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

In [2]:
```
df = pd.read_csv('/home/inshad/Downloads/WA_Fn-UseC_-Telco-Customer-Churn.csv')
df
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	Streaming
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	I
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	I
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	I
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	I
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	Yes	Yes	Y
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	Yes	No	Y
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	No	No	I
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	No	No	I
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	Yes	Yes	Y

7043 rows × 21 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

The data type of total charges is object while it is a numerical column, so we have to convert it into numerical column.

And it seems like an character or multiple character are present in it.So we will replace all non- numerical values with ' ' and then later replace it with np.nan

In [4]:
```
df['TotalCharges'] = df['TotalCharges'].str.replace('\W','',regex=True)
```

In [5]:
```
df['TotalCharges']=df['TotalCharges'].replace('',np.nan)
```

In [6]:
```
# converting the 'total charges' to numeric
df['TotalCharges']=df['TotalCharges'].apply(pd.to_numeric)
```

In [7]:
```
df.dtypes
```

Out[7]:
```
customerID          object
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
```

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

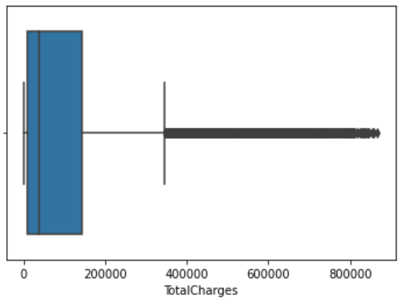
Out[8]:
```
customerID           0
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

In [9]:
```
print("mean:",df['TotalCharges'].mean())
print("median:",df['TotalCharges'].median())
print("mode:",df['TotalCharges'].mode())
```

```
mean: 126251.90841865756
median: 36769.0
mode: 0    202.0
dtype: float64
```

In [10]:
```
sns.boxplot(x=df['TotalCharges'],data=df)
```

Out[10]: <AxesSubplot:xlabel='TotalCharges'>



In [11]:
```
len(df[df['TotalCharges']<36769])
```

Out[11]: 3516

In [12]:
```
# The dataset have very extensive outliers, so we will replace missing values with median(around 36000)
df['TotalCharges'].fillna(value=df['TotalCharges'].median(),inplace=True)
```

In [13]:
```
df['TotalCharges'].isna().sum()
```

Out[13]: 0

From the primary anlaysis all the columns seems relevant except 'customer id' so we will drop it.

For the rest of the columns we label encode the columns and find correlation, then decide whether to drop or not.

In [14]:
```
df.drop('customerID',axis=1,inplace=True)
```

In [15]:
```
df
```

Out[15]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingT
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	N
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	N
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	N
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	N
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	N
...	...	...	...	...	...	...	...	...	...	...	...	...	.
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	No	Yes	Yes	Ye

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingT
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	Yes	Yes	No	Ye
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	No	No	No	N
7041	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	No	No	No	N
7042	Male	0	No	No	66	Yes	No	Fiber optic	Yes	No	Yes	Yes	Ye

In [16]:
```
df.columns
```

Out[16]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

Most of the columns have yes or no values so we will directly Label encode them. For the other columns we will check the value counts.

In [17]:
```
for i in ['MultipleLines','InternetService','Contract','PaymentMethod']:
    print(df[i].value_counts())
```

```
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
Fiber optic    3096
DSL            2421
No             1526
Name: InternetService, dtype: int64
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
```

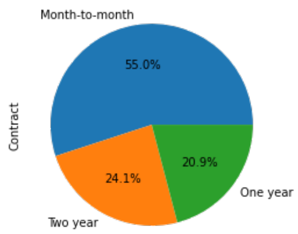
In [18]:
```
df.groupby('PaymentMethod')['TotalCharges'].mean().sort_values(ascending=False)
```

Out[18]:
```
PaymentMethod
Bank transfer (automatic)    171491.021373
Credit card (automatic)      165116.934297
Electronic check             117298.687526
Mailed check                  58750.851117
Name: TotalCharges, dtype: float64
```

Visualizations

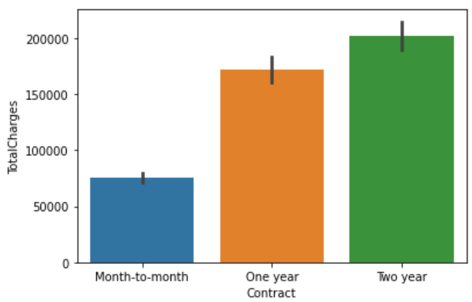
In [19]:
```
# diff types of contract
df['Contract'].value_counts().plot(kind='pie',autopct='%1.1f%%')
```

Out[19]: <AxesSubplot:ylabel='Contract'>



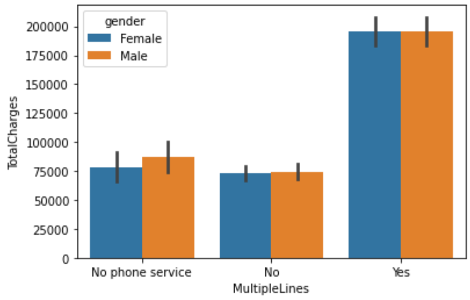
In [20]:
```
# contract type contributing more to the total charges
sns.barplot(x=df['Contract'],y=df['TotalCharges'],data=df)
```

Out[20]: <AxesSubplot:xlabel='Contract', ylabel='TotalCharges'>



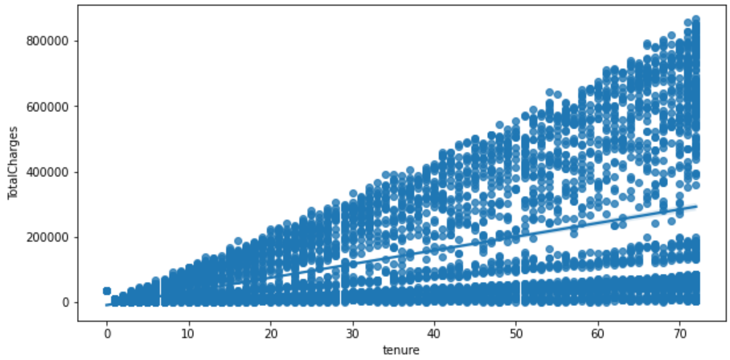
In [21]:
```
# people having multiple lines are contributing more to the total charges
sns.barplot(x=df['MultipleLines'],y=df['TotalCharges'],data=df,hue='gender')
```

Out[21]: <AxesSubplot:xlabel='MultipleLines', ylabel='TotalCharges'>



In [22]:
```
# with increase in the tenure there is increase in TotalCharges
plt.figure(figsize=(10,5))
sns.regplot(x=df['tenure'],y=df['TotalCharges'],data=df)
```

Out[22]: <AxesSubplot:xlabel='tenure', ylabel='TotalCharges'>



Label encoding the data set except numerical columns.

In [23]:
```
for i  in ['gender', 'Partner', 'Dependents',
        'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
        'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
        'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn']:
    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    df[i]=le.fit_transform(df[i])
df
```

Out[23]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingT
0	0	0	1	0	1	0	1	0	0	2	0	0	
1	1	0	0	0	34	1	0	0	2	0	2	0	
2	1	0	0	0	2	1	0	0	2	2	0	0	
3	1	0	0	0	45	0	1	0	2	0	2	2	
4	0	0	0	0	2	1	0	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	0	2	0	2	2	
7039	0	0	1	1	72	1	2	1	0	2	2	0	
7040	0	0	1	1	11	0	1	0	2	0	0	0	
7041	1	1	1	0	4	1	2	1	0	0	0	0	
7042	1	0	0	0	66	1	0	1	2	0	2	2	

7043 rows × 20 columns

In [24]:
```
df.describe()
```

Out[24]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupp
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000
mean	0.504756	0.162147	0.483033	0.299588	32.371149	0.903166	0.940508	0.872923	0.790004	0.906432	0.904444	0.797
std	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752	0.948554	0.737796	0.859848	0.880162	0.879949	0.861
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	1.000000	0.000000	0.000000	0.000000	29.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	2.000000	1.000000	2.000000	2.000000	2.000000	2.000
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000

In [25]:
```
df.corr()
```

Out[25]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechS
gender	1.000000	-0.001874	-0.001808	0.010517	0.005106	-0.006488	-0.006739	-0.000863	-0.015017	-0.012057	0.000549	-0.0
SeniorCitizen	-0.001874	1.000000	0.016479	-0.211185	0.016567	0.008576	0.146185	-0.032310	-0.128221	-0.013632	-0.021398	-0.1

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSu
Partner	-0.001808	0.016479	1.000000	0.452676	0.379697	0.017706	0.142410	0.000891	0.150828	0.153130	0.166330	0.1
Dependents	0.010517	-0.211185	0.452676	1.000000	0.159712	-0.001762	-0.024991	0.044590	0.152166	0.091015	0.080537	0.1
tenure	0.005106	0.016567	0.379697	0.159712	1.000000	0.008448	0.343032	-0.030359	0.325468	0.370876	0.371105	0.3
PhoneService	-0.006488	0.008576	0.017706	-0.001762	0.008448	1.000000	-0.020538	0.387436	-0.015198	0.024105	0.003727	-0.0
MultipleLines	-0.006739	0.146185	0.142410	-0.024991	0.343032	-0.020538	1.000000	-0.109216	0.007141	0.117327	0.122318	0.0
InternetService	-0.000863	-0.032310	0.000891	0.044590	-0.030359	0.387436	-0.109216	1.000000	-0.028416	0.036138	0.044944	-0.0
OnlineSecurity	-0.015017	-0.128221	0.150828	0.152166	0.325468	-0.015198	0.007141	-0.028416	1.000000	0.185126	0.175985	0.2
OnlineBackup	-0.012057	-0.013632	0.153130	0.091015	0.370876	0.024105	0.117327	0.036138	0.185126	1.000000	0.187757	0.1
DeviceProtection	0.000549	-0.021398	0.166330	0.080537	0.371105	0.003727	0.122318	0.044944	0.175985	0.187757	1.000000	0.2
TechSupport	-0.006825	-0.151268	0.126733	0.133524	0.322942	-0.019158	0.011466	-0.026047	0.285028	0.195748	0.240593	1.0
StreamingTV	-0.006421	0.030776	0.137341	0.046885	0.289373	0.055353	0.175059	0.107417	0.044669	0.147186	0.276652	0.1
StreamingMovies	-0.008743	0.047266	0.129574	0.021321	0.296866	0.043870	0.180957	0.098350	0.055954	0.136722	0.288799	0.1
Contract	0.000126	-0.142554	0.294806	0.243187	0.671607	0.002247	0.110842	0.099721	0.374416	0.280980	0.350277	0.4
PaperlessBilling	-0.011754	0.156530	-0.014877	-0.111377	0.006152	0.016505	0.165146	-0.138625	-0.157641	-0.013370	-0.038234	-0.1
PaymentMethod	0.017352	-0.038551	-0.154798	-0.040292	-0.370436	-0.004184	-0.176793	0.086140	-0.096726	-0.124847	-0.135750	-0.1
MonthlyCharges	-0.014569	0.220173	0.096848	-0.113890	0.247900	0.247398	0.433576	-0.323260	-0.053878	0.119777	0.163652	-0.0
TotalCharges	0.002091	0.091066	0.201766	0.043723	0.534920	0.073512	0.299457	-0.115772	0.165065	0.252791	0.245631	0.1

Machine Learning

In [26]:
```
X = df.drop('Churn',axis=1).values
y = df['Churn'].values
```

In [27]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Support Vector Machine

In [30]:
```
from sklearn.svm  import SVC
svc_model = SVC()
svc_model.fit(X_train,y_train)
```

Out[30]: SVC()

In [31]:
```
y_pred = svc_model.predict(X_test)
```

In [32]:
```
from sklearn.metrics import classification_report,ConfusionMatrixDisplay
```

In [33]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      1697
           1       0.70      0.50      0.58       628

    accuracy                           0.81      2325
   macro avg       0.77      0.71      0.73      2325
weighted avg       0.80      0.81      0.80      2325
```

Naive Bayes

In [34]:
```
from sklearn.naive_bayes import GaussianNB
naive_model = GaussianNB()
naive_model.fit(X_train,y_train)
```

Out[34]: GaussianNB()

In [35]:
```
naive_pred = naive_model.predict(X_test)
```

In [36]:
```
print(classification_report(naive_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.90      0.82      1410
           1       0.78      0.53      0.63       915

    accuracy                           0.76      2325
   macro avg       0.76      0.72      0.72      2325
weighted avg       0.76      0.76      0.74      2325
```

K-Neighbours classifier

In [37]:
```
from sklearn.neighbors import KNeighborsClassifier
knnmodel = KNeighborsClassifier()
knnmodel.fit(X_train,y_train)
```
Out[37]:
```
KNeighborsClassifier()
```
In [38]:
```
knn_pred = knnmodel.predict(X_test)
```
In [39]:
```
print(classification_report(knn_pred,y_test))
```
```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83      1697
           1       0.54      0.54      0.54       628

    accuracy                           0.75      2325
   macro avg       0.69      0.69      0.69      2325
weighted avg       0.75      0.75      0.75      2325
```

Logistic Regression

In [40]:
```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(X_train,y_train)
```
Out[40]:
```
LogisticRegression()
```
In [41]:
```
lg_pred = lg.predict(X_test)
```
In [42]:
```
print(classification_report(lg_pred,y_test))
```
```
              precision    recall  f1-score   support

           0       0.90      0.85      0.88      1786
           1       0.59      0.68      0.63       539

    accuracy                           0.82      2325
   macro avg       0.74      0.77      0.75      2325
weighted avg       0.83      0.82      0.82      2325
```

Decision Tree

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(criterion='entropy')
tree_model.fit(X_train,y_train)
```
Out[43]:
```
DecisionTreeClassifier(criterion='entropy')
```
In [44]:
```
tree_pred = tree_model.predict(X_test)
```
In [45]:
```
print(classification_report(y_test,tree_pred))
```
```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1697
           1       0.51      0.50      0.51       628

    accuracy                           0.73      2325
   macro avg       0.66      0.66      0.66      2325
weighted avg       0.73      0.73      0.73      2325
```

Random Forest

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train,y_train)
```
Out[46]:
```
RandomForestClassifier()
```
In [47]:
```
rf_pred = rf_model.predict(X_test)
```
In [48]:
```
print(classification_report(y_test,rf_pred))
```
```
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1697
           1       0.66      0.51      0.58       628

    accuracy                           0.80      2325
   macro avg       0.75      0.71      0.72      2325
weighted avg       0.79      0.80      0.79      2325
```

Our data set is an imbalanced data set ,so we will apply SMOTE (oversampling only) to check whether our model performs better.

In [49]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
Xo,yo = smote.fit_resample(X,y)
```

Support Vector Machine using SMOTE

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(Xo, yo, test_size=0.33, random_state=42)
```

In [51]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [52]:
```
from sklearn.svm  import SVC
svc_model = SVC()
svc_model.fit(X_train,y_train)
```

Out[52]:
```
SVC()
```

In [53]:
```
y_pred_os = svc_model.predict(X_test)
```

In [54]:
```
print(classification_report(y_test,y_pred_os))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      1730
           1       0.85      0.84      0.84      1685

    accuracy                           0.84      3415
   macro avg       0.84      0.84      0.84      3415
weighted avg       0.84      0.84      0.84      3415
```

Naive Bayes using SMOTE

In [55]:
```
from sklearn.naive_bayes import GaussianNB
naive_model = GaussianNB()
naive_model.fit(X_train,y_train)
```

Out[55]:
```
GaussianNB()
```

In [56]:
```
naive_pred_os = naive_model.predict(X_test)
```

In [57]:
```
print(classification_report(naive_pred_os,y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.81      0.78      1604
           1       0.82      0.76      0.79      1811

    accuracy                           0.78      3415
   macro avg       0.78      0.78      0.78      3415
weighted avg       0.78      0.78      0.78      3415
```

K-Neighbours classifier using SMOTE

In [58]:
```
from sklearn.neighbors import KNeighborsClassifier
knnmodel = KNeighborsClassifier()
knnmodel.fit(X_train,y_train)
```

Out[58]:
```
KNeighborsClassifier()
```

In [59]:
```
knn_pred_os = knnmodel.predict(X_test)
```

In [60]:
```
print(classification_report(knn_pred_os,y_test))
```

```
              precision    recall  f1-score   support

           0       0.69      0.86      0.77      1400
           1       0.88      0.74      0.80      2015

    accuracy                           0.79      3415
   macro avg       0.79      0.80      0.78      3415
weighted avg       0.80      0.79      0.79      3415
```

Logistic Regression using SMOTE

In [61]:
```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(X_train,y_train)
```

Out[61]:
```
LogisticRegression()
```

In [62]:
```
lg_pred_os = lg.predict(X_test)
```

In [63]:
```
print(classification_report(lg_pred_os,y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.81      0.78      1599
           1       0.82      0.76      0.79      1816

    accuracy                           0.79      3415
   macro avg       0.79      0.79      0.79      3415
weighted avg       0.79      0.79      0.79      3415
```

Decision Tree using SMOTE

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(criterion='entropy')
tree_model.fit(X_train,y_train)
```

Out[64]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [65]:
```
tree_pred_os = tree_model.predict(X_test)
```

In [66]:
```
print(classification_report(tree_pred_os,y_test))
```

```
              precision    recall  f1-score   support

           0       0.81      0.82      0.82      1713
           1       0.82      0.81      0.81      1702

    accuracy                           0.81      3415
   macro avg       0.81      0.81      0.81      3415
weighted avg       0.81      0.81      0.81      3415
```

Random Forest using SMOTE

In [67]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train,y_train)
```

Out[67]:
```
RandomForestClassifier()
```

In [68]:
```
rf_pred_os = rf_model.predict(X_test)
```

In [69]:
```
print(classification_report(y_test,rf_pred_os))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1730
           1       0.89      0.82      0.86      1685

    accuracy                           0.86      3415
   macro avg       0.87      0.86      0.86      3415
weighted avg       0.87      0.86      0.86      3415
```

Conclusion

In [70]:
```
# Dataframe of our ML model of various algorithms with and without smote
```

In [71]:
```
overview = pd.DataFrame({
    "Algorithms":['SVM','Naive_Bayes','KNN','Logistic_Regression','Decision_tree','Random_Forest'],
    "Accuracy":[0.81,0.76,0.75,0.82,0.73,0.80],
    "Accuracy_SMOTE":[0.84,0.78,0.79,0.79,0.82,0.87]
})
```

In [72]:
```
overview
```

Out[72]:

	Algorithms	Accuracy	Accuracy_SMOTE
0	SVM	0.81	0.84
1	Naive_Bayes	0.76	0.78
2	KNN	0.75	0.79
3	Logistic_Regression	0.82	0.79
4	Decision_tree	0.73	0.82
5	Random_Forest	0.80	0.87

When SMOTE is applied our model performs much better than our actual model.

Among that Random Forest classifier performs well with an accuracy score of 0.87. Also Random forest using smote have much better f1 score and recall score