit(X_train,Y_train)
xg_pred = xgboost.predict(X_test)
```

In [69]:

```
confusion_matrix(Y_test,xg_pred)
```

Out[69]:

```
array([[129,  39],
       [ 29, 133]], dtype=int64)
```

In [70]:

```
accuracy_score(Y_test,xg_pred)
```

Out[70]:

```
0.793939393939394
```