TELECOM CHURN PREDICTION MODEL USING ML

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
#Load dataset
df=pd.read_csv("/content/drive/MyDrive/ML project /telecom_churn.csv")
```

```
#check first 5 records
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	Nc

```
#Check number of rows and columns
df.shape
```

```
(7043, 21)
```

QPYBK

```
#Check column name present in data set
df.columns
```

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

```
#check number of duplicate values
df.duplicated().sum()
```

```
0
```

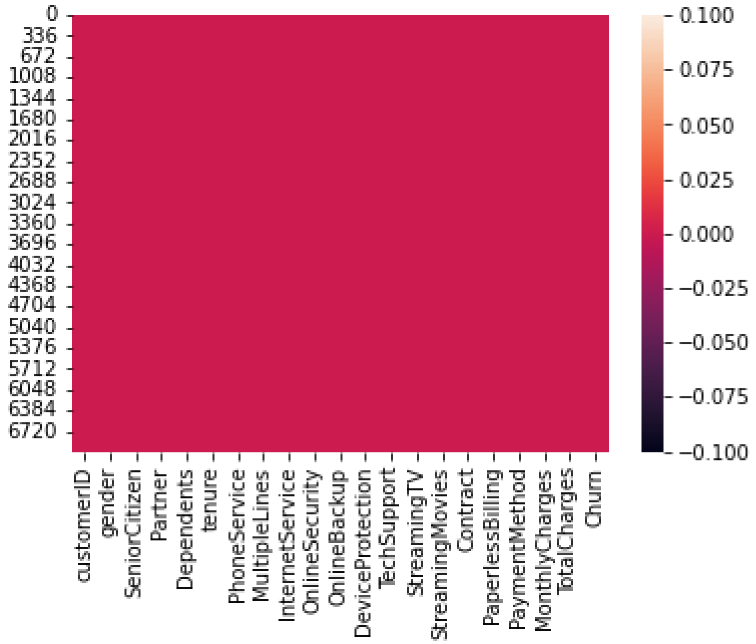
```
#Chexk null values
df.isnull().sum()
```

```
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        0
Churn               0
dtype: int64
```

```
#visualize null values
sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7888821a10>
```



```
#check data types of column
df.dtypes
```

```
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         object
Churn                object
dtype: object
```

```
df.sample(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecur
933	4750-ZRXIU	Female	1	No	No	4	Yes	Yes	Fiber optic	
3847	8439-LTUGF	Male	0	No	No	10	Yes	No	No	No inte ser
1280	2388-LAESQ	Female	1	Yes	No	72	Yes	Yes	Fiber optic	

```
#check unvanted value in Totalcharges column
df["TotalCharges"].unique()
```

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

```
#change the unvanted value in null values
df["TotalCharges"]=pd.to_numeric(df["TotalCharges"].astype(str),errors="coerce")
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	Nc
	5575-									

```
df.dtypes
```

```
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
dtype: object
```

```
#find the mean of totalcharges column
m=df["TotalCharges"].mean()
m
```

```
2283.3004408418656
```

```
#check null values
```

```
df.isnull().sum()
```

```
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
```

```
#fill null value with mean of the column
df['TotalCharges'].fillna(m,inplace=True)
```

```
#check null value
df.isnull().sum()
```

```
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        0
Churn               0
dtype: int64
```

```
df['Churn'].value_counts().plot(kind='bar', figsize=(8, 6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f78858aeb90>
```

```
#how many percantage 1 and 0
100*df['Churn'].value_counts()/len(df['Churn'])
```

```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

```
df["Churn"].value_counts()
```

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No

5 rows × 21 columns

```
#REMOVE CUSTOMERID COLUMN PERMANENT
df.drop('customerID',axis=1,inplace=True)
```

```
#SEPRATE ALL NUM TYPE DATA HOLD IN DF_NUM
df_num=df.select_dtypes(['int64','float64'])
#SEPRATE ALL OBJECT TYPE DATA HOLD IN DF_CAT
df_cat=df.select_dtypes(object)
```

```
#TO CONVERT OBJECT TYPE DATA IN NUMBER USING LABELENCODER
from sklearn.preprocessing import LabelEncoder
for col in df_cat:
  le=LabelEncoder()
  df_cat[col]=le.fit_transform(df_cat[col])
```

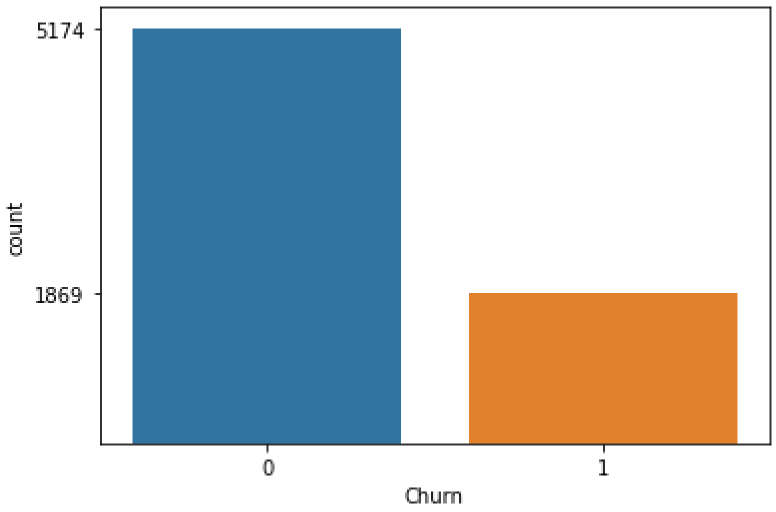
```
#CONCATENATE BOTH DATAFRAME
df_new=pd.concat([df_num,df_cat],axis=1)
```

```
df_new.head()
```

```
df_new["Churn"].value_counts()
```

```
0    5174
1    1869
Name: Churn, dtype: int64
```

```
#VISUALIZS
sns.countplot(data=df_new,x='Churn')
f=df_new['Churn'].value_counts()
plt.yticks(f)
plt.show()
```




```
#HERE IS CLEAR UNDARSTAND DATA IS IMBALANCE
#WE HAVE TO BALANCE DATASET
```

```
#DIVIDE DATA INTO 70% AND 30% FOR TRAIN AND TEST
```

```
#SELECT INPUT AND OUTPUT FROM DATASET
```

```
x=df_new.drop('Churn',axis=1) #INPUT VARIABLE
y=df_new['Churn'] #OUTPUT TARGET


#TRAIN TEST SPLIT
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=1)
```

APPLY SCALING

```
#WORK ON STANDARDSCALER
from sklearn.preprocessing import StandardScaler
#FIT_TRANSFORM TRAING DATA X_TRAIN
#TRANSFORM USE ONLY TESTING DATA MEANS X_TEST
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.transform(x_test)


#APPLY RANDOMEOVERSAMPLER
#FIRST CREATE THE OBJECT OF CLASS RANDOMOVERSAMPLER
from imblearn.over_sampling import RandomOverSampler


#CREATE OBJECT OF RANDOMOVERSAMPLER CLASS
ros=RandomOverSampler(random_state=1)



#APPLY OVERSAMPLER TRAINING DATA(70%)
x_train1,y_train1=ros.fit_resample(x_train,y_train)
```

```
#CHECK AFTER APPLY RANDOMOVERSAMPLER
y_train1.value_counts()
```

```
0    3589
1    3589
Name: Churn, dtype: int64
```

```
#BEFORE APPLY RANDOMOVERSAPMLER TESTING DATA
y_test.value_counts()
```

```
0    1585
1     528
Name: Churn, dtype: int64
```

```
#ALSO APPLY RANDOMOVERSAMPLER TESTING DATA(30%)
x_test1,y_test1=ros.fit_resample(x_test,y_test)
```

```
#CHECK AFTER APLLY RANDOMOVERSAMPLER
y_test1.value_counts()
```

```
0    1585
1    1585
Name: Churn, dtype: int64
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
#CREATE A FUNCTION
def create_model(model):
 model.fit(x_train1,y_train1) #TRAIN THE MODAL
 y_pred=model.predict(x_test1) #TEST MODEL
 print(classification_report(y_test1,y_pred))
 print('confusion matrix :' )
 #CONFUSTION MATRIX
```

```
    print(confusion_matrix(y_test1,y_pred))
    return model,y_pred


#BASE LINE MODEL MENS LOGISTICREGRESSION (WE PREDICT YES OR NO VALUE THEN USE CLA
from sklearn.linear_model import LogisticRegression


#GIVEN TRANING DATA LOGISTICREGRESSION ALGORITHEM
#CREATE LOGISTIC CLASS
lr=LogisticRegression()



#CALL FUNCTION TRAIN AND TEST THE MODEL
lr,y_pred=create_model(lr)
```

```
                  precision    recall  f1-score   support

               0       0.80      0.73      0.77      1585
               1       0.75      0.82      0.79      1585

        accuracy                           0.78      3170
       macro avg       0.78      0.78      0.78      3170
    weighted avg       0.78      0.78      0.78      3170

     confusion matrix :
     [[1162  423]
      [ 284 1301]]
```

```
#Note:- Here logisticregression we have to focus recall score here recall score for class
 # 0 is 73 and reacll score class 1 is 82
#Let's test the data with other model(algorithem)
```

DecisionTreeClassifier

```
#now perfome data set with the help of decisiontreeclassifier
from sklearn.tree import DecisionTreeClassifier



#create object of decisiontreeclassifier
dt=DecisionTreeClassifier()


#call function train and test the model
dt=create_model(dt)
```

```
                  precision    recall  f1-score   support

               0       0.64      0.81      0.71      1585
               1       0.74      0.54      0.62      1585

        accuracy                           0.67      3170
       macro avg       0.69      0.67      0.67      3170
    weighted avg       0.69      0.67      0.67      3170

    confusion matrix :
    [[1285  300]
     [ 731  854]]
```

```
#here is clearly undrstand that the model is over fit, so reduced the overfitting
#over fiting situation by using the pruning technique
#1:-max_depth
#2:-min_samples_leaf


#1:- max_depth
dt1=DecisionTreeClassifier(max_depth=3,random_state=1)


#call function train and test the model
dt1=create_model(dt1)
```

```
                  precision    recall  f1-score   support
```

```
               0       0.80      0.69      0.74      1585
               1       0.73      0.83      0.78      1585

        accuracy                           0.76      3170
       macro avg       0.77      0.76      0.76      3170
    weighted avg       0.77      0.76      0.76      3170

    confusion matrix :
    [[1099  486]
     [ 272 1313]]
```

```
#2:-min_samples_leaf
dt2=DecisionTreeClassifier(min_samples_leaf=70,random_state=1)



#call function train and test the model
dt2=create_model(dt2)
```

```
                  precision    recall  f1-score   support

               0       0.77      0.74      0.75      1585
               1       0.75      0.78      0.76      1585

        accuracy                           0.76      3170
       macro avg       0.76      0.76      0.76      3170
    weighted avg       0.76      0.76      0.76      3170

    confusion matrix :
    [[1168  417]
     [ 347 1238]]
```

```
#here decision tree classifier with use pruning technique max_depth give about ac
#we have focur recall score of class 0 is 69 and 1 is 83
#here decision tree classifier with use pruning technique min_depth give about ac
#we have focuse on recall score of class 0 is 0.74 and 1 is 0.78
#test data with other model
```

Random Forest Classifier

```
#call random forest tree
from sklearn.ensemble import RandomForestClassifier
```

```
#create the object on randomforestclassifer class
rfc=RandomForestClassifier(n_estimators=15,max_features=12,random_state=1)
```

```
#call function train and test the model
rfc=create_model(rfc)
```

```
              precision    recall  f1-score   support

           0       0.69      0.82      0.75      1585
           1       0.78      0.64      0.70      1585

    accuracy                           0.73      3170
   macro avg       0.74      0.73      0.73      3170
weighted avg       0.74      0.73      0.73      3170

confusion matrix :
[[1301  284]
 [ 572 1013]]
```

```
#here random forestreeclassifier class recall o is 0.82 and 1 is 0.64
#test data with other model
```

Boosting Technique

ensembling technique

```
#1. ADA boost
from sklearn.ensemble import AdaBoostClassifier
#create the object ADAboost
ada=AdaBoostClassifier(n_estimators=6,random_state=1)


#call function train and test the model
ada=create_model(ada)
```

```
              precision    recall  f1-score   support

           0       0.84      0.66      0.74      1585
           1       0.72      0.87      0.79      1585

    accuracy                           0.77      3170
   macro avg       0.78      0.77      0.76      3170
weighted avg       0.78      0.77      0.76      3170

confusion matrix :
[[1043  542]
 [ 199 1386]]
```

```
#2. gradientboostingclassifer
from sklearn.ensemble import GradientBoostingClassifier


#create the object gradientboosting
gbc=GradientBoostingClassifier(n_estimators=50,random_state=1)


#call function train and test the model
gbc=create_model(gbc)
```

```
              precision    recall  f1-score   support

           0       0.80      0.73      0.76      1585
```

```
              1       0.75      0.82      0.78      1585

       accuracy                           0.77      3170
      macro avg       0.78      0.77      0.77      3170
   weighted avg       0.78      0.77      0.77      3170

   confusion matrix :
   [[1154  431]
    [ 287 1298]]
```

```
#3. XGB classifier
from xgboost import XGBClassifier


#create the object xgradient
xgb=XGBClassifier(n_estimators=8,reg_alpha=1,random_state=1)


#call function train and test the model
xgb=create_model(xgb)
```

```
                 precision    recall  f1-score   support

              0       0.82      0.68      0.75      1585
              1       0.73      0.85      0.79      1585

       accuracy                           0.77      3170
      macro avg       0.78      0.77      0.77      3170
   weighted avg       0.78      0.77      0.77      3170

   confusion matrix :
   [[1083  502]
    [ 230 1355]]
```

```
#here ADA boost give about accuracy 0.77 score but here
#we have focur recall score of class 0 is 66 and 1 is 87
#here gradient boost give about accuracy 0.77 score but here
#we have focur recall score of class 0 is 73 and 1 is 82
```

```
#here XGBgradient boost give about accuracy 0.78 score but here
#we have focur recall score of class 0 is 68 and 1 is 85
#Let's test data with other model
```

K-NN(KNeighborsClassifier):

```
from sklearn.neighbors import KNeighborsClassifier
#create the object k-nn
knc=KNeighborsClassifier(n_neighbors=15, metric='minkowski',p=2)
```

```
#call function train and test the model
knc=create_model(knc)
```

```
                  precision    recall  f1-score   support

               0       0.81      0.67      0.73      1585
               1       0.72      0.84      0.77      1585

        accuracy                           0.75      3170
       macro avg       0.76      0.75      0.75      3170
    weighted avg       0.76      0.75      0.75      3170

     confusion matrix :
     [[1058  527]
      [ 252 1333]]
```

```
#here k-nn give about accuracy 0.75 score but here
#we have focur recall score of class 0 is 67 and 1 is 84
#Let's test data with other model
```

Suport Vector Machine

```
from sklearn.svm import LinearSVC
#create the object svm
svc=LinearSVC(random_state=1)
#call function train and test the model
svc=create_model(svc)
```

```
              precision    recall  f1-score   support

           0       0.82      0.72      0.77      1585
           1       0.75      0.84      0.79      1585

    accuracy                           0.78      3170
   macro avg       0.78      0.78      0.78      3170
weighted avg       0.78      0.78      0.78      3170

confusion matrix :
[[1143  442]
 [ 259 1326]]
```

```
#here accuracy is 0.78 which is good but we can more better
#add some external error on trainig time object od LinerSVS
```

```
#create the object svm1
svc1=LinearSVC(random_state=1,C=0.7)
```

```
#call function train test the model
svc1=create_model(svc1)
```

```
              precision    recall  f1-score   support

           0       0.82      0.72      0.77      1585
           1       0.75      0.84      0.79      1585

    accuracy                           0.78      3170
```

```
   macro avg       0.78      0.78      0.78      3170
weighted avg       0.78      0.78      0.78      3170

confusion matrix :
[[1143  442]
 [ 259 1326]]
```

```
#here data is non-linear changing the value 'c' above acuracy not change
#adding external error traning time but any no changes on score
#mens data is linear if any changes then data is non-linear
```

```
#work on non-linear dataset
from sklearn.svm import SVC
#create the object poly
svc2=SVC(random_state=1,kernel='poly')
```

```
#call function train and test the model
svc2=create_model(svc2)
```

```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      1585
           1       0.75      0.79      0.77      1585

    accuracy                           0.77      3170
   macro avg       0.77      0.77      0.77      3170
weighted avg       0.77      0.77      0.77      3170

confusion matrix :
[[1173  412]
 [ 331 1254]]
```

```
#work on radial basis
svc3=SVC(random_state=1,kernel='rbf')
```

```
#call function train and test the model
svc3=create_model(svc3)
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      1585
           1       0.76      0.78      0.77      1585

    accuracy                           0.77      3170
   macro avg       0.77      0.77      0.77      3170
weighted avg       0.77      0.77      0.77      3170

confusion matrix :
[[1197  388]
 [ 345 1240]]
```

```
#here poly give about accuracy 0.77 score but here
#we have focur recall score of class 0 is 0.74 and 1 is 0.79
#Let's test data with other model
#here radiel give about accuracy 0.77 score but here
#we have focur recall score of class 0 is 0.76 and 1 is 0.78
#Let's test data with other model
```

Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
```

```
#create object navie bayes classifier
gnb=GaussianNB()
#call function train and test the model
gnb=create_model(gnb)
```

```
              precision    recall  f1-score   support
```

```
           0       0.80      0.72      0.76      1585
           1       0.75      0.83      0.78      1585

    accuracy                           0.77      3170
   macro avg       0.78      0.77      0.77      3170
weighted avg       0.78      0.77      0.77      3170

confusion matrix :
[[1140  445]
 [ 277 1308]]
```

```
#here naive bayes give about accuracy 0.77 score but here
#we have focur recall score of class 0 is 0.72 and 1 is 0.83
#Let's test data with other model
```

CONCLUSION

here we will recommend Suport Vector Machine algorithem for the give data for telecom_churn, here accuracy is 0.78 which is good better,we have focur recall score of class 0 is 0.72 and 1 is 0.84

✓ 0s completed at 1:06 PM