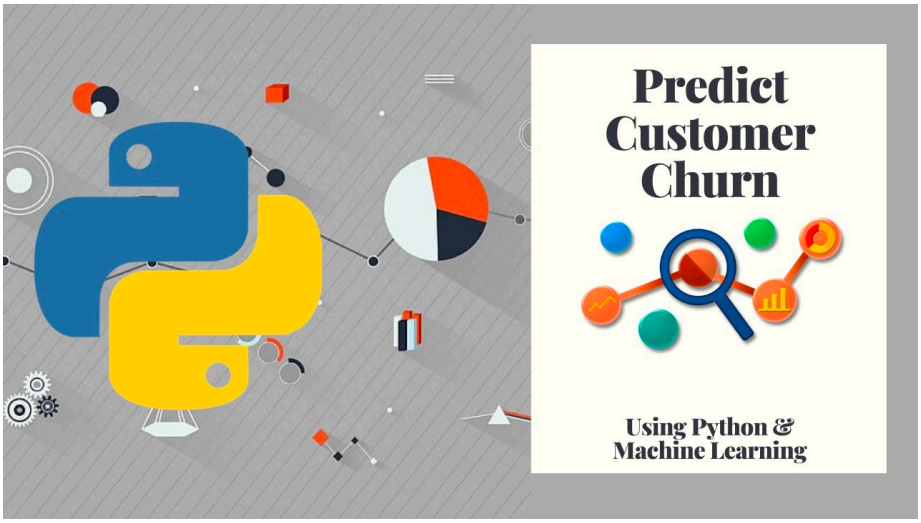
# Customer Churn

## 1) Problem Definition:

Customer churn is a common problem across businesses in many sectors. If you want to grow as a company, you have to invest in acquiring new clients. Every time a client leaves, it represents a significant investment lost. Both time and effort need to be channelled into replacing them. Being able to predict when a client is likely to leave, and offer them incentives to stay, can offer huge savings to a business.

Here in this dataset customer churn (*target variable*) is provided with yes no type nominal data, which itself is clear that it is a Classification problem where we have to predict whether a customer will churn or not. To predict this we have been given various data due to which customers are more often to change the service provider. In this project we will be using many classification models that can help the companies to make decisions by predicting, how and why the customers churn will evolve in the future.



## 2) Data Analysis:

- Reading the CSV file and doing initial statistical analysis (shape, values etc).
- Solving missing values.
- Data Pre-processing: Reading the unique values for each column and removing those which won't be significant in the analysis further.
- Model building and saving the best model with pickle.

## Dataset contains:

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

All given data describes its meaning itself and most columns are having nominal data.

Here, only 4 columns are Numerical, rest are Categorical.

```
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
```

At first I didn't got any nan values in data but soon after certain analysis I found that there are values with not marked as nan.

```
df['TotalCharges'].value_counts()
```

```
           11
20.2       11
19.75       9
20.05       8
19.9        8
           ..
6849.4      1
692.35      1
130.15      1
3211.9      1
6844.5      1
Name: TotalCharges, Length: 6531, dtype: int64
```

Empty space later converted to mean of that column value

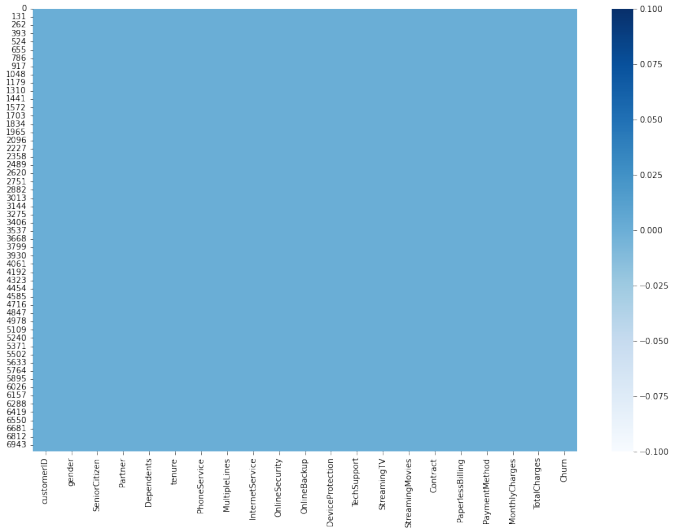
```
read_csv(r"https://raw.githubusercontent.com/dsrscientist/DSData/master/Telecom_customer_churn.csv",na_values = ' ')
```

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

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

Converted the empty space to nan first with this above function than to its mean.

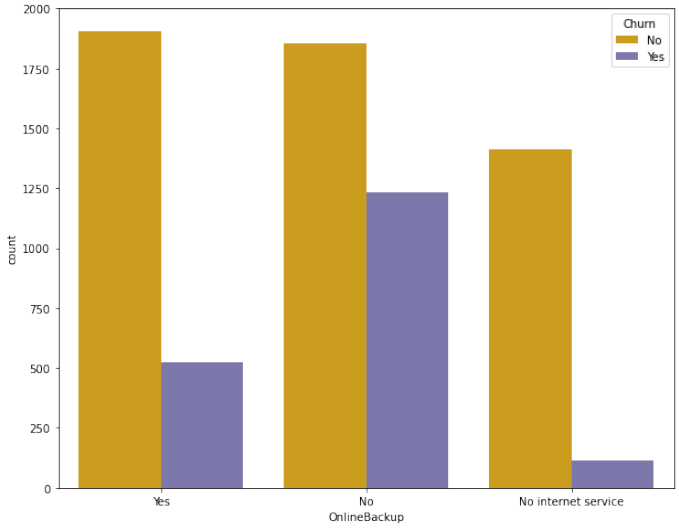
```
df['TotalCharges']=df['TotalCharges'].fillna(np.mean(df['TotalCharges']))

df.head()
```

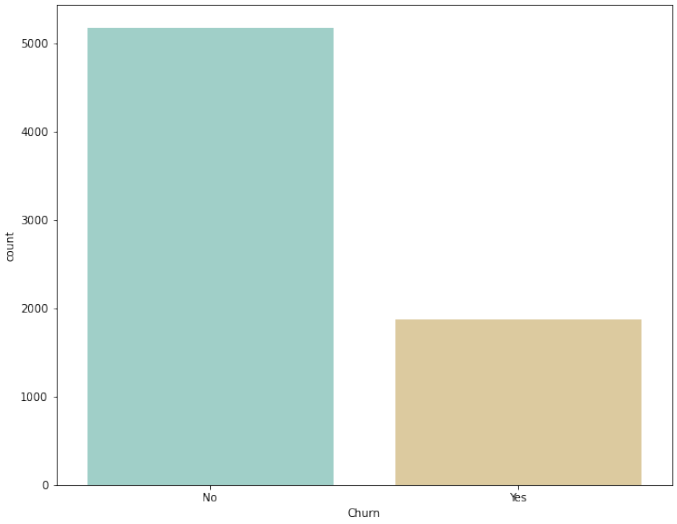


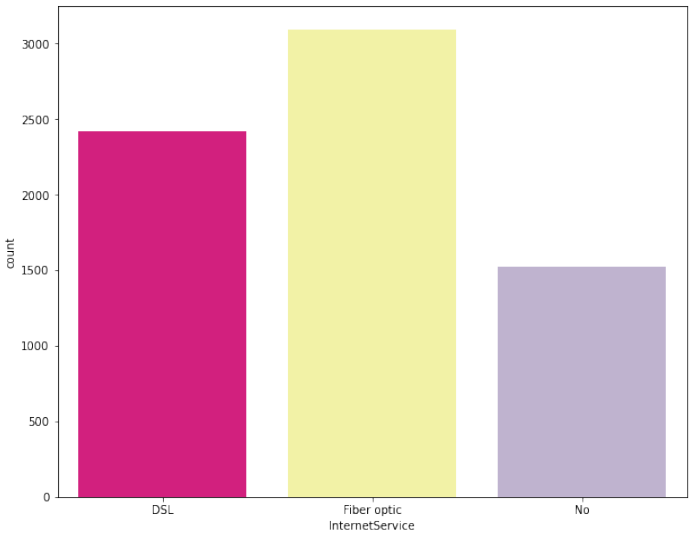
Finally there were no missing value then.

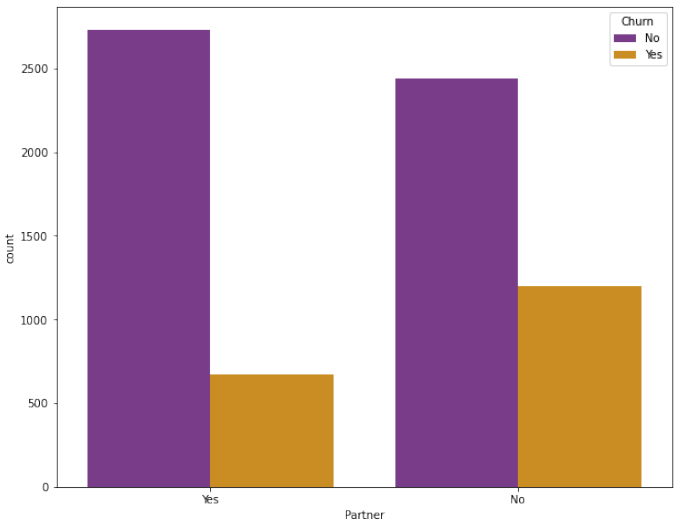
So then I filled that empty space with nan and again filled that nan values or missing values with mean.now my data is fulfilled with no missing values.

After all fixing missing values and reading all data I started visualization as to understand the graphical behaviour with churn rate for knowing what are the points related to customer churn.



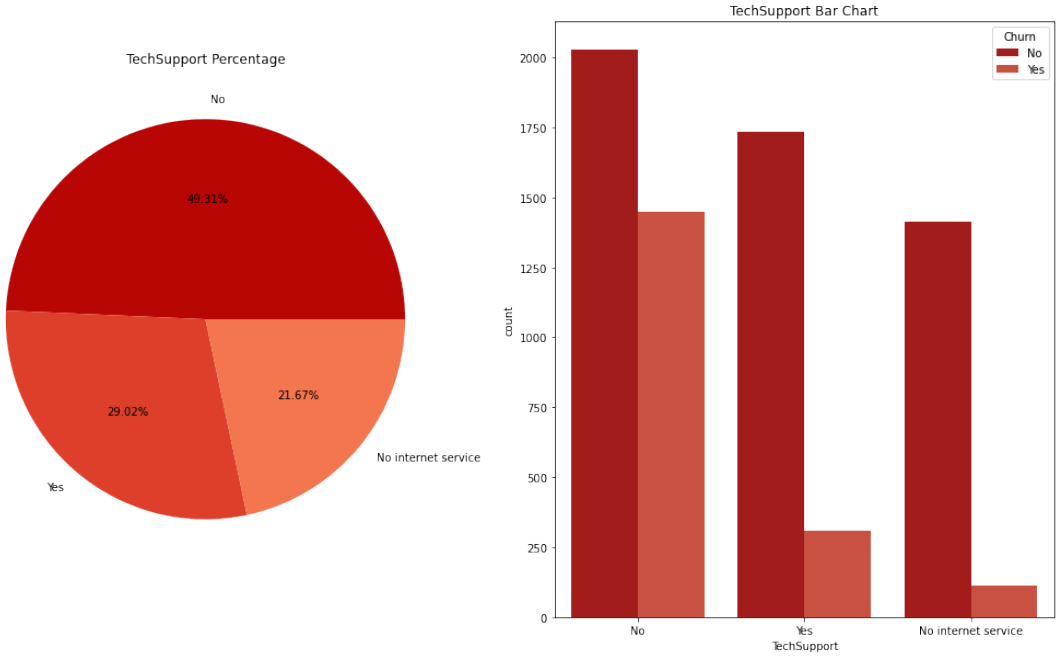
Plot relating churn with various aspects are shown here for better understanding.

Comments:

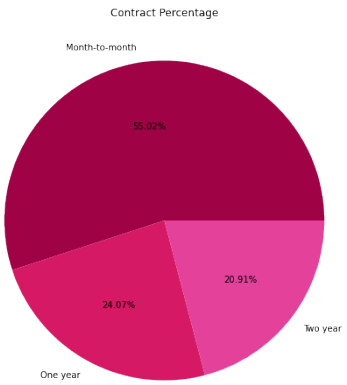
- Churning rate for the company are low but is still a problem.
- As we can see from above graphs company who has no online backups are having high churn rate.As customer need all benefits for their comfort and smooth process.
- Fiber optic service provider are also high which boom their company and attract new customers leading to customer churn for those company who don't have this in present.

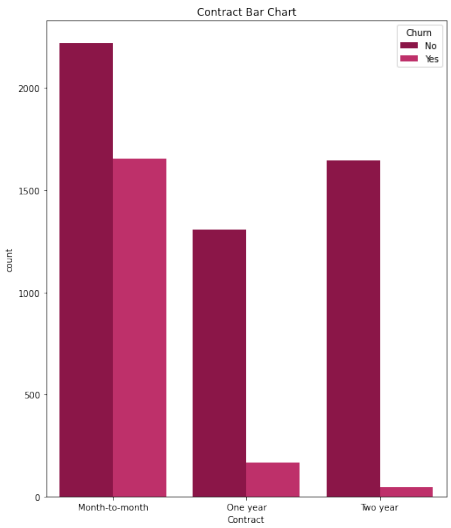
As the telecommunications sector is one of such sector where changing brand or moving to other company is relatively very easy and transparent market.Customers are more tend to attract to those company which gives more benefit and charge less.

During visualisation I found various factors which are same and can be dropped after correlation.



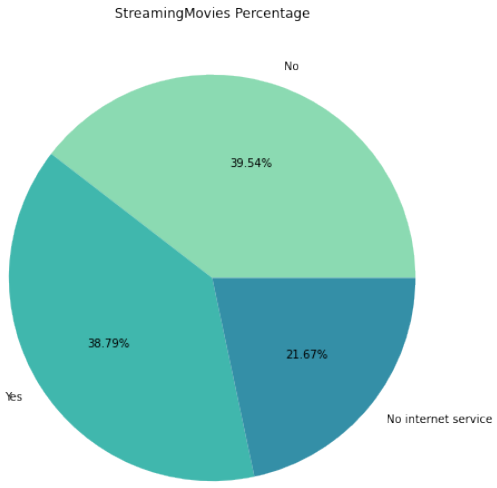

Company having tech support are having low churn rate.

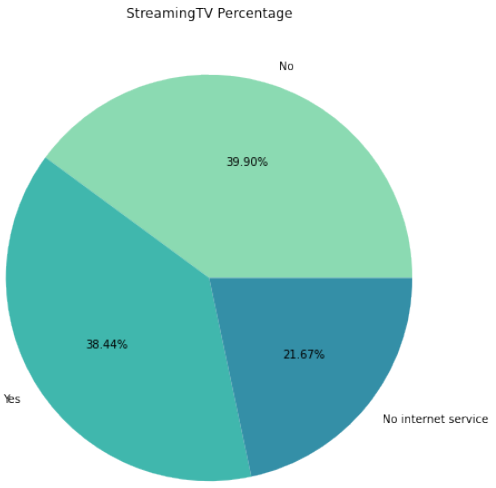



Contacting or subscription is the best way to reduce the churn rate. As month to month contract subscribers are more tending to churn as they can opt for other company at any moment.

```
In [5]: colors=sns.color_palette('mako_r')
        labels=df['StreamingMovies'].dropna().unique()

        plt.figure(figsize=(18,10))

        plt.subplot(1,2,1)
        plt.title('StreamingMovies Percentage')
        plt.pie(df['StreamingMovies'].value_counts(),
                labels=labels,
                colors=colors,
                autopct='%.2f%%')


        colors1=sns.color_palette('mako_r')
        labels1=df['StreamingTV'].dropna().unique()

        plt.subplot(1,2,2)
        plt.title('StreamingTV Percentage')
        plt.pie(df['StreamingTV'].value_counts(),
                labels=labels1,
                colors=colors1,
                autopct='%.2f%%')

        df.StreamingTV.value_counts(dropna=False),df.StreamingMovies.value_counts(dropna=False)

Out[5]: (No                     2810
         Yes                    2707
         No internet service    1526
         Name: StreamingTV, dtype: int64,
         No                     2785
         Yes                    2732
         No internet service    1526
         Name: StreamingMovies, dtype: int64)
```




Streaming movies and streaming tv are having equivalent data so decided to drop one after correlation.also the code written for plotting used to compare those categorial data.We can also see value counts of them.

Now,As majority are categorial columns , so first applying **preprocessing** and converting all categorical data to numerical by **Label Encoder.**

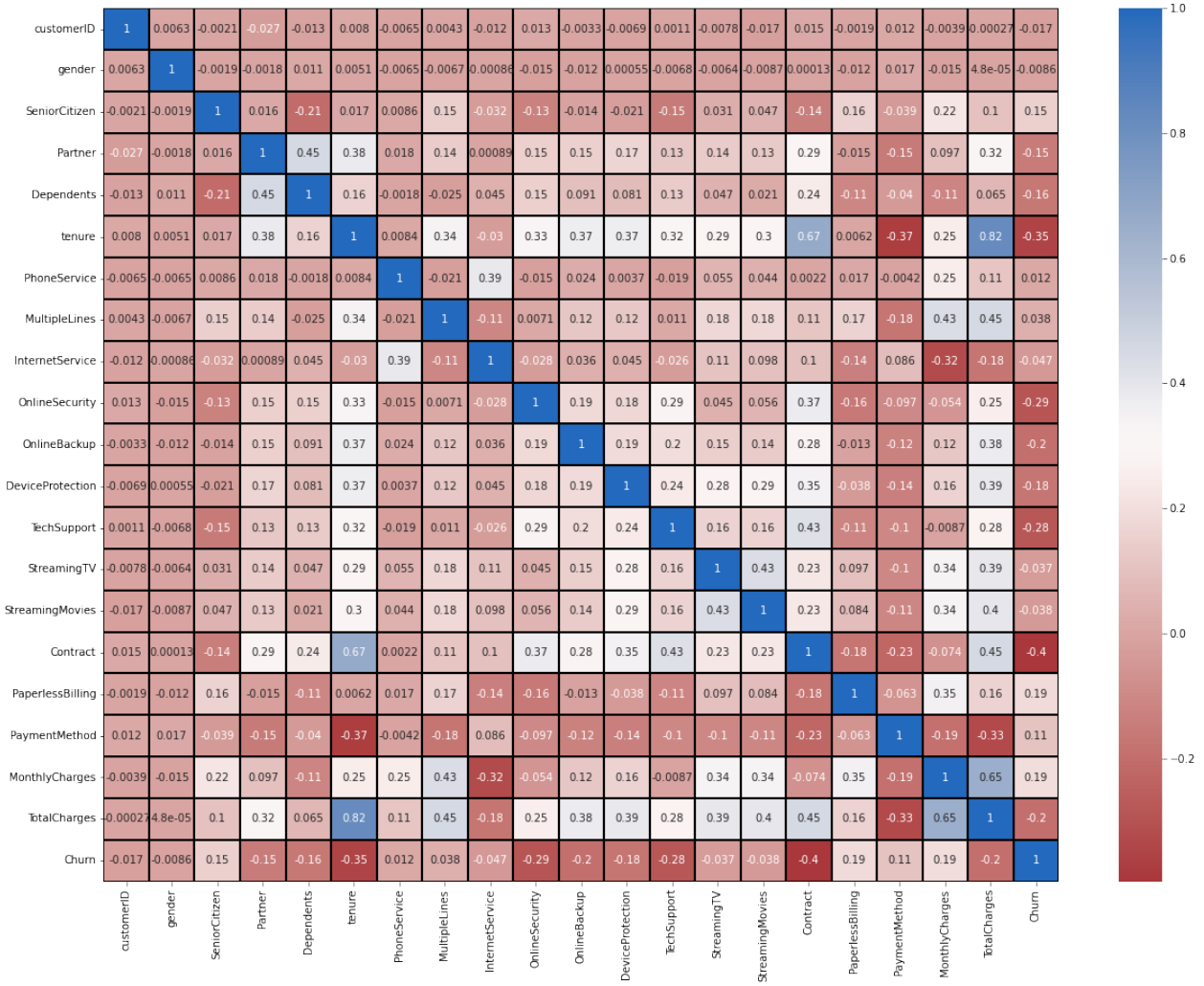
**Label Encoding**

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()

object=['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'Churn']


for i in object:
    df[i] = LE.fit_transform(df[i])

df.head()
```

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... | DeviceProtection | TechSu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5375 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | ... | 0 | |
| **1** | 3962 | 1 | 0 | 0 | 0 | 34 | 1 | 0 | 0 | 2 | ... | 2 | |
| **2** | 2564 | 1 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 2 | ... | 0 | |
| **3** | 5535 | 1 | 0 | 0 | 0 | 45 | 0 | 1 | 0 | 2 | ... | 2 | |
| **4** | 6511 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 1 | 0 | ... | 0 | |

5 rows × 21 columns

Next, The correlation between different features of the dataset showed that customers with low satisfaction level left. The correlation heat map is shown below:



From graph we can see that there are normal correlation between some columns, hence will drop after checking VIF.

- Streaming tv and streaming movies are having correlation near 0.2.
- Device protection and tech support are having correlation near 0.2.
- Online security and online backup are also having almost same values.
- Since cases of multicollinearity is seen ,later by applying VIF it we be solved.

# 3) EDA Concluding Remark:

- After EDA I came to conclusion that customers tend to shift from one company to other only because of some extra benefits or low pricing or easy and smooth process.
- Technical service to be provided to user.
- Relationship could be seen between many features by matplotlib and seaborn.

We have made all the data transformation by changing the categorical columns to numerical for further process . Now it is time to check the outliers and skewness in the dataset.

# 4) Pre-Processing Pipeline:

I used box-plots to identify the outliers and found no outliers as some were present in categorical datas which was due to label encoding so moving further with preparing dataset into training and testing set, separating features and target columns and then checking Skewness.

Skewness refers to a distortion or asymmetry that deviates from the symmetrical bell curve, or normal distribution, in a set of data. If the curve is shifted to the left or to the right, it is said to be skewed. Skewness can be quantified as a representation of the extent to which a given distribution varies from a normal distribution. A normal distribution has a skew of zero, while a lognormal distribution, for example, would exhibit some degree of right-skew.

# Separating Feature and Target column:

## Separating Dataset

```
#independent column
x=df.drop(['Churn'],axis=1)
#target
y=df['Churn']
```

```
x.shape
```

```
(7043, 20)
```

```
y.shape
```

```
(7043,)
```

# Skewness Check

I had used dist-plot to show skewness in columns ,and observed skewness in two columns :

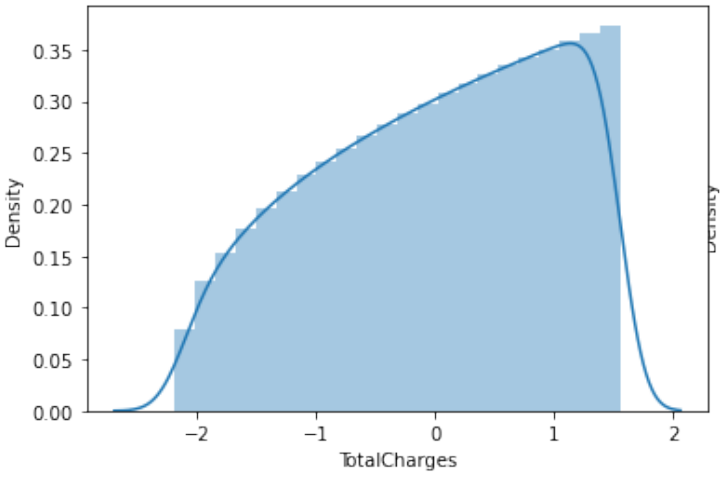
- Total charges
- Monthly charges

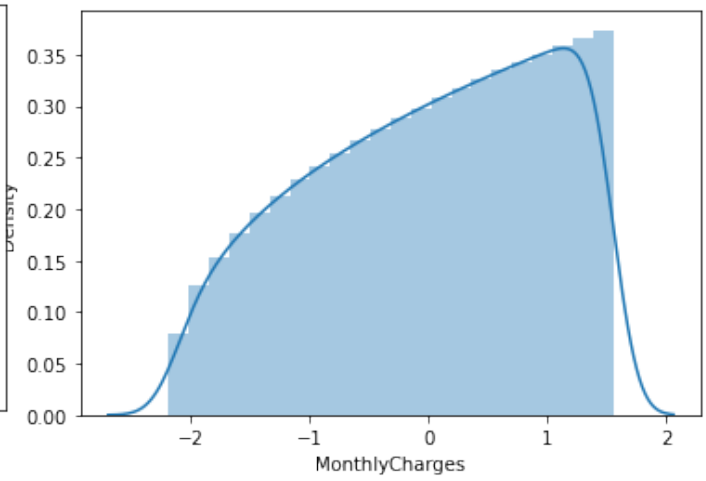
Hence removing skewness by power transform.

```
from sklearn.preprocessing import power_transform
df['MonthlyCharges']=power_transform(df,method='yeo-johnson')
df['TotalCharges']=power_transform(df,method='yeo-johnson')
df.head()
```

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecur |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5375 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| **1** | 3962 | 1 | 0 | 0 | 0 | 34 | 1 | 0 | 0 | |
| **2** | 2564 | 1 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | |
| **3** | 5535 | 1 | 0 | 0 | 0 | 45 | 0 | 1 | 0 | |
| **4** | 6511 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 1 | |

5 rows × 21 columns



Since the skewness of the data is in the acceptable range and the data is also normally distributed in the columns, in such case we can make use of Standard Scaler method else we can make use of Min Max scaler method.
Here we used standard scaler method.

# Standard Scaler method:

Standard Scaler helps to get standardised distribution, which makes mean = 0 and scales the data to unit variance. It helps in improving our model accuracy also solve the issue of data bias.

## Scalling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x
```

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | C |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.911890 | -1.009559 | -0.439916 | 1.034530 | -0.654012 | -1.277445 | -3.054010 | 0.062723 | -1.183234 | |
| **1** | 0.216906 | 0.990532 | -0.439916 | -0.966622 | -0.654012 | 0.066327 | 0.327438 | -0.991588 | -1.183234 | |
| **2** | -0.470701 | 0.990532 | -0.439916 | -0.966622 | -0.654012 | -1.236724 | 0.327438 | -0.991588 | -1.183234 | |
| **3** | 0.990587 | 0.990532 | -0.439916 | -0.966622 | -0.654012 | 0.514251 | -3.054010 | 0.062723 | -1.183234 | |
| **4** | 1.470632 | -1.009559 | -0.439916 | -0.966622 | -0.654012 | -1.236724 | 0.327438 | -0.991588 | 0.172250 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | 0.655145 | 0.990532 | -0.439916 | 1.034530 | 1.529024 | -0.340876 | 0.327438 | 1.117034 | -1.183234 | |
| **7039** | -0.981733 | -1.009559 | -0.439916 | 1.034530 | 1.529024 | 1.613701 | 0.327438 | 1.117034 | 0.172250 | |
| **7040** | -0.075745 | -1.009559 | -0.439916 | 1.034530 | 1.529024 | -0.870241 | -3.054010 | 0.062723 | -1.183234 | |
| **7041** | 1.186835 | 0.990532 | 2.273159 | 1.034530 | -0.654012 | -1.155283 | 0.327438 | 1.117034 | 0.172250 | |
| **7042** | -0.636946 | 0.990532 | -0.439916 | -0.966622 | -0.654012 | 1.369379 | 0.327438 | -0.991588 | 0.172250 | |

7043 rows × 20 columns

In the heat map we have found some features having high correlation between each other which leads to multicollinearity problem. In order to solve multicollinearity problem, we will check VIF values. If we find VIF values greater than 10 in any features that means the features causing multicollinearity issue in the data. To overcome with this problem, we need to drop that particular feature column.

# Checking Variance Influence Factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF values"] = [variance_inflation_factor(x.values,i)
                     for i in range(len(x.columns))]
vif["Features"] = x.columns

# Let's check the values
vif
```

| | VIF values | Features |
|---|---|---|
| 0 | 1.002557 | SeniorCitizen |
| 1 | 1.001974 | Dependents |
| 2 | 1.149760 | tenure |
| 3 | 1.462157 | PhoneService |
| 4 | 1.382370 | MultipleLines |
| 5 | 7.214940 | InternetService |
| 6 | 1.622496 | OnlineSecurity |
| 7 | 1.394365 | TechSupport |
| 8 | 1.826752 | StreamingMovies |
| 9 | 1.269022 | DeviceProtection |
| 10 | 1.218732 | customerID |
| 11 | 1.298269 | gender |
| 12 | 1.321581 | Partner |
| 13 | 1.446971 | OnlineBackup |
| 14 | 1.446455 | StreamingTV |
| 15 | 2.467740 | Contract |
| 16 | 1.203124 | PaperlessBilling |
| 17 | 1.187300 | PaymentMethod |
| 18 | 4.924796 | MonthlyCharges |
| 19 | 10.354925 | TotalCharges |

A variance inflation factor is a tool to help identify the degree of multicollinearity.The dependent variable is the outcome that is being acted upon by the independent variables—the inputs into the model. Multicollinearity exists when there is a linear relationship, or correlation, between one or more of the independent variables or inputs.

Here total charges was having via more than 10 so dropped it and moved further for balancing the dataset as our target column was bias.

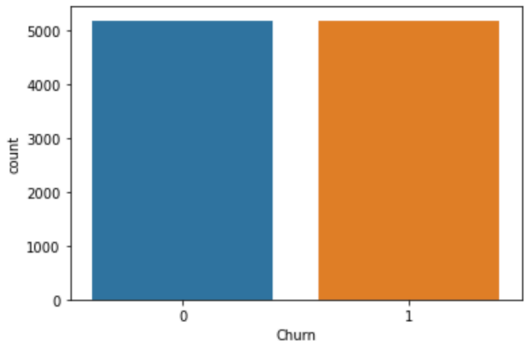
# Smote for oversampling

```
#SMOTE(synthetic Minority over sampling technique)
#It's use for Balancing the dataset

from imblearn.over_sampling import SMOTE
sm=SMOTE()

x_train,y_train=sm.fit_resample(x,y)
y_train
```

```
0        0
1        0
2        1
3        0
4        1
        ..
10343    1
10344    1
10345    1
10346    1
10347    1
Name: Churn, Length: 10348, dtype: int64
```

```
sns.countplot(y_train)
```

```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```

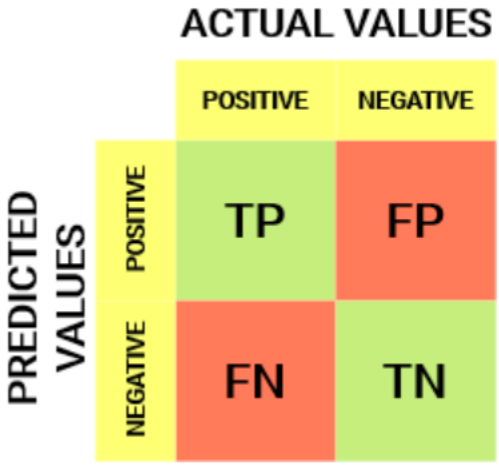



Now when our data set is balanced, so moving forward for model building and choosing the best model.

# 5) Building Machine Learning Models:

After making sure our data is good and ready we can continue to building our model. In this notebook I have build 7 model but will show best 4 different models with different algorithm. In this step we will create a baseline model for each algorithm using the default parameters set by sklearn and after building all 4 of our models we will compare them to see which works best for our case.

# Confusion matrix to evaluate our model




We will use 5 metrics below to evaluate models:

- Accuracy: the proportion of true results among the total number of cases examined.

$$Accuracy = \frac{Correct\ Predictions}{All\ Predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: used to calculate how much proportion of all data that was predicted positive was actually positive.

- Recall: used to calculate how much proportion of actual positives is correctly classified.

- F1 score: a number between 0 and 1 and is the harmonic mean of precision and recall.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

$TP$ = True positive

$TN$ = True negative

$FP$ = False positive

$FN$ = False negative

- Area Under Curve (AUC): indicates how well the probabilities from the positive classes are separated from the negative classes.

Now the dataset is ready for next step which is splitting dataset at a test size of 0.22 for best accuracy score, for which a code is set to find best random state.

## Model Building

```
# importing all the required libraries of models
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import neighbors
from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix

from sklearn.model_selection import train_test_split

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

```
# creating instances of models
models = [('lo',LogisticRegression()),('rfc',RandomForestClassifier()),('ETC',ExtraTreesClas
         ('bc',BaggingClassifier()),('dtc',DecisionTreeClassifier()),('knc',KNeighborsClassi
          ('gbc',GradientBoostingClassifier())]
```

```
#function to return best score of individual models
def model_selection(md,model):
    best_score=0
    for i in range(201):
        from sklearn.model_selection import train_test_split
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.22, random_s

        md = model
        md.fit(x_train,y_train)
        b_score=md.score(x_test,y_test)
        pred_md = md.predict(x_test)
        b_score
        if b_score>best_score:
            best_score=b_score
            random_state=i
    print("Best Score for model ",model," is = {} For Random state = {}".format(best_score*1
```

```
for i,j in models:
    model_selection(i,j)
```

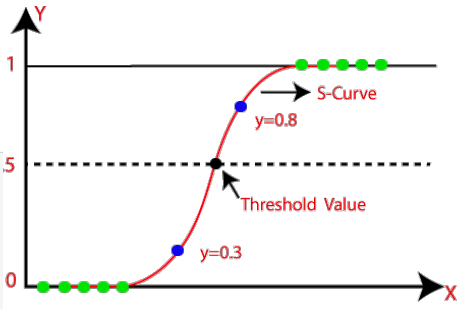
```
Best Score for model  LogisticRegression()  is = 82.51612903225806 For Random state = 35
Best Score for model  RandomForestClassifier()  is = 82.64516129032258 For Random state = 3
5
Best Score for model  ExtraTreesClassifier()  is = 80.51612903225806 For Random state = 56
Best Score for model  BaggingClassifier()  is = 80.7741935483871 For Random state = 1
Best Score for model  DecisionTreeClassifier()  is = 74.83870967741936 For Random state = 1
58
Best Score for model  KNeighborsClassifier()  is = 77.87096774193549 For Random state = 18
Best Score for model  GradientBoostingClassifier()  is = 82.3225806451613 For Random state
= 35
```

By running this code we get best random state for different models.

Now ,As I said in earlier par of this article I have build 7 model but will show only 4 model which gave best results :Logistic Regression, Random Forest, Support vector classifier and Gradient boosting .Before we start below are a simple definition of each algorithms and how they work. If you aren't familiar with any of the said algorithm you should definitely try to read more in-depth explanation about them before you continue.

# Logistic Regression

In statistics, the Logistic model is a statistical model that models the probability of one event taking place by having the log-odds for the event be a linear combination of one or more independent variables. In regression analysis, logistic regression is estimating the parameters of a logistic model.



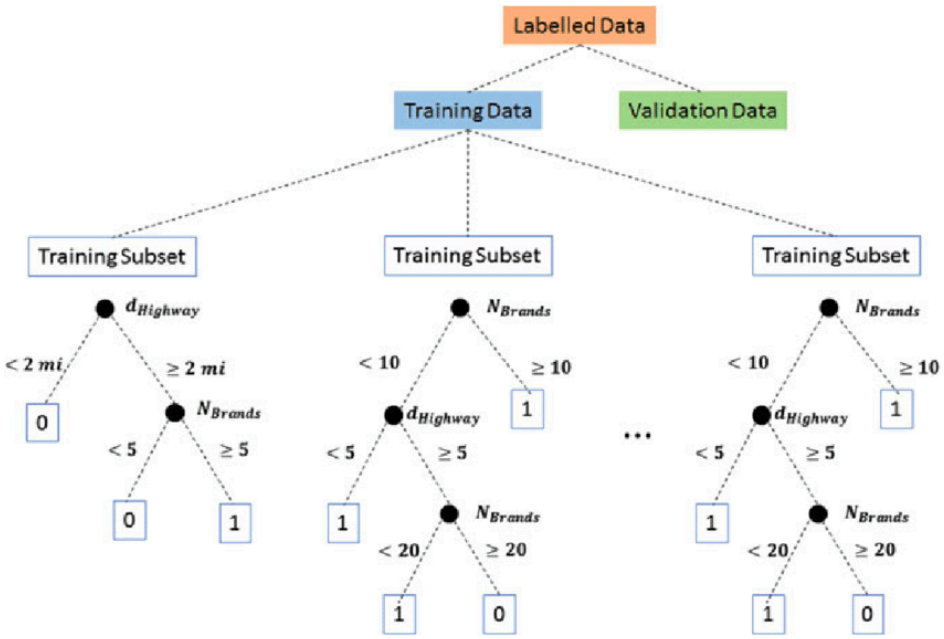

**1)LOGISTIC REGRESSION**

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix

lo=LogisticRegression()
lo.fit(x_train,y_train)
lo.score(x_train,y_train)
predlo=lo.predict(x_test)


print('Accuracy Score is''\n',accuracy_score(y_test,predlo))
print('Confusion Matrix''\n',confusion_matrix(y_test,predlo))
print(classification_report(y_test,predlo))
```

```
Accuracy Score is
 0.8251612903225807
Confusion Matrix
 [[1066  104]
 [ 167  213]]
              precision    recall  f1-score   support

           0       0.86      0.91      0.89      1170
           1       0.67      0.56      0.61       380

    accuracy                           0.83      1550
   macro avg       0.77      0.74      0.75      1550
weighted avg       0.82      0.83      0.82      1550
```

Accuracy score for Logistic Regression is 82.5%.

# RandomForest Classifier




Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

### 3) RandomForestClassifier

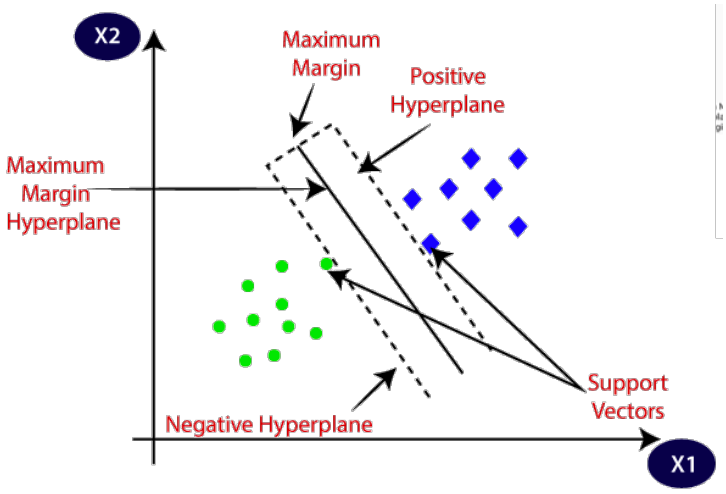
```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
pred_rfc=rfc.predict(x_test)
print('Accuracy score''\n',accuracy_score(y_test,pred_rfc))
print('Confusion Matrix''\n',confusion_matrix(y_test,pred_rfc))
print('Classification Report''\n',classification_report(y_test,pred_rfc))
```

```
Accuracy score
 0.82
Confusion Matrix
 [[1078   92]
 [ 187  193]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.92      0.89      1170
           1       0.68      0.51      0.58       380

    accuracy                           0.82      1550
   macro avg       0.76      0.71      0.73      1550
weighted avg       0.81      0.82      0.81      1550
```

It also have good accuracy score of 82%.

# SUPPORT VECTOR CLASSIFIER

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification or regression. It is mostly used in Classification and is called support vector classifier. In SVM the data is plotted in n (Number of features.) dimensional space where a hyperplane differentiates between classes very well. Then a maximum space is created for differentiating two classes this space is called margin.



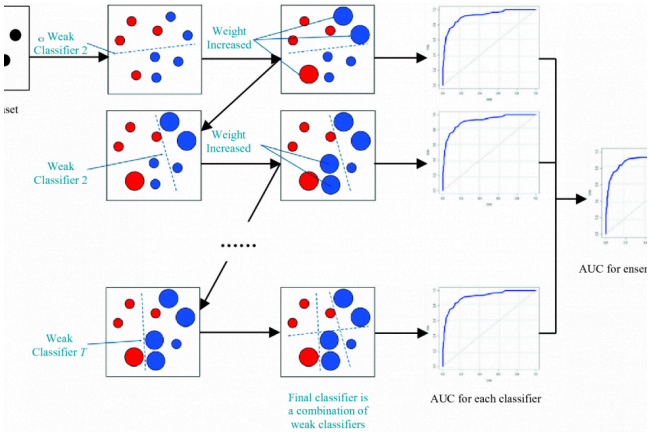

### 4) SUPPORT VECTOR CLASSIFIER

```
from sklearn.svm import SVC

svc=SVC()
svc.fit(x_train,y_train)
pred_svc=svc.predict(x_test)
print('Accuracy score''\n',accuracy_score(y_test,pred_svc))
print('Confusion Matrix''\n',confusion_matrix(y_test,pred_svc))
print('Classification Report''\n',classification_report(y_test,pred_svc))
```

```
Accuracy score
 0.8187096774193549
Confusion Matrix
 [[1086   84]
 [ 197  183]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      1170
           1       0.69      0.48      0.57       380

    accuracy                           0.82      1550
   macro avg       0.77      0.70      0.73      1550
weighted avg       0.81      0.82      0.81      1550
```

Thus SVM classifier gives us accuracy score of 81.87%

# GradientBoostingClassifier




Gradient boosting are the supervised machine learning algorithm that can be used for both regression and classification. It basically uses various weak learning models together and gives the best output. It has two necessary parts:1)a weak learner and 2) an additive component. The weak learner are decision trees and additive component are the gradient boosting model comes from the fact that trees are added to the model over time with manipulating existing trees and also minimizes errors.

## 5) GradientBoostingClassifier

```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
pred_gbk = gbk.predict(x_test)
print('Accuracy score''\n',accuracy_score(pred_gbk,y_test))
print('Confusion Matrix''\n',confusion_matrix(y_test,pred_gbk))
print('Classification Report''\n',classification_report(y_test,pred_gbk))
```

```
Accuracy score
 0.8232258064516129
Confusion Matrix
 [[1081   89]
 [ 185  195]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.92      0.89      1170
           1       0.69      0.51      0.59       380

    accuracy                           0.82      1550
   macro avg       0.77      0.72      0.74      1550
weighted avg       0.81      0.82      0.81      1550
```

Here, accuracy score is 82.32%

# Cross-validation

Cross-validation is a statistical method used to estimate the performance (or accuracy) of machine learning models. It is used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited.
Here we could check the best cv score and accuracy score which gives us the best model knowledge.
As best model is the one which has the least difference between accuracy score and cv score. Below are the cross validation results for all above model with best cross fold value:

**CROSS VALIDATION**

```
test_accuracy= accuracy_score(y_test,predlo)

from sklearn.model_selection import cross_val_score
for i in range(2,10):
    cv_score=cross_val_score(lo,x,y,cv=i,n_jobs=2)
    cv_mean=cv_score.mean()
    print(f"At cross fold {i} the Cross Val score is {cv_mean*100} and Accuracy score is {test_accuracy*100}")
```

# Logistic Regression

```
At cross fold 2 the Cross Val score is 80.02263453432082 and Accuracy score is 82.51612903225806
At cross fold 3 the Cross Val score is 80.3066947620254 and Accuracy score is 82.51612903225806
At cross fold 4 the Cross Val score is 80.34936244902173 and Accuracy score is 82.51612903225806
At cross fold 5 the Cross Val score is 80.39190028388929 and Accuracy score is 82.51612903225806
At cross fold 6 the Cross Val score is 80.16478324287768 and Accuracy score is 82.51612903225806
At cross fold 7 the Cross Val score is 80.2498951531272 and Accuracy score is 82.51612903225806
At cross fold 8 the Cross Val score is 80.30656665978742 and Accuracy score is 82.51612903225806
At cross fold 9 the Cross Val score is 80.36354008906949 and Accuracy score is 82.51612903225806
```

```
test_accuracy= accuracy_score(y_test,pred_rfc)

from sklearn.model_selection import cross_val_score
for i in range(2,10):
    cv_score=cross_val_score(rfc,x,y,cv=i,n_jobs=2)
    cv_mean=cv_score.mean()
    print(f"At cross fold {i} the Cross Val score is {cv_mean*100} and Accuracy score is {test_accuracy*100}")
```

# Random Forest

```
At cross fold 2 the Cross Val score is 78.88681942578326 and Accuracy score is 82.0
At cross fold 3 the Cross Val score is 79.3269864727574 and Accuracy score is 82.0
At cross fold 4 the Cross Val score is 79.14268590934903 and Accuracy score is 82.0
At cross fold 5 the Cross Val score is 79.19937374991936 and Accuracy score is 82.0
At cross fold 6 the Cross Val score is 79.35563233515019 and Accuracy score is 82.0
At cross fold 7 the Cross Val score is 79.55412651061991 and Accuracy score is 82.0
At cross fold 8 the Cross Val score is 79.27027976988958 and Accuracy score is 82.0
At cross fold 9 the Cross Val score is 79.19936182961533 and Accuracy score is 82.0
```

```
test_accuracy= accuracy_score(y_test,pred_svc)

from sklearn.model_selection import cross_val_score
for i in range(2,10):
    cv_score=cross_val_score(svc,x,y,cv=i)
    cv_mean=cv_score.mean()
    print(f"At cross fold {i} the Cross Val score is {cv_mean*100} and Accuracy score is {test_accuracy*100}")
```

# Support Vector Classifier

```
At cross fold 2 the Cross Val score is 79.31277025121115 and Accuracy score is 81.87096774193549
At cross fold 3 the Cross Val score is 79.6962280069498 and Accuracy score is 81.87096774193549
At cross fold 4 the Cross Val score is 79.9802701202829 and Accuracy score is 81.87096774193549
At cross fold 5 the Cross Val score is 79.76739426737208 and Accuracy score is 81.87096774193549
At cross fold 6 the Cross Val score is 79.92346730065503 and Accuracy score is 81.87096774193549
At cross fold 7 the Cross Val score is 79.49740907653808 and Accuracy score is 81.87096774193549
At cross fold 8 the Cross Val score is 79.75325043855123 and Accuracy score is 81.87096774193549
At cross fold 9 the Cross Val score is 79.69657146735274 and Accuracy score is 81.87096774193549
```

```
test_accuracy= accuracy_score(y_test,pred_gbk)
```

**# Gradient Boosting Classifier**

```
from sklearn.model_selection import cross_val_score
for i in range(2,10):
    cv_score=cross_val_score(gbk,x,y,cv=i,n_jobs=2)
    cv_mean=cv_score.mean()
    print(f"At cross fold {i} the Cross Val score is {cv_mean*100} and Accuracy score is {test_accuracy*100}")
```

```
At cross fold 2 the Cross Val score is 80.10790211275545 and Accuracy score is 82.3225806451613
At cross fold 3 the Cross Val score is 79.96600345457743 and Accuracy score is 82.3225806451613
At cross fold 4 the Cross Val score is 79.92341967373909 and Accuracy score is 82.3225806451613
At cross fold 5 the Cross Val score is 80.07969264146072 and Accuracy score is 82.3225806451613
At cross fold 6 the Cross Val score is 80.22160546810136 and Accuracy score is 82.3225806451613
At cross fold 7 the Cross Val score is 79.98004313458166 and Accuracy score is 82.3225806451613
At cross fold 8 the Cross Val score is 80.0511589361263 and Accuracy score is 82.3225806451613
At cross fold 9 the Cross Val score is 80.13638424070463 and Accuracy score is 82.3225806451613
```

Now all models are performing good ,picking the best for hyper-parameter.i.e, **Gradient Boosting Classifier** as it has the least difference between cv score and accuracy score.

# Hyper Parameter Tuning for best model

Thus Gradient Boosting showed the best result after hyper parameter and so plotting arc roc curve for all models.

### Hyper Parameter Tuning for best model

```
#GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

#creating the parameter list to pass in GrideSearchcv
parameters={'max_features':['auto','sqrt','log2'],
            'max_depth':[4,5,6,7,8,9,10]}

gcv=GridSearchCV(GradientBoostingClassifier(),parameters,cv=5,scoring="accuracy")
gcv.fit(x_train,y_train)
gcv.best_params_
```

```
{'max_depth': 4, 'max_features': 'log2'}
```

```
gcv_pred=gcv.best_estimator_.predict(x_test)#predicting with best parameters
accuracy_score(y_test,gcv_pred)#checking the final score

print('Accuracy score''\n',accuracy_score(y_test,gcv_pred))
print('Confusion Matrix''\n',confusion_matrix(y_test,gcv_pred))
print('Classification Report''\n',classification_report(y_test,gcv_pred))
```

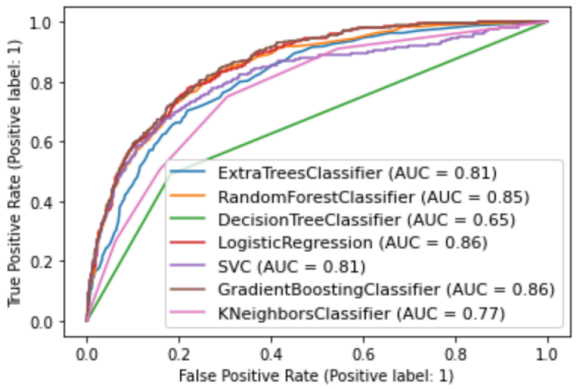
```
Accuracy score
 0.827741935483871
Confusion Matrix
 [[1075   95]
 [ 172  208]]
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.92      0.89      1170
           1       0.69      0.55      0.61       380

    accuracy                           0.83      1550
   macro avg       0.77      0.73      0.75      1550
weighted avg       0.82      0.83      0.82      1550
```

Thus after hyperparmeter tuning accuracy score for gradient boosting classifier is increased to 82.72%.

# Arc roc curve:

**COMPARING ROC AUC PLOT WITH MODELS**

```
# Plotting for all the models used here
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn.metrics import plot_roc_curve


disp = plot_roc_curve(ETC,x_test,y_test)     # ax_=Axes with confusion matrix
plot_roc_curve(rfc, x_test, y_test, ax=disp.ax_)
plot_roc_curve(dt, x_test, y_test, ax=disp.ax_)
plot_roc_curve(lo, x_test, y_test, ax=disp.ax_)
plot_roc_curve(svc, x_test, y_test, ax=disp.ax_)
plot_roc_curve(gbk, x_test, y_test, ax=disp.ax_)
plot_roc_curve(knc, x_test, y_test, ax=disp.ax_)

plt.legend(prop={'size':11}, loc='lower right')
plt.show()
```




Thus in arc roc curve also Gradient Boosting is having highest score. So now saving the model with it.

# Saving the best Model

```
# save the model to disk
import pickle

filename = 'Customer_churn.pkl'
pickle.dump(gcv.best_estimator_, open(filename, 'wb'))
```

```
# load the model from disk

loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(x_test,y_test)
print(result)
```

```
0.827741935483871
```

Saving the model with pickle and then loading again for prediction.

### Final Prediction Submission

```
import numpy as np
a = np.array(y_test)
predicted = np.array(loaded_model.predict(x_test))
df = pd.DataFrame(zip(a,predicted), columns = ["Original","Predicted"])
df
```

| | Original | Predicted |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 0 |
| **2** | 0 | 0 |
| **3** | 0 | 0 |
| **4** | 1 | 1 |
| **...** | ... | ... |
| **1545** | 0 | 0 |
| **1546** | 0 | 0 |
| **1547** | 0 | 0 |
| **1548** | 1 | 1 |
| **1549** | 0 | 1 |

1550 rows × 2 columns

Final submission is showing the comparison between original and production for customer churn.

# 6) Concluding Remarks:

- The accuracy of the technique is critical to the success of any proactive retention efforts. If the Marketer is unaware of a customer about to churn, no action will be taken to retain that customer.
- To succeed at retaining customers who are ready to abandon your business, Marketers & Customer Success experts must be able to predict in advance which customers are going to churn and set up a plan of marketing actions that will have the greatest retention impact on each customer. The key here is to to be proactive and engage with these customers.
- Special retention-focused offers or incentives may be provided to happy, active customers, resulting in reduced revenues for no good reason

- Bringing it all together, predicting customer churn is important. Effective action can be taken to retain the customer before it is too late. The ability to predict that a customer is at high risk of churning while there is still time to do something about it represents a huge additional potential revenue source for every online business.

# Thank you