BLOG SUBMISSION

For

Customer Churn Analysis

Submitted By

HariharaSudhan

INTRODUCTION:

Churn Analysis is a major problem faced by companies these days. Churn is the nightmare for every business and is faced by almost all companies. Customer churn occurs when a company's customers cease to conduct business with it. Churn is closely monitored by businesses since maintaining an existing client is significantly less expensive than obtaining a new one. Working leads through a sales funnel and using marketing and sales expenditures to obtain additional clients are all part of starting a new firm. Existing customers have a larger volume of service usage and are more likely to refer new customers.

Good customer service and merchandise can help you keep your customers. However, the most efficient strategy for a corporation to prevent client attrition is to fully understand them. Churn prediction models can be built using the massive amounts of data collected about customers. Knowing who is most likely to defect allows a corporation to focus its marketing efforts on that segment of its client base.

In the telecommunications industry, preventing client turnover is vital, as the obstacles to entry are high.

Importing Libraries:

```
In [1]: #Importing Libraries

        import warnings
        warnings.filterwarnings("ignore")

        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline

        import seaborn as sns
        from plotly.subplots import make_subplots
        from sklearn import metrics

        from sklearn import preprocessing
        from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import MinMaxScaler
        from scipy.stats import zscore
        from sklearn.model_selection import train_test_split

        from sklearn.preprocessing import PowerTransformer
        from sklearn.model_selection import KFold
        from sklearn.feature_selection import RFE

        from sklearn.pipeline import Pipeline
        from sklearn.pipeline import make_pipeline

        from statsmodels.stats.outliers_influence import variance_inflation_factor
        from sklearn.metrics import silhouette_score
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import cross_val_score
```

Importing DataSet:

```
In [2]: data=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/DSData/master/Telecom_customer_churn.csv")
        data.head(10)
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupp
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	
5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No	...	Yes	
6	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	...	No	
7	6713-OKOMC	Female	0	No	No	10	No	No phone service	DSL	Yes	...	No	
8	7892-POOKP	Female	0	Yes	No	28	Yes	Yes	Fiber optic	No	...	Yes	
9	6388-TABGU	Male	0	No	Yes	62	Yes	No	DSL	Yes	...	No	

10 rows × 21 columns

Here we have imported the libraries and imported the dataset which was in .csv format in the Jupyter notebook.

This data set contains Independent and Dependent(target) variables.

<u>Independent Variable</u>: These are the known as Input variables. These are the input for a process that is being analyzed.

<u>Dependent Variable</u>: These are known as Output or Target variables. These are dependent on Indepenndent variables for their outcome.

After importing we will display a sample data. The variables in this dataset are as follows:

- customerID
- gender
- SeniorCitizen
- Partner
- Dependents
- tenure
- PhoneService
- MultipleLines
- InternetService
- OnlineSecurity
- OnlineBackup
- DeviceProtection
- TechSupport
- StreamningTV
- StreamingMovies
- Contract
- PaperlessBilling
- PaymentMethod
- MonthlyCharges
- TotalCharges
- Churn

EDA & Data Analysing

Shape of DataSet: 7043 * 21

So the dataset contains 7043 rows and 21 columns..

DATA INFORMATION:

```
<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
```

Data Types:

```
n [7]: data.dtypes

ut[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        object
       Churn               object
       dtype: object
```

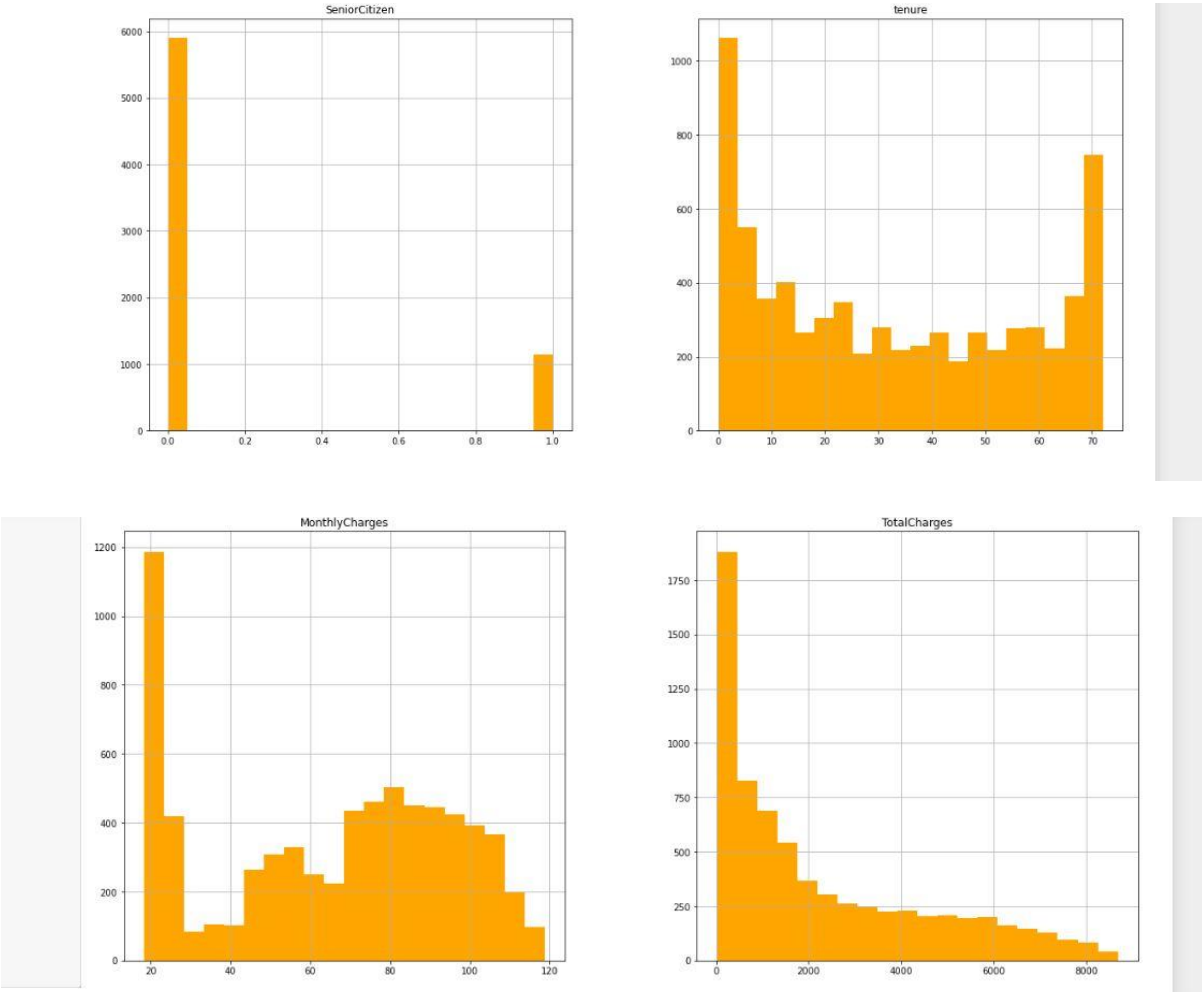
Null Values:

```
In [8]: data.isnull().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        0
        Churn               0
        dtype: int64

        There are no null values present in the dataset..
```

There are no null values in the dataset, as we can see. If the dataset contained null values. If there were any null values in the dataset, the mean, median, or mode would have been used to replace them.

Data Visualization and EDA Concluding Remarks:

The 'Churn' feature or variable is the Target feature or variable in the given data. This characteristic has only two distinct values, Y and N (Yes and No), implying that it has only two classifications. As there are only two distinct values, this is referred to as a 'Classification Problem.'



- Majority of the customer are Male. There is no much difference in gender count as well.
- Majority of the customers not Senior Citizens.
- Majority of the customers have partner and also there is not larger difference under this column data types.
- Majority of the customers doesn’t have dependents.

- Majority of the customers are with PhoneService.
- Majority of the customers has Paperless Billing.
- Majority of the customers has no Multiple Lines followed by having Multiple Lines and least are with no phone service.
- Majority of the customers has Fiber optic Internet Service followed by DSL and least are with no Internet Service.
- Majority of the customers doesn’t have Online Security followed by customers with online security and least are with no Internet Service.
- Majority of the customers doesn’t have/ use Online Backup service followed by customers with online Backup service and least are with no Internet Service.
- Majority of the customers doesn’t have/ use Device Protection service followed by customers with Device Protection service and least are with no Internet Service.
- Majority of the customers doesn’t have/ use Technical Support service followed by customers using Tech Support service and least are with no Internet Service.
- Majority of the customers doesn’t have/ use Streaming TV service followed by customers using Streaming TV service and least are with no Internet Service.
- Majority of the customers are under Month-to-month contract followed by two year contract and least are under one year contract.
- Majority of the customers use Electronic check as PaymentMethod followed by Mailed check and Bank transfer

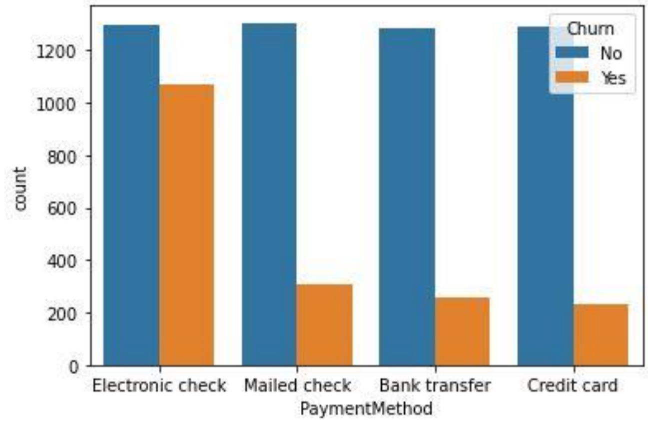
(automatic) payment methods and least are through Credit card (automatic).

The plot of every feature column against target variable 'Churn' is as follows:

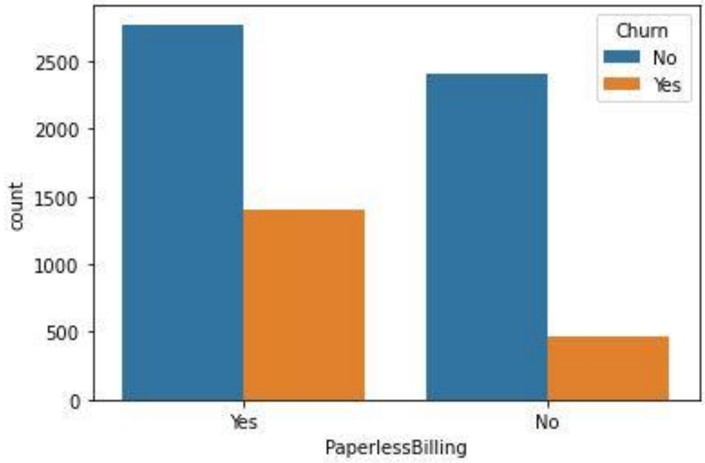
- PaperlessBilling - Customers who use paperless billing has higher Churn rate.
- gender - There is no much difference in the Churn rate with respect to Gender.
- Partner - Customers who doesn't have partner are having higher Churn rate.
- Dependents - Customers who doesnt have any dependents are having higher Churn rate.
- PhoneService - Customers who use Phone services has higher Churn rate.
- MultipleLines - Churn rate is almost same for the customers with and without Multiple Lines.
- InternetService - Customers who use Fiber optics based internet service has higher Churn rate.
- OnlineSecurity - Customers who doesn't have/ use online security service are having higher Churn rate.
- OnlineBackup - Customers who doesn't have/ use online backup service are having higher Churn rate.
- DeviceProtection - Customers who doesn't have/ use device protection service are having higher Churn rate.
- TechSupport - Customers who doesn't have/ use Tech Support service are having higher Churn rate.
- StreamingTV - Customers who doesn't have/ use streaming TV service are having higher Churn rate.
- StreamingMovies - Customers who doesn't have/ use streaming movies service are having higher Churn rate.
- Contract - Customers who are under the contract of Month to month are having higher Churn rate.

PaperlessBilling

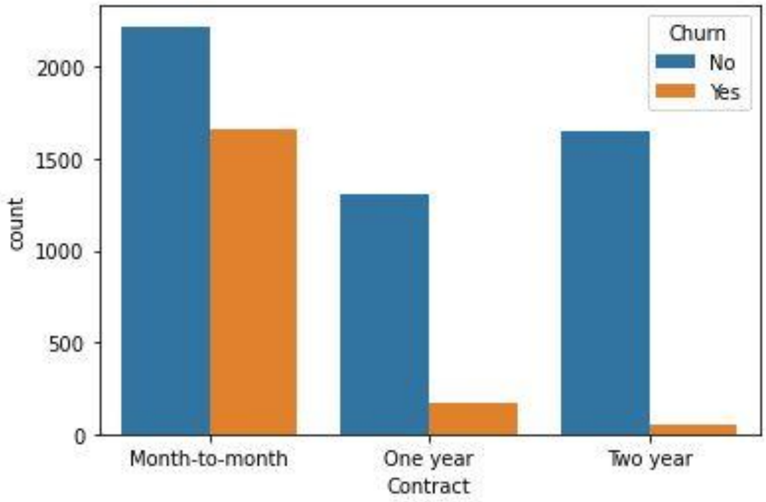
In [32]:
```
sns.countplot(x = 'PaymentMethod', hue = 'Churn', data = data)
plt.show()
```



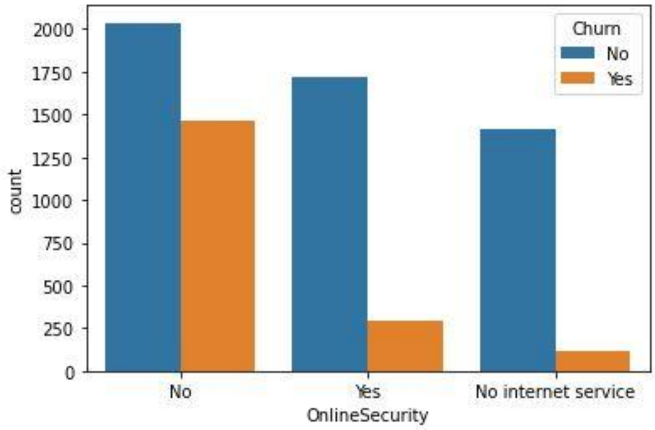
[31]:
```
sns.countplot(x = 'PaperlessBilling', hue = 'Churn', data = data)
plt.show()
```



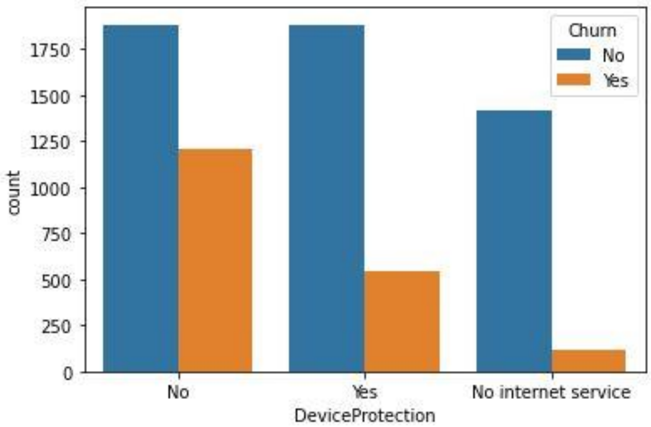
In [30]:
```
sns.countplot(x = 'Contract', hue = 'Churn', data = data)
plt.show()
```



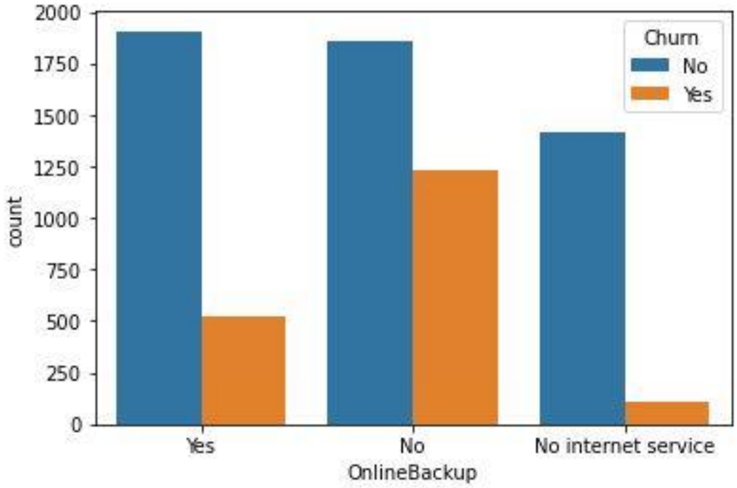
In [29]:
```
sns.countplot(x = 'OnlineSecurity', hue = 'Churn', data = data)
plt.show()
```



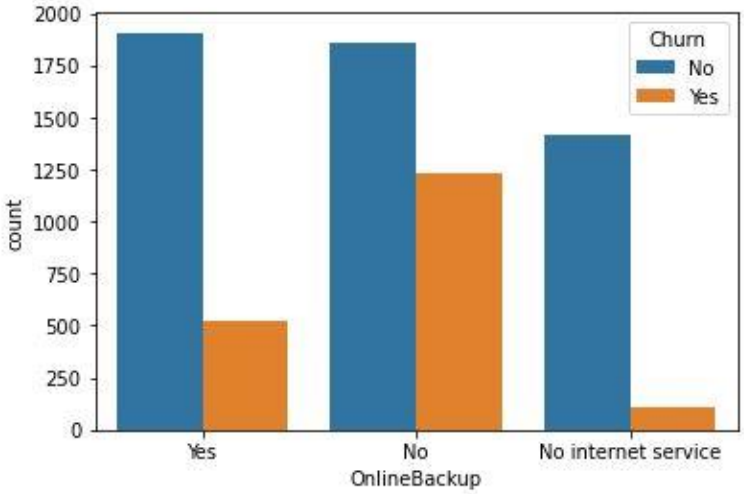
In [28]:
```
sns.countplot(x = 'DeviceProtection', hue = 'Churn', data = data)
plt.show()
```



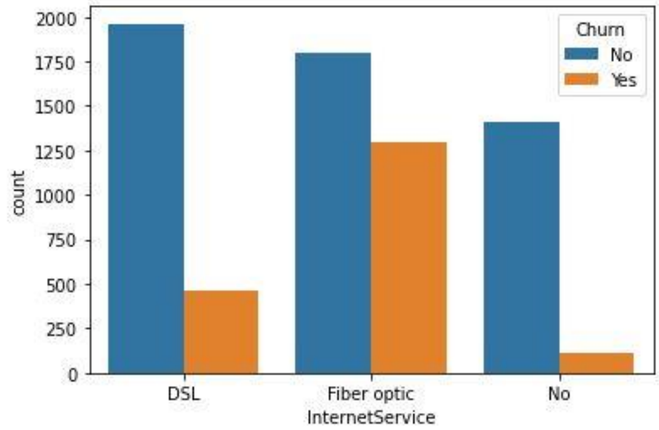
```
In [27]: sns.countplot(x = 'OnlineBackup', hue = 'Churn', data = data)
         plt.show()
```



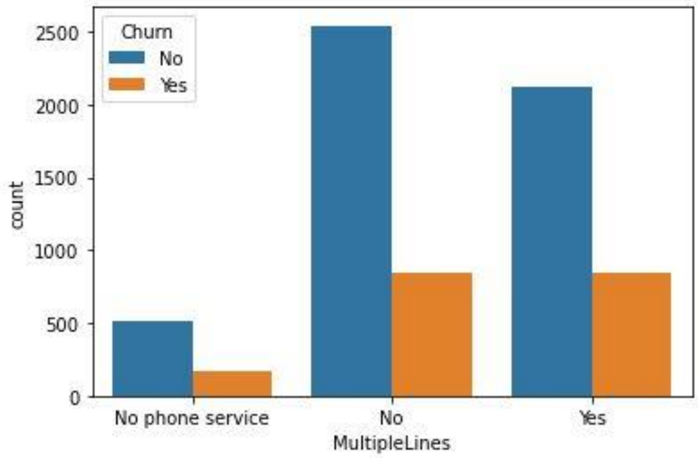

```
In [27]: sns.countplot(x = 'OnlineBackup', hue = 'Churn', data = data)
         plt.show()
```



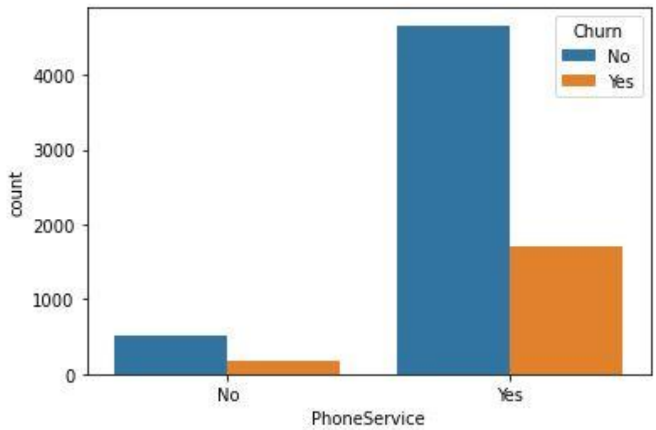
In [26]:
```
sns.countplot(x = 'InternetService', hue = 'Churn', data = data)
plt.show()
```



In [25]:
```
sns.countplot(x = 'MultipleLines', hue = 'Churn', data = data)
plt.show()
```

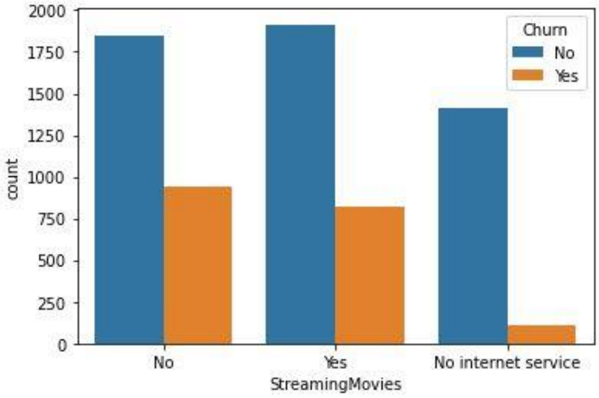


In [24]:
```
sns.countplot(x = 'PhoneService', hue = 'Churn', data = data)
plt.show()
```

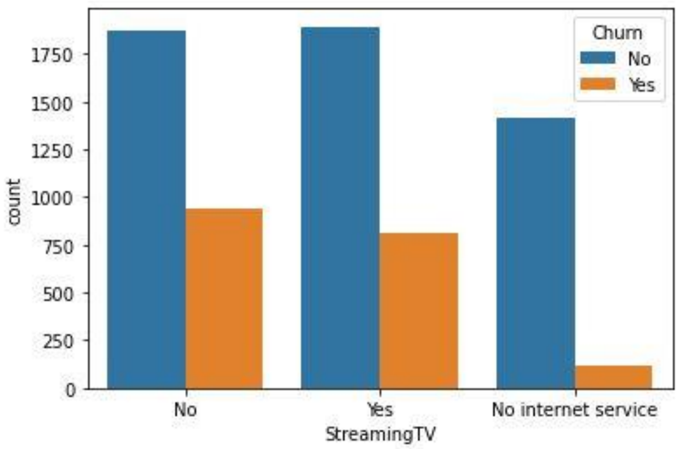


StreamingTV

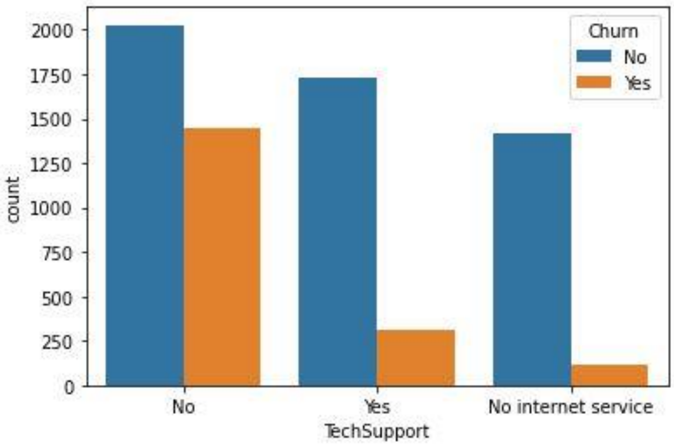
In [23]:
```
sns.countplot(x = 'StreamingMovies', hue = 'Churn', data = data)
plt.show()
```



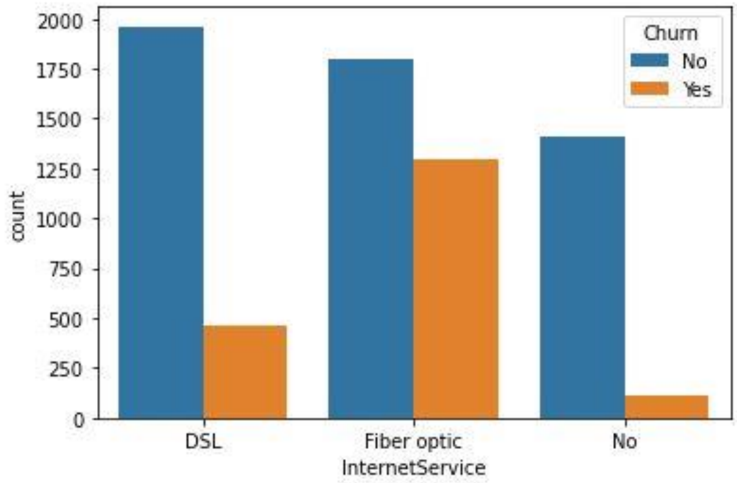
In [22]:
```
sns.countplot(x = 'StreamingTV', hue = 'Churn', data = data)
plt.show()
```



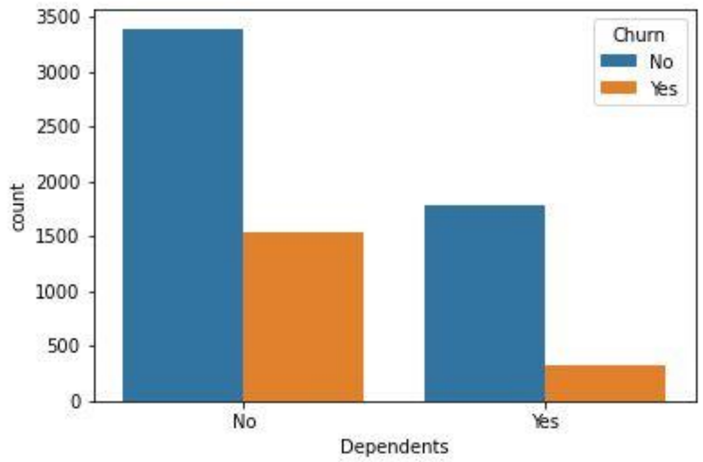
In [21]:
```
sns.countplot(x = 'TechSupport', hue = 'Churn', data = data)
plt.show()
```



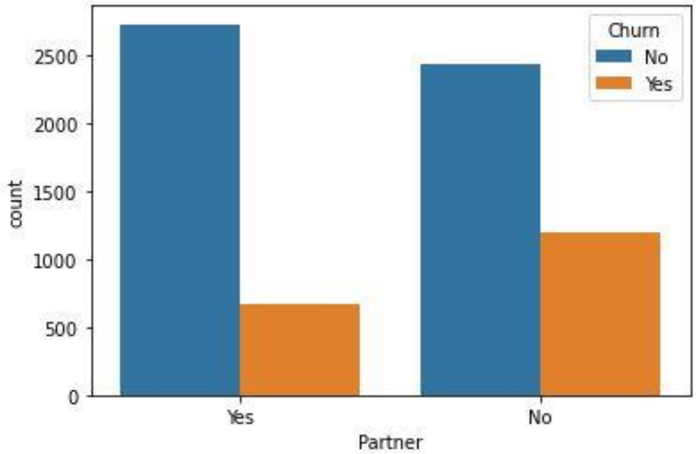
In [20]:
```
# Visualize the churn count for internet service
sns.countplot(x = 'InternetService', hue = 'Churn', data = data)
plt.show()
```



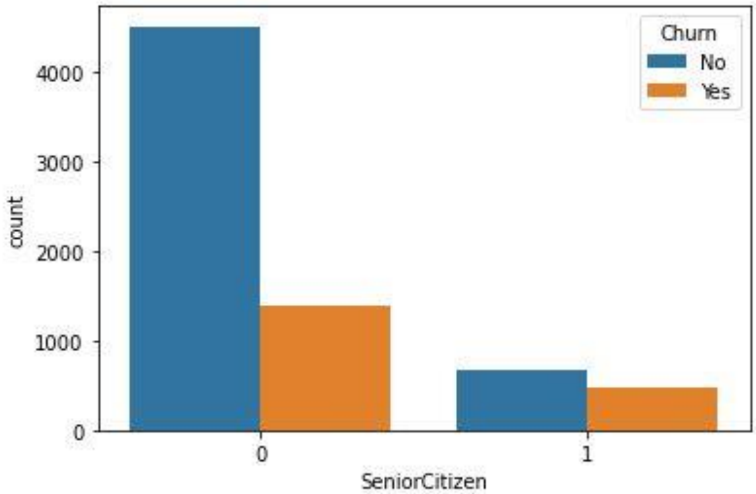
In [19]:
```
sns.countplot(x = 'Dependents', hue = 'Churn', data = data)
plt.show()
```



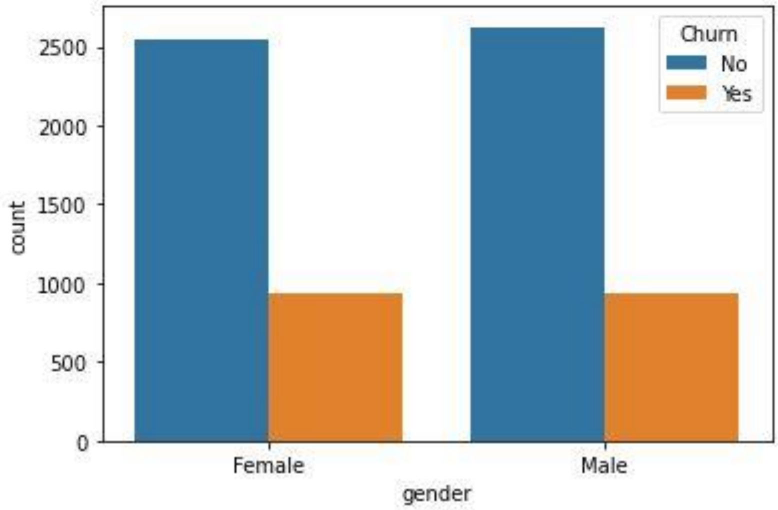
In [18]:
```
sns.countplot(x = 'Partner', hue = 'Churn', data = data)
plt.show()
```



In [17]:
```
sns.countplot(x = 'SeniorCitizen', hue = 'Churn', data = data)
plt.show()
```



In [16]:
```
# Visualize the churn count for both male and females
sns.countplot(x = 'gender', hue = 'Churn', data = data)
plt.show()
```



- Customers who does payment through Electronic check has higher attrition rate / Churn rate and rest 3 payment modes customers have almost same Churn rate.

<u>Pre-processing Pipeline:</u>

There are also object-type variables, as we can see. They contain strings that the machine learning model won't be able to recognise because it doesn't understand string data types. It is only capable of recognising numerical data.

As a result, we'll convert it to numerical data using Label Encoding. The target variable 'Churn' has two distinct values, Y and N, which will be changed to 0 and 1 after encoding. Similarly, if there were three distinct values, they were changed to 0, 1, and 2.

```
In [33]: from sklearn.preprocessing import LabelEncoder
```

```
In [34]: data.columns.to_series().groupby(data.dtypes).groups
```

```
Out[34]: {int64: ['SeniorCitizen', 'tenure'], float64: ['MonthlyCharges', 'TotalCharges'], object: ['customerID', 'gender', 'Partner',
'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSu
pport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn']}
```

```
In [35]: lab=LabelEncoder()
lab_count=0
for col in data.columns[1:]:
    if data[col].dtype == 'object':
        if len(list(data[col].unique())) <= 2:
            lab.fit(data[col])
            data[col] = lab.transform(data[col])
            lab_count += 1
            print(col)
print('{} columns were label encoded.'.format(lab_count))
```

```
gender
Partner
Dependents
PhoneService
PaperlessBilling
Churn
6 columns were label encoded.
```

Checking the correlation between Independent and Target Variable:

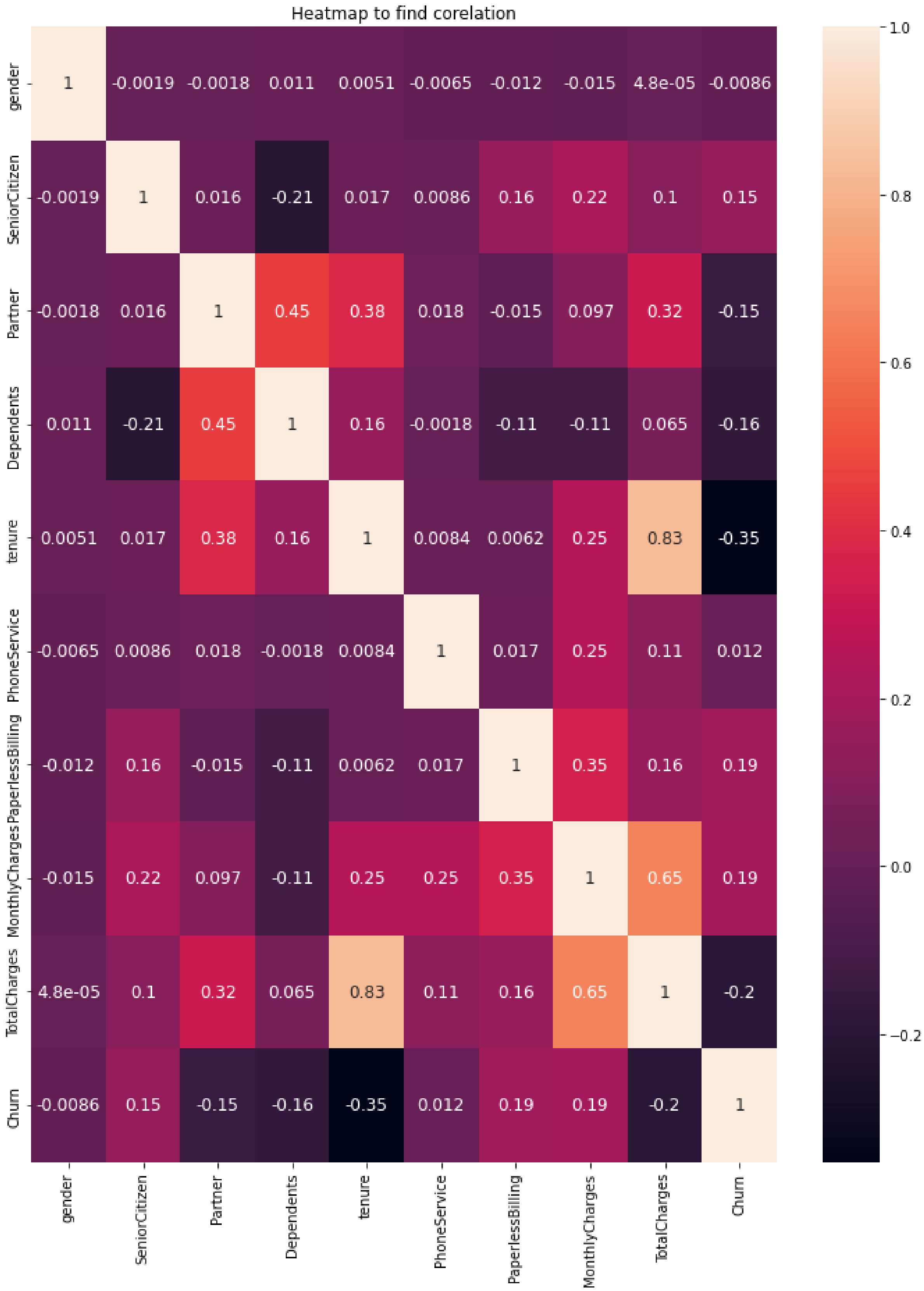
The heat map of correlation between all independent variable and target column churn is shown below.

```
In [36]: data.corr()
```

Out[36]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn
gender	1.000000	-0.001874	-0.001808	0.010517	0.005106	-0.006488	-0.011754	-0.014569	0.000048	-0.008612
SeniorCitizen	-0.001874	1.000000	0.016479	-0.211185	0.016567	0.008576	0.156530	0.220173	0.102411	0.150889
Partner	-0.001808	0.016479	1.000000	0.452676	0.379697	0.017706	-0.014877	0.096848	0.319072	-0.150448
Dependents	0.010517	-0.211185	0.452676	1.000000	0.159712	-0.001762	-0.111377	-0.113890	0.064653	-0.164221
tenure	0.005106	0.016567	0.379697	0.159712	1.000000	0.008448	0.006152	0.247900	0.825880	-0.352229
PhoneService	-0.006488	0.008576	0.017706	-0.001762	0.008448	1.000000	0.016505	0.247398	0.113008	0.011942
PaperlessBilling	-0.011754	0.156530	-0.014877	-0.111377	0.006152	0.016505	1.000000	0.352150	0.157830	0.191825
MonthlyCharges	-0.014569	0.220173	0.096848	-0.113890	0.247900	0.247398	0.352150	1.000000	0.651065	0.193356
TotalCharges	0.000048	0.102411	0.319072	0.064653	0.825880	0.113008	0.157830	0.651065	1.000000	-0.199484
Churn	-0.008612	0.150889	-0.150448	-0.164221	-0.352229	0.011942	0.191825	0.193356	-0.199484	1.000000

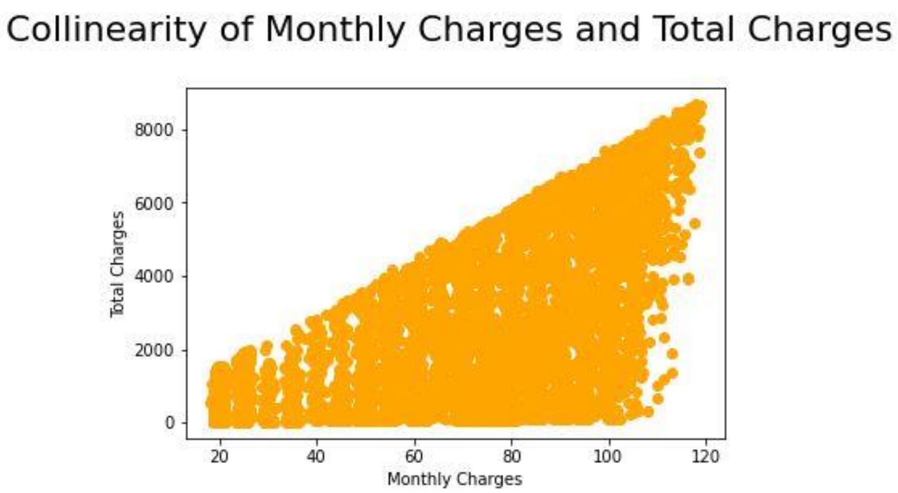
Finding Corelation


Heatmap to find corelation
gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
PaperlessBilling
MonthlyCharges
TotalCharges
Churn
1 -0.0019 -0.0018 0.011 0.0051 -0.0065 -0.012 -0.015 4.8e-05 -0.0086
-0.0019 1 0.016 -0.21 0.017 0.0086 0.16 0.22 0.1 0.15
-0.0018 0.016 1 0.45 0.38 0.018 -0.015 0.097 0.32 -0.15
0.011 -0.21 0.45 1 0.16 -0.0018 -0.11 -0.11 0.065 -0.16
0.0051 0.017 0.38 0.16 1 0.0084 0.0062 0.25 0.83 -0.35
-0.0065 0.0086 0.018 -0.0018 0.0084 1 0.017 0.25 0.11 0.012
-0.012 0.16 -0.015 -0.11 0.0062 0.017 1 0.35 0.16 0.19
-0.015 0.22 0.097 -0.11 0.25 0.25 0.35 1 0.65 0.19
4.8e-05 0.1 0.32 0.065 0.83 0.11 0.16 0.65 1 -0.2
-0.0086 0.15 -0.15 -0.16 -0.35 0.012 0.19 0.19 -0.2 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

We can see that the datasets are linearly related, thus we don't need to remove any columns. We must remove any strongly linked columns from the dataset since they alter the dataset and bias the model towards it.

Now that we've done the preprocessing, we can move on to data modelling and prediction.

Collinearity



Reducing VIF

Out[41]:

	variables	VIF
0	gender	1.878863
1	SeniorCitizen	1.323160
2	Partner	2.812757
3	Dependents	1.904657
4	tenure	3.299933
5	PhoneService	5.967552
6	PaperlessBilling	2.748477
7	MonthlyCharges	7.465415

Standard Scaler:

In [50]:
```python
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X_scaled
```

Out[50]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	MultipleLines_No	MultipleLines_No phone service	...	Strea
0	-1.009430	-0.440327	1.035617	-0.652305	-1.280248	-3.056334	0.828939	-1.161694	-0.963411	3.056334	...	
1	0.990658	-0.440327	-0.965608	-0.652305	0.064303	0.327189	-1.206361	-0.260878	1.037979	-0.327189	...	
2	0.990658	-0.440327	-0.965608	-0.652305	-1.239504	0.327189	0.828939	-0.363923	1.037979	-0.327189	...	
3	0.990658	-0.440327	-0.965608	-0.652305	0.512486	-3.056334	-1.206361	-0.747850	-0.963411	3.056334	...	
4	-1.009430	-0.440327	-0.965608	-0.652305	-1.239504	0.327189	0.828939	0.196178	1.037979	-0.327189	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
7027	0.990658	-0.440327	1.035617	1.533025	-0.343137	0.327189	0.828939	0.664868	-0.963411	-0.327189	...	
7028	-1.009430	-0.440327	1.035617	1.533025	1.612573	0.327189	0.828939	1.276493	-0.963411	-0.327189	...	
7029	-1.009430	-0.440327	1.035617	1.533025	-0.872808	-3.056334	0.828939	-1.170004	-0.963411	3.056334	...	
7030	0.990658	2.271039	1.035617	-0.652305	-1.158016	0.327189	0.828939	0.319168	-0.963411	-0.327189	...	
7031	0.990658	-0.440327	-0.965608	-0.652305	1.368109	0.327189	0.828939	1.357932	1.037979	-0.327189	...	

To remove the mean and to scale the features to be all in unit variance we have used StandardScaler to transform the data as shown in above fig.

Building Machine Learning Models:

We'll now divide the data into training and testing datasets and determine which random state is the best.

In [52]:
```python
from sklearn.linear_model import LogisticRegression
max_accu = 0
max_rs = 0
for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.25, random_state = i)
    LR = LogisticRegression()
    LR.fit(x_train,y_train)
    pred = LR.predict(x_test)
    acc = accuracy_score(y_test,pred)
    if acc > max_accu:
        max_accu = acc
        max_rs = i
print("Best accuracy is",max_accu,"on Random State",max_rs)
```

```
Best accuracy is 0.8253697383390216 on Random State 99
```

We found that 99 is the greatest random state for this, so we'll use that for our model.We'll now apply a Machine Learning model to learn from the training dataset and forecast the testing set.Because the target variable 'Churn' contains categorical data, it is a categorical problem.To anticipate the data, we'll apply five different Machine Learning models and pick the best one.

- LogisticRegression
- KNeighbors Classifier
- DecisionTree Classifier
- RandomForest Classifier
- Ada Boost Classifier

Logisticregression:

```
In [55]: LR.fit(x_train,y_train)
         LR_pred=LR.predict(x_test)

         print(accuracy_score(y_test,LR_pred))
         print(confusion_matrix(y_test,LR_pred))
         print(classification_report(y_test,LR_pred))
         print("Training accuracy::",LR.score(x_train,y_train))
         print("Test accuracy::",LR.score(x_test,y_test))

0.8253697383390216
[[1200  109]
 [ 198  251]]
              precision    recall  f1-score   support

           0       0.86      0.92      0.89      1309
           1       0.70      0.56      0.62       449

    accuracy                           0.83      1758
   macro avg       0.78      0.74      0.75      1758
weighted avg       0.82      0.83      0.82      1758

Training accuracy:: 0.798445202882063
Test accuracy:: 0.8253697383390216

In [56]: print(cross_val_score(LR,X,Y,cv=5).mean())

0.800907674591885
```

KNeighborsClassifier:

In [57]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [58]:
```
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
pred_knn=knn.predict(x_test)
print(accuracy_score(y_test,pred_knn))
print(confusion_matrix(y_test,pred_knn))
print(classification_report(y_test,pred_knn))
print("Training accuracy::",knn.score(x_train,y_train))
print("Test accuracy::",knn.score(x_test,y_test))
```

```
0.78839590443686
[[1151  158]
 [ 214  235]]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1309
           1       0.60      0.52      0.56       449

    accuracy                           0.79      1758
   macro avg       0.72      0.70      0.71      1758
weighted avg       0.78      0.79      0.78      1758

Training accuracy:: 0.8392112248767539
Test accuracy:: 0.78839590443686
```

In [59]:
```
print(cross_val_score(knn,X,Y,cv=5).mean())
```

```
0.7713292913607133
```

DecoisionTreeClassifier:

In [61]:
```
DT=DecisionTreeClassifier()
DT.fit(x_train,y_train)
pred_DT=DT.predict(x_test)
print(accuracy_score(y_test,pred_DT))
print(confusion_matrix(y_test,pred_DT))
print(classification_report(y_test,pred_DT))
print("Training accuracy::",DT.score(x_train,y_train))
print("Test accuracy::",DT.score(x_test,y_test))
```

```
0.7622298065984073
[[1107  202]
 [ 216  233]]
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      1309
           1       0.54      0.52      0.53       449

    accuracy                           0.76      1758
   macro avg       0.69      0.68      0.68      1758
weighted avg       0.76      0.76      0.76      1758

Training accuracy:: 0.997155858930603
Test accuracy:: 0.7622298065984073
```

In [62]:
```
print(cross_val_score(DT,X,Y,cv=5).mean())
```

```
0.7191408331235511
```

RandomForest Classifier:

```
In [64]: RF=RandomForestClassifier()
         RF.fit(x_train,y_train)
         pred_RF=RF.predict(x_test)
         print(accuracy_score(y_test,pred_RF))
         print(confusion_matrix(y_test,pred_RF))
         print(classification_report(y_test,pred_RF))
         print("Training accuracy::",RF.score(x_train,y_train))
         print("Test accuracy::",RF.score(x_test,y_test))

         0.7946530147895335
         [[1185  124]
          [ 237  212]]
                       precision    recall  f1-score   support

                    0       0.83      0.91      0.87      1309
                    1       0.63      0.47      0.54       449

             accuracy                           0.79      1758
            macro avg       0.73      0.69      0.70      1758
         weighted avg       0.78      0.79      0.78      1758

         Training accuracy:: 0.997155858930603
         Test accuracy:: 0.7946530147895335

In [65]: print(cross_val_score(RF,X,Y,cv=5).mean())

         0.7838425228056022
```

AdaBoostClassifier

```
In [67]: ADA=AdaBoostClassifier()
         ADA.fit(x_train,y_train)
         pred_ADA=ADA.predict(x_test)
         print(accuracy_score(y_test,pred_ADA))
         print(confusion_matrix(y_test,pred_ADA))
         print(classification_report(y_test,pred_ADA))
         print("Training accuracy::",ADA.score(x_train,y_train))
         print("Test accuracy::",ADA.score(x_test,y_test))

         0.8242320819112628
         [[1207  102]
          [ 207  242]]
                       precision    recall  f1-score   support

                    0       0.85      0.92      0.89      1309
                    1       0.70      0.54      0.61       449

             accuracy                           0.82      1758
            macro avg       0.78      0.73      0.75      1758
         weighted avg       0.82      0.82      0.82      1758

         Training accuracy:: 0.7986348122866894
         Test accuracy:: 0.8242320819112628

In [68]: print(cross_val_score(ADA,X,Y,cv=5).mean())

         0.8020453513776372
```

GridSearchCV:

Random Forest GCV:

```
In [82]: RF = RandomForestClassifier(random_state=0)

         param_dist = {"n_estimators": [1600,1700,1900,2000],
                      'max_depth': [None,2],
                      "min_samples_leaf": [1,2],
                      "min_samples_split": [2,3], #0.5
                      "criterion": ["gini"]}

         #RF_rs = RandomizedSearchCV(RF_cls, param_distributions=param_dist, cv=5, n_iter =25,verbose=True,random_state=0,scoring='recall
         RF_gs2 = GridSearchCV(estimator=RF,param_grid=param_dist,scoring='roc_auc', verbose=True, n_jobs=-1).fit(x_train,y_train)


         print('Best params achieve a train score of', abs(RF_gs2.best_score_),'with the params:')
         RF_gs2.best_params_

         Fitting 5 folds for each of 32 candidates, totalling 160 fits
         Best params achieve a train score of 0.8288863970285574 with the params:
```

AdaBoostGCV:

```
In [81]: param_dist = {'n_estimators': range(400,600,25),
                       'learning_rate': [0.01],
                       'algorithm':['SAMME.R']}

         ADA = AdaBoostClassifier(DT,random_state=0)

         ADA_cv = GridSearchCV(ADA, param_dist, cv=5, n_jobs=-1,verbose=1).fit(x_train,y_train)


         # Print the tuned parameters and score
         print("Tuned Decision Tree Parameters: {}".format(ADA_cv.best_params_))
         print("Best score is {}".format(ADA_cv.best_score_))

         Fitting 5 folds for each of 8 candidates, totalling 40 fits
         Tuned Decision Tree Parameters: {'algorithm': 'SAMME.R', 'learning_rate': 0.01, 'n_estimators': 450}
         Best score is 0.776826353228954
```

DecisionTreeGCV:

```
In [78]: DT = DecisionTreeClassifier(random_state=0)

         param_dist = {
                       'max_depth': [None,2,4,6],
                       "min_samples_leaf": [2,4,6],
                       "min_samples_split": [6,7,8,9], #0.5
                       "criterion": ["gini"],
                       "max_features":[None]}

         DT_rs = GridSearchCV(DT,param_dist,scoring='roc_auc', verbose=1,n_jobs=-1).fit(x_train,y_train)

         print('Best params achieve a train score of', abs(DT_rs.best_score_),'with the params:')
         DT_rs.best_params_

         Fitting 5 folds for each of 48 candidates, totalling 240 fits
         Best params achieve a train score of 0.8255361196776224 with the params:

Out[78]: {'criterion': 'gini',
          'max_depth': 4,
          'max_features': None,
          'min_samples_leaf': 4,
          'min_samples_split': 6}
```

KneighborsGCV:

```
In [75]: param_dist = {'n_neighbors': range(1,10), 'weights': ['uniform','distance'], 'algorithm':['auto', 'ball_tree', 'kd_tree', 'brut

         knn = KNeighborsClassifier().fit(x_train,y_train)


         knn_cv = GridSearchCV(knn, param_dist,n_jobs=-1, verbose=1).fit(x_train,y_train)


         # Print the tuned parameters and score
         print("Tuned Decision Tree Parameters: {}".format(knn_cv.best_params_))
         print("Best score is {}".format(knn_cv.best_score_))

         Fitting 5 folds for each of 144 candidates, totalling 720 fits
         Tuned Decision Tree Parameters: {'algorithm': 'auto', 'n_neighbors': 8, 'p': 1, 'weights': 'uniform'}
         Best score is 0.7787255051844925
```

LogisticGCV:

```
In [74]: lg = LogisticRegression(random_state=0)

         param_grid = {'solver':['liblinear'],
                       'penalty': ['l1'],
                      'C': np.linspace(2,1,50)}

         lg_cv = GridSearchCV(lg, param_grid,scoring="roc_auc", verbose=1).fit(x_train,y_train)


         print("Tuned Decision Logistic Regression: {}".format(lg_cv.best_params_))
         print("Best score is {}".format(lg_cv.best_score_))

         Fitting 5 folds for each of 50 candidates, totalling 250 fits
         Tuned Decision Logistic Regression: {'C': 1.4285714285714286, 'penalty': 'l1', 'solver': 'liblinear'}
         Best score is 0.8363470707513564
```

Machine Learning Model	**Accuracy**	**Cross_Validation score**	**HyperParameter Tuning(GridCV)**
LogisticRegression	0.82	0.80	0.83
KNeighbors Classifier	0.78	0.77	0.77
DecisionTreeClassifier	0.76	0.79	0.82
RandomForestClassifier	0.79	0.78	0.82
AdaBoostClassifier	0.82	0.80	0.77

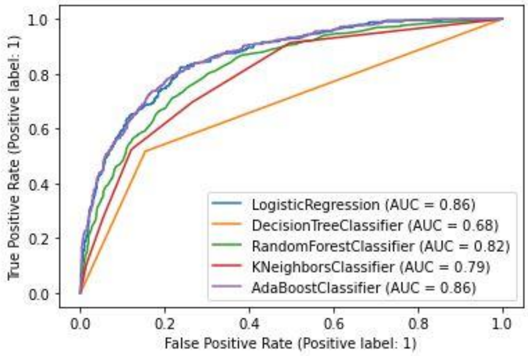
We can see from the Cross Val Score and accuracy score that the RandomForestClassifier has the least difference and has a high accuracy, hence we will choose the RandomForestClassifier model.

ROC-AUC Score

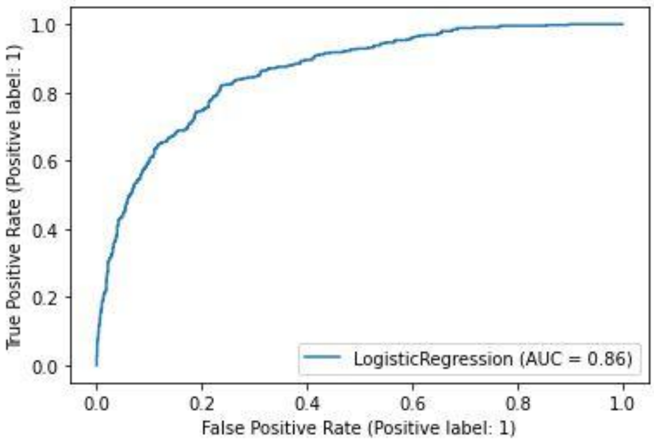
```
In [89]: model = LogisticRegression( solver = 'liblinear', penalty='l1')

         model.fit(x_train,y_train)
         pred = model.predict(x_test)

         print(f"Accuracy Score: {accuracy_score(y_test,pred)*100}%")
         print("--------------------------------------------------")

         print(f"roc_auc_score: {roc_auc_score(y_test,LR_pred)*100}%")
         print("--------------------------------------------------")

         print(f"Confusion Matrix : \n {confusion_matrix(y_test,pred)}\n")
         print("--------------------------------------------------")

         print(f"CLASSIFICATION REPORT : \n {classification_report(y_test,pred)}")
         print("--------------------------------------------------")

         Accuracy Score: 82.53697383390217%
         --------------------------------------------------
         roc_auc_score: 73.78751865192321%
         --------------------------------------------------
         Confusion Matrix :
          [[1200  109]
          [ 198  251]]
```

ROC-Curve:

```
In [69]: #Lets plot roc curve and check auc and performance of all algorithms
         from sklearn.metrics import plot_roc_curve
         disp = plot_roc_curve(LR, x_test, y_test)
         plot_roc_curve(DT, x_test, y_test, ax = disp.ax_)
         plot_roc_curve(RF, x_test, y_test, ax = disp.ax_)
         plot_roc_curve(knn, x_test, y_test, ax = disp.ax_)
         plot_roc_curve(ADA, x_test, y_test, ax = disp.ax_)

         plt.legend(prop={"size" :10} ,loc = 'lower right')
         plt.show()
```



```
In [90]: plot_roc_curve(model, x_test, y_test)
         plt.show()
```



As we can see here that the accuracy score of the model has been increased.

After this we will save the model.

```
In [92]: import joblib
         joblib.dump(model,"Customer_CHURN.pkl")

Out[92]: ['Customer_CHURN.pkl']
```

Concluding Remarks:

In this type of situation, The most crucial step is pre-processing and data cleaning. We must properly manage both categorical and numerical data, as well as check by generating multiple ML models on the same dataset. We must examine each model's accuracy and cross-val score and select the one that has the best of both.

The results show an accuracy of 82 percent, indicating that our algorithm correctly forecasts client retention 82% of the time. Customer churn forecast is critical to a company's long-term financial survival. This concludes our procedure. With an accuracy of 82 percent, we have successfully trained our model to predict customer data from Sample Data Sets with the goal of constructing and evaluating different customer churn prediction models.

Conclusion

```
In [91]: print("Logistic Regression Classifier: {:.2f}% Accuracy".format( 100 * accuracy_score(LR_pred, y_test)))
         print("Random Forest Classifier: {:.2f}% Accuracy".format( 100 * accuracy_score(pred_RF, y_test)))
         print("K-Nearest Neighbors Classifier: {:.2f}% Accuracy".format( 100 * accuracy_score(pred_knn, y_test)))
         print("DecisionTreeClassifier: {:.2f}% Accuracy".format( 100 * accuracy_score(pred_DT, y_test)))
         print("AdaBoostClassifier: {:.2f}% Accuracy".format( 100 * accuracy_score(pred_ADA, y_test)))

         Logistic Regression Classifier: 82.54% Accuracy
         Random Forest Classifier: 79.47% Accuracy
         K-Nearest Neighbors Classifier: 78.84% Accuracy
         DecisionTreeClassifier: 76.22% Accuracy
         AdaBoostClassifier: 82.42% Accuracy
```

Thank You