About the dataset

The dataset conntains information about individual passengers such as their age,gender,ticket,class,fare,cabin,and whether survived or not.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

DATASET

```
df=pd.read_csv('/content/drive/MyDrive/Datasets/titanic.csv')
df
```

	PassengerId	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	Braund, Mr. Owen Harris	3	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	female	38.0	1	0	PC 17599	71.2833	C85
2	3	Heikkinen, Miss. Laina	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	female	35.0	1	0	113803	53.1000	C123
4	5	Allen, Mr. William Henry	3	male	35.0	0	0	373450	8.0500	NaN

Next steps: Generate code with df View recommended plots

DATA PREPROCESSING AND CLEANING

```
#Table columns
df.columns
```

```
Index(['PassengerId', 'Name', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked', 'Survived'],
      dtype='object')
```

```
#columns datatypes
df.dtypes
```

```
PassengerId      int64
Name            object
Pclass           int64
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
```

```
Embarked        object
Survived         int64
dtype: object
```

```
# Finding missing values
df.isnull().sum()
```

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

```
df.drop_duplicates(inplace=True)
```

```
# Drop unwanted columns
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
```

```
#Missing values handling
df['Age'].fillna(df['Age'].mean(),inplace=True)
```

```
df.Embarked.isna().sum()
df.dropna(inplace=True)
```

```
df.isna().sum()
```

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
Survived    0
dtype: int64
```

```
df.describe()
```

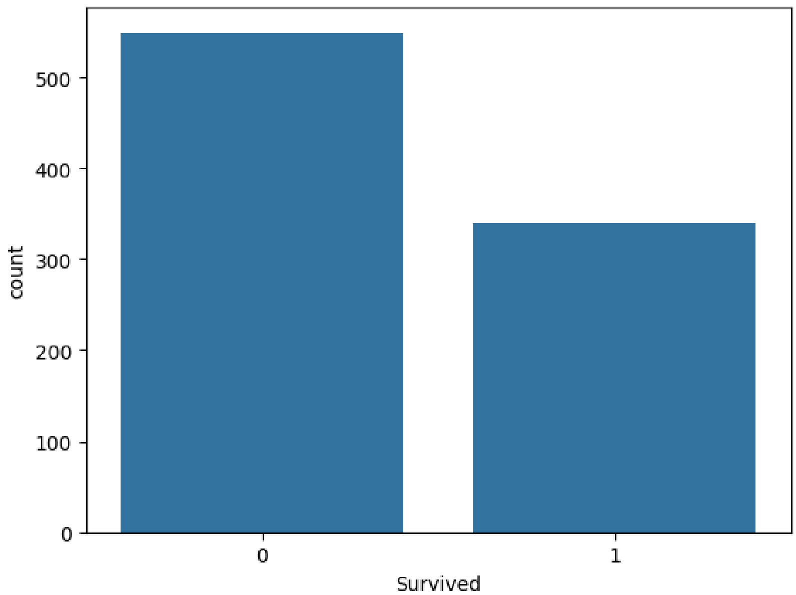
	Pclass	Age	SibSp	Parch	Fare	Survived
count	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000
mean	2.311586	29.653446	0.524184	0.382452	32.096681	0.382452
std	0.834700	12.968366	1.103705	0.806761	49.697504	0.486260
min	1.000000	0.420000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	22.000000	0.000000	0.000000	7.895800	0.000000
50%	3.000000	29.699118	0.000000	0.000000	14.454200	0.000000
75%	3.000000	35.000000	1.000000	0.000000	31.000000	1.000000
max	3.000000	80.000000	8.000000	6.000000	512.329200	1.000000

```
df.Survived.value_counts()
```

```
0    549
1    340
Name: Survived, dtype: int64
```

DATA VISUALIZATION

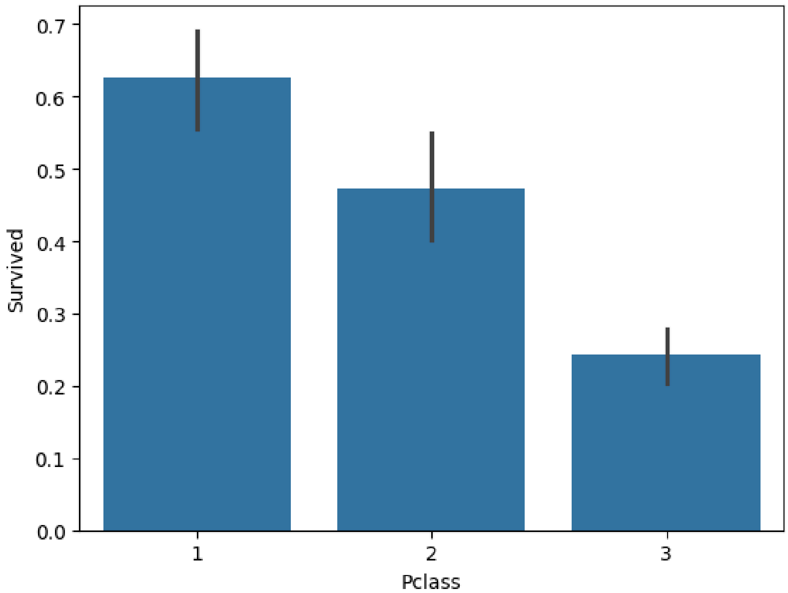
```
# count plot of survivels
sns.countplot(x=df['Survived'])
plt.show()
```




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

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

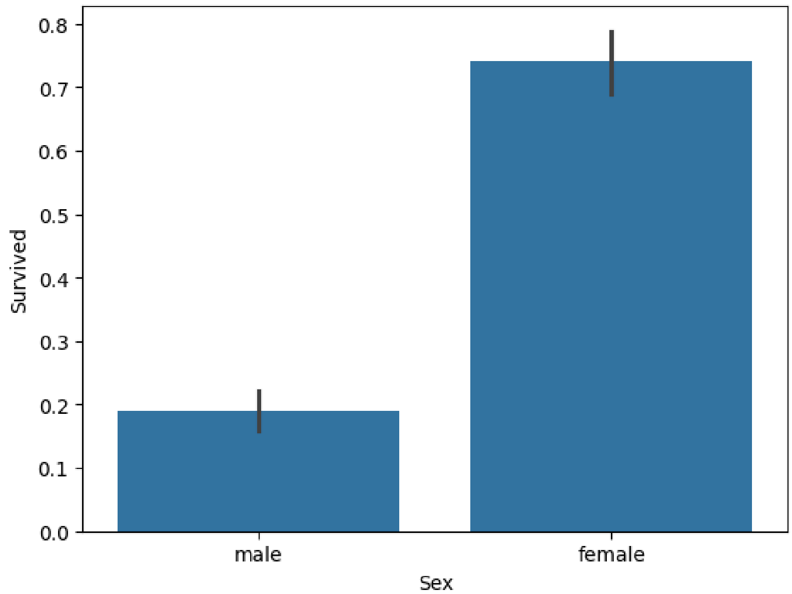
```
sns.barplot(x='Pclass',y='Survived',data=df)
plt.show()
```




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

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

```
sns.barplot(x='Sex',y='Survived',data=df)
plt.show()
```




```
df['SibSp'].value_counts()
```

```
0    606
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

```
df['Parch'].value_counts()
```

```
0    676
1    118
2     80
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64
```

```
df['Fare'].value_counts()
```

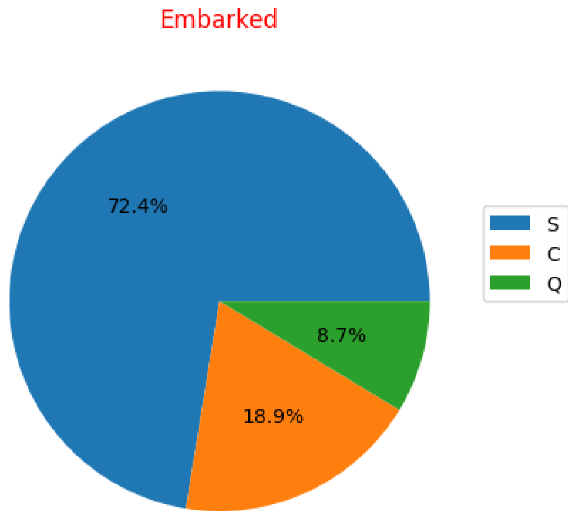
```
8.0500     43
13.0000    42
7.8958     38
7.7500     34
26.0000    31
           ..
35.0000     1
28.5000     1
6.2375      1
14.0000     1
10.5167     1
Name: Fare, Length: 247, dtype: int64
```

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

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

```
plt.pie(df['Embarked'].value_counts(),autopct='%1.1f%%')
plt.legend(df['Embarked'].value_counts().index,loc=(1,0.5))
plt.title('Embarked',color='red')
```

```
Text(0.5, 1.0, 'Embarked')
```



LabelEncoding

LabelEncoding Machine learning models work only with numerical values.so categorical columns are converted in to numerical values.

```
from sklearn.preprocessing import LabelEncoder
list=['Sex','Embarked']
dict1={}
for columns in list:
   dict1[columns]=LabelEncoder()
   df[columns]=dict1[columns].fit_transform(df[columns])
dict1
```

```
{'Sex': LabelEncoder(), 'Embarked': LabelEncoder()}
```

```
df.dtypes
```

```
Pclass        int64
Sex           int64
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Embarked      int64
Survived      int64
dtype: object
```

Take input variable and output variable

```
x=df.drop(columns='Survived')
y=df['Survived']
```

TRAIN TEST SPLIT

A train test split function is used to split data in to training set and testing set.Training set are used to train the model and test their accuracy on unseen data(testing data).

```
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)
x_train
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
351	1	1	29.699118	0	0	35.0000	2
125	3	1	12.000000	1	0	11.2417	0
578	3	0	29.699118	1	0	14.4583	0
423	3	0	28.000000	1	1	14.4000	2
119	3	0	2.000000	4	2	31.2750	2
...	...	...	...	...	...	...	...
837	3	1	29.699118	0	0	8.0500	2
193	2	1	3.000000	1	1	26.0000	2
630	1	1	80.000000	0	0	30.0000	2
560	3	1	29.699118	0	0	7.7500	1
685	2	1	25.000000	1	2	41.5792	0

622 rows × 7 columns

Next steps: Generate code with `x_train` View recommended plots

```
x_test
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
14	3	0	14.000000	0	0	7.8542	2
159	3	1	29.699118	8	2	69.5500	2
763	1	0	36.000000	1	2	120.0000	2
741	1	1	36.000000	1	0	78.8500	2
483	3	0	63.000000	0	0	9.5875	2
...	...	...	...	...	...	...	...
620	3	1	27.000000	1	0	14.4542	0
821	3	1	27.000000	0	0	8.6625	2
65	3	1	29.699118	1	1	15.2458	0
884	3	1	25.000000	0	0	7.0500	2
52	1	0	49.000000	1	0	76.7292	0

267 rows × 7 columns

Next steps: Generate code with `x_test` View recommended plots

```
y_train
```

```
351    0
125    1
578    0
423    0
119    0
      ..
837    0
193    1
630    1
560    0
```

```
685    0
Name: Survived, Length: 622, dtype: int64
```

```
y_test
```

```
14     0
159    0
763    1
741    0
483    1
      ..
620    0
821    1
65     1
884    0
52     1
Name: Survived, Length: 267, dtype: int64
```

SCALING USING STANDARDSCALER

Normalization in machine learning is the process of translating data into the range[0,1] or simply transforming data onto the unit sphere.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_scaled=sc.fit_transform(x_train)
x_train_scaled
```

```
array([[-1.59995219,  0.72423049,  0.00627074, ..., -0.45521828,
         0.07980737,  0.59263555],
       [ 0.82433591,  0.72423049, -1.32578257, ..., -0.45521828,
        -0.40736542, -1.92353724],
       [ 0.82433591, -1.38077589,  0.00627074, ..., -0.45521828,
        -0.34140784, -1.92353724],
       ...,
       [-1.59995219,  0.72423049,  3.79196551, ..., -0.45521828,
        -0.02271949,  0.59263555],
       [ 0.82433591,  0.72423049,  0.00627074, ..., -0.45521828,
        -0.47896403, -0.66545084],
       [-0.38780814,  0.72423049, -0.34738956, ...,  1.91420238,
         0.21471632, -1.92353724]])
```

```
x_test_scaled=sc.transform(x_test)
x_test_scaled
```

```
array([[ 0.82433591, -1.38077589, -1.17526057, ..., -0.45521828,
        -0.47682737,  0.59263555],
       [ 0.82433591,  0.72423049,  0.00627074, ...,  1.91420238,
         0.78826799,  0.59263555],
       [-1.59995219, -1.38077589,  0.48048146, ...,  1.91420238,
         1.82276404,  0.59263555],
       ...,
       [ 0.82433591,  0.72423049,  0.00627074, ...,  0.72949205,
        -0.32525985, -1.92353724],
       [ 0.82433591,  0.72423049, -0.34738956, ..., -0.45521828,
        -0.49331779,  0.59263555],
       [-1.59995219, -1.38077589,  1.45887447, ..., -0.45521828,
         0.93548016, -1.92353724]])
```

MODEL CREATION

K-Nearest Neighbors algorithm(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=21)
knn.fit(x_train_scaled,y_train)
y_pred_knn=knn.predict(x_test_scaled)
y_pred_knn
```

```
array([1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
```

```
           1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
           1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
           0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
           1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
           0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
           0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
           1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
           0, 0, 1])
```

```
np.array(y_test)
```

```
    array([0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
           1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
           1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
           0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1,
           0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1,
           1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
           1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
           1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
           1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
           1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
           1, 0, 1])
```

NAIVE BAYES

```
from sklearn.naive_bayes import BernoulliNB
naive=BernoulliNB()
naive.fit(x_train_scaled,y_train)
y_pred_naive=naive.predict(x_test_scaled)
y_pred_naive
```

```
    array([0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
           0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
           1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
           1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
           0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
           1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
           0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
           0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
           1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
           0, 0, 1])
```

SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
sv=SVC()
sv.fit(x_train_scaled,y_train)
y_pred_sv=sv.predict(x_test_scaled)
y_pred_sv
```

```
    array([1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
           0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
           1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
           0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
           1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
           0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
           0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
           1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
           1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
           0, 0, 1])
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier(max_depth=20,criterion='gini')
tree.fit(x_train_scaled,y_train)
y_pred_tree=tree.predict(x_test_scaled)
y_pred_tree
```

```
array([1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1])
```

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100,max_depth=100,min_samples_split=5,criterion='gini',bootstrap=False)
rf.fit(x_train_scaled,y_train)
y_pred_rf=rf.predict(x_test_scaled)
y_pred_rf
```

```
array([1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 1])
```

```
import xgboost
xb=xgboost.XGBClassifier()
xb.fit(x_train_scaled,y_train)
y_pred_xb=xb.predict(x_test_scaled)
y_pred_xb
```

```
array([1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 1])
```

PERFORMANCE EVALUATION

```
#knn
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,ConfusionMatrixDisplay
print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.77      0.90      0.83       157
           1       0.82      0.61      0.70       110

    accuracy                           0.78       267
   macro avg       0.79      0.76      0.76       267
weighted avg       0.79      0.78      0.78       267
```

```
print(accuracy_score(y_test,y_pred_knn))
```

```
0.7827715355805244
```

```
#naive_bayes
print(classification_report(y_test,y_pred_naive))
```

```
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       157
           1       0.73      0.65      0.69       110

    accuracy                           0.76       267
   macro avg       0.75      0.74      0.74       267
weighted avg       0.75      0.76      0.75       267
```

```
print(accuracy_score(y_test,y_pred_naive))
```

```
0.7565543071161048
```

```
#svm
print(classification_report(y_test,y_pred_sv))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       157
           1       0.80      0.67      0.73       110

    accuracy                           0.79       267
   macro avg       0.79      0.78      0.78       267
weighted avg       0.79      0.79      0.79       267
```

```
print(accuracy_score(y_test,y_pred_sv))
```

```
0.7940074906367042
```

```
#decision tree
print(classification_report(y_test,y_pred_tree))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       157
           1       0.74      0.74      0.74       110

    accuracy                           0.78       267
   macro avg       0.78      0.78      0.78       267
weighted avg       0.78      0.78      0.78       267
```

```
print(accuracy_score(y_test,y_pred_tree))
```

```
0.7827715355805244
```

```
#random forest
print(classification_report(y_test,y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       157
           1       0.79      0.72      0.75       110

    accuracy                           0.81       267
   macro avg       0.80      0.79      0.80       267
weighted avg       0.80      0.81      0.80       267
```

```
print(accuracy_score(y_test,y_pred_rf))
```

```
0.8052434456928839
```

```
#XGBclassifier
print(classification_report(y_test,y_pred_xb))
```

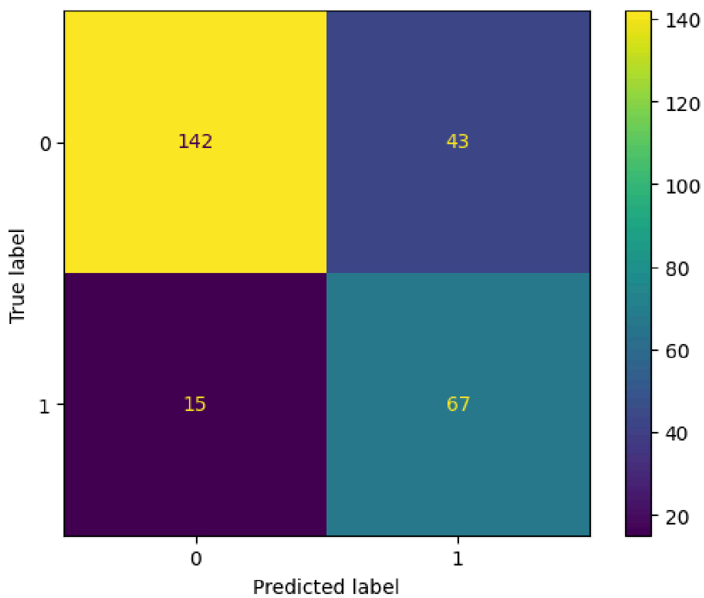
```
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       157
           1       0.76      0.70      0.73       110

    accuracy                           0.79       267
   macro avg       0.78      0.77      0.78       267
weighted avg       0.79      0.79      0.78       267
```

```
print(accuracy_score(y_test,y_pred_xb))
```

```
0.7865168539325843
```

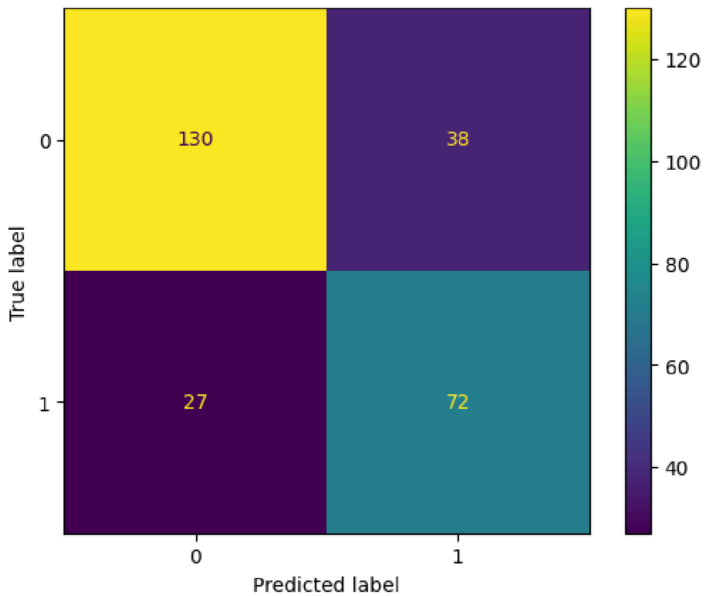
```
label=[0,1]
matx_knn=confusion_matrix(y_pred_knn,y_test)
print(matx_knn)
cmd=ConfusionMatrixDisplay(matx_knn,display_labels=label)
cmd.plot()
```

```
[[142  43]
 [ 15  67]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e831de173d0>
```



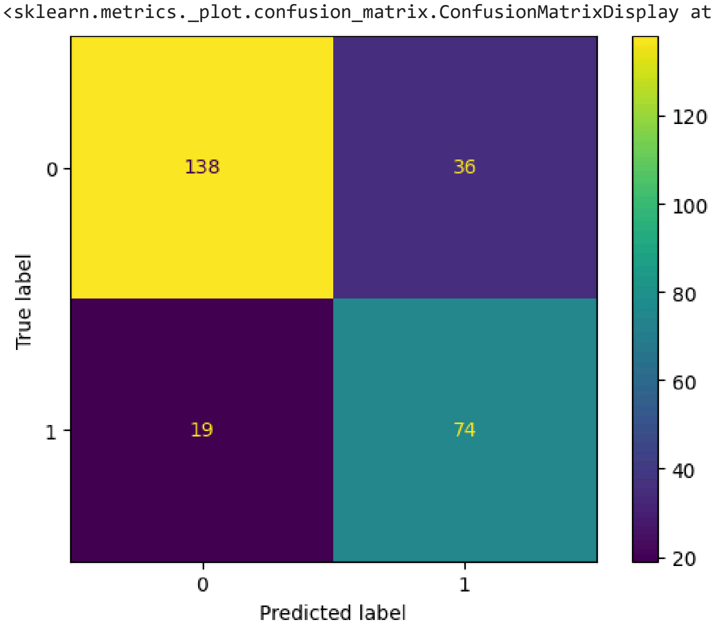

```
label=[0,1]
matx_naive=confusion_matrix(y_pred_naive,y_test)
print(matx_naive)
cmd=ConfusionMatrixDisplay(matx_naive,display_labels=label)
cmd.plot()
```

```
[[130  38]
 [ 27  72]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e831dc770d0>
```



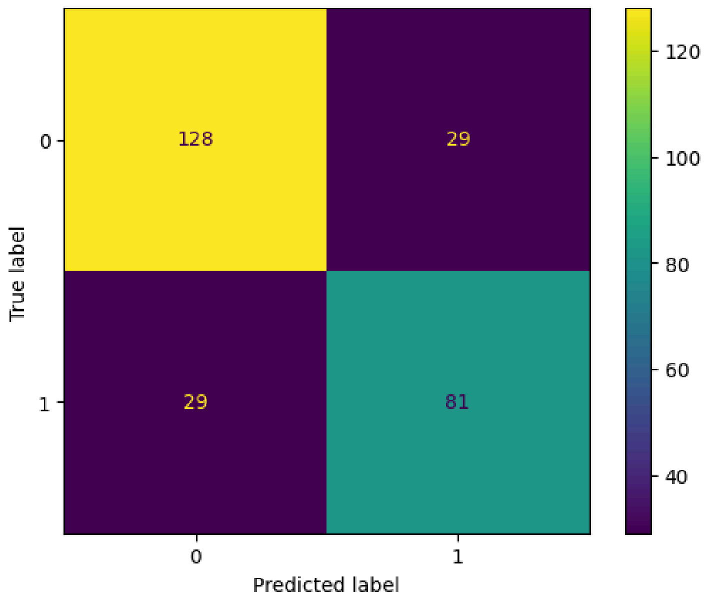

```
label=[0,1]
matx_sv=confusion_matrix(y_pred_sv,y_test)
print(matx_sv)
cmd=ConfusionMatrixDisplay(matx_sv,display_labels=label)
cmd.plot()
```

```
[[138  36]
 [ 19  74]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e8376a21060>
```



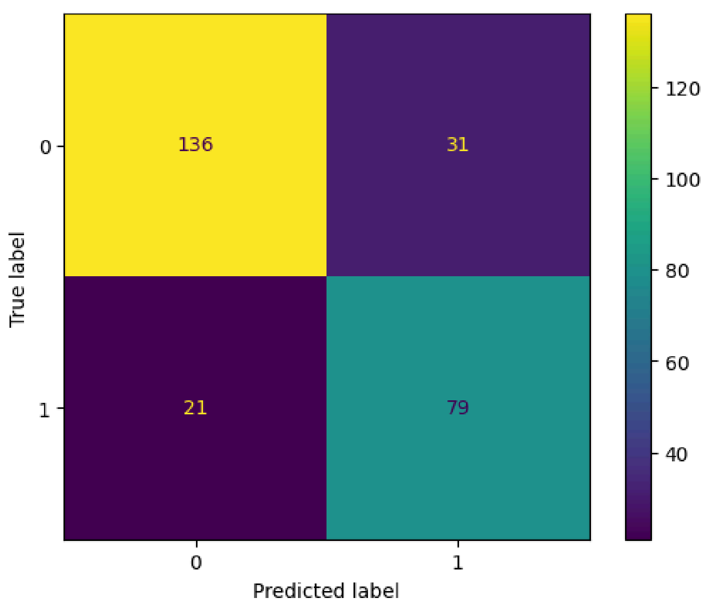

```
label=[0,1]
matx_tree=confusion_matrix(y_pred_tree,y_test)
print(matx_tree)
cmd=ConfusionMatrixDisplay(matx_tree,display_labels=label)
cmd.plot()
```

```
[[128  29]
 [ 29  81]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e83376b2230>
```




```
label=[0,1]
matx_rf=confusion_matrix(y_pred_rf,y_test)
print(matx_rf)
cmd=ConfusionMatrixDisplay(matx_rf,display_labels=label)
cmd.plot()
```

```
[[136  31]
 [ 21  79]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e831e0c8100>
```




```
label=[0,1]
matx_xb=confusion_matrix(y_pred_xb,y_test)
print(matx_xb)
cmd=ConfusionMatrixDisplay(matx_xb,display_labels=label)
cmd.plot()
```

```
[[133  33]
 [ 24  77]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e8337caf520>
```