```
#import packages
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import seaborn as sns
from sklearn.preprocessing import LabelEncoder,StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from imblearn.over_sampling import SMOTE,RandomOverSampler,ADASYN
from imblearn.under_sampling import RandomUnderSampler,NearMiss
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report,ConfusionMatrixDisplay
warnings.filterwarnings('ignore')
```

```
#load dataset
df=pd.read_csv("/content/drive/MyDrive/MLDataset/IT_customer_churn.csv")
df.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
0	Female	0	Yes	No	1	No	
1	Male	0	No	No	34	Yes	
2	Male	0	No	No	2	Yes	
3	Male	0	No	No	45	No	
4	Female	0	No	No	2	Yes	

```
#analysis of data
df.shape
```

```
(7043, 20)
```

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

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
```

```
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

```
df.dtypes
```

```
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

```
df["TotalCharges"].values
```

```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

```
#converting to numeric and then check missing values
df.TotalCharges = pd.to_numeric(df.TotalCharges, errors='coerce')
df.isnull().sum()
```

```
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

```
df.dtypes
```

```
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges       float64
Churn               object
dtype: object
```

```
df.loc[df['TotalCharges'].isnull() == True]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
488	Female	0	Yes	Yes	0	No	No ph ser
753	Male	0	No	Yes	0	Yes	
936	Female	0	Yes	Yes	0	Yes	
1082	Male	0	Yes	Yes	0	Yes	
1340	Female	0	Yes	Yes	0	No	No ph ser
3331	Male	0	Yes	Yes	0	Yes	
3826	Male	0	Yes	Yes	0	Yes	
4380	Female	0	Yes	Yes	0	Yes	
5218	Male	0	Yes	Yes	0	Yes	
6670	Female	0	Yes	Yes	0	Yes	
6754	Male	0	No	Yes	0	Yes	

```
#dropping the row contain Null value
df.dropna(how = 'any', inplace = True)
df.shape
```

```
(7032, 20)
```

```
#checking unique values of each column
def printunique(df):
      for i in df:
```

```
            if df[i].dtypes=='object':
                print(f'{i}: {df[i].unique()}')
printunique(df)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No phone service' 'No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes' 'No internet service']
OnlineBackup: ['Yes' 'No' 'No internet service']
DeviceProtection: ['No' 'Yes' 'No internet service']
TechSupport: ['No' 'Yes' 'No internet service']
StreamingTV: ['No' 'Yes' 'No internet service']
StreamingMovies: ['No' 'Yes' 'No internet service']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

```
df.replace('No internet service','No',inplace=True)
df.replace('No phone service','No',inplace=True)
```

```
printunique(df)
```

```
gender: ['Female' 'Male']
Partner: ['Yes' 'No']
Dependents: ['No' 'Yes']
PhoneService: ['No' 'Yes']
MultipleLines: ['No' 'Yes']
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: ['No' 'Yes']
OnlineBackup: ['Yes' 'No']
DeviceProtection: ['No' 'Yes']
TechSupport: ['No' 'Yes']
StreamingTV: ['No' 'Yes']
StreamingMovies: ['No' 'Yes']
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: ['Yes' 'No']
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn: ['No' 'Yes']
```

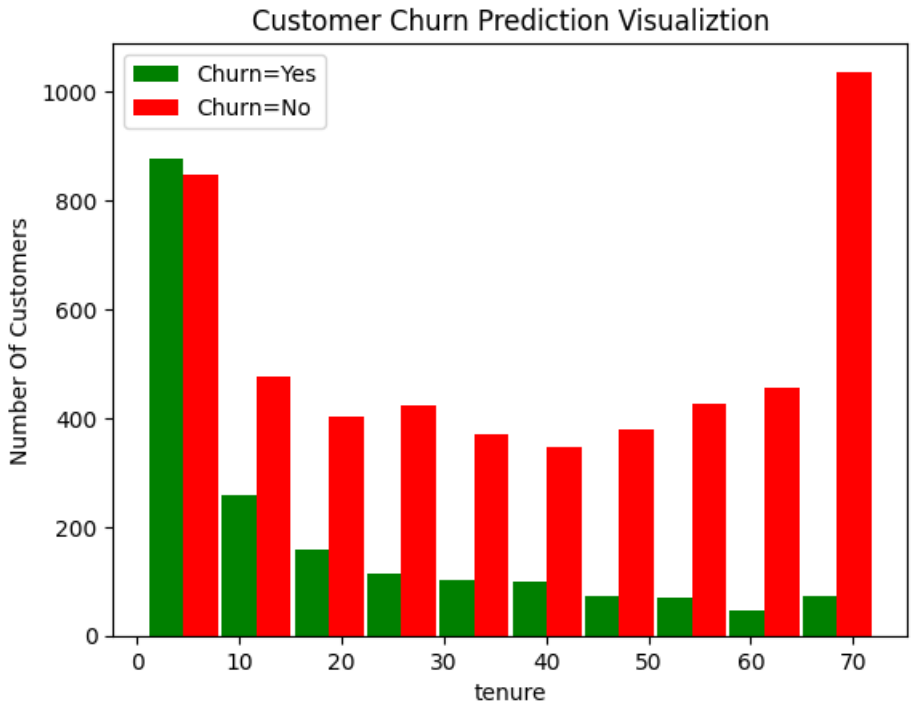
```
sns.countplot(x="Churn",data=df)
```

```
<Axes: xlabel='Churn', ylabel='count'>
```


```
100*df['Churn'].value_counts()/len(df['Churn'])

No     73.421502
Yes    26.578498
Name: Churn, dtype: float64
```

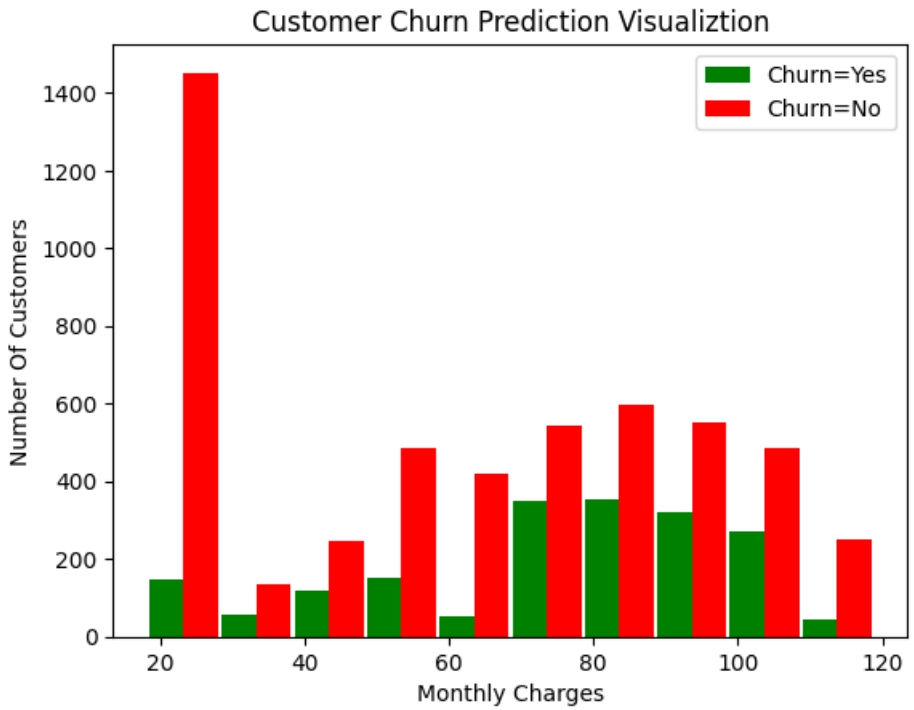
```
 #Data is highly imbalanced, ratio = 73:27
df['Churn'].value_counts()

No     5163
Yes    1869
Name: Churn, dtype: int64
```

```
t_c_n = df[df.Churn=='No'].tenure
t_c_y = df[df.Churn=='Yes'].tenure
plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")
plt.hist([t_c_y, t_c_n], rwidth=0.95, color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
plt.show()
```

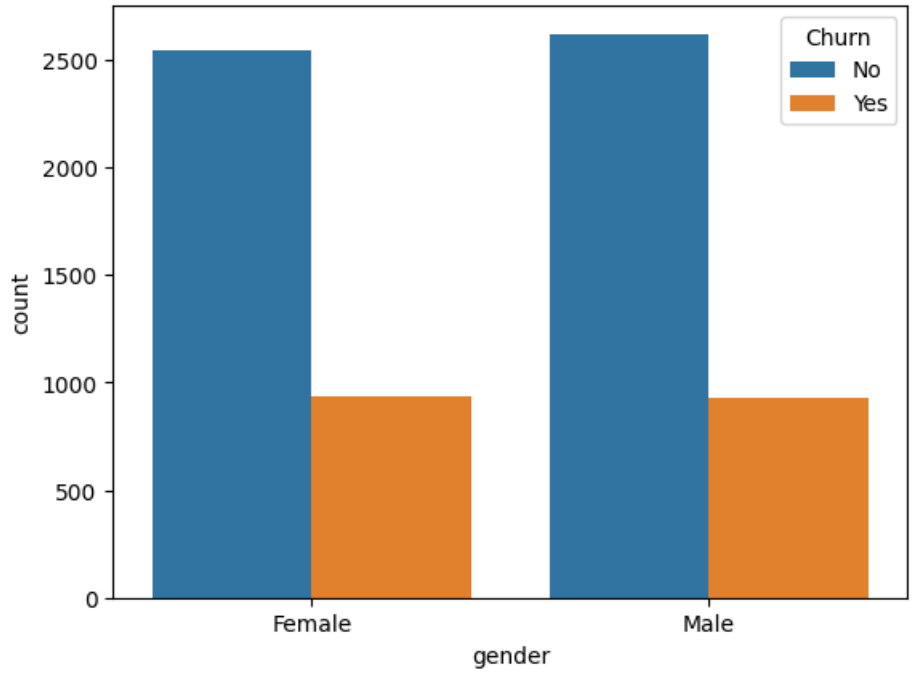



```
m_c_n = df[df.Churn=='No'].MonthlyCharges
m_c_y = df[df.Churn=='Yes'].MonthlyCharges

plt.xlabel("Monthly Charges")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualiztion")
```

```
plt.hist([m_c_y, m_c_n], rwidth=0.95, color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()
plt.show()
```




```
for i, predictor in enumerate(df.drop(columns=['Churn', 'TotalCharges', 'MonthlyCharges'])):
    plt.figure(i)
    sns.countplot(data=df, x=predictor, hue='Churn')
```

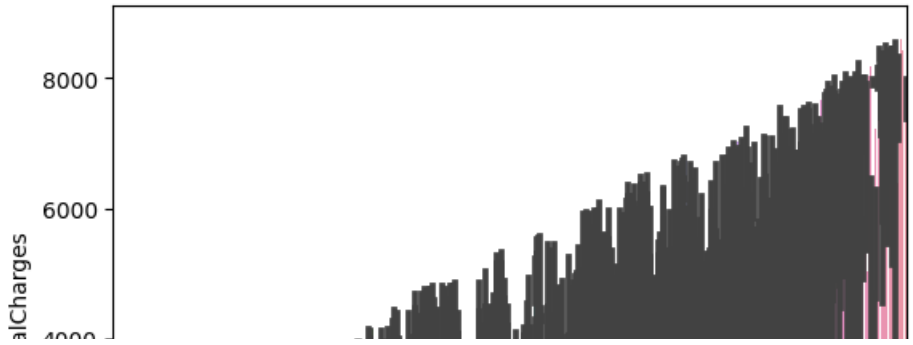



```
le=LabelEncoder()
lst=["gender","Partner","Dependents","PhoneService","MultipleLines","InternetService","OnlineSecurity","OnlineBackup"
for i in lst:
  df[i]=le.fit_transform(df[i])
df.dtypes
```

```
gender                int64
SeniorCitizen         int64
Partner               int64
Dependents            int64
tenure                int64
PhoneService          int64
MultipleLines         int64
InternetService       int64
OnlineSecurity        int64
OnlineBackup          int64
DeviceProtection      int64
TechSupport           int64
StreamingTV           int64
StreamingMovies       int64
Contract              int64
PaperlessBilling      int64
PaymentMethod         int64
MonthlyCharges      float64
TotalCharges        float64
Churn                 int64
dtype: object
```

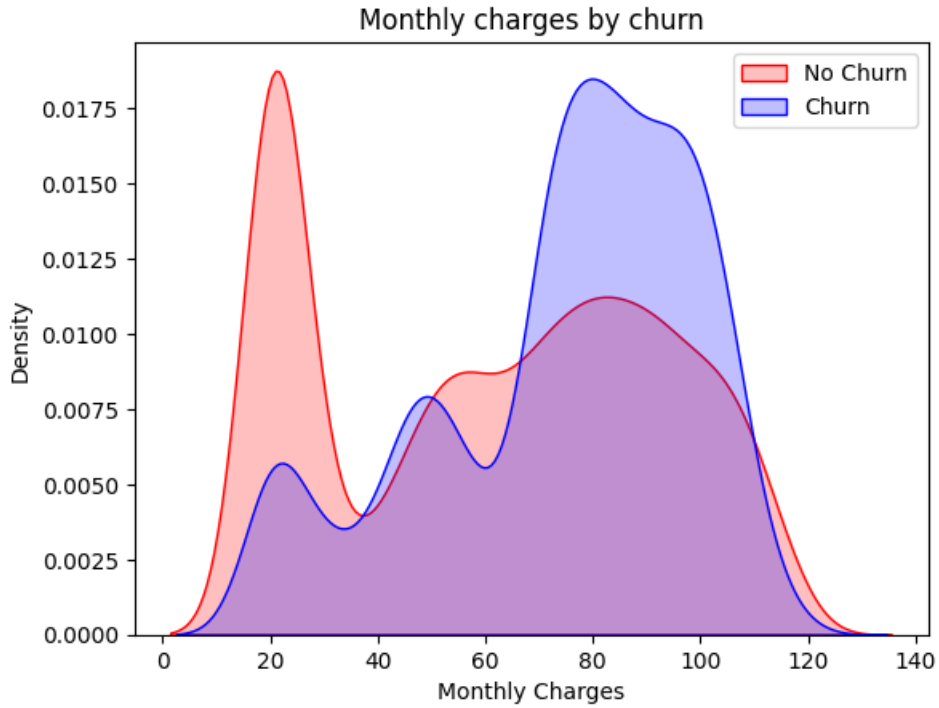

```
sns.barplot(data=df, x='MonthlyCharges', y='TotalCharges')#Total Charges increase as Monthly Charges increase .
```

```
<Axes: xlabel='MonthlyCharges', ylabel='TotalCharges'>
```




```
mon = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 0) ],
                color="Red", shade = True)
mon = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 1) ],
                ax =mon, color="Blue", shade= True)
mon.legend(["No Churn","Churn"],loc='upper right')
mon.set_ylabel('Density')
mon.set_xlabel('Monthly Charges')
mon.set_title('Monthly charges by churn')
```

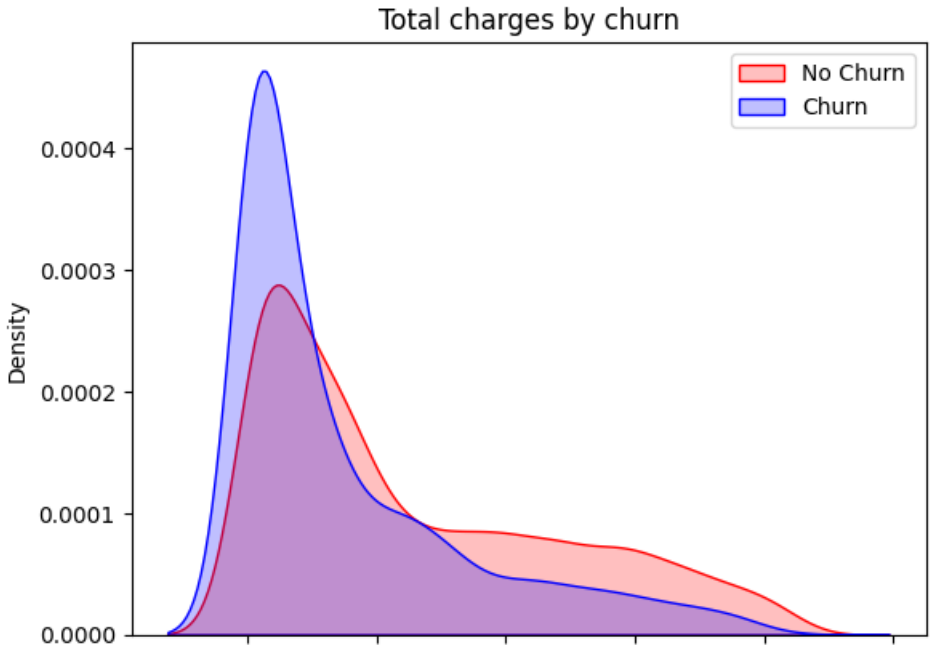
```
Text(0.5, 1.0, 'Monthly charges by churn')
```




```
#Churn is high when Monthly Charges are high
```

```
Tot = sns.kdeplot(df.TotalCharges[(df["Churn"] == 0) ],
                color="Red", shade = True)
Tot = sns.kdeplot(df.TotalCharges[(df["Churn"] == 1) ],
                ax =Tot, color="Blue", shade= True)
Tot.legend(["No Churn","Churn"],loc='upper right')
Tot.set_ylabel('Density')
Tot.set_xlabel('Total Charges')
Tot.set_title('Total charges by churn')
```

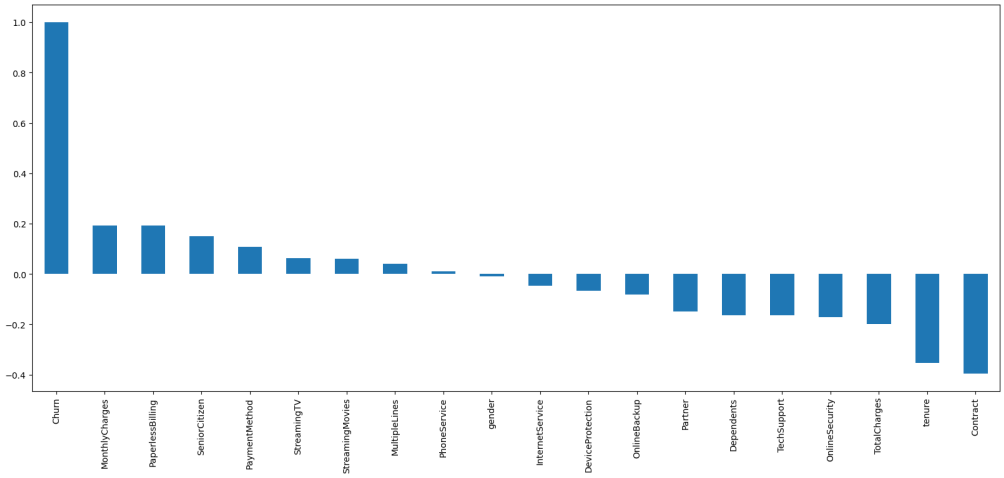
```
Text(0.5, 1.0, 'Total charges by churn')
```




```
#as higher Churn at lower Total Charges
```

```
plt.figure(figsize=(20,8))
df.corr()['Churn'].sort_values(ascending = False).plot(kind='bar')
```

```
<Axes: >
```



```
#HIGH Churn seen in case of Monthly charges,paperless billing etc..
#LOW Churn is seens in case of contract,tenure,totalcharges etc..
#Factors like gender, Availability of PhoneService and # of multiple lines have alomost NO impact on Churn
```

Double-click (or enter) to edit

```
X=df.iloc[:,:-1]
X
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	0	0	1	0	1	0	
1	1	0	0	0	34	1	
2	1	0	0	0	2	1	
3	1	0	0	0	45	0	
4	0	0	0	0	2	1	
...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	
7039	0	0	1	1	72	1	
7040	0	0	1	1	11	0	
7041	1	1	1	0	4	1	
7042	1	0	0	0	66	1	

7032 rows × 19 columns

```
y=df.iloc[:,-1]
y
```

```
0       0
1       0
2       1
3       0
4       1
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn, Length: 7032, dtype: int64
```

```
#ms=MinMaxScaler()
#X_sc=ms.fit_transform(X)
#X_sc
```

DeviceProtection

```
st=StandardScaler()
X_st=st.fit_transform(X)
```

```
sm=SMOTE(random_state=10)
X_sm,y_sm=sm.fit_resample(X_st,y)
```

```
y_sm.value_counts()
```

```
0    5163
1    5163
Name: Churn, dtype: int64
```

```
X_train,X_test,y_train,y_test=train_test_split(X_sm,y_sm,random_state=1,test_size=0.02)
```

```
#pca=PCA(n_components=1,random_state=1)
#X_train=pca.fit_transform(X_train)
#X_test=pca.transform(X_test)
#X_train
```

```
knn=KNeighborsClassifier()
params={'n_neighbors':[3,5,7,9],'weights':['uniform','distance'],'algorithm':['auto','ball_tree','kd_tree','brute']}
clf=GridSearchCV(knn,params,cv=10,scoring='accuracy')
clf.fit(X_train,y_train)
print(clf.best_params_)
```

```
{'algorithm': 'ball_tree', 'n_neighbors': 3, 'weights': 'distance'}
```

```
knn=KNeighborsClassifier(algorithm ='auto',n_neighbors = 3,weights ='distance')
svm=SVC(gamma="auto",kernel="rbf")
nb=GaussianNB()
rf=RandomForestClassifier(criterion= 'entropy', n_estimators=100,max_depth=8,max_features=6,min_samples_leaf=7)
gb=GradientBoostingClassifier(n_estimators=100)
dc=DecisionTreeClassifier(criterion = "entropy",max_depth=6, min_samples_leaf=8,random_state=5)
lg=LogisticRegression()
```

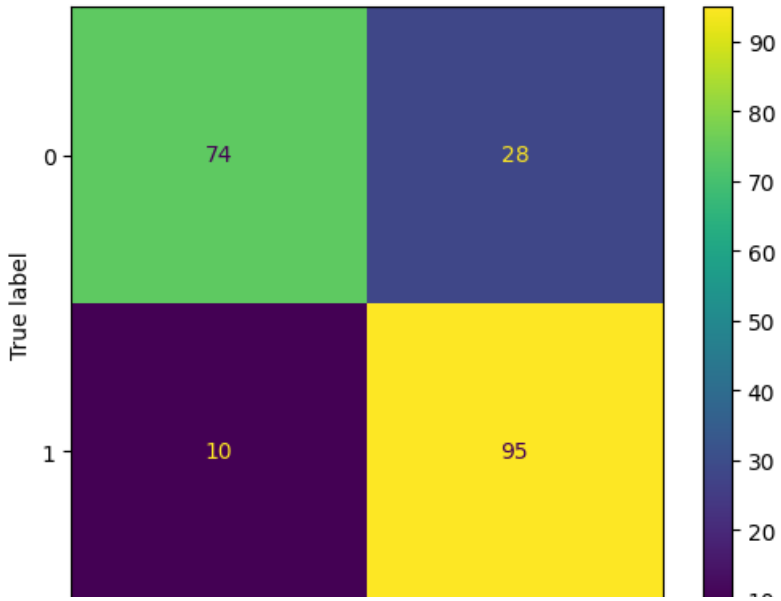
```
lst2=[knn,svm,nb,rf,gb,dc,lg]
for i in lst2:
  print(i)
  i.fit(X_train,y_train)
  y_pred=i.predict(X_test)
  print(classification_report(y_test,y_pred))
```

```
KNeighborsClassifier(n_neighbors=3, weights='distance')
              precision    recall  f1-score   support

           0       0.88      0.73      0.80       102
           1       0.77      0.90      0.83       105

    accuracy                           0.82       207
   macro avg       0.83      0.82      0.81       207
weighted avg       0.83      0.82      0.81       207

SVC(gamma='auto')
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       102
           1       0.81      0.86      0.83       105

    accuracy                           0.83       207
   macro avg       0.83      0.83      0.83       207
weighted avg       0.83      0.83      0.83       207

GaussianNB()
              precision    recall  f1-score   support

           0       0.80      0.65      0.72       102
           1       0.71      0.85      0.77       105
```

```
           accuracy                           0.75       207
          macro avg       0.76      0.75      0.75       207
       weighted avg       0.76      0.75      0.75       207

       RandomForestClassifier(criterion='entropy', max_depth=8, max_features=6,
                              min_samples_leaf=7)
                     precision    recall  f1-score   support

                  0       0.88      0.77      0.82       102
                  1       0.80      0.90      0.85       105

           accuracy                           0.84       207
          macro avg       0.84      0.83      0.83       207
       weighted avg       0.84      0.84      0.84       207

       GradientBoostingClassifier()
                     precision    recall  f1-score   support

                  0       0.89      0.83      0.86       102
                  1       0.85      0.90      0.87       105

           accuracy                           0.86       207
          macro avg       0.87      0.86      0.86       207
       weighted avg       0.87      0.86      0.86       207

       DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_leaf=8,
                              random_state=5)
                     precision    recall  f1-score   support

                  0       0.84      0.83      0.84       102
                  1       0.84      0.85      0.84       105
```

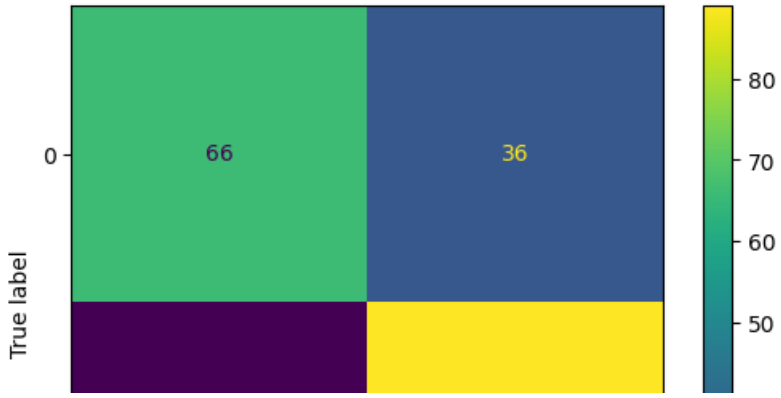
```
lst2=[knn,svm]
for i in lst2:
  print(i)
  i.fit(X_train,y_train)
  y_pred=i.predict(X_test)

  print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
KNeighborsClassifier(n_neighbors=3, weights='distance')
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232a6
SVC(gamma='auto')
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f236a1
```




```
lst2=[nb,rf]
for i in lst2:
  print(i)
  i.fit(X_train,y_train)
  y_pred=i.predict(X_test)
  print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
GaussianNB()
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232a4
RandomForestClassifier(criterion='entropy', max_depth=8, max_features=6,
                       min_samples_leaf=7)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232a6
```




```
lst2=[gb,dc,lg]
for i in lst2:
  print(i)
  i.fit(X_train,y_train)
  y_pred=i.predict(X_test)
  print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
GradientBoostingClassifier()
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232
DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_leaf=8,
                       random_state=5)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232
LogisticRegression()
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f232
```

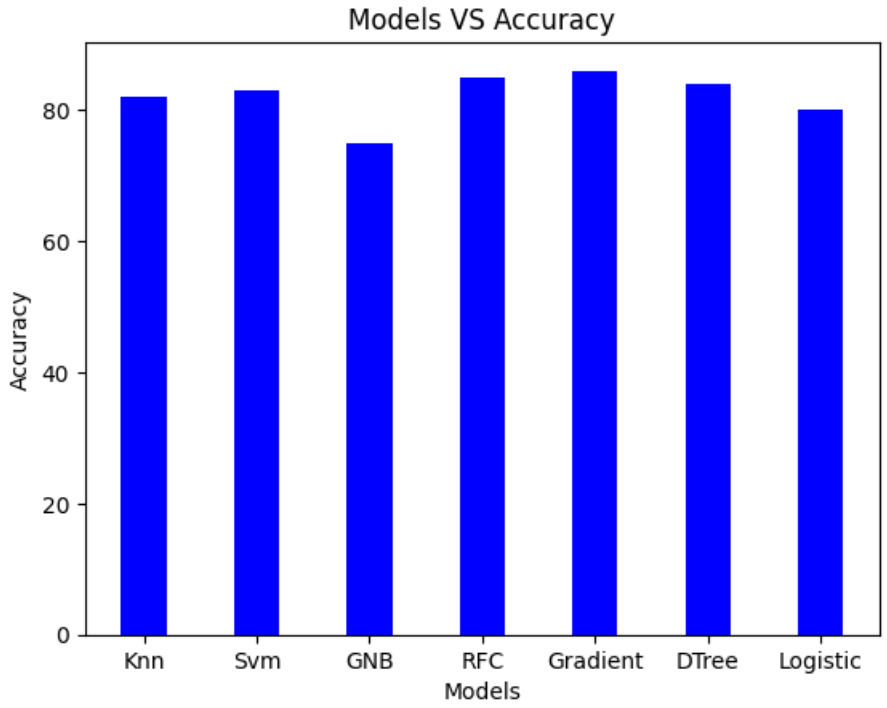

```
d={'Model':['K-Nearest Neighbors','Support Vector Classifier','GaussianNB','Random Forest Classifier','GradientBoosti
df_new=pd.DataFrame(d,index=[1,2,3,4,5,6,7],columns=['Model','Prediction_Accuracy'])
df_new.style.highlight_max(subset=['Prediction_Accuracy'],color='green')
```

	Model	Prediction_Accuracy
1	K-Nearest Neighbors	0.820000
2	Support Vector Classifier	0.830000
3	GaussianNB	0.750000
4	Random Forest Classifier	0.850000
5	GradientBoostingClassifier	0.860000
6	DecisionTreeClassifier	0.840000
7	logisticRegression	0.800000

```
df_new.style.highlight_min(subset=['Prediction_Accuracy'],color='red')
```

	Model	Prediction_Accuracy
1	K-Nearest Neighbors	0.820000
2	Support Vector Classifier	0.830000
3	GaussianNB	0.750000
4	Random Forest Classifier	0.850000
5	GradientBoostingClassifier	0.860000
6	DecisionTreeClassifier	0.840000
7	logisticRegression	0.800000


```
y=[82,83,75,85,86,84,80]
x=['Knn','Svm','GNB','RFC','Gradient','DTree','Logistic']
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.title("Models VS Accuracy")
plt.bar(x,y,color="b",width=0.4)
plt.show()
```


Models VS Accuracy
Accuracy
0
20
40
60
80
Knn
Svm
GNB
RFC
Gradient
DTree
Logistic
Models