CUSTOMER CHURN PREDICTION USING MACHINE LEARNING

In this blog-post, I will go through the whole process of creating a machine learning model from loading the dataset to saving the best model on the customer churn dataset from IBM Sample Data Sets. It provides information on the churn rate of customers, summarized according to Age, Tenure, Contract, Monthly charges, Total charges and other features.

What is Customer Churn?

Customer churn is when a company's customers stop doing business with that company. Businesses are very keen on measuring churn because keeping an existing customer is far less expensive than acquiring a new customer. New business involves working leads through a sales funnel, using marketing and sales budgets to gain additional customers. Existing customers will often have a higher volume of service consumption and can generate additional customer referrals.

Why is Customer Retention so important?

There are some excellent reasons to focus on customer retention:

- It's up to five times cheaper to retain your current customers than it is to acquire new ones.
- The probability of making a sale to an existing customer is 60-70%
- Retaining your current customers increases word-of-mouth recommendations and loyalty

and many more.

Customer retention can suffer when you attract new customers and grow quickly but struggle to implement a strong customer service strategy as a foundation to support that growth.

Customer Retention strategies:

Customer retention can be achieved with good customer service and products. But the most effective way for a company to prevent attrition of customers is to truly know them. The vast volumes of data collected about customers can be used to build churn prediction models. Knowing who is most likely to defect means that a company can prioritise focused marketing efforts on that subset of their customer base.

Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

Importing libraries:

```
#linear algebra

import numpy as np

#data processing

import pandas as pd

#data visualization

import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

#algorithms

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier , AdaBoostClassifier , GradientBoostingClassifier

#importing libraries

from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_curve,auc
```

Getting the data:

```
#Loading the dataset

df=pd.read_csv("customerchurn.csv")
df
```

Data Analysis and Preprocessing:

Let’s have a look at the data:

```
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   float64
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(17)
```

A sample from the dataset:

```
In [9]: pd.set_option('display.max_columns',None)
        df.sample(10)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	Tech
745	2499-AJYUA	Female	1	Yes	No	72	Yes	Yes	Fiber optic	No	Yes	Yes	
1711	9964-WBQDJ	Female	0	Yes	No	71	Yes	Yes	No	No internet service	No internet service	No internet service	No
3370	5307-UVGNB	Female	0	Yes	Yes	53	No	No phone service	DSL	Yes	Yes	Yes	
360	9294-TDIPC	Male	0	No	Yes	5	Yes	No	No	No internet service	No internet service	No internet service	No
2552	3547-LQRIK	Female	0	Yes	No	47	Yes	Yes	No	No internet service	No internet service	No internet service	No
6734	8182-PNAGI	Male	0	No	No	12	Yes	No	DSL	No	No	No	
217	0230-WEQUW	Male	0	Yes	No	66	No	No phone service	DSL	Yes	Yes	Yes	
	9933												

The dataset has 7043 examples and 20 features + the target variable (Churn). 2 of the features are floats, 2 are integers and 17 are objects. Below I have listed the variables with a short description:

CustomerID: A unique ID that identifies each customer.
Gender: The customer's gender: Male, Female
SeniorCitizen: Indicates if the customer is 65 or older: Yes, No
Partner: Indicates if the customer has a partner: Yes, No
Dependents: Indicates if the customer lives with any dependents: Yes, No. Dependents could be children, parents, grandparents, etc.
tenure: Indicates the total amount of months that the customer has been with the company.

PhoneService: Indicates if the customer subscribes to home phone service with the company: Yes, No
MultipleLines: Indicates if the customer subscribes to multiple telephone lines with the company: Yes, No
InternetService: Indicates if the customer subscribes to Internet service with the company: No, DSL, Fiber Optic.
OnlineSecurity: Indicates if the customer subscribes to an additional online security service provided by the company: Yes, No
OnlineBackup: Indicates if the customer subscribes to an additional online backup service provided by the company: Yes, No
DeviceProtection: Indicates if the customer subscribes to an additional device protection plan for their Internet equipment provided by the company: Yes, No
TechSupport: Indicates if the customer subscribes to an additional technical support plan from the company with reduced wait times: Yes, No
StreamingTV: Indicates if the customer uses their Internet service to stream television programing from a third party provider: Yes, No. The company does not charge an additional fee for this service.
StreamingMovies: Indicates if the customer uses their Internet service to stream movies from a third party provider: Yes, No. The company does not charge an additional fee for this service.
Contract: Indicates the customer's current contract type: Month-to-Month, One Year, Two Year.
PaperlessBilling: Indicates if the customer has chosen paperless billing: Yes, No
PaymentMethod: Indicates how the customer pays their bill: Electronic check, Bank transfer (automatic), Credit Card (automatic), Mailed check
MonthlyCharge: Indicates the customer's current total monthly charge for all their services from the company.

TotalCharges: Indicates the customer's total charges, calculated to the end of the quarter specified above.
Churn: Customers who left within the last month: Yes, No

Describing the dataset further:

```
df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2279.765853
std	0.368612	24.559481	30.090047	2266.762876
min	0.000000	0.000000	18.250000	18.800000
25%	0.000000	9.000000	35.500000	398.550000
50%	0.000000	29.000000	70.350000	1394.550000
75%	0.000000	55.000000	89.850000	3786.600000
max	1.000000	72.000000	118.750000	8684.800000

Now, first thing that I would do is to check for empty or null values, like if they are present in the dataset.

```
#Lets check for missing 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
```

So as we can see that the dataset is showing zero empty values for every variable. Now, let's look out for data-type of each variable for a better understanding of the dataset:

```
#checking for datatype

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
```

We can see that variable 'TotalCharges' has been assigned 'object' datatype instead of 'float64' , as it has float values. So, we must convert its datatype from 'object' to 'float64'.

Upon checking into 'TotalCharges' before converting its datatype I found that this variables have some empty strings present. In order to convert its datatype first we must replace these empty strings. So, I replaced those empty strings with numpy NaN value and again checked for the count of now null values.

```
df.replace(' ',np.nan,inplace=True)
```

```
#now again checking for missing 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
```

‘TotalCharges’ now has 11 null values. Before converting its datatype, I am going to impute those null values using SimpleImputer from sklearn.

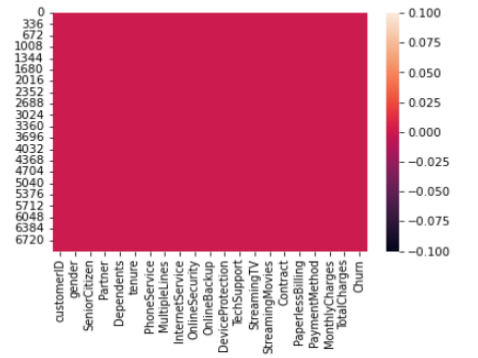
As the variable is of ‘object’ datatype, SimpleImputer with strategy as ‘most_frequent’ which represents the mode of the data, should be used. After that I converted the variable ‘TotalCharges’ to ‘float64’ using pandas ‘to_numeric’ function.

```
from sklearn.impute import SimpleImputer

df['TotalCharges']=SimpleImputer(strategy='most_frequent').fit_transform(df['TotalCharges'].values.reshape(-1,1))

df['TotalCharges']=pd.to_numeric(df['TotalCharges'])
```

Again checking for null values:

```
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



All the null values has been replaced. Now we can move towards exploratory data analysis (EDA).

Since, the preprocessing hasn’t been completed yet, I am going to perform EDA first and just after that I am going to do the left preprocessing i.e. the conversion of ‘object’ datatype variables to integer datatype.

Exploratory Data Analysis:

```
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')
```

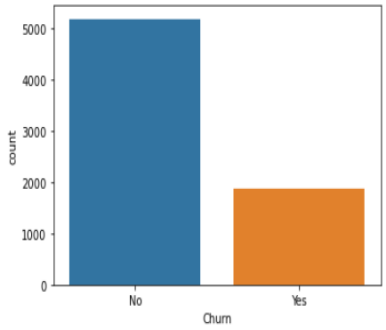
Above you can see the 2 features + the target variable (Churn). What do you think which variables could contribute to the churn rate.

I think to me it would make sense if everything except ‘customerID’, ‘gender’, ‘partner’ and some of the services provided by the providers would be correlated with churn rate.

For, the EDA I am going to plot countplot using seaborn library for the ‘object’ datatype variables and for the numerical variables I am going to use catplot with the target(Churn) variable.

1. Churn:

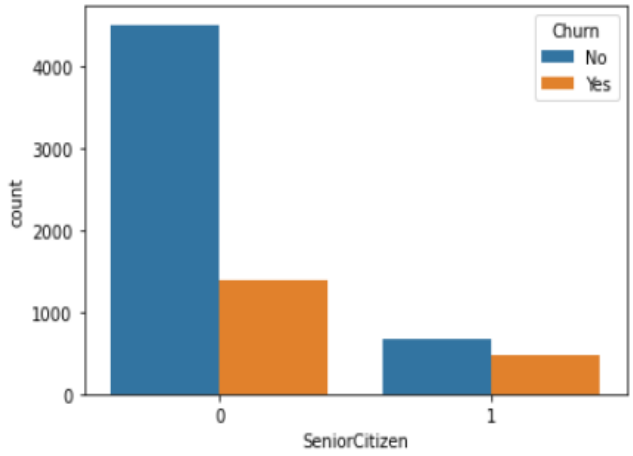
```
sns.countplot(df['Churn'])
plt.show()
```



We can see that the labels or classes of the target variable has a imbalanced distribution. Label ‘No’ of Churn has more than twice value count than ‘Yes’ Churn label.

2. SeniorCitizen and Churn:

```
sns.countplot(df['SeniorCitizen'],hue=df['Churn'])
plt.show()
```



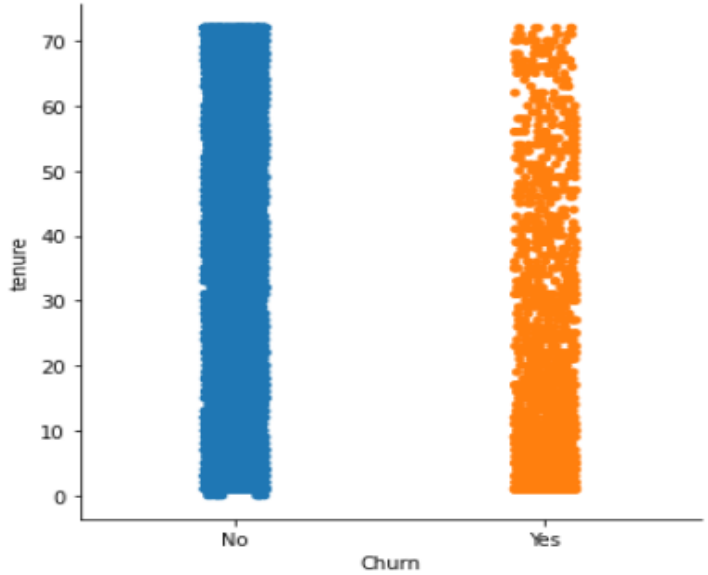
You can see that SeniorCitizen must be correlated with the target variable. Mostly customers are below age of 65.

But the thing is that senior citizens i.e. the customers aged above 65 has a higher rate of churn than the customers below age 65.

This is why the correlation should be good with the target variable.

3. Tenure and Churn:

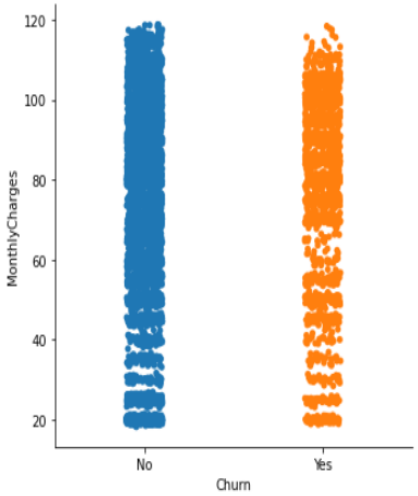
```
sns.catplot('Churn','tenure',data=df)
plt.show()
```



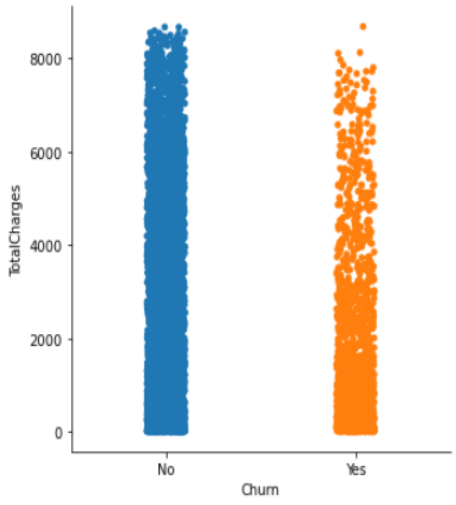
From the above catplot it can be concluded that while new customers i.e. of tenure below 20 months have higher rate of churn, old customers i.e. of tenure more than 40 months have reasonably lesser rate of churn.

4. MonthlyCharges , TotalCharges , Churn:

```
sns.catplot('Churn','MonthlyCharges',data=df)
plt.show()
```




```
sns.catplot('Churn','TotalCharges',data=df)
plt.show()
```



While higher monthly charges have higher churn rate, higher total charges have a lower churn rate.

For some lower ranges of monthly charges data for both the labels of target variable is distributed well to some extent.

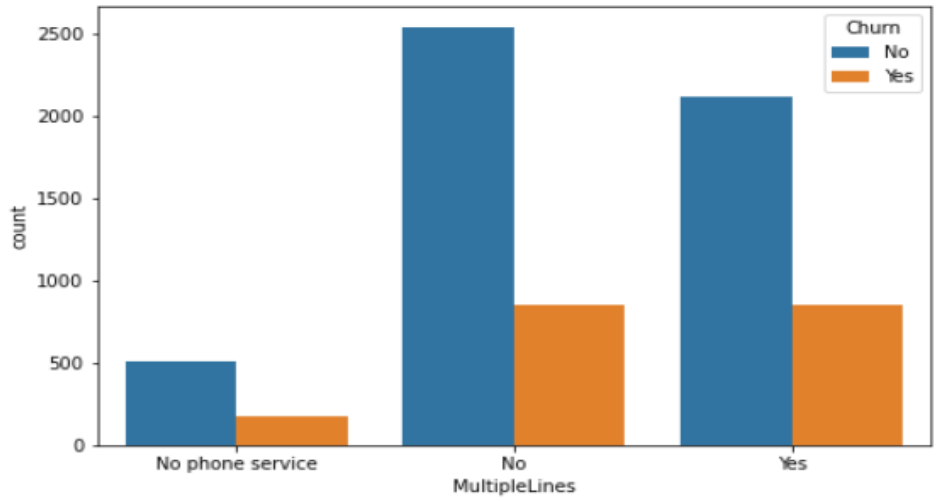
5. MultipleLines and Churn:

So, as I was going through the EDA I created a user defined function that plots countplot for object datatype variable with Churn as hue.

```
def count(i):
    plt.figure(figsize=(8,5))
    sns.countplot(df[i],hue=df['Churn'])
    plt.show()
```

So,

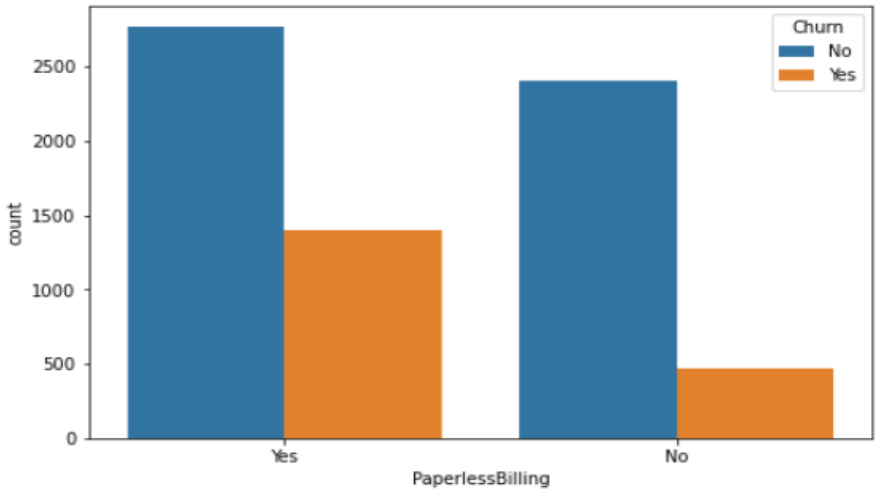
```
count('MultipleLines')
```



From this you can see that most customers do not have multiple telephone lines. Moreover for customers with multiple telephone lines churn rate is higher than customers with no multiple telephone lines.

6. PaperlessBilling and Churn:

```
count('PaperlessBilling')
```

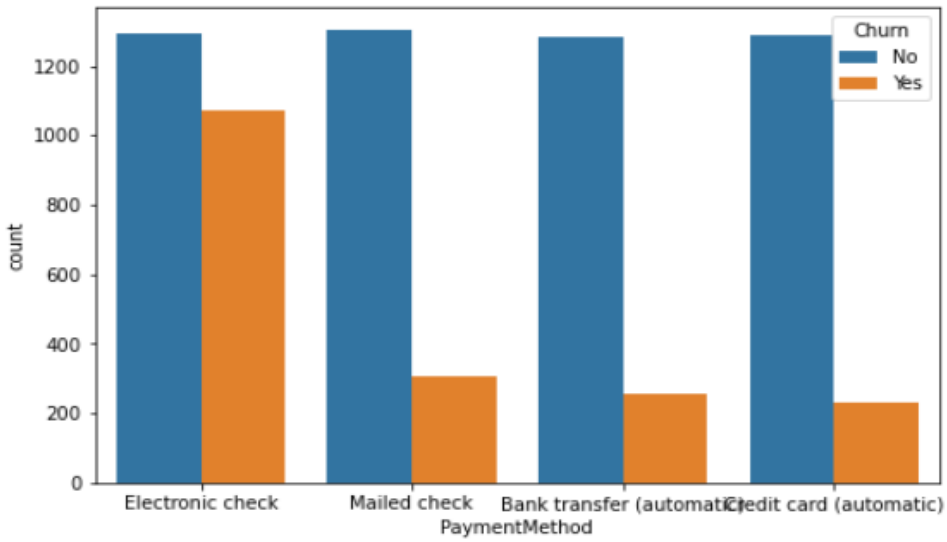


Both the classes of the above variable have reasonable count. Though more customers prefer paperless billing these customers have found to be with higher churn rate.

Churn rate for no paperless billing customers is exceptionally low.

7. PaymentMethods and Churn:

```
count('PaymentMethod')
```



From here you can see that the data is well distributed for the different classes of above variable.

Electronic check is what most customers used based on the dataset but it also has an exceptionally high churn rate. For all the other classes of this variable churn rate is low and almost similar.

This variable seems to be correlated with the target variable.

So, now as the EDA has been done before checking for correlation matrix and describing our dataset further conversion of categorical variables should be done.

We have already dealt with datatype conversion of one variable and have handled the missing values too, I am going to create a list of categorical

variables present in the dataset and I will drop ‘customerID’ from the list and later from the dataset too, because it shouldn’t contribute to a customer churn probability.

Then I am iterating the list using for loop and passing the categorical variables through LabelEncoder of sklearn,

```
#now lets LabelEncode the object datatype variables

cat=[]  #creating an empty list for categorical variables
for i in df.columns:
    if df[i].dtypes=='object':
        cat.append(i)
    else:
        pass


# now removing customerID, as this variable has all unique values so it would be tricky to encode it

cat.remove('customerID')

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in cat:
    df[i]=le.fit_transform(df[i])

df.sample()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtec
1666	8388-DMKAE	0	0	0	0	8	1	0	2	1	1	

which gives us:

```
#now checking for dtypes

df.dtypes
```

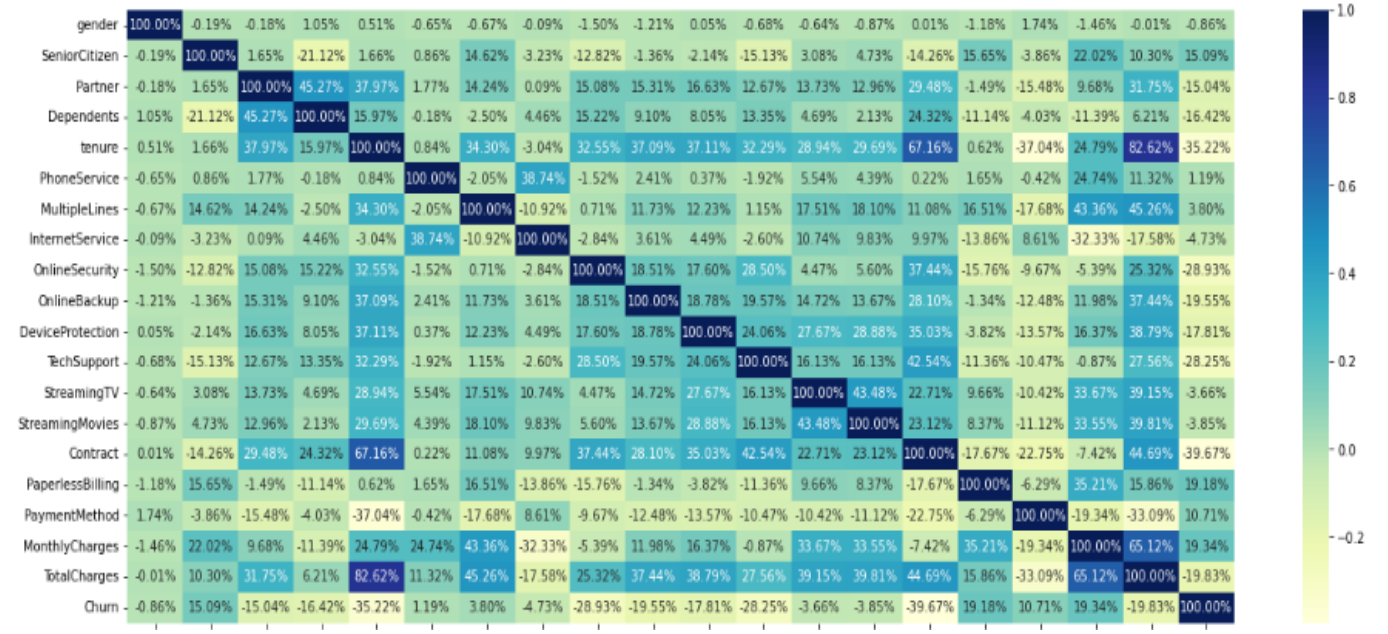
```
customerID           object
gender                int32
SeniorCitizen         int64
Partner               int32
Dependents            int32
tenure                int64
PhoneService          int32
MultipleLines         int32
InternetService       int32
OnlineSecurity        int32
OnlineBackup          int32
DeviceProtection      int32
TechSupport           int32
StreamingTV           int32
StreamingMovies       int32
Contract              int32
PaperlessBilling      int32
PaymentMethod         int32
MonthlyCharges      float64
TotalCharges        float64
Churn                 int32
dtype: object
```

Correlation matrix:

Correlation matrix with respect to the target variable:

```
plt.figure(figsize=(21,8))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu',fmt='.2%')
plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```



From the above matrix we can conclude that:

'MonthlyCharges', 'PaperlessBilling', 'SeniorCitizen' have fine correlation bond with the target variable 'Churn'.

While 'MultipleLines' and 'PhoneServices' have very weak correlation bond with tha target variable, all the other variables that are left have negative correlation bond with the target variable 'Churn'.

Now before selecting our feature and target variables , I am going to check for outliers using scipy zscore.

```
#lets check out for outliers using zscore

from scipy.stats import zscore

z=np.abs(zscore(df))

#threhold=3
np.where(z>3)
```

Now, I am going to check the data loss percent and if the data loss is lower than 10% I am going to use the dataset without outliers for further process.

```
#so outliers are present, so we are going to see how much data is being lost

df_new=df[(z<3).all(axis=1)]
```

```
df.shape
```

```
(7043, 20)
```

```
df_new.shape
```

```
(6361, 20)
```

```
print('Data Loss is:',(7043-6361)/7043*100,'%')
```

```
Data Loss is: 9.683373562402386 %
```

- so data loss is less than 10%, so we will use dataset with no outliers, as due to imbalance data we are going to do over sampling later

```
# now selecting our feature and target variable for further process

x=df_new.drop('Churn',axis=1)
y=df_new['Churn']

print(x.shape)
print(y.shape)
```

```
(6361, 19)
(6361,)
```

Now as we have selected the variables I am going to check for multicollinearity between the feature variables using variance inflation factor (VIF).

```
#checking for multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor

def vif_calc():
    vif=pd.DataFrame()
    vif['VIF Factor']=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    vif['features']=x.columns
    print(vif)
```

```
vif_calc()
```

```
    VIF Factor          features
0     1.001906            gender
1     1.152849     SeniorCitizen
2     1.468116           Partner
3     1.384518        Dependents
4    41.413310            tenure
5          NaN      PhoneService
6     1.409040     MultipleLines
7     1.698310   InternetService
8     1.340613    OnlineSecurity
9     1.217116      OnlineBackup
10    1.305294  DeviceProtection
11    1.403960       TechSupport
12    1.443696       StreamingTV
13    1.445687   StreamingMovies
14    2.511665          Contract
15    1.215684  PaperlessBilling
16    1.179855     PaymentMethod
17   11.565568    MonthlyCharges
18   57.624659      TotalCharges
```

Three variables have VIF greater than 10, which means multicollinearity is present among the feature variables. So I am going to drop the feature variable with high VIF and weak correlation with the target variable and again check for VIF.

```
#dropping TotalCharges

x.drop('TotalCharges',axis=1,inplace=True)

#now checking for VIF

vif_calc()
```

	VIF Factor	features
0	1.001843	gender
1	1.152802	SeniorCitizen
2	1.468110	Partner
3	1.384442	Dependents
4	2.621278	tenure
5	NaN	PhoneService
6	1.408747	MultipleLines
7	1.477634	InternetService
8	1.336128	OnlineSecurity
9	1.214003	OnlineBackup
10	1.301420	DeviceProtection
11	1.397659	TechSupport
12	1.441703	StreamingTV
13	1.445276	StreamingMovies
14	2.363900	Contract
15	1.215054	PaperlessBilling
16	1.177636	PaymentMethod
17	2.437063	MonthlyCharges

From this you can see that the VIF has been reduced to below five for all feature variables, thus, reduction in multicollinearity.

Now I am going to scale the data using StandardScaler for better results before passing it into the algorithms.

```
#scaling the input data

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

Now, as we have already seen that the target variable has imbalanced class, if we continue with that the dataset might give us poor recall. To handle this I am

going to do over sampling using SMOTE. SMOTE would create equal number of counts for both the classes of target variable.

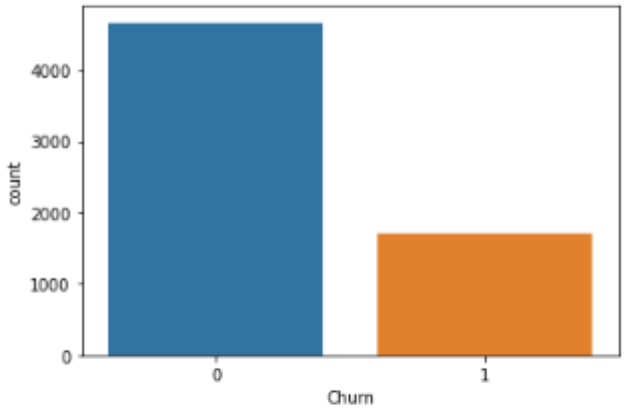
So it is giving us:

```
#now checking if the dataset is imbalanced or not

y.value_counts()
```

```
0    4662
1    1699
Name: Churn, dtype: int64
```

- data is imabalanced, so we will use SMOTE

```
sns.countplot(y)
plt.show()
```




```
from imblearn.over_sampling import SMOTE

train_x,train_y=SMOTE().fit_resample(x,y)

print(train_x.shape)
print(train_y.shape)
```

```
(9324, 18)
(9324,)
```

The imbalanced data has been treated now we can process with the algorithms.

Building Machine Learning Models:

Now we will train several Machine Learning models and compare their results. As the problem has a target variable with two labels or classes this is a binary classification problem.

We are going to compare accuracy score, errors and later cross validation and ROC curve and AUC to find the best model.

Logistic Regression:

```
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
lr.fit(x_train,y_train)
predtrain=lr.predict(x_train)
predlr=lr.predict(x_test)
```

K Nearest Neighbor:

```
knn=KNeighborsClassifier()

x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
knn.fit(x_train,y_train)
predtrain=knn.predict(x_train)
predknn=knn.predict(x_test)
```

Decision Tree:

```
dtc=DecisionTreeClassifier()
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
dtc.fit(x_train,y_train)
predtrain=dtc.predict(x_train)
preddtc=dtc.predict(x_test)
```

Support Vector:

```
svc=SVC()
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
svc.fit(x_train,y_train)
predtrain=svc.predict(x_train)
predsvc=svc.predict(x_test)
```

Random Forest:

```
rf=RandomForestClassifier()
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
rf.fit(x_train,y_train)
predtrain=rf.predict(x_train)
predrf=rf.predict(x_test)
```

Ada Boost:

```
ada=AdaBoostClassifier()
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
ada.fit(x_train,y_train)
predtrain=ada.predict(x_train)
predada=ada.predict(x_test)
```

Gradient Boosting:

```
gb=GradientBoostingClassifier()
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
gb.fit(x_train,y_train)
predtrain=gb.predict(x_train)
predgb=gb.predict(x_test)
```

Best Model:

```
from sklearn.metrics import accuracy_score as acs

algodf=pd.DataFrame({'Mean Squared Error':[mselr,mseknn,msedtc,msesvc,mseada,msegb,mserf],'Accuracy Score':[acs(y_test,predlr),ac
algodf
```

Giving us:

	Mean Squared Error	Accuracy Score
LogisticRegression	0.220872	0.779128
KNeighborsClassifier	0.220515	0.779485
DecisionTreeClassifier	0.215511	0.784489
SVC	0.189421	0.810579
AdaBoostClassifer	0.187991	0.812009
GradientBoostingClassifier	0.158327	0.841673
RabdomForestClassifier	0.140457	0.859543

As we can see, the Random Forest Classifier goes on the first place with highest accuracy score and lowest mean squared error among all the other trained models. But first, let us check, how random-forest performs, when we use cross validation.

So I am going to compute the score in a smaller range to see how the model is working.

```
for j in range(2,7):
    rfs=cross_val_score(rf,train_x,train_y,cv=j)
    rfc=rfs.mean()
    print('At cv=',j)
    print('Cross Validation Score is:',rfc*100)
    print('Accuracy score is:',accuracy_score(y_test,predrf)*100,'\n')
```

```
At cv= 2
Cross Validation Score is: 83.27970827970827
Accuracy score is: 85.9542530378842

At cv= 3
Cross Validation Score is: 84.42728442728443
Accuracy score is: 85.9542530378842

At cv= 4
Cross Validation Score is: 84.94208494208495
Accuracy score is: 85.9542530378842

At cv= 5
Cross Validation Score is: 84.96411764029041
Accuracy score is: 85.9542530378842

At cv= 6
Cross Validation Score is: 85.27456027456027
Accuracy score is: 85.9542530378842
```

As you can see the accuracy score of the model is coming out to be around the cross validation score with a tiny deviation the model is performing really well.

Since Random Forest is an easy to use model, we will try to increase its performance in the following section.

Hyperparameter Tuning:

Below you can see the code of the hyperparamter tuning for the parameters ‘criterion’ and ‘max features’ using Grid Search CV. As the code takes very long time to run, I have used two parameters only.

```
from sklearn.model_selection import GridSearchCV

rf=RandomForestClassifier()
parameters={'criterion':["gini", "entropy"],'max_features':['sqrt','log2','auto']}
clf=GridSearchCV(rf,parameters)
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=37,test_size=.30)
clf.fit(x_train,y_train)

print(clf.best_params_)
```

```
{'criterion': 'entropy', 'max_features': 'sqrt'}
```

As you can see the best parameter values has been founded. After finding the best suited parameters I put them through the algorithm and found a random state with a good accuracy score.

```
rf=RandomForestClassifier(criterion='entropy', max_features= 'sqrt')
x_train,x_test,y_train,y_test=train_test_split(train_x,train_y,random_state=57,test_size=.30)
rf.fit(x_train,y_train)
predtrain=rf.predict(x_train)
predrf=rf.predict(x_test)
```

Evaluation:

Now I am going to check for cross validation score with tuned parameters, just to see how the model is working. So, I iterated the CV for a small range, to find the best value of CV that is most close the accuracy score of the model,

```
rf=RandomForestClassifier(criterion='entropy', max_features= 'sqrt')
rfs=cross_val_score(rf,train_x,train_y,cv=17)
rfc=rfs.mean()
```

turns out:

```
print('The model is performing extremely well with RandomForestClassifier because we are getting both accuracy score and cross va
print('Accuracy:',round(accuracy_score(y_test,predrf)*100,1))
print('Cross Validation Score:',round(rfc*100,1))
```

```
The model is performing extremely well with RandomForestClassifier because we are getting both accuracy score and cross validat
ion score as 86.1

Accuracy: 86.1
Cross Validation Score: 86.1
```

We got both the scores similar up to one decimal place.

Now checking for the errors:

```
print('Mean Squared Error:',mean_squared_error(y_test,predrf))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test,predrf)))
```

```
Mean Squared Error: 0.13867047891350964
Root Mean Squared Error: 0.37238485322782616
```

As you can see the errors have been further reduced after hyperparameter tuning. The more the errors are close to zero the better the model is performing.

Further Evaluation:

Now that we have a proper model, we can start evaluating its performance in a more accurate way.

Confusion Matrix and Classification Report:

```
print(classification_report(y_test,predrf))
print(confusion_matrix(y_test,predrf))
```

```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      1439
           1       0.84      0.88      0.86      1359

    accuracy                           0.86      2798
   macro avg       0.86      0.86      0.86      2798
weighted avg       0.86      0.86      0.86      2798

[[1215  224]
 [ 164 1195]]
```

Confusion Matrix:

The first row is about the not-churn-predictions: 1215 customers were correctly classified as not churn (called true negatives) and 224 were wrongly classified as not churn (false positives).

The second row is about the churn-predictions: 164 customers were wrongly classified as churn (false negatives) and 1195 were correctly classified as survived (true positives).

Classification Report:

Precision and Recall and F-Score:

- The model predicts 86% of the time, a customer churns correctly (precision).
- The recall tells us that it predicted the churn of 86 % of the people who actually churn.
- For a binary classification clearly, the higher the f1-score the better, with 0 the worst possible and 1 being the best.

  Our model is giving f1-score of 0.86 which is certainly closer to 1.

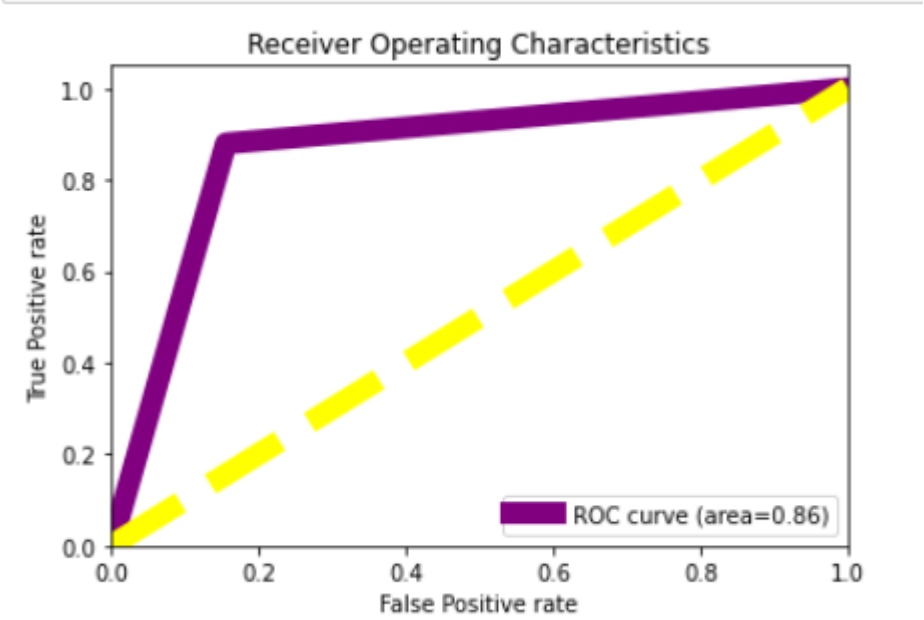
ROC Curve and AUC SCORE:

ROC AUC is used to evaluate and compare your binary classifier in another way and AUC Score is basically the corresponding score to area under curve (AUC).

As the ROC Curve is a plot for true positive rate (recall) against false positive rate I plotted the ROC Curve and placed AUC score in the figure only,

```
fpr,tpr,thresholds=roc_curve(y_test,predrf)

roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='purple',lw=10,label='ROC curve (area=%.2f)'%roc_auc)
plt.plot([0,1],[0,1],color='yellow',lw=10,linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive rate')
plt.ylabel('True Positive rate')
plt.title('Receiver Operating Characteristics')
plt.legend(loc='lower right')
plt.show()
```

which came up like:



So, the more area under the curve the better the classifier is as compared to the base classifier i.e. a completely random classifier with an AUC score of 0.50.

The AUC score for our classifier is coming out to be 0.8, so our Random Forest model seems to do a good job.

Saving the model:

I am going to use pickle dump to save the best fit model. Pickle module is used for serializing and de-serializing an object structure.

```
#saving the best model

import pickle

pickle.dump(rf,open('churn.pkl','wb'))
```

Summary:

We began with the importing of required libraries and loading the dataset. Afterwards, we got an understanding of the dataset by exploratory data analysis, checked about missing data and datatypes. During this process we used seaborn and matplotlib to do the visualizations which helped us in grasping the relation of important features with the target variable. During the data preprocessing part, we imputed the missing data, converted categorical features into numeric ones and converted the datatype where required. Then we proceeded with training 7 different machine learning models, picked one of them (Random Forest) and checked for cross validation. Then we optimized its performance by tuning its hyperparameter values. Afterwards, we came to the evaluation part and checked out its confusion matrix and computed the model's precision, recall and f1-score. Lastly, we compared our classifier with the base classifier by using ROC Curve and AUC score and saved the model.

There is still a room for improvement. The model's performance can be further optimized by doing a more extensive feature engineering i.e. comparing and plotting the features against each other and identifying and removing the noisy features. Another way is by doing a better hyperparameter tuning.