In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv("C:\\Users\\DELL\\Downloads\\Train_psolI3n.csv.zip")
df
```

Out[2]:

	Email_ID	Email_Type	Subject_Hotness_Score	Email_Source_Type	Customer_Location
0	EMA00081000034500	1	2.2	2	E
1	EMA00081000045360	2	2.1	1	NaN
2	EMA00081000066290	2	0.1	1	E
3	EMA00081000076560	1	3.0	2	E
4	EMA00081000109720	1	0.0	2	C
...	...	...	...	...	..
68348	EMA00089995974500	2	0.4	1	F
68349	EMA00089998225300	1	1.3	1	C
68350	EMA00089998436500	1	2.2	2	NaN
68351	EMA00089999168800	1	0.4	1	E
68352	EMA00089999316900	1	1.5	1	G

68353 rows × 12 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 68353 entries, 0 to 68352
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Email_ID                   68353 non-null  object
 1   Email_Type                 68353 non-null  int64
 2   Subject_Hotness_Score      68353 non-null  float64
 3   Email_Source_Type          68353 non-null  int64
 4   Customer_Location          56758 non-null  object
 5   Email_Campaign_Type        68353 non-null  int64
 6   Total_Past_Communications  61528 non-null  float64
 7   Time_Email_sent_Category   68353 non-null  int64
 8   Word_Count                 68353 non-null  int64
 9   Total_Links                66152 non-null  float64
 10  Total_Images               66676 non-null  float64
 11  Email_Status               68353 non-null  int64
dtypes: float64(4), int64(6), object(2)
memory usage: 6.3+ MB
```

In [4]:
```
df.describe()
```

Out[4]:

	Email_Type	Subject_Hotness_Score	Email_Source_Type	Email_Campaign_Type	Total_Past_Com
count	68353.000000	68353.000000	68353.000000	68353.000000	6
mean	1.285094	1.095481	1.456513	2.272234	
std	0.451462	0.997578	0.498109	0.468680	
min	1.000000	0.000000	1.000000	1.000000	
25%	1.000000	0.200000	1.000000	2.000000	
50%	1.000000	0.800000	1.000000	2.000000	
75%	2.000000	1.800000	2.000000	3.000000	
max	2.000000	5.000000	2.000000	3.000000	

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Email_ID                      0
Email_Type                    0
Subject_Hotness_Score         0
Email_Source_Type             0
Customer_Location         11595
Email_Campaign_Type           0
Total_Past_Communications  6825
Time_Email_sent_Category      0
Word_Count                    0
Total_Links                2201
Total_Images               1677
Email_Status                  0
dtype: int64
```

In [6]:
```
df.shape
```

Out[6]:
```
(68353, 12)
```

In [7]:
```
df=df.dropna()
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Email_ID                  0
Email_Type                0
Subject_Hotness_Score     0
Email_Source_Type         0
Customer_Location         0
Email_Campaign_Type       0
Total_Past_Communications 0
Time_Email_sent_Category  0
Word_Count                0
Total_Links               0
Total_Images              0
Email_Status              0
dtype: int64
```

In [9]:
```
df.shape
```

Out[9]:
```
(48291, 12)
```

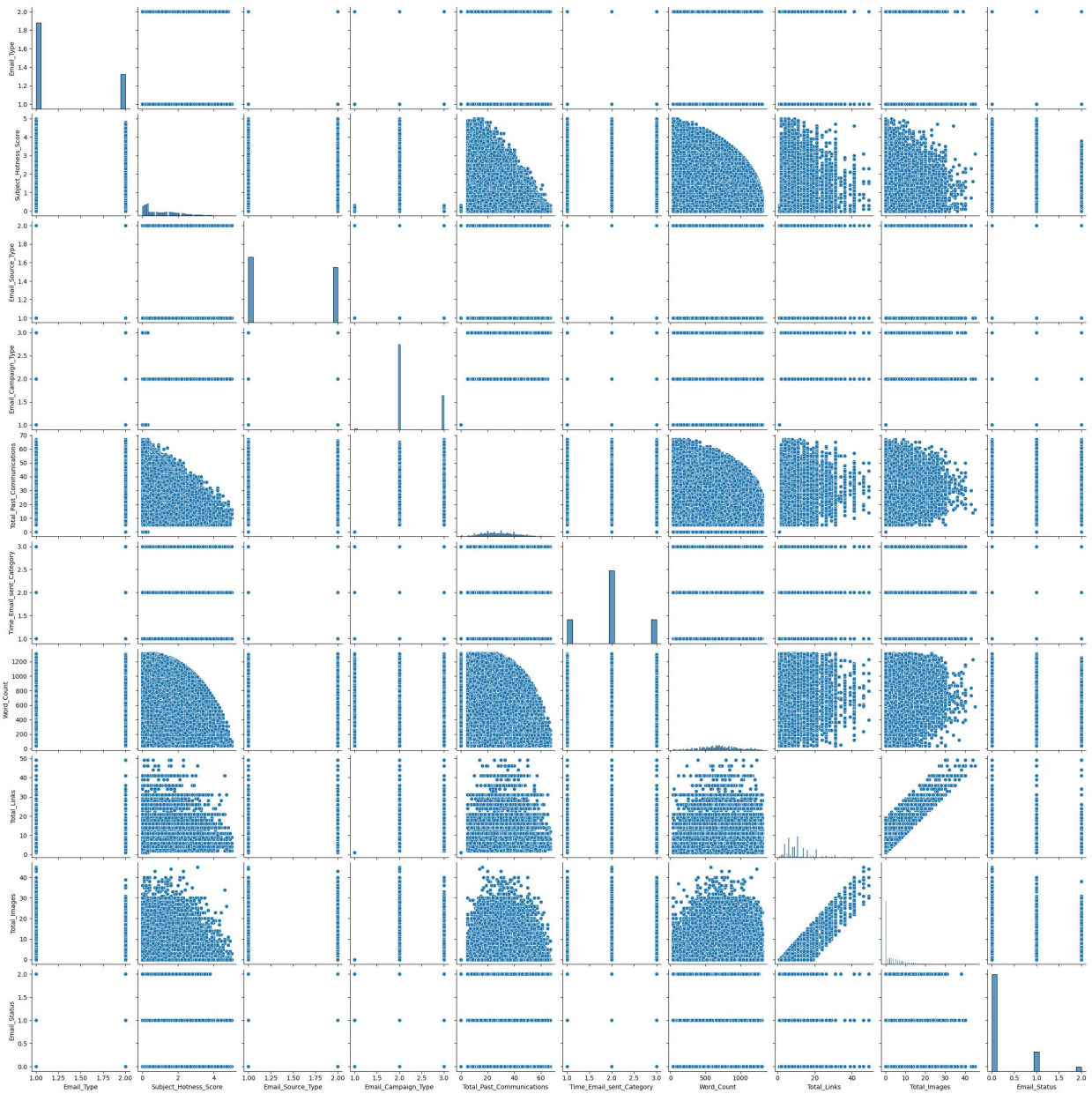
In [10]:
```
#No nominal nor ordinal variables so no encoding
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Email_ID', 'Email_Type', 'Subject_Hotness_Score', 'Email_Source_Type',
       'Customer_Location', 'Email_Campaign_Type', 'Total_Past_Communications',
       'Time_Email_sent_Category', 'Word_Count', 'Total_Links', 'Total_Images',
       'Email_Status'],
      dtype='object')
```

In [12]:
```
df=df.drop(columns=['Email_ID'])
```

In [13]:
```
sns.pairplot(df)
plt.show()
```



In [14]:
```
#Data wrangling
```

In [15]:
```
df['Customer_Location'].value_counts()
```

Out[15]:
```
G    19756
E     8660
D     6257
C     4893
F     3771
B     3726
A     1228
Name: Customer_Location, dtype: int64
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Customer_Location']=le.fit_transform(df['Customer_Location'])
```

In [17]:
```
df.shape
```

Out[17]:
```
(48291, 11)
```

In [18]:
```
df
```

Out[18]:

	Email_Type	Subject_Hotness_Score	Email_Source_Type	Customer_Location	Email_Campaign_Typ
0	1	2.2	2	4	
2	2	0.1	1	1	
3	1	3.0	2	4	
4	1	0.0	2	2	
6	1	3.2	1	4	
...	...	...	...	...	
68346	1	1.2	2	4	
68348	2	0.4	1	5	
68349	1	1.3	1	2	
68351	1	0.4	1	4	
68352	1	1.5	1	6	

48291 rows × 11 columns

In [19]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit_transform(df)
```

Out[19]:
```
array([[-0.63202467,  1.10511906,  1.08781195, ..., -0.38184034,
        -0.6342954 , -0.46597325],
       [ 1.58221672, -0.99909607, -0.91927653, ..., -0.85143988,
        -0.6342954 ,  1.54370026],
       [-0.63202467,  1.90672482,  1.08781195, ...,  0.87042509,
        -0.6342954 , -0.46597325],
       ...,
       [-0.63202467,  0.20331258, -0.91927653, ...,  3.21842278,
         2.22789084, -0.46597325],
       [-0.63202467, -0.69849391, -0.91927653, ...,  0.55735873,
         0.08125116, -0.46597325],
       [-0.63202467,  0.40371402, -0.91927653, ..., -0.6949067 ,
        -0.6342954 , -0.46597325]])
```

Train test split

In [20]:
```
x=df.drop('Email_Status',axis=1)
y=df['Email_Status']
```

In [21]:
```
x.shape
```

Out[21]:
```
(48291, 10)
```

In [22]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
x=scaler.fit_transform(df)
```

In [23]:
```
x
```

Out[23]:
```
array([[0.        , 0.44      , 1.        , ..., 0.14583333, 0.        ,
        0.        ],
       [1.        , 0.02      , 0.        , ..., 0.08333333, 0.        ,
        0.5       ],
       [0.        , 0.6       , 1.        , ..., 0.3125    , 0.        ,
        0.        ],
       ...,
       [0.        , 0.26      , 0.        , ..., 0.625     , 0.35555556,
        0.        ],
       [0.        , 0.08      , 0.        , ..., 0.27083333, 0.08888889,
        0.        ],
       [0.        , 0.3       , 0.        , ..., 0.10416667, 0.        ,
        0.        ]])
```

In [ ]:
```
```

In [24]:
```
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=32)
```

In [25]:
```
#Modelling
```

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

In [27]:
```
log_model=LogisticRegression()
dt_model=DecisionTreeClassifier()
rf_model=RandomForestClassifier()
gb_model=GradientBoostingClassifier()
xg_model=XGBClassifier()
```

In [28]:
```
log_model.fit(x_train,y_train)#http://localhost:8888/notebooks/regression%20project.ip

dt_model.fit(x_train,y_train)
rf_model.fit(x_train,y_train)
gb_model.fit(x_train,y_train)
xg_model.fit(x_train,y_train)
```

Out[28]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...)
```

In [29]:
```
y_pred=log_model.predict(x_test)
y_pred5=dt_model.predict(x_test)
y_pred6=rf_model.predict(x_test)
y_pred7=gb_model.predict(x_test)
y_pred8=xg_model.predict(x_test)
```

In [30]:
```
from sklearn.metrics import accuracy_score
```

In [ ]:
```

```

In [31]:
```
print(accuracy_score(y_test,y_pred))
print(accuracy_score(y_test,y_pred5))
print(accuracy_score(y_test,y_pred6))
print(accuracy_score(y_test,y_pred7))
print(accuracy_score(y_test,y_pred8))
```

```
1.0
1.0
1.0
1.0
1.0
```

In [ ]:
```

```

In [ ]:
```

```

In [32]:
```
from sklearn.model_selection import cross_val_score
scores=cross_val_score(log_model,x,y,cv=5)
scores.mean()
```

Out[32]:
```
1.0
```

In [ ]:

In [ ]:

In [ ]:

In [33]:
```
scores=cross_val_score(dt_model,x,y,cv=5)
scores.mean()
```

Out[33]:
```
1.0
```

In [34]:
```
scores=cross_val_score(rf_model,x,y,cv=5)
scores.mean()
```

Out[34]:
```
1.0
```

In [35]:
```
scores=cross_val_score(gb_model,x,y,cv=5)
scores.mean()
```

Out[35]:
```
1.0
```

In [36]:
```
scores=cross_val_score(xg_model,x,y,cv=5)
scores.mean()
```

Out[36]:
```
1.0
```

In [37]:
```
#Hyperparameter tuning for dt,rf,gb,xg
```

In [38]:
```
from sklearn.model_selection import GridSearchCV
estimator=DecisionTreeClassifier(random_state=32)
param_grid={'criterion':['gini','entropy'],
              'max_depth':[1,2,3,4]}
grid=GridSearchCV(estimator,param_grid,scoring='accuracy',cv=5)
grid.fit(x_train,y_train)
grid.best_params_
```

Out[38]:
```
{'criterion': 'gini', 'max_depth': 2}
```

In [39]:
```
dt_bhp=DecisionTreeClassifier(criterion='gini',max_depth=2,random_state=32)
dt_bhp.fit(x_train,y_train)

ypred_train=dt_bhp.predict(x_train)
ypred_test=dt_bhp.predict(x_test)

print(accuracy_score(ypred_train,y_train))
print(accuracy_score(ypred_test,y_test))

scores=cross_val_score(dt_bhp,x,y,cv=5)
scores.mean()
```

Out[39]:
```
1.0
1.0
1.0
```

In [40]:
```
from sklearn.model_selection import GridSearchCV
estimator=RandomForestClassifier(random_state=32)
param_grid={'n_estimators':list(range(1,21))}
grid=GridSearchCV(estimator,param_grid,scoring='accuracy',cv=5)
grid.fit(x_train,y_train)
grid.best_params_
```

Out[40]:
```
{'n_estimators': 9}
```

In [41]:
```
rf_bhp=RandomForestClassifier(n_estimators=9,random_state=32)
rf_bhp.fit(x_train,y_train)

ypred_train=rf_bhp.predict(x_train)
ypred_test=rf_bhp.predict(x_test)

print(accuracy_score(ypred_train,y_train))
print(accuracy_score(ypred_test,y_test))

scores=cross_val_score(rf_bhp,x,y,cv=5)
scores.mean()
```

Out[41]:
```
1.0
0.9999309773605742
1.0
```

In [42]:
```
from sklearn.model_selection import GridSearchCV
estimator=GradientBoostingClassifier()
param_grid={'n_estimators':[1,5,10,20,40,100],
            'learning_rate':[0.1,0.2,0.3,0.5,0.8,1]}
grid=GridSearchCV(estimator,param_grid,scoring='accuracy',cv=5)
grid.fit(x_train,y_train)
grid.best_params_
```

Out[42]:
```
{'learning_rate': 0.1, 'n_estimators': 5}
```

In [43]:
```
gb_bhp=GradientBoostingClassifier(n_estimators=5,learning_rate=0.1)
gb_bhp.fit(x_train,y_train)

ypred_train=gb_bhp.predict(x_train)
ypred_test=gb_bhp.predict(x_test)

print(accuracy_score(ypred_train,y_train))
print(accuracy_score(ypred_test,y_test))

scores=cross_val_score(gb_bhp,x,y,cv=5)
scores.mean()
```

Out[43]:
```
1.0
1.0
1.0
```

In [44]:
```
from sklearn.model_selection import GridSearchCV
estimator=XGBClassifier()
param_grid={'n_estimators':[10,20,40,100],
            'max_depth':[3,4,5],
            'gamma':[0,0.15,0.3,0.5,1]}
```

```
grid=GridSearchCV(estimator,param_grid,scoring='accuracy',cv=5)
grid.fit(x_train,y_train)
grid.best_params_
```

Out[44]:
```
{'gamma': 0, 'max_depth': 3, 'n_estimators': 10}
```

In [45]:
```
xg_bhp=XGBClassifier(n_estimators=10,gamma=0,max_depth=3)
xg_bhp.fit(x_train,y_train)

ypred_train=xg_bhp.predict(x_train)
ypred_test=xg_bhp.predict(x_test)

print(accuracy_score(ypred_train,y_train))
print(accuracy_score(ypred_test,y_test))

scores=cross_val_score(xg_bhp,x,y,cv=5)
scores.mean()
```

Out[45]:
```
1.0
1.0
1.0
```

In [46]:
```
#Test data
```

In [47]:
```
df1=pd.read_csv("C:\\Users\\DELL\\Downloads\\Test_09JmpYa.csv.zip")
df1
```

Out[47]:

	Email_ID	Email_Type	Subject_Hotness_Score	Email_Source_Type	Customer_Location
0	EMA00081000168000	1	0.3	2	H
1	EMA00081000187610	1	1.3	2	G
2	EMA00081000244770	2	0.0	1	H
3	EMA00081000245260	1	2.3	1	G
4	EMA00081000264690	1	1.2	2	NaN
...	...	...	...	...	...
45973	EMA00089995144100	1	2.7	2	C
45974	EMA00089996757700	1	2.1	2	G
45975	EMA00089997159800	2	0.1	1	E
45976	EMA00089997508900	2	1.8	1	NaN
45977	EMA00089997599500	1	1.2	2	E

45978 rows × 11 columns

In [48]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45978 entries, 0 to 45977
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Email_ID                   45978 non-null  object
 1   Email_Type                 45978 non-null  int64
 2   Subject_Hotness_Score      45978 non-null  float64
 3   Email_Source_Type          45978 non-null  int64
 4   Customer_Location          38135 non-null  object
 5   Email_Campaign_Type        45978 non-null  int64
 6   Total_Past_Communications  41288 non-null  float64
 7   Time_Email_sent_Category   45978 non-null  int64
 8   Word_Count                 45978 non-null  int64
 9   Total_Links                44555 non-null  float64
 10  Total_Images               44885 non-null  float64
dtypes: float64(4), int64(5), object(2)
memory usage: 3.9+ MB
```

In [49]:
```
df1['Email_Status']=df['Email_Status']
```

In [50]:
```
df1.isnull().sum()
```

Out[50]:
```
Email_ID                       0
Email_Type                     0
Subject_Hotness_Score          0
Email_Source_Type              0
Customer_Location           7843
Email_Campaign_Type            0
Total_Past_Communications    4690
Time_Email_sent_Category       0
Word_Count                     0
Total_Links                 1423
Total_Images                1093
Email_Status               13517
dtype: int64
```

In [51]:
```
df1=df1.dropna()
```

In [52]:
```
df1.isnull().sum()
```

Out[52]:
```
Email_ID                   0
Email_Type                 0
Subject_Hotness_Score      0
Email_Source_Type          0
Customer_Location          0
Email_Campaign_Type        0
Total_Past_Communications  0
Time_Email_sent_Category   0
Word_Count                 0
Total_Links                0
Total_Images               0
Email_Status               0
dtype: int64
```

In [53]:
```
df1=df1.drop(columns=['Email_ID'])
```

In [ ]:

In [54]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df1['Customer_Location']=le.fit_transform(df1['Customer_Location'])
```

In [55]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit_transform(df1)
```

Out[55]:
```
array([[-0.62753738, -0.80090079,  1.08903321, ...,  0.56219952,
        -0.62753572, -0.46646972],
       [ 1.59353058, -1.10283597, -0.91824564, ..., -0.3707017 ,
        -0.26862205,  1.54778744],
       [-0.62753738,  1.21200046, -0.91824564, ..., -0.21521817,
         0.26974845, -0.46646972],
       ...,
       [-0.62753738,  1.61458071,  1.08903321, ...,  0.09574891,
        -0.62753572, -0.46646972],
       [-0.62753738,  1.01071034,  1.08903321, ...,  0.09574891,
         0.09029161, -0.46646972],
       [ 1.59353058, -1.00219091, -0.91824564, ..., -1.30360293,
        -0.62753572,  1.54778744]])
```

In [56]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
df1=scaler.fit_transform(df1)
```

In [57]:
```
df1
```

Out[57]:
```
array([[0.        , 0.06      , 1.        , ..., 0.27083333, 0.        ,
        0.        ],
       [1.        , 0.        , 0.        , ..., 0.14583333, 0.04255319,
        0.5       ],
       [0.        , 0.46      , 0.        , ..., 0.16666667, 0.10638298,
        0.        ],
       ...,
       [0.        , 0.54      , 1.        , ..., 0.20833333, 0.        ,
        0.        ],
       [0.        , 0.42      , 1.        , ..., 0.20833333, 0.08510638,
        0.        ],
       [1.        , 0.02      , 0.        , ..., 0.02083333, 0.        ,
        0.5       ]])
```

In [58]:
```
df1.shape
```

Out[58]:
```
(22942, 11)
```

In [59]:
```
log_model
```

Out[59]:
```
LogisticRegression()
```

In [60]:
```
predict=log_model.predict(df1)
```

ML Pipeline

In [61]:
```
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
```

In [66]:
```
logisticregressionpipeline=Pipeline([("myscaler",MinMaxScaler()),
                                     ('mypca',PCA(n_components=0.95)),
                                     ('logistic_classifier',log_model)])
```

In [67]:
```
dtpipeline=Pipeline([("myscaler",MinMaxScaler()),
                                     ('mypca',PCA(n_components=0.95)),
                                     ('dt_classifier',dt_model)])
```

In [69]:
```
gbpipeline=Pipeline([("myscaler",MinMaxScaler()),
                                     ('mypca',PCA(n_components=0.95)),
                                     ('gb_classifier',gb_model)])
```

In [70]:
```
xgpipeline=Pipeline([("myscaler",MinMaxScaler()),
                                     ('mypca',PCA(n_components=0.95)),
                                     ('xg_classifier',xg_model)])
```

In [71]:
```
mypipeline=[logisticregressionpipeline,dtpipeline,gbpipeline,xgpipeline]
```

In [72]:
```
for mypipe in mypipeline:
    mypipe.fit(x_train,y_train)
```

In [73]:
```
pipelinedict={0:'Logistic Regression',1:'Decision Tree',2:'Gradient boosting',3:'XGBoo
```

In [74]:
```
accuracy=1.0
classifier=0
pipeline=""
```

In [76]:
```
for i,model in enumerate(mypipeline):
    print('{} Test Accuracy:{}'.format(pipelinedict[i],model.score(x_test,y_test)))
```

```
Logistic Regression Test Accuracy:1.0
Decision Tree Test Accuracy:0.9977912755383765
Gradient boosting Test Accuracy:1.0
XGBoost Test Accuracy:1.0
```

In [77]:
```
for i,model in enumerate(mypipeline):
    if model.score(x_test,y_test)>accuracy:
        accuracy=model.score(x_test,y_test)
        pipeline=model
        classifier=i
print('classifier with best accuracy:{}'.format(pipelinedict[classifier]))
```

```
classifier with best accuracy:Logistic Regression
```

In [ ]: