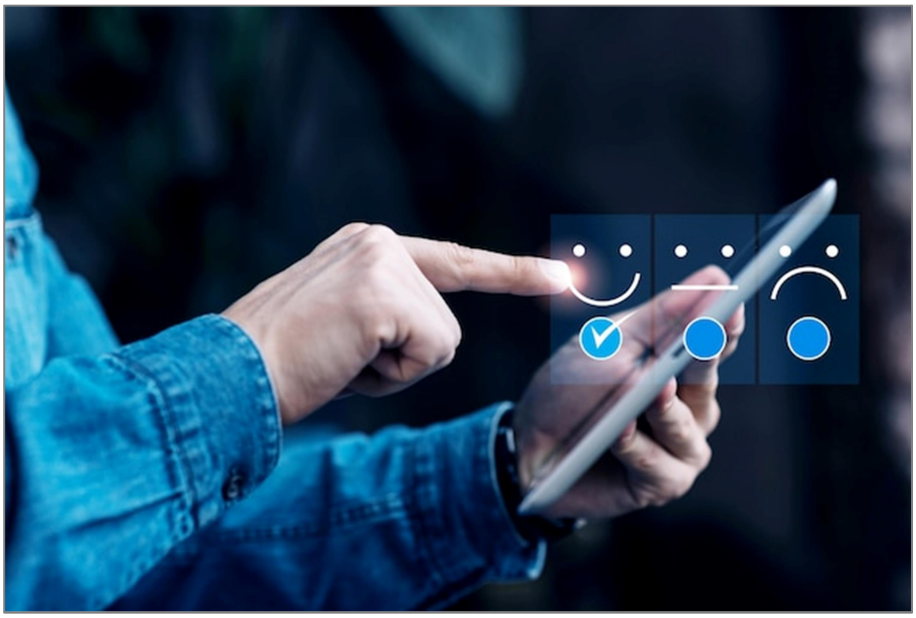
Telco Customer Churn Predictive Modeling

Introduction

With costs increasing at the fastest pace in decades, inflation has unsurprisingly risen to the top of the agenda for telecommunications executives. The effect on each Telco will vary, of course. But inflation will put pressure on most of the budget, particularly personnel costs, energy, external spending on services, leases, and capital expenditures.

Therefore, it's a no-brainer that customer retention is one of the most critical goals for businesses. Increasing customer retention rates by a mere 5% could increase profits by 25% to 95%. Many Telco will continue to focus on delivering a great customer experience which will make it easier for customers to accept price increases and not switch to competitors.



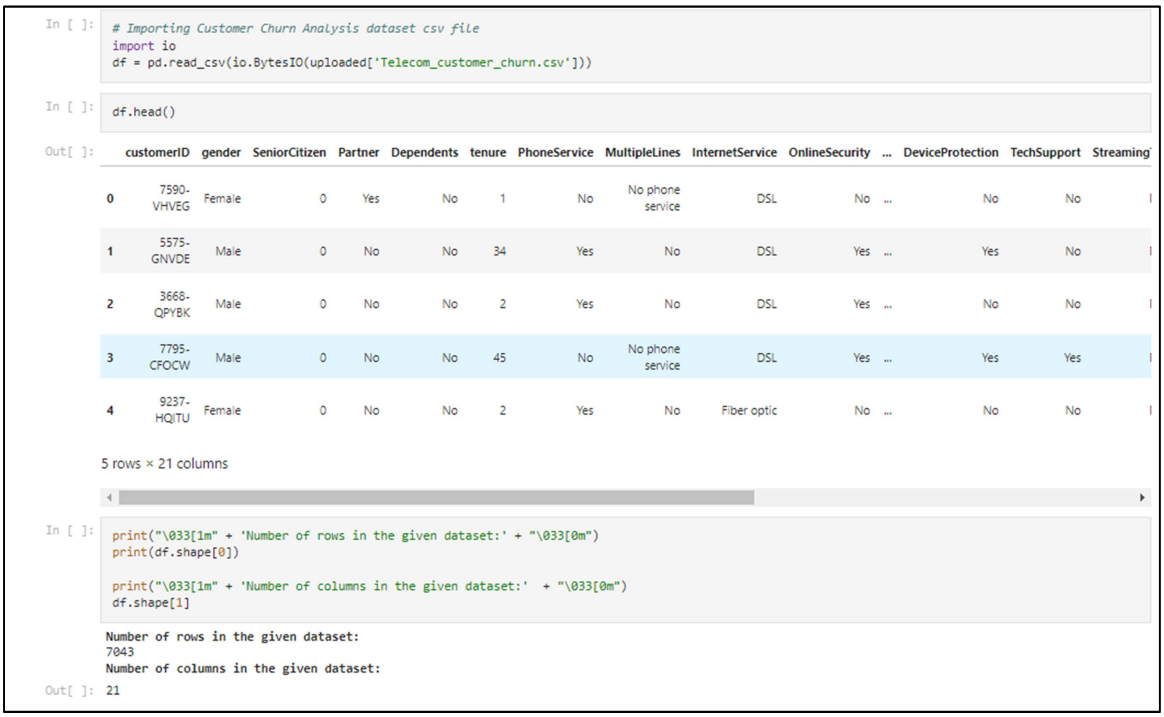
What is attrition?

Churn — or **customer attrition** — is simply defined as the number or percentage of customers lost within a specific period of time, netting off the new acquisitions during that time. In other words, the metric tracks how successful or not you have been at keeping your customers engaged.

For a telecom company you may identify a customer as ‘attrited’ is he/she hasn’t used your network for a month. Most companies today, design specific initiatives to address attrition. The heart of churn management lies in being able to identify the early warning signals from potential attritors. If I know early enough that a specific customer is likely to leave my business, I can take proactive steps to prevent it from happening. And this is where analytics can play a transformational role.

Problem Definition

Studying historical data around negative customer experiences and how customers of different types have responded to them can help develop a robust model for predicting reactive churn (Customers react to specific negative experiences that trigger their move away from your business is known as ‘Reactive’ churn). In this blog, we will examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models. The company provides home and internet services to **7043 customers** in California. Our challenge is to help the company predict behavior to retain customers and analyze all relevant customer data to develop focused customer retention programs.



In [ ]:
```
# Importing Customer Churn Analysis dataset csv file
import io
df = pd.read_csv(io.BytesIO(uploaded['Telecom_customer_churn.csv']))
```

In [ ]:
```
df.head()
```

Out[ ]:

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

5 rows × 21 columns

In [ ]:
```
print("\033[1m" + 'Number of rows in the given dataset:' + "\033[0m")
print(df.shape[0])

print("\033[1m" + 'Number of columns in the given dataset:'  + "\033[0m")
df.shape[1]
```

```
Number of rows in the given dataset:
7043
Number of columns in the given dataset:
```

Out[ ]:
```
21
```

Now, let's have a look at the **21 columns** present in the dataset:

```
[7]  1 # Sorting columns on the basis of its data type
     2 df.columns.to_series().groupby(df.dtypes).groups
```

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

Data Analysis

Checking statistics part of numerical data:

```
[8]  1 df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Observation:

- ✓ Senior Citizen is a categorical variable but it is listed as int64,need to change its data type to object
- ✓ Total Charges feature is numerical in nature but categories as object data types. This implies that there **is presence of string variable** in this column or might be data error

✓ Customer ID is unnecessary variable from our analytical & modeling viewpoint and hence will drop 'Customer ID' column

```
[ ]  1 df = df.drop('customerID',axis=1)

[ ]  1 df['SeniorCitizen'] = df['SeniorCitizen'].astype(object)

[ ]  1 df.loc[df['TotalCharges']==" "]
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	Yes	Yes	No	Two year	Yes	Bank transfe (automatic
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed checl
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	No	Yes	Yes	Two year	No	Mailed checl
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed checl
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	Yes	Yes	No	Two year	No	Credit carc (automatic
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed checl
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed checl
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed checl
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	One year	Yes	Mailed checl
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	Yes	Yes	No	Two year	No	Mailed checl
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	Yes	No	No	Two year	Yes	Bank transfe (automatic

```
1 # Replacing void spaces into null values
2 df['TotalCharges'] = df['TotalCharges'].replace(" ",np.nan)
```

✓ Here we can see that there are 11 blank spaces in the Total charges column

✓ Converting blank spaces into null values

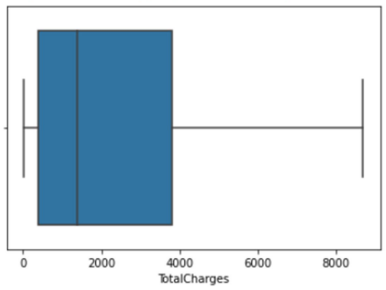
```
[13]  1 # Checking missing values in the dataset
      2 df.isnull().sum()
```

```
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
```

- ✓ Now we need to impute these null values in the **Total Charges** column
- ✓ Converting Total Charges column from object to float data type
- ✓ Checking outliers in Total Charges column, as mean is more sensitive to the existence of outliers than the median

```
[16]  1 sns.boxplot(df['TotalCharges'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8827424820>
```



Since there are no outliers present in the Total Charges column, imputation of missing values can be done using mean strategy

```
1 # Imputing the missing values of Totalcharges column with mean
2 df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].mean())
```

```
1 # Imputing the missing values of Totalcharges column with mean
2 df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].mean())
```

```
[ ]  1 df.isnull().sum()
```

```
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
```

Now, there is no null values in the dataset, we can proceed further.

Checking duplicated data present in the dataset:

Checking Duplicates in the dataset

```
[ ]  1 df.duplicated().sum()
```

```
22
```

```
[ ]  1 # Dropping the duplicated entries
     2 df.drop_duplicates(inplace=True)
```

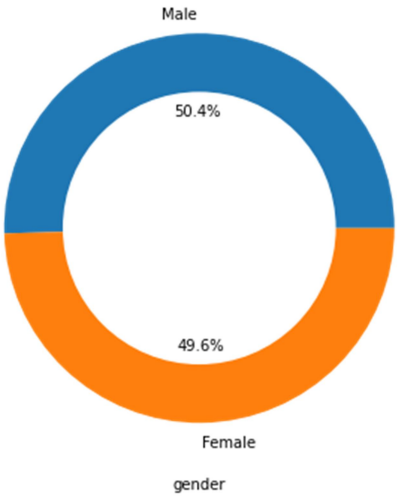
```
1 print("\033[1m" + 'After removing duplicates,number of rows in the given dataset:' + "\033[0m")
2 print(df.shape[0])
3
4 print("\033[1m" + 'After removing duplicates,number of columns in the given dataset:'  + "\033[0m")
5 df.shape[1]
```

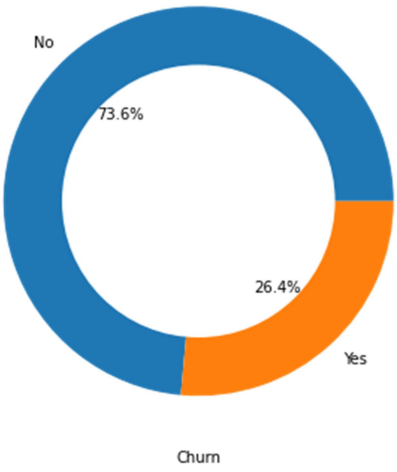
```
After removing duplicates,number of rows in the given dataset:
7021
After removing duplicates,number of columns in the given dataset:
20
```

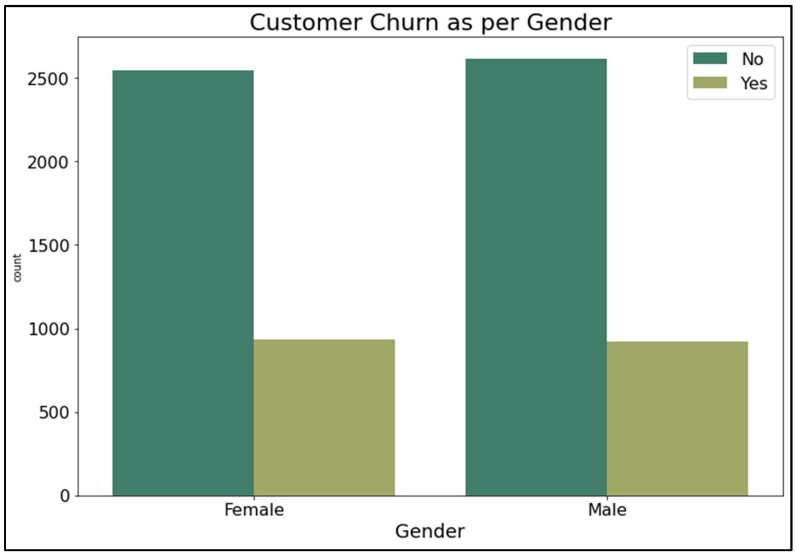
- ✓ Removing the duplicate data from the dataset and making it clean for further investigation
- ✓ The shape of the dataset after removing duplicates becomes **7021 rows and 20 columns**

Exploratory Data Analysis

1. **Customer Churn as per Gender**

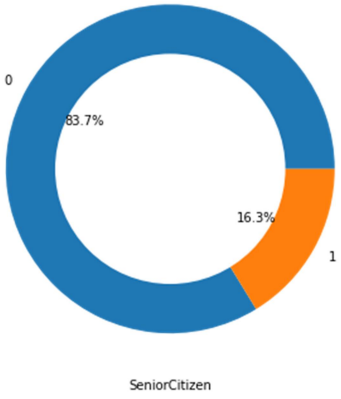


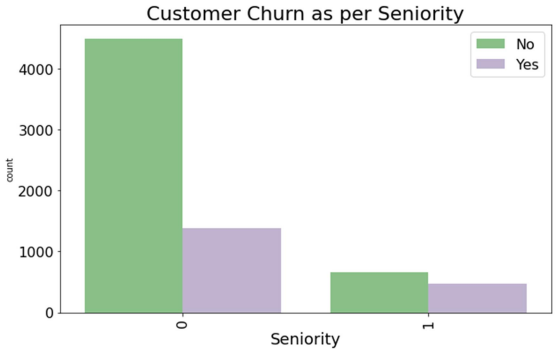




- ✓ From the above plots, equal number of male and female looks dissatisfied with the telecom company

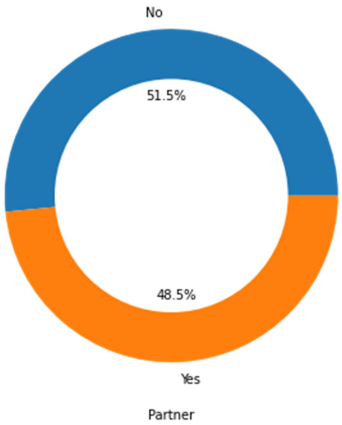
2. Customer Churn as per Seniority

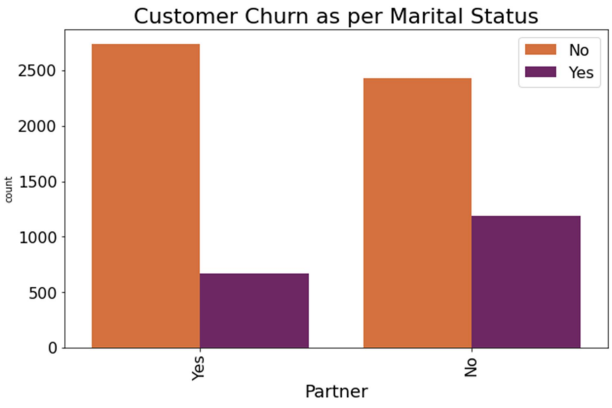




- ✓ More number of young people are dissatisfied and switching one telecom operator to other but it is obvious as almost 84% of younger people were present in the observation
- ✓ Non-Senior Citizens are high Churners

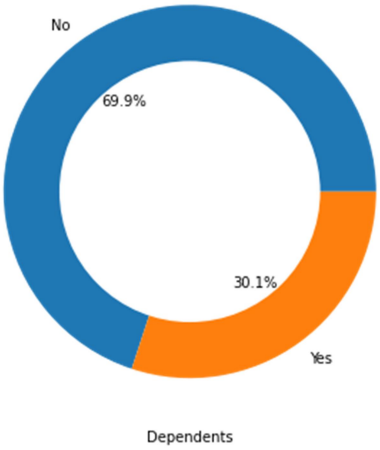
3. Customer Churn as per Marital Status

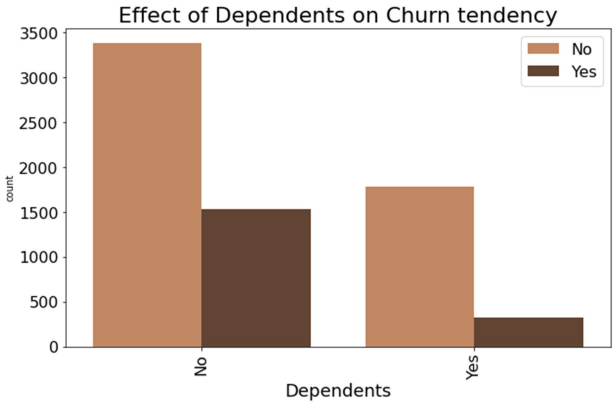




- ✓ There are 51.5% of married customer present in the dataset
- ✓ Bachelors are more reluctant to churn

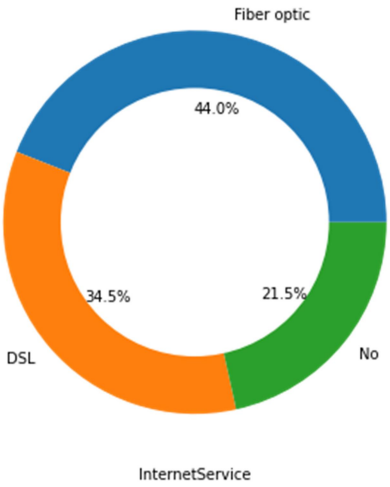
4. Effect of dependents on Churn tendency

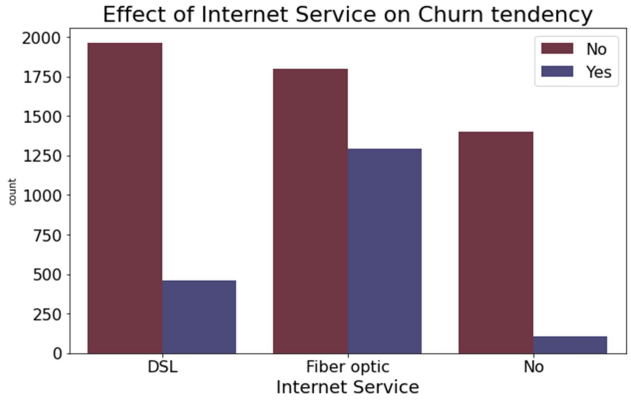




- ✓ Number of dependent customers is 2110 and number of independent customer is 4911
- ✓ Customer having depedents have less tendency to Churn

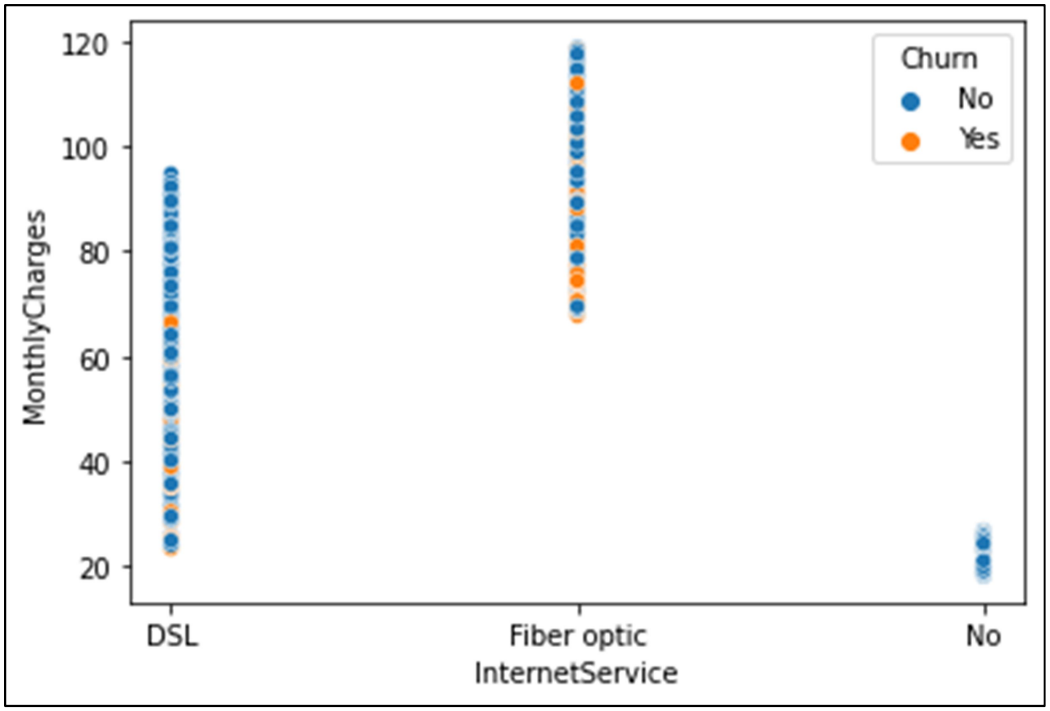
5. Effect of Internet Service on Churn tendency





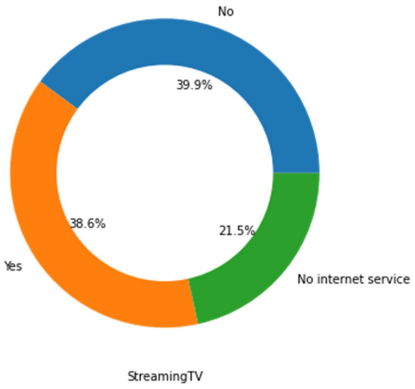
- ✓ Number of customers using Internet service is 5509 and customers with no internet service is 1512

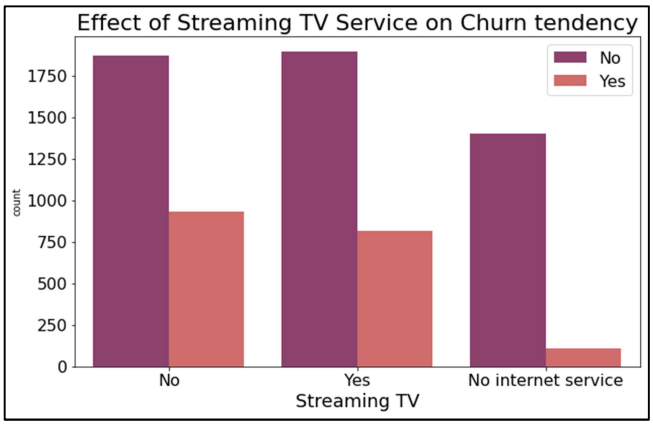
- ✓ Out of 5509 internet users, 56% of customers uses Fiber optic as an Internet service
- ✓ 44% of Internet users opt for DSL service
- ✓ Customers using Fiber optic Internet service have more tendency to churn when compared with DSL and No Internet service



- ✓ High monthly charges can be seen among customers using fiber optic compare to DSL. So, high charges might be the reason of customer churn

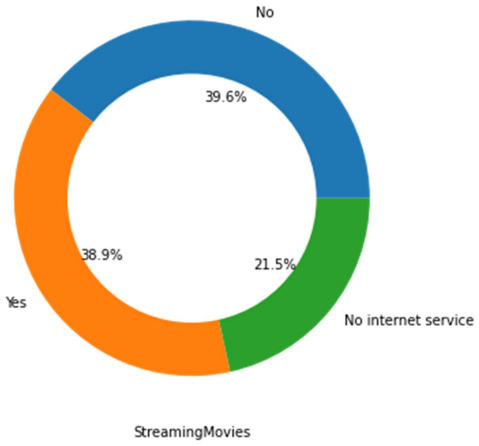
6. Effect of Streaming TV Service on Churn tendency

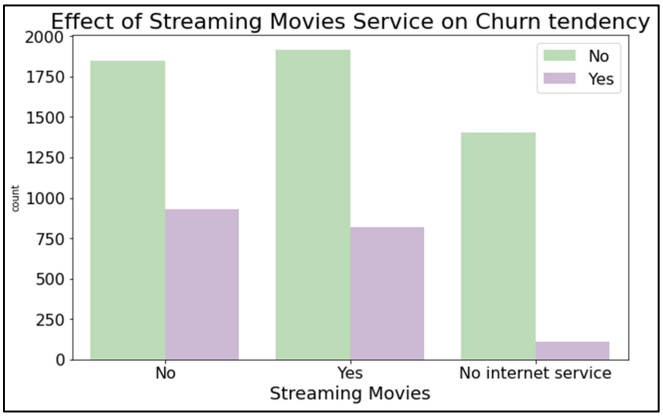




- ✓ Number of customers who don't use Internet service is 1512 and customers using Streaming TV service is 2707 and who don't use Streaming TV service is 2802
- ✓ Equal amount of churning can be seen for both Streaming TV users and non-users

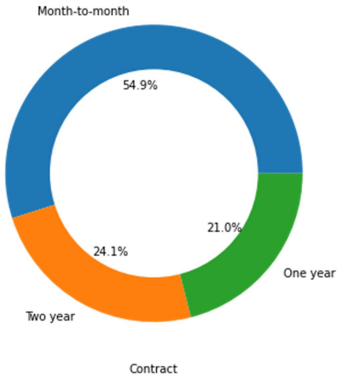
7. Effect of Streaming Movies on Churn tendency

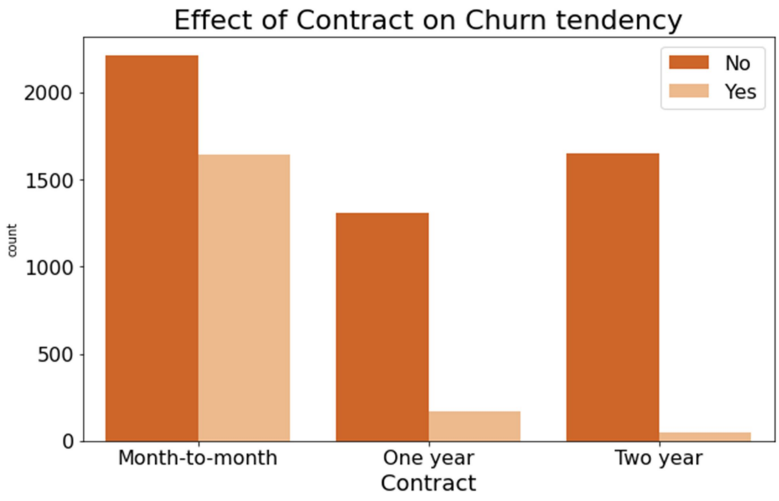




- ✓ Number of customers who don't use Internet service is 1512 and customers using Streaming Movies service is 2732 and who don't use Streaming Movies service is 2777
- ✓ Nothing much to conclude from Streaming movies service as equal amount of churning and non-churning can be seen for users and non-users

8. Effect of Contract on Churn tendency

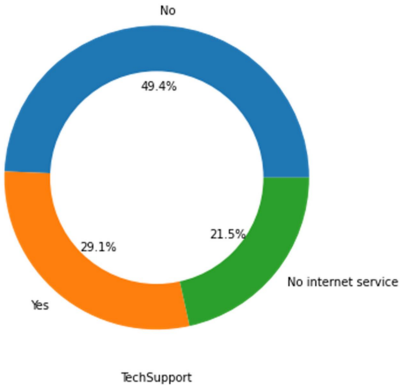


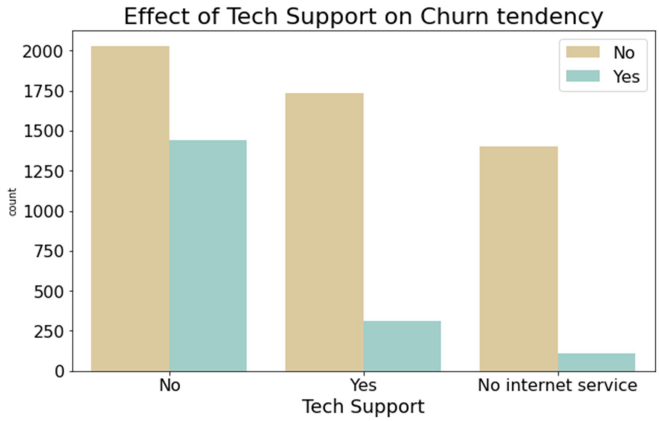


✓ Almost 55% customer prefer month to month contract compare to others

✓ Monthly contract customers are more likely to churn as they are free-to-go customers

9. Effect of Tech Support on Churn tendency

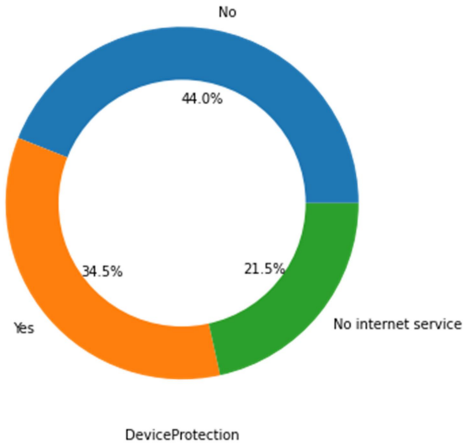


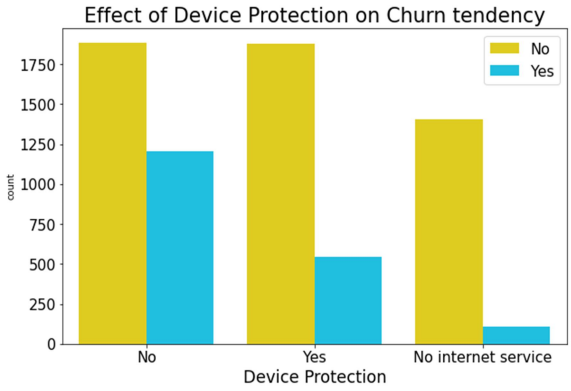


✓ Number of customers who don't use Internet service is 1512 and customers with tech support is 2044 and who don't use tech support is 3465

✓ Customers with no Technical support are more reluctant to churn

10. Effect of Device Protection on Churn tendency

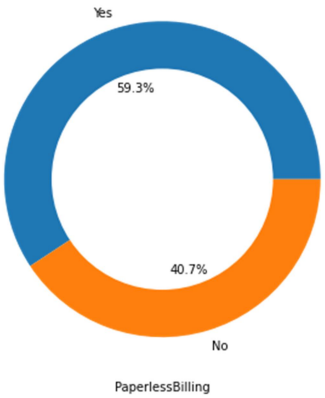


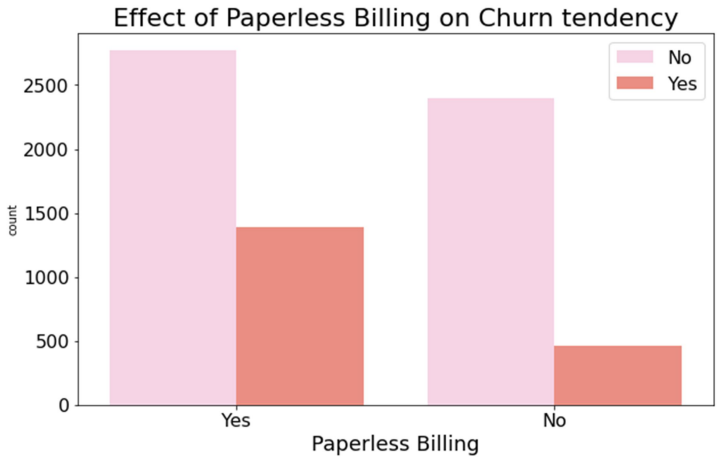


✓ Only 34.5 % customers uses Device Protection service

✓ Customers with no device protection have more tendency to churn

11. Effect of Paperless Billing on Churn tendency





- ✓ 60% of customers prefer paperless billing
- ✓ Customers using paperless billing has high tendency to churn

Concluding remarks after EDA

- ❖ Non-Senior Citizens are high Churners
- ❖ Bachelors are more reluctant to churn
- ❖ Customer having depedents have less tendency to Churn
- ❖ High monthly charges of Fiber Optic Internet might be the reason of customer churn
- ❖ Monthly contract customers are more likely to churn as they are free-to-go customers
- ❖ Customers with no Technical support are more reluctant to churn
- ❖ Customers with no device protection have more tendency to churn
- ❖ Customers using paperless billing has high tendency to churn

Pre-Processing

1. Encoding the categorical columns

Label Encoder is used for encoding the categorical variable. Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

ENCODING THE CATEGORICAL VARIABLES

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [ ]:
```
for i in df.columns:
    if df[i].dtypes == "object":
        df[i] = le.fit_transform(df[i].values.reshape(-1,1))
```

In [ ]:
```
df.head()
```

Out[ ]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV
0	0	0	1	0	1	0	1	0	0	2	0	0	0
1	1	0	0	0	34	1	0	0	2	0	2	0	0
2	1	0	0	0	2	1	0	0	2	2	0	0	0
3	1	0	0	0	45	0	1	0	2	0	2	2	0
4	0	0	0	0	2	1	0	1	0	0	0	0	0

2. Skewness Reduction

A power transform will make the probability distribution of a variable more Gaussian.This is often described as removing a skew in the distribution, although more generally is described as stabilizing the variance of the distribution. Power transformer is used to reduce the skewness in the numerical variables.

SKEWNESS REDUCTION

In [ ]:
```
# Considering skewness reduction through PowerTransformer

from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
```

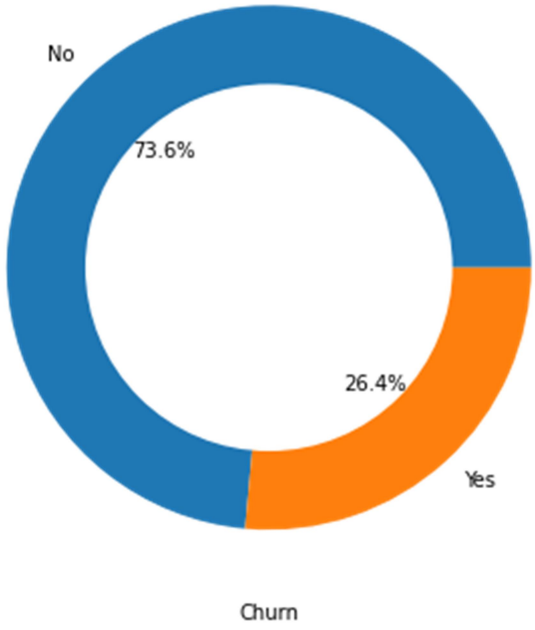
In [ ]:
```
# Seperating features and target variable for classification task

X = df.drop('Churn',axis=1)

y = df['Churn']
```

In [ ]:
```
# Applying Power transformer

X_new_pt = pt.fit_transform(X)

# Converting numpy array(X_new_pt) into Dataframe and reassigning the values

X = pd.DataFrame(X_new_pt,columns= X.columns)

# Checking Skewness

X.skew().sort_values(ascending=False)
```

Out[ ]:
```
SeniorCitizen       1.829987
Dependents          0.870322
Contract            0.297934
OnlineSecurity      0.150084
TechSupport         0.140414
Partner             0.061857
MultipleLines       0.028603
DeviceProtection    0.000849
OnlineBackup       -0.001635
gender             -0.017381
InternetService    -0.071493
StreamingTV        -0.098052
StreamingMovies    -0.106539
TotalCharges       -0.143822
PaymentMethod      -0.206316
tenure             -0.243646
```

3. Balancing the target variable using SMOTE

SMOTE (Synthetic Minority Oversampling Technique) is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.



73.6% customers are satisfied with their current telecom company. But as far as model building is concerned, we need a balanced dataset i.e no bias to any class. Hence we will be using oversampling technique to balance the target class of our dataset.

BALANCING TARGET VARIABLE USING SMOTE

In [ ]:
```
from imblearn.over_sampling import SMOTE
over_smp = SMOTE(0.8)
```

In [ ]:
```
print("The number of target classes before fit{}".format(Counter(y)))
```

```
The number of target classes before fitCounter({0: 5164, 1: 1857})
```

Clearly visible the data is imbalanced

In [ ]:
```
X,y = over_smp.fit_resample(X,y)
print("The number of target classes after fit{}".format(Counter(y)))
```

```
The number of target classes after fitCounter({0: 5164, 1: 4131})
```

4. Scaling of Data using Standard Scaler

Standard Scaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation.

Standard Scaling

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
X_scaler = scaler.fit_transform(X)
```

5. Multicollinearity Check

A variance inflation factor (VIF) is a measure of the amount of multicollinearity in regression analysis. Multicollinearity exists when there is a correlation between multiple independent variables in a multiple regression model.

In [ ]:

```
# Checking multicollinearity in the dataset
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['vif'] = [variance_inflation_factor(X_scaler,i) for i in range(X.shape[1])]
vif['Features'] = X.columns

vif
```

Out[ ]:

	vif	Features
0	1.003089	gender
1	1.151983	SeniorCitizen
2	1.500458	Partner
3	1.409321	Dependents
4	32.957930	tenure
5	1.739492	PhoneService
6	1.426152	MultipleLines
7	1.764817	InternetService
8	1.403631	OnlineSecurity
9	1.238512	OnlineBackup
10	1.320332	DeviceProtection
11	1.446367	TechSupport
12	1.453910	StreamingTV
13	1.445920	StreamingMovies
14	2.488099	Contract
15	1.218408	PaperlessBilling
16	1.186351	PaymentMethod
17	8.655797	MonthlyCharges
18	43.730559	TotalCharges

To deal with multicollinearity:

- Remove some of the highly correlated independent variables
- Linearly combine the independent variables, such as adding them together
- Perform an analysis designed for highly correlated variables, such as principal components analysis or partial least squares regression

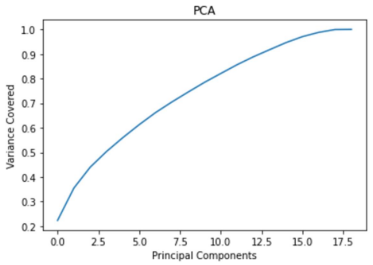
Tenure and TotalCharges showing vif value greater than 10. It means multicollinearity is present in the dataset.

6. Principal Component Analysis

PCA (Principal Component Analysis) takes advantage of multicollinearity and combines the highly correlated variables into a set of uncorrelated variables called principal components. Therefore, PCA can effectively eliminate multicollinearity between features.PCA aims to reduce dimensionality in our dataset.

```
In [ ]: from sklearn.decomposition import PCA
        pca = PCA()
        pca.fit_transform(X_scaler)

Out[ ]: array([[-2.75309717,  0.4355675 ,  2.87372997, ...,  0.69582799,
                -0.40640662, -0.02235923],
               [-0.50109664,  1.52360373,  0.81451602, ...,  0.19495256,
                 0.03416362,  0.04606429],
               [-2.65408977,  0.87146781,  0.42507995, ...,  0.68719124,
                 0.229324  , -0.06453815],
               ...,
               [-1.37210034, -0.0189085 ,  0.50176903, ..., -0.84635268,
                -0.27746002,  0.09619082],
               [ 2.14523347,  0.47785425,  2.73434674, ...,  0.0339959 ,
                -0.30123352,  0.00452573],
               [ 1.01134221, -2.1934633 , -0.74207206, ..., -0.1004391 ,
                 0.36248154,  0.04855424]])
```

```
In [ ]: # Lets plot scree plot to check the best components

        plt.figure()
        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.xlabel('Principal Components')
        plt.ylabel('Variance Covered')
        plt.title('PCA')
        plt.show()
```



Around 16 components are able to explain more than 95% variance in the dataset and hence its safe to consider 16 PC's

- Out of 19 independent features, we are getting 16 components explaining more than 95% of variance in the dataset and hence it is safe to consider 16 Principal components.

In [ ]:
```
pca = PCA(n_components=16)
new_pcomp = pca.fit_transform(X_scaler)
princ_comp = pd.DataFrame(new_pcomp,columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11','PC12','PC13','PC14','PC15','PC16'])
princ_comp
```

Out[ ]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16
0	-2.753097	0.435568	2.873730	0.192411	0.875418	1.235275	1.135920	-1.610368	-1.488920	-1.068593	0.153262	-0.151215	0.082051	0.088376	-0.035944	-1.494853
1	-0.501097	1.523604	0.814516	0.939438	-0.588768	-1.344997	-0.585908	0.343482	2.102353	1.134424	1.338381	-0.561788	-1.535690	-0.276805	-0.956656	0.411905
2	-2.654090	0.871468	0.425080	0.356416	-0.578846	-1.152757	-0.552390	-2.365402	0.012008	0.621626	0.337749	0.608602	-1.502712	0.125617	0.614953	-0.104337
3	0.725332	1.969884	3.389159	2.594794	-0.236451	-0.950245	-0.419248	1.670113	-0.085313	0.187093	-0.211230	-1.070294	-0.055813	-0.212132	0.469412	0.034258
4	-3.231858	-0.335700	-0.432627	-0.397421	-0.769395	0.875748	-0.844085	-0.191641	-0.256335	0.287600	0.006709	-0.341756	0.011242	-0.000568	-0.609547	0.010598
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9290	-2.599865	-0.370120	-0.034716	-0.130948	-0.742029	0.755259	-0.073561	0.173142	2.027556	-1.313456	-0.213248	-0.521907	0.413409	-0.112652	-0.122743	0.045864
9291	3.256598	-1.484173	-0.621103	-0.808693	1.123158	-0.891429	1.638201	-0.651508	-0.157308	0.828164	0.536822	-0.037462	1.534217	0.163804	0.806008	0.983557
9292	-1.372100	-0.018909	0.501769	-2.922276	0.143476	-1.008490	-0.446098	-0.141983	-0.299861	0.458622	0.073993	-0.154168	0.068935	0.060849	-0.811216	0.140466
9293	2.145233	0.477854	2.734347	-0.073382	2.379884	-0.394748	0.616380	-0.158581	-1.229355	-1.211253	0.248238	1.162573	-0.899706	-0.072087	-0.394923	0.443460
9294	1.011342	-2.193463	-0.742072	-0.427176	1.716237	-0.741841	-0.334500	0.520615	0.502771	-0.204979	0.370479	-0.984139	-0.432381	-0.064066	-0.384133	-1.093126

9295 rows × 16 columns

Building Machine Learning Models

First, we will find the best random state. The random state hyper parameter in the train_test_split function controls the shuffling process. With random state=None , we get different train and test sets across different executions and the shuffling process is out of control.

BEST RANDOM STATE

In [ ]:
```
from sklearn.model_selection import train_test_split
maxAccu = 0 # Max Accuracy
maxRS = 0   # Best random state for which maximum accuracy is achieved

for i in range(0,100):
    X_train,X_test,y_train,y_test = train_test_split(princ_comp,y,test_size=0.2,random_state=i)

    rf = RandomForestClassifier()

    rf.fit(X_train,y_train)    # Training the model

    pred_rf = rf.predict(X_test)    # Predicting the target variable

    acc_rf = accuracy_score(y_test,pred_rf)

    if acc_rf>maxAccu:

        maxAccu = acc_rf

        maxRS = i

print("Maximum Accuracy:",maxAccu,"at random state ", maxRS)
```

```
Maximum Accuracy: 0.8380849919311458 at random state  15
```

We get random state = 15 as we are getting the maximum R2 score for our base model (Random Forest Classifier).

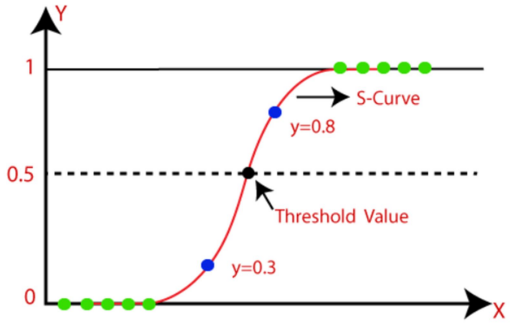
Then we will split the data into train and test dataset using the best random state.

Train test Split using best Random State

```
[65]: X_train,X_test,y_train,y_test = train_test_split(princ_comp,y,test_size=0.2,random_state=15)
```

Model 1 – Logistic Regression

Logistic Regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.



Logistic Regression

```
In [67]: lr = LogisticRegression()

         lr.fit(X_train,y_train)

         pred_lr = lr.predict(X_test)    # Predicted values

         metric_score(lr,X_train,X_test,y_train,y_test,train=True)

         metric_score(lr,X_train,X_test,y_train,y_test,train=False)

         cr_lr = cross_val_score(lr,princ_comp,y,cv=5)

         print("Cross validation score of Logistic regression model :",cr_lr.mean()*100)
```

```
----------Train Result----------
Train Accuracy Score : 76.45%
----------Test Result----------
=====Confusion Matrix=======
 [[820 218]
 [181 640]]
Test Accuracy Score : 78.54%

 Test Classification Report
              precision    recall  f1-score   support

           0       0.82      0.79      0.80      1038
           1       0.75      0.78      0.76       821

    accuracy                           0.79      1859
   macro avg       0.78      0.78      0.78      1859
weighted avg       0.79      0.79      0.79      1859

Cross validation score of Logistic regression model : 76.82625067240453
```

The result looks good. We will try to improve the performance of the model by tuning the parameters.

Tuning parameteres for Logistic regression

In [68]:
```
lr_params = {'penalty':['l1', 'l2'],
             'tol':[0.0001,0.001,0.01],
             'solver':['newton-cg', 'lbfgs', 'liblinear'],
             'multi_class':['auto', 'ovr', 'multinomial']}

grd_lr = GridSearchCV(lr,param_grid = lr_params, n_jobs =-1)

grd_lr.fit(X_train,y_train)

print("Best parameters : ",grd_lr.best_params_)
```

```
Best parameters :  {'multi_class': 'auto', 'penalty': 'l1', 'solver': 'liblinear', 'tol': 0.01}
```

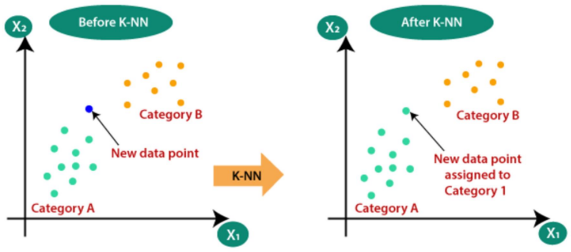
We get the following result:

In [69]:
```
lr = grd_lr.best_estimator_

lr.fit(X_train,y_train)

pred_lr = lr.predict(X_test)    # Predicted values

metric_score(lr,X_train,X_test,y_train,y_test,train=True)

metric_score(lr,X_train,X_test,y_train,y_test,train=False)

cr_lr = cross_val_score(lr,princ_comp,y,cv=5)

print("Cross validation score of Logistic regression model :",cr_lr.mean()*100)
```

```
----------------Train Result--------------------
Train Accuracy Score : 76.44%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[820 218]
 [181 640]]
Test Accuracy Score : 78.54%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.82      0.79      0.80      1038
           1       0.75      0.78      0.76       821

    accuracy                           0.79      1859
   macro avg       0.78      0.78      0.78      1859
weighted avg       0.79      0.79      0.79      1859

Cross validation score of Logistic regression model : 76.85852608929532
```

Not much difference after tuning the parameters. Lets go for another model.

Model 2 – K Nearest Neighbor Classifier



K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

KNN Classifier

In [70]:

```
knn = KNeighborsClassifier()

knn.fit(X_train,y_train)

pred_knn = knn.predict(X_test)    # Predicted values

metric_score(knn,X_train,X_test,y_train,y_test,train=True)

metric_score(knn,X_train,X_test,y_train,y_test,train=False)

cr_knn = cross_val_score(knn,princ_comp,y,cv=5)

print("Cross validation score of K Neighbors Classifier model :",cr_knn.mean()*100)
```

```
----------------Train Result--------------------
Train Accuracy Score : 84.48%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[764 274]
 [113 708]]
Test Accuracy Score : 79.18%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.87      0.74      0.80      1038
           1       0.72      0.86      0.79       821

    accuracy                           0.79      1859
   macro avg       0.80      0.80      0.79      1859
weighted avg       0.80      0.79      0.79      1859

Cross validation score of K Neighbors Classifier model : 77.52555137170522
```

Better results can be seen for KNN Classifier model when compared to Logistic Regression. Let's tune the parameters for KNN model

Hyperparameter tuning for KNN Classifier:

In [71]:

```
knn_params = { 'n_neighbors':[4,5],
      'weights':['uniform','distance'],
    'algorithm':['auto','ball_tree','brute'],
    'leaf_size':[25,30]}

grd_knn = GridSearchCV(knn,param_grid = knn_params, n_jobs =-1)

grd_knn.fit(X_train,y_train)

print("Best parameters : ",grd_knn.best_params_)
```

```
Best parameters :  {'algorithm': 'auto', 'leaf_size': 25, 'n_neighbors': 4, 'weights': 'distance'}
```

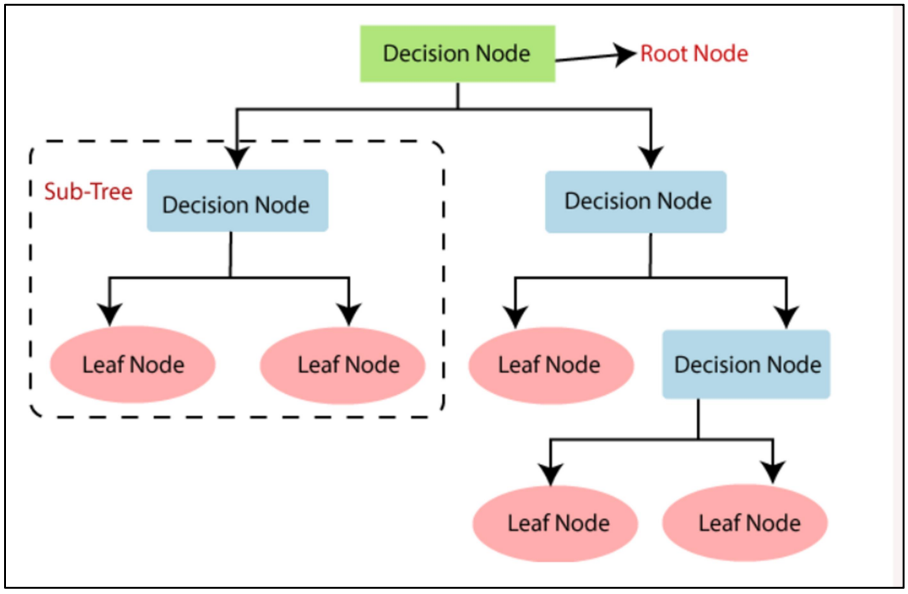
In [72]:

```
knn = grd_knn.best_estimator_

knn.fit(X_train,y_train)

pred_knn = knn.predict(X_test)    # Predicted values

metric_score(knn,X_train,X_test,y_train,y_test,train=True)

metric_score(knn,X_train,X_test,y_train,y_test,train=False)

cr_knn = cross_val_score(knn,princ_comp,y,cv=5)

print("Cross validation score of K Neighbors Classifier model :",cr_knn.mean()*100)
```

```
----------------Train Result--------------------
Train Accuracy Score : 99.87%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[813 225]
 [128 693]]
Test Accuracy Score : 81.01%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.86      0.78      0.82      1038
           1       0.75      0.84      0.80       821

    accuracy                           0.81      1859
   macro avg       0.81      0.81      0.81      1859
weighted avg       0.82      0.81      0.81      1859

Cross validation score of K Neighbors Classifier model : 80.43033889187734
```

We get better result after hyperparameter tuning for this model. Let's check another model.

Model 3 – Decision Tree Classifier



It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

Decision Tree Classifier

In [73]:

```
dt = DecisionTreeClassifier()

dt.fit(X_train,y_train)      # Model training

pred_dt = dt.predict(X_test)    # Predicted values

metric_score(dt,X_train,X_test,y_train,y_test,train=True)  # Training result

metric_score(dt,X_train,X_test,y_train,y_test,train=False)  # Test result

cr_dt = cross_val_score(dt,princ_comp,y,cv=5)

print("Cross validation score of Decision Tree Classifier model :",cr_dt.mean()*100)
```

```
----------------Train Result-------------------
Train Accuracy Score : 99.87%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[798 240]
 [243 578]]
Test Accuracy Score : 74.02%

 Test Classification Report
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      1038
           1       0.71      0.70      0.71       821

    accuracy                           0.74      1859
   macro avg       0.74      0.74      0.74      1859
weighted avg       0.74      0.74      0.74      1859

Cross validation score of Decision Tree Classifier model : 75.06186121570737
```

Performing hyper parameter tuning for Decision tree classifier:

Tuning parameters for Decision Tree

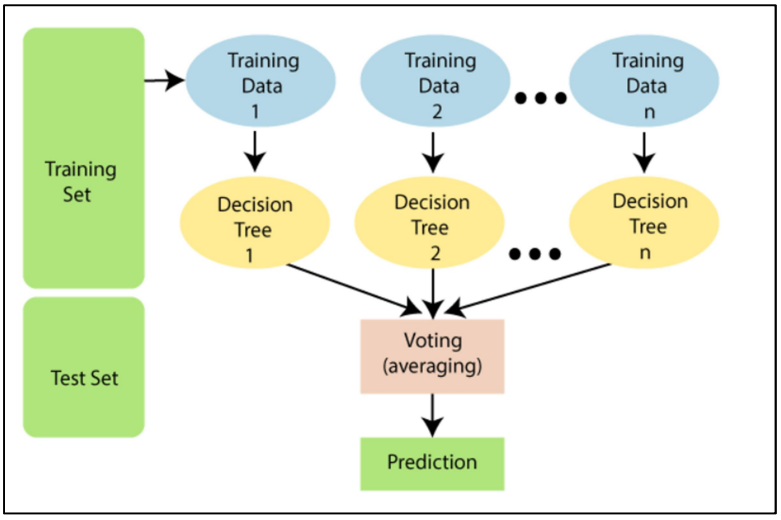
In [75]:
```
grid_param = {'criterion':['gini','entropy'],
              'max_depth': range(9,15),
              'min_samples_leaf': range(3,7),
              'min_samples_split': range(2,6)
             }

grid_search = GridSearchCV(estimator = dt, param_grid = grid_param,n_jobs =-1)

grid_search.fit(X_train,y_train)

print("Best parameters : ",grid_search.best_params_)
```

```
Best parameters :  {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 3, 'min_samples_split': 5}
```

In [76]:
```
dt = grid_search.best_estimator_

dt.fit(X_train,y_train)      # Model training

pred_dt = dt.predict(X_test)    # Predicted values

metric_score(dt,X_train,X_test,y_train,y_test,train=True)  # Training result

metric_score(dt,X_train,X_test,y_train,y_test,train=False)  # Test result

cr_dt = cross_val_score(dt,princ_comp,y,cv=5)

print("Cross validation score of Decision Tree Classifier model :",cr_dt.mean()*100)
```

```
----------------Train Result--------------------
Train Accuracy Score : 87.76%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[778 260]
 [183 638]]
Test Accuracy Score : 76.17%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.81      0.75      0.78      1038
           1       0.71      0.78      0.74       821

    accuracy                           0.76      1859
   macro avg       0.76      0.76      0.76      1859
weighted avg       0.77      0.76      0.76      1859

Cross validation score of Decision Tree Classifier model : 75.45992469069391
```

Model 4 – Random Forest Classifier



Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase. New data points are assigned to the category that wins the majority votes.

Random Forest Classifier

In [77]:
```
rf = RandomForestClassifier()

rf.fit(X_train,y_train)      # Model training

pred_rf = rf.predict(X_test)    # Predicted values

metric_score(rf,X_train,X_test,y_train,y_test,train=True)  # Training result

metric_score(rf,X_train,X_test,y_train,y_test,train=False)  # Test result

cr_rf = cross_val_score(rf,princ_comp,y,cv=5)

print("Cross validation score of Random Forest Classifier model :",cr_rf.mean()*100)
```

```
----------Train Result----------
Train Accuracy Score : 99.87%
----------Test Result----------
=====Confusion Matrix=======
 [[890 148]
 [167 654]]
Test Accuracy Score : 83.06%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.84      0.86      0.85      1038
           1       0.82      0.80      0.81       821

    accuracy                           0.83      1859
   macro avg       0.83      0.83      0.83      1859
weighted avg       0.83      0.83      0.83      1859

Cross validation score of Random Forest Classifier model : 82.27003765465304
```

Far better result can be seen for Random Forest Classifier until now

Hyper parameter tuning is performed on Random Forest:

Tuning parameters for Random Forest

In [78]:
```
params_rf = {'criterion':['gini','entropy'],
        'max_depth': [10,12,25,30],
        'min_samples_split' :[2,3,4],
        'min_samples_leaf' :[2,3,4,5]}

grd_rf = GridSearchCV(rf,param_grid = params_rf,n_jobs =-1)

grd_rf.fit(X_train,y_train)

print("Best parameters : ",grd_rf.best_params_)
```

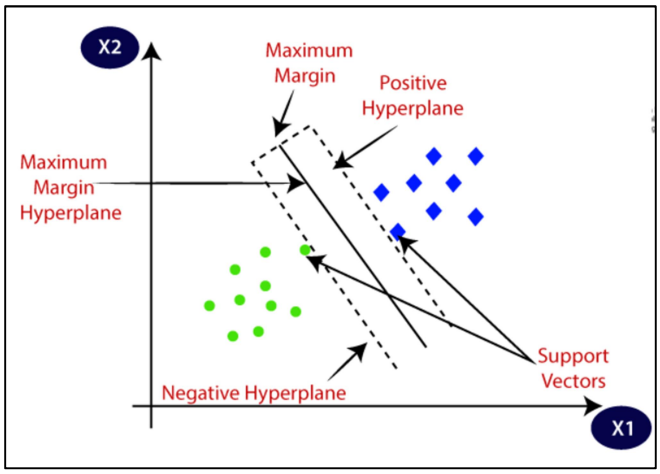
```
Best parameters :  {'criterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 2, 'min_samples_split': 2}
```

In [79]:
```
rf = grd_rf.best_estimator_

rf.fit(X_train,y_train)

pred_rf = rf.predict(X_test)    # Predicted values

metric_score(rf,X_train,X_test,y_train,y_test,train=True)  # Training result

metric_score(rf,X_train,X_test,y_train,y_test,train=False)  # Test result

cr_rf = cross_val_score(rf,princ_comp,y,cv=5)

print("Cross validation score of Random Forest Classifier model :",cr_rf.mean()*100)
```

```
----------Train Result----------
Train Accuracy Score : 99.06%
----------Test Result----------
=====Confusion Matrix=======
 [[885 153]
 [154 667]]
Test Accuracy Score : 83.49%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.85      0.85      0.85      1038
           1       0.81      0.81      0.81       821

    accuracy                           0.83      1859
   macro avg       0.83      0.83      0.83      1859
weighted avg       0.83      0.83      0.83      1859

Cross validation score of Random Forest Classifier model : 81.85045723507261
```

Not much difference was observed after hyper parameter tuning for Random forest model. But we get better training and testing accuracy and max test accuracy of 83.49%.

Model 5 – Support Vector Classifier



The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

Support Vector Classifier

In [80]:

```
svc = SVC()

svc.fit(X_train,y_train)

pred_svc = svc.predict(X_test)    # Predicted values

metric_score(svc,X_train,X_test,y_train,y_test,train=True)

metric_score(svc,X_train,X_test,y_train,y_test,train=False)

cr_svc = cross_val_score(svc,princ_comp,y,cv=5)

print("Cross validation score of Support Vector Classifier model :",cr_svc.mean()*100)
```

```
----------------Train Result-------------------
Train Accuracy Score : 81.62%
----------------Test Result--------------------
=====Confusion Matrix=======
 [[847 191]
 [161 660]]
Test Accuracy Score : 81.07%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.84      0.82      0.83      1038
           1       0.78      0.80      0.79       821

    accuracy                           0.81      1859
   macro avg       0.81      0.81      0.81      1859
weighted avg       0.81      0.81      0.81      1859

Cross validation score of Support Vector Classifier model : 79.11780527165142
```

Difference between training and test accuracy is minimum and hence best result can be seen for the given dataset as far as Support Vector Classifier is concerned

We get the result with minimum difference between training and test accuracy. Tuning parameters for Support Vector Classifier:

Tuning parameters for SVC

In [81]:
```
svc_params = {'kernel':['rbf','linear','poly','sigmoid'],
              'gamma':['scale','auto'],
              'decision_function_shape':['ovr','ovo']
              }

grd_svc = GridSearchCV(svc,param_grid = svc_params,n_jobs =-1)

grd_svc.fit(X_train,y_train)

print("Best parameters : ",grd_svc.best_params_)
```

```
Best parameters :  {'decision_function_shape': 'ovr', 'gamma': 'auto', 'kernel': 'rbf'}
```

In [82]:
```
svc = grd_svc.best_estimator_

svc.fit(X_train,y_train)

pred_svc = svc.predict(X_test)    # Predicted values

metric_score(svc,X_train,X_test,y_train,y_test,train=True)

metric_score(svc,X_train,X_test,y_train,y_test,train=False)

cr_svc = cross_val_score(svc,princ_comp,y,cv=5)

print("Cross validation score of Support Vector Classifier model :",cr_svc.mean()*100)
```

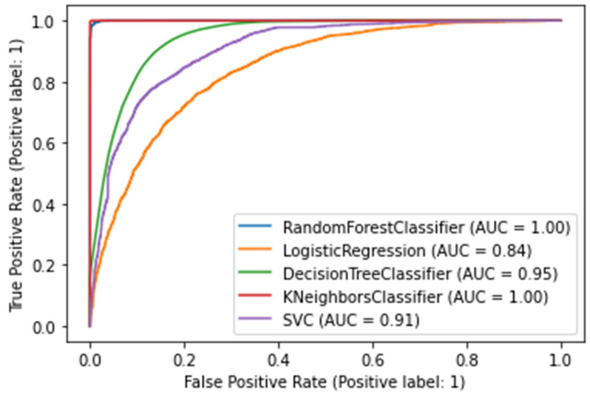
```
----------------Train Result-------------------
Train Accuracy Score : 82.26%
----------------Test Result-------------------
=====Confusion Matrix=======
 [[846 192]
 [161 660]]
Test Accuracy Score : 81.01%

 Test Classification Report
               precision    recall  f1-score   support

           0       0.84      0.82      0.83      1038
           1       0.77      0.80      0.79       821

    accuracy                           0.81      1859
   macro avg       0.81      0.81      0.81      1859
weighted avg       0.81      0.81      0.81      1859

Cross validation score of Support Vector Classifier model : 79.32221624529316
```

There is very little improvement in the result after tuning the parameters.

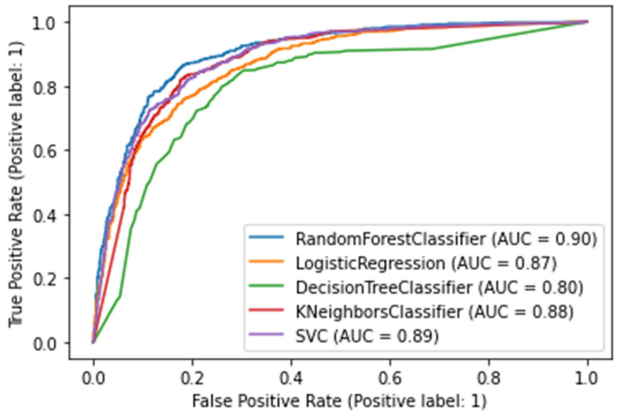
Concluding Remark

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'.

ROC Curve for training dataset:



ROC Curve for test dataset:



As per ROC AUC Curve, **SVC model fits best for the given dataset as there is minimum difference between training and testing accuracy.**