CUSTOMER CHURN ANALYSIS

Customer churn is measured using customer churn rate. That's the number of people who stopped being customers during a set period of time, such as a year, a month, or a financial quarter.

Well, it's important because it costs more to acquire new customers than it does to retain existing customers. In fact, an increase in customer retention of just 5% can create at least a 25% increase in profit. This is because returning customers will likely spend 67% more on your company's products and services.

Problem statement:

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

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

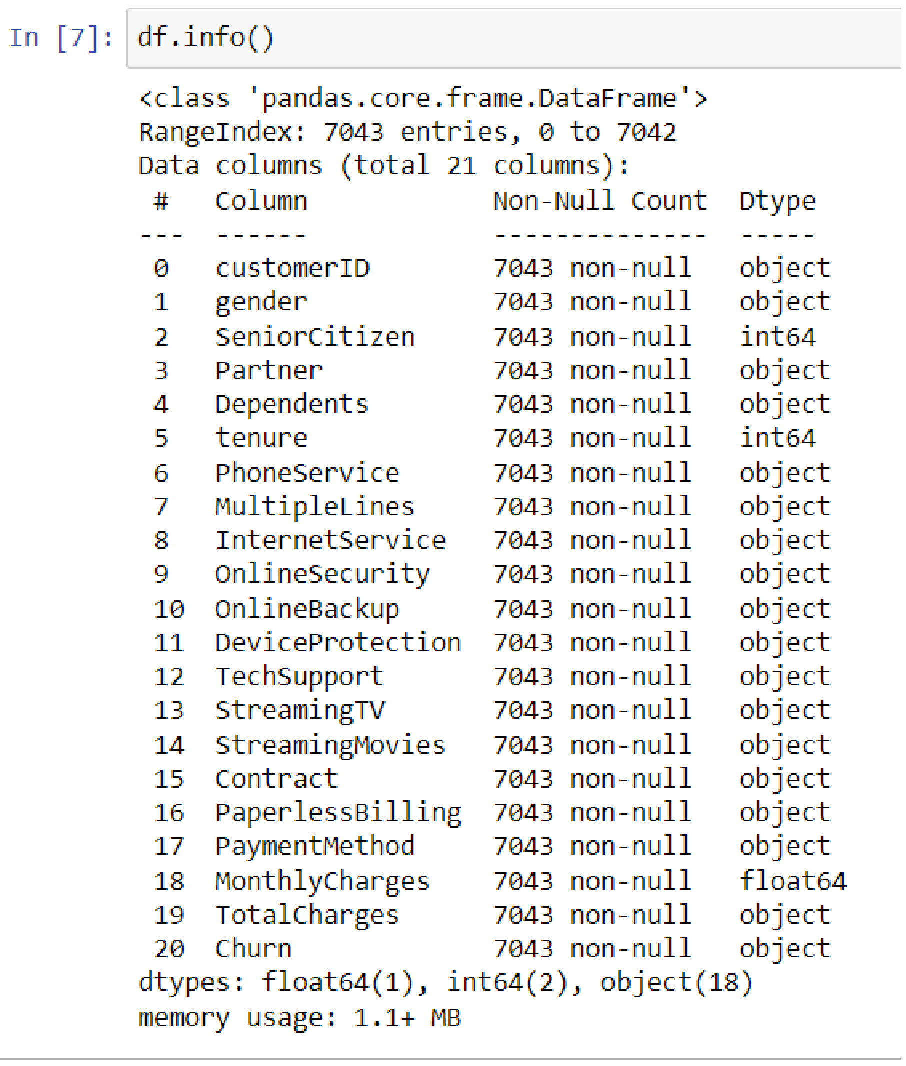
Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

Now will examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

DATA ANALYSIS

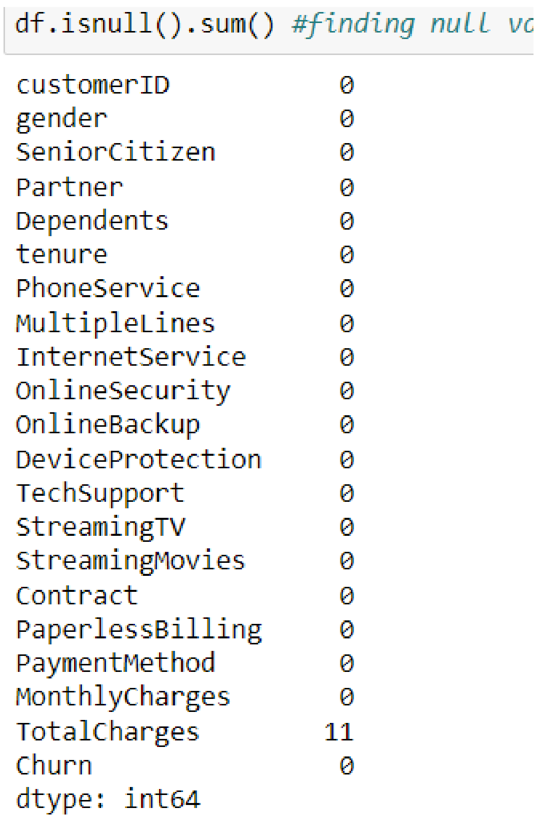
We have been given with a dataset which contains the details of the customer churn analysis data along with the customer details. It also has the details of the customers stop doing business with that company.

The provided dataset contains 7,043 rows and 21 columns . Some of the column's name in the dataset are Gender, Tenure, customer ID, churn, total payments and internet service etc.



```
In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

As per our observations, we found that some null values or missing values were present in the dataset as values '?'.



```
df.isnull().sum() #finding null vc
```

```
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
```

The null values are present in the column Total Charges

# Changing the datatype of the column 'TotalCharges' from Object to float and replacing the blank values with NaN

df["TotalCharges"]=df["TotalCharges"].apply(lambda x: float(x) if len(x)>1 else np.nan) and the Heat map is below.

There are many featured columns but our target column is "churn".

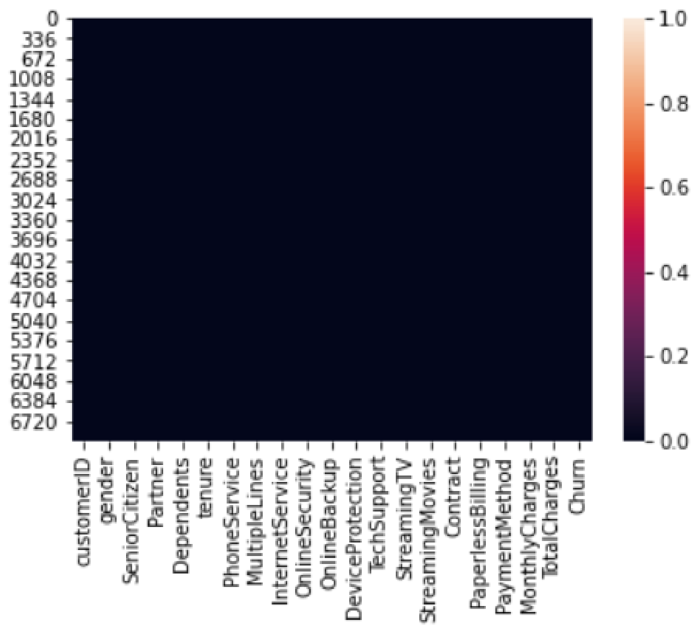
We found that there are 11 missing values in the Total Charges of the column of the dataset and there are no missing values in other columns.

Dataset contains float, integer and object data type.

There are 17 object datatype columns, 2 integer datatype columns, and 2 float datatype columns.

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

```
<AxesSubplot:>
```



```
# Let's locate the rows of the 'TotalCharges' column which contain missing values:

df.loc[df['TotalCharges'].isna(),['tenure', 'MonthlyCharges', 'TotalCharges']]
```

	tenure	MonthlyCharges	TotalCharges
488	0	52.55	NaN
753	0	20.25	NaN
936	0	80.85	NaN
1082	0	25.75	NaN
1340	0	56.05	NaN
3331	0	19.85	NaN
3826	0	25.35	NaN
4380	0	20.00	NaN
5218	0	19.70	NaN
6670	0	73.35	NaN
6754	0	61.90	NaN

```
# Since the value of 'tenure' is 0 so we'll replace the missing values of the 'TotalCharges' column with 0.

df['TotalCharges'].fillna(0, inplace=True)
```

STATISTICAL SUMMARY

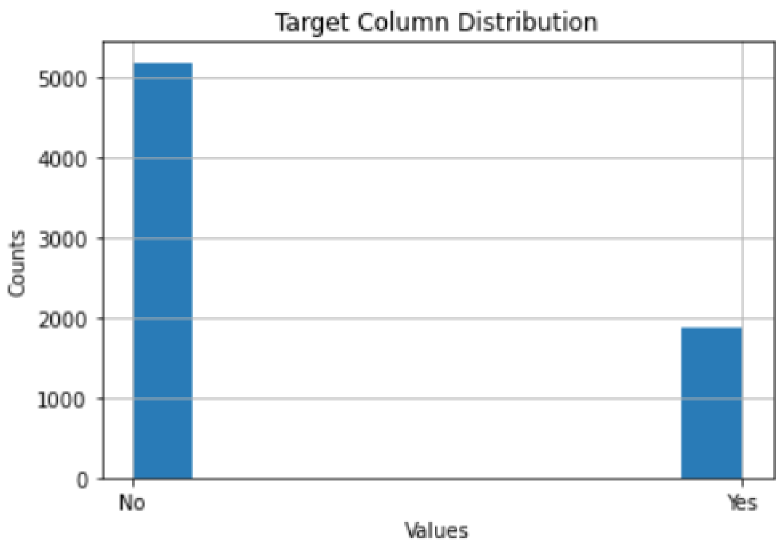
```
df.describe() #statistical summary
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2279.734304
std	0.368612	24.559481	30.090047	2266.794470
min	0.000000	0.000000	18.250000	0.000000
25%	0.000000	9.000000	35.500000	398.550000
50%	0.000000	29.000000	70.350000	1394.550000
75%	0.000000	55.000000	89.850000	3786.600000
max	1.000000	72.000000	118.750000	8684.800000

EDA(EXPLORATORY DATA ANALYSIS)

```
# Plotting histogram to show the distribution of target columnn

plt.figure(figsize=[6,4])
df['Churn'].hist(grid=True)
plt.xlabel('Values')
plt.ylabel('Counts')
plt.title('Target Column Distribution')
```

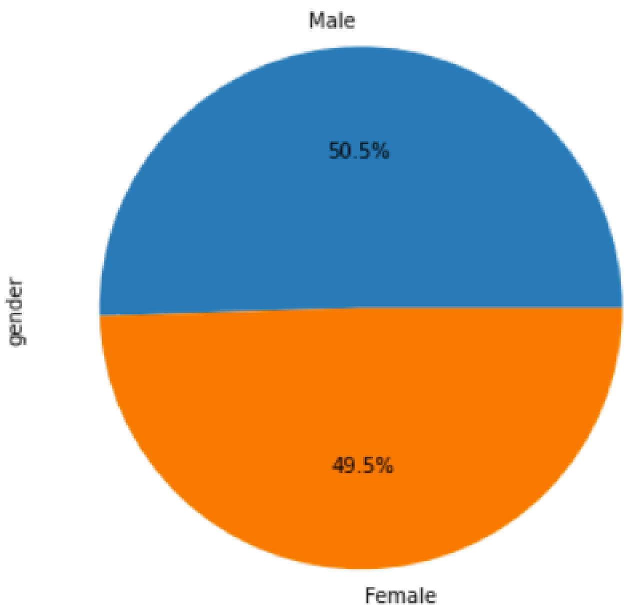
```
Text(0.5, 1.0, 'Target Column Distribution')
```



The above graph shows the Target column is imbalance and we have to balance it.

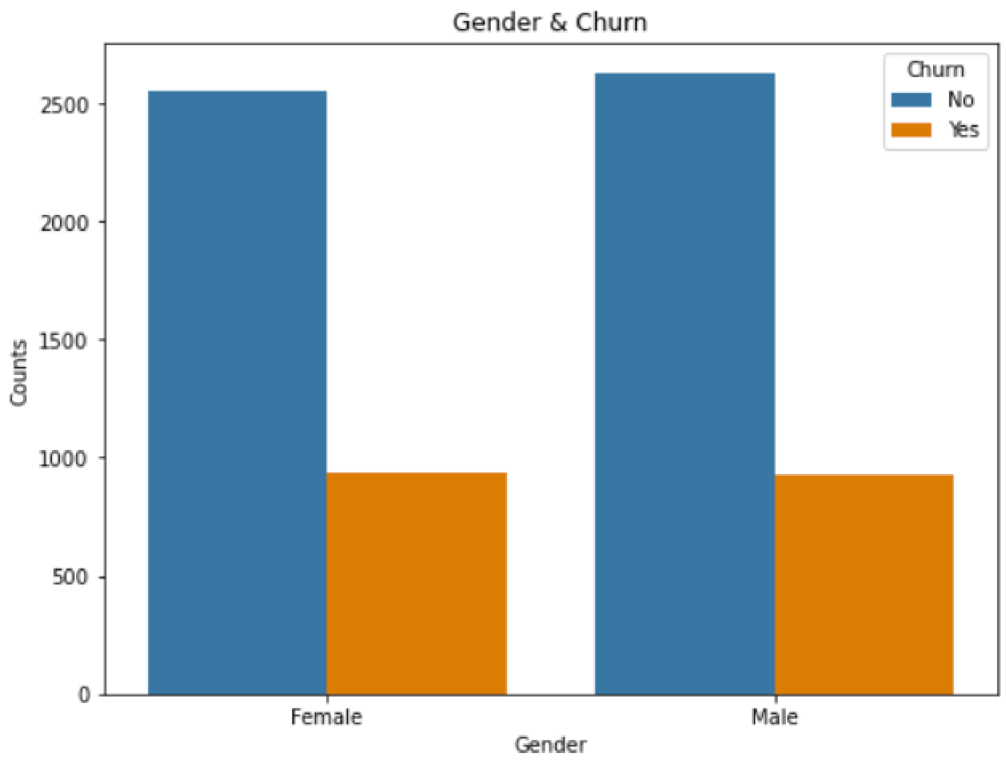
Distribution of gender column:

```
<AxesSubplot:ylabel='gender'>
```



The distribution of male and female dataset is almost same.

```
# Checking for the gender wise distribution of the target column

plt.figure(figsize=[8,6])
sns.countplot(x=df['gender'], hue=df['Churn'], data=df)
plt.title('Gender & Churn')
plt.xlabel('Gender')
plt.ylabel('Counts')
plt.show()
```

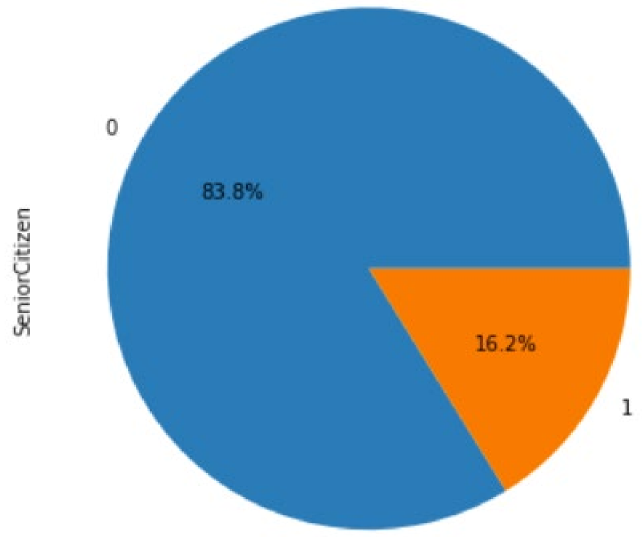


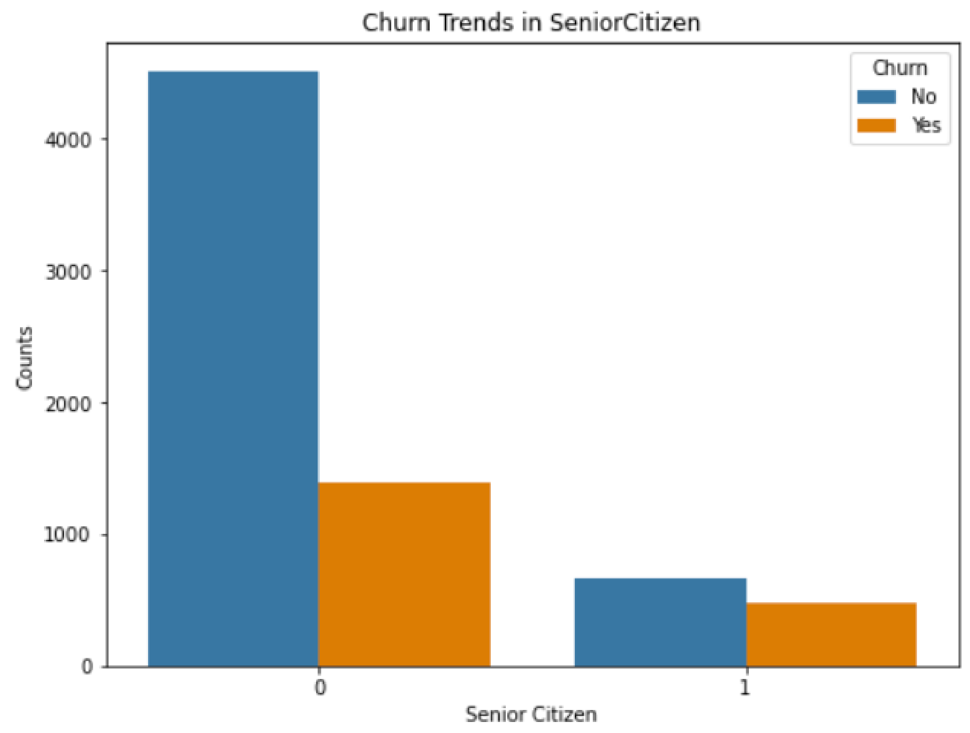
Most of the male and female prefer the same telecom company.

Senior citizen column:

Most of the customers nearly 83% are below 60 years of age and the remaining 16% are senior citizens.

We can examine by observing the below graph

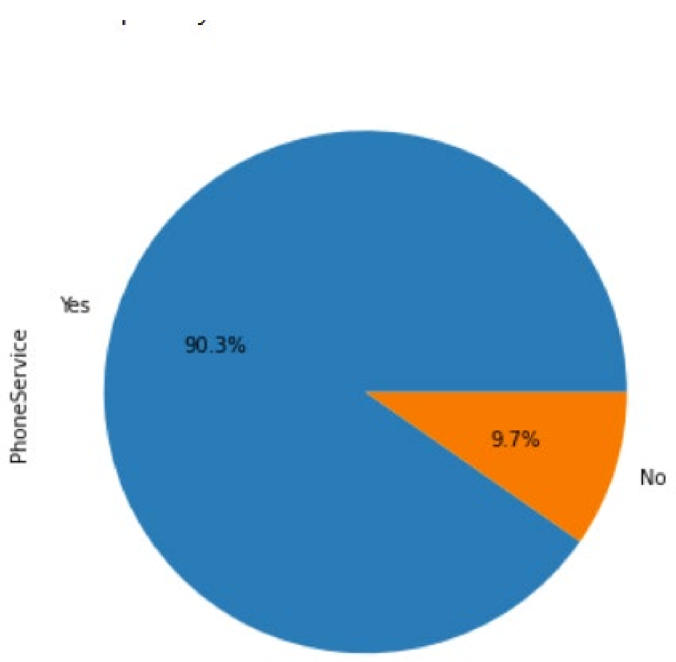




The percentage of churn trend in the customers who are not senior citizens is less.

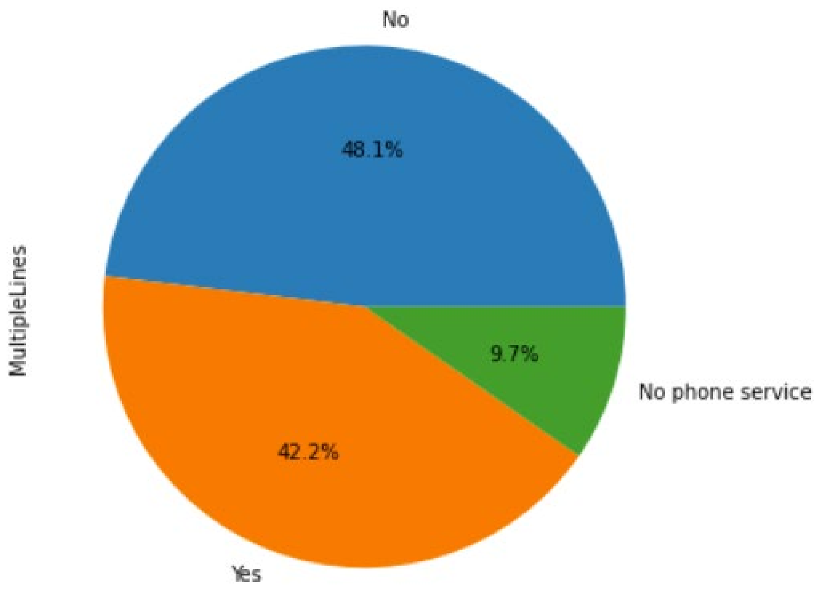
In case of senior citizen the trend of churn is very high.

Phone service column:



The above graph shows 90% have opted for the phone service.

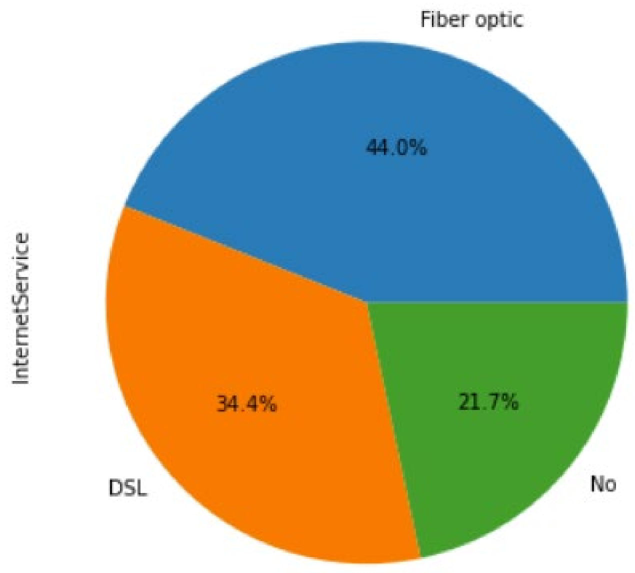
Multiple lines column:



- 42.2% customers are using multiple lines service.

- 48.1% customers are using single line.
- 9.7% customers are not using phone service.

Internet service column:



44% are using Fiber optic internet

34.4% are using DSL internet service

21.7% are not using internet service

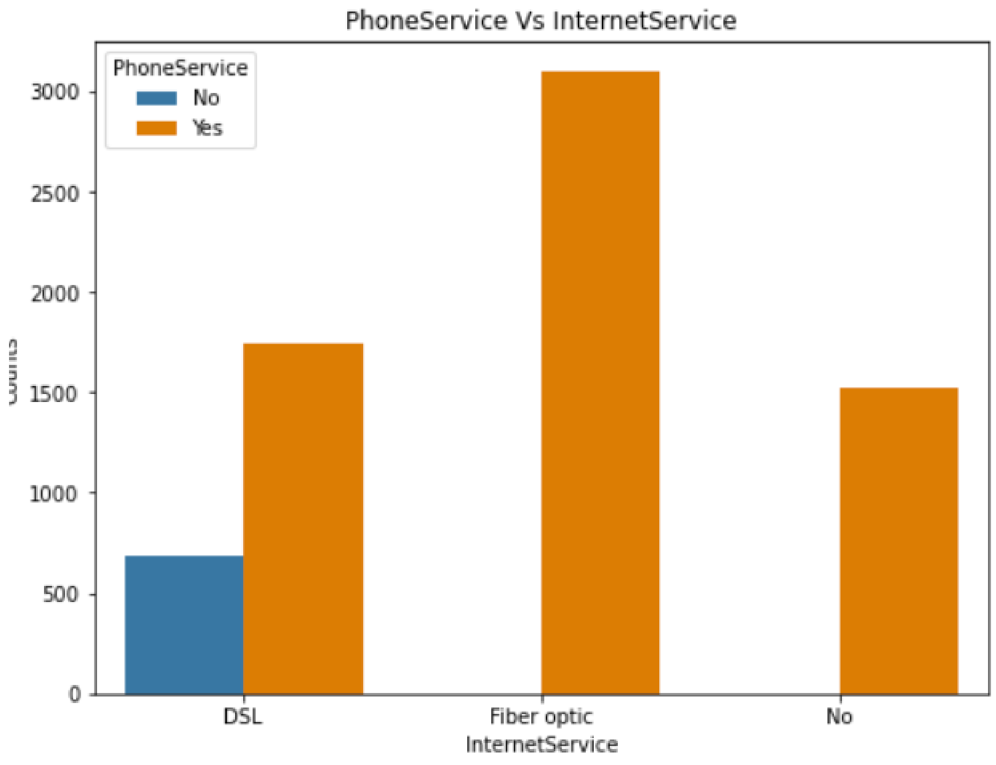
Checking the relation b/n phone service and internet service:

Approx 1500 customers is using phone service but they are not using any internet service.

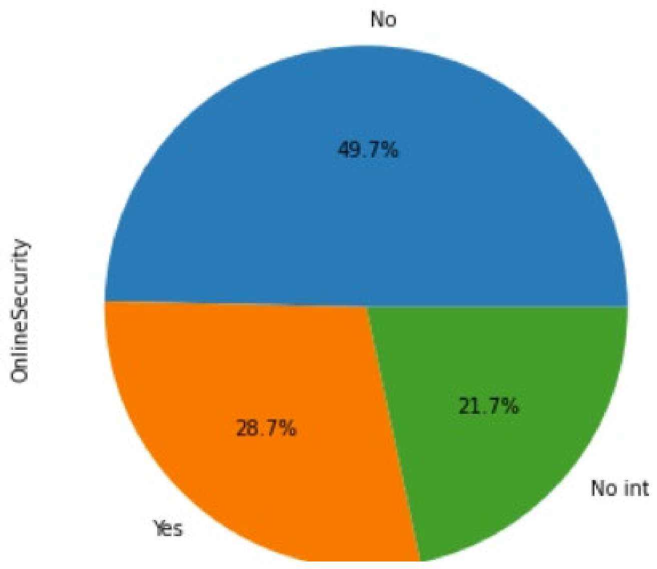
All the customers who are using fibre optic internet connection is also using phone service.

Most of the customers who are using DSL internet connection are also using phone service but some of them are not using phone service they are only using internet service.

The graph is plotted below:



Internet security:



28.7% having online security where as 49.7% having no security and moreover 21.7% are not having internet service.

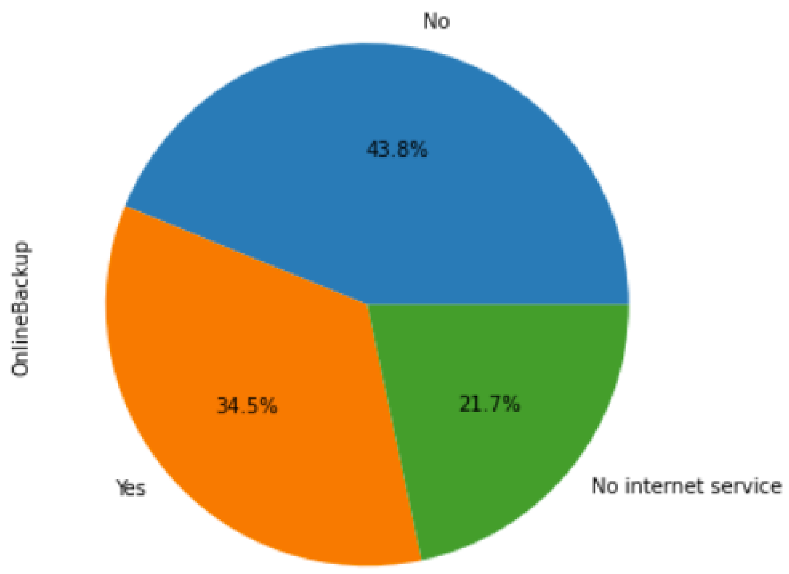
ONLINE BACKUP:

From the below graph we observe that

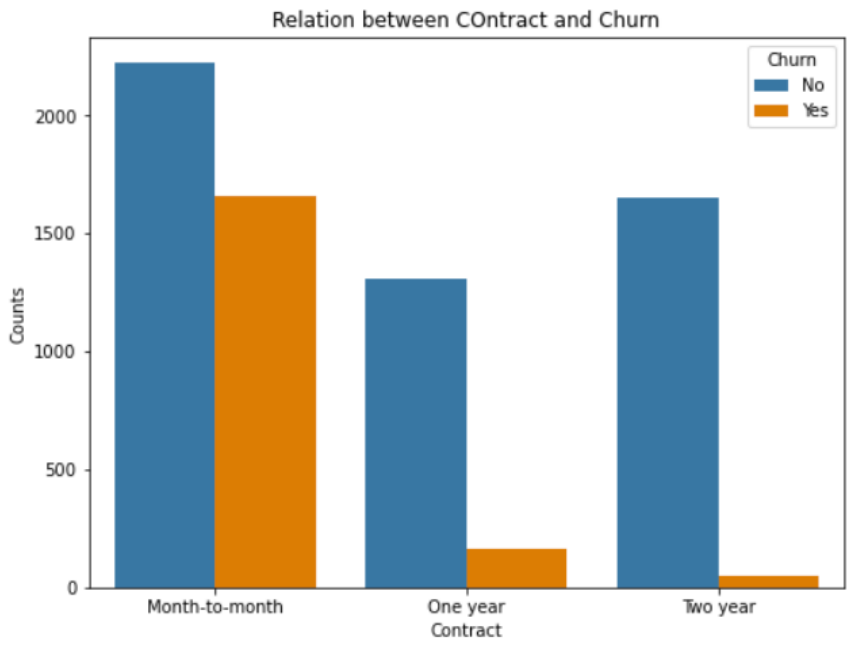
34.5% are having online backup

43.8% are not having online backup

```
<AxesSubplot:ylabel='OnlineBackup'>
```



RELATION B/N CUSTOMER CHURN AND CONTRACT:



From the graph we can say that customers having monthly contract have high rate of churn as compared to the customers having one or two years of contract.

Data Pre-processing:

Data pre-processing is very important step in Machine Learning to get highly accurate and insightful results. The greater quality of data will help us to make highly efficient model. This will lead to the highly reliability of the predicted results.

Incomplete, Noisy, and Inconsistent data will lead to the inherent nature of real-world problems. Data pre-processing help us in increasing the quality of data by treating the missing values, removing the noises, and resolving inconsistency.

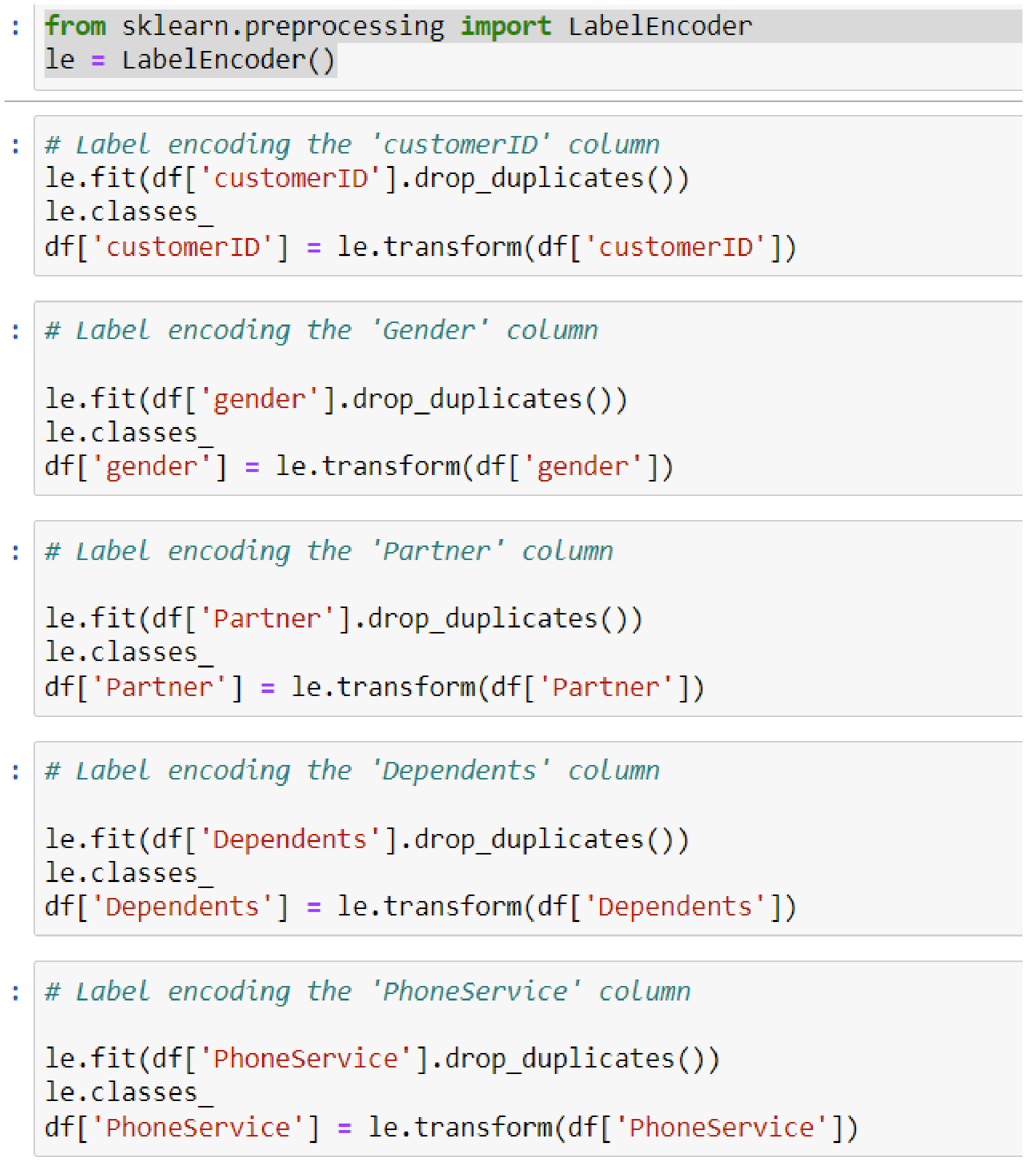
- Incomplete Data: There can be a number of reasons for this. There may be some misunderstanding or technical fault which lead to the incomplete data.
- Noisy Data: This can be the incorrect feature values present in the dataset. There can be many reasons for this. The instrument used for the data collection might be faulty. There may be human error at the time of data entry.

Stages of Data Pre-processing:

- Data Cleaning: In this stage we work on the dataset to remove the unwanted feature columns (columns which are not contributing to the dataset), treating the missing or null values, removing the outliers and skewness if present.
- Data integration: integrates data from a multitude if source into single data warehouse.
- Data transformation: such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency ofmining algorithms involving distance measurement.
- Data reduction: Reduce the data size by dropping out redundant features. Feature selection and feature extraction technique can be used.

- Label Encoding: (Converting the categorical features to numerical):

- In dataset there can be some columns which will be containing the categorical columns (columns with the datatype of object). For machine learning model we need to convert the categorical data to numerical. We can use label encoding for the conversion.



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

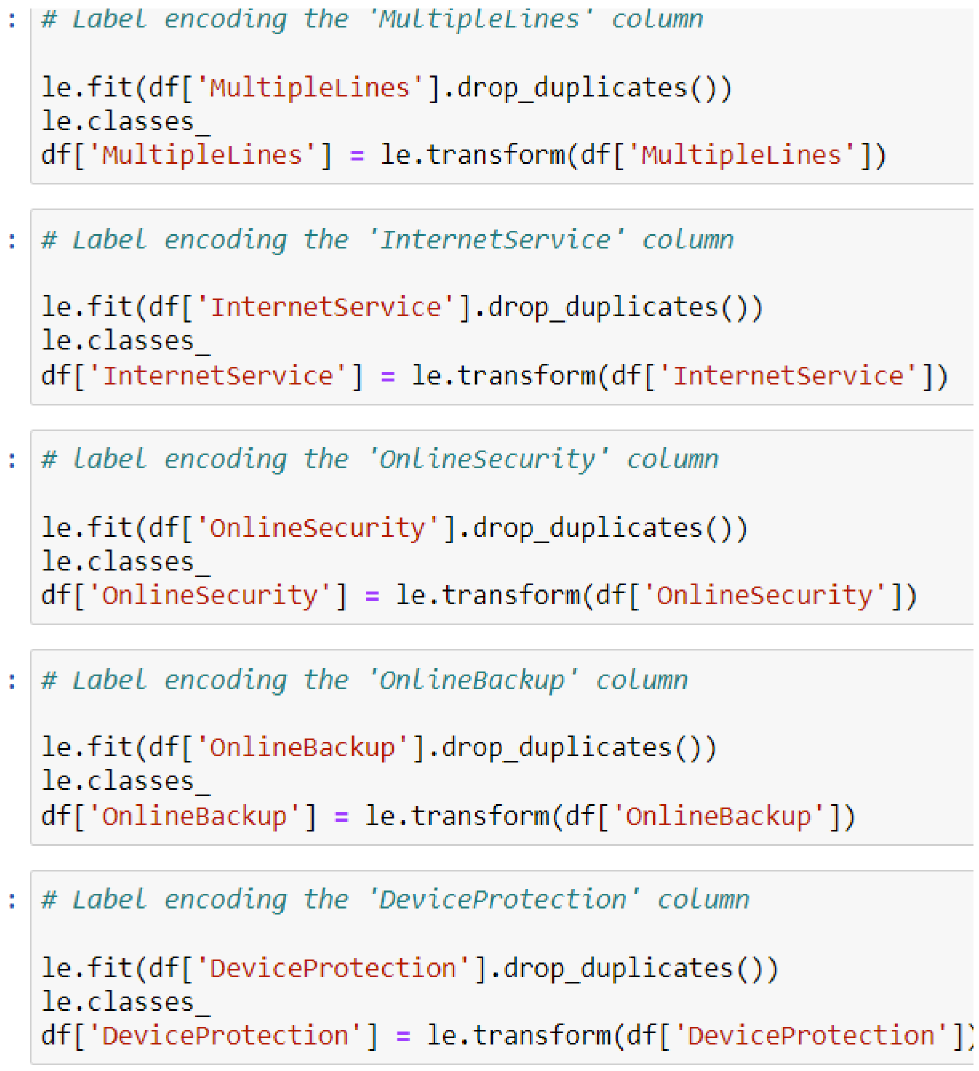
```
: # Label encoding the 'customerID' column
  le.fit(df['customerID'].drop_duplicates())
  le.classes_
  df['customerID'] = le.transform(df['customerID'])
```

```
: # Label encoding the 'Gender' column

  le.fit(df['gender'].drop_duplicates())
  le.classes_
  df['gender'] = le.transform(df['gender'])
```

```
: # Label encoding the 'Partner' column

  le.fit(df['Partner'].drop_duplicates())
  le.classes_
  df['Partner'] = le.transform(df['Partner'])
```

```
: # Label encoding the 'Dependents' column

  le.fit(df['Dependents'].drop_duplicates())
  le.classes_
  df['Dependents'] = le.transform(df['Dependents'])
```

```
: # Label encoding the 'PhoneService' column

  le.fit(df['PhoneService'].drop_duplicates())
  le.classes_
  df['PhoneService'] = le.transform(df['PhoneService'])
```



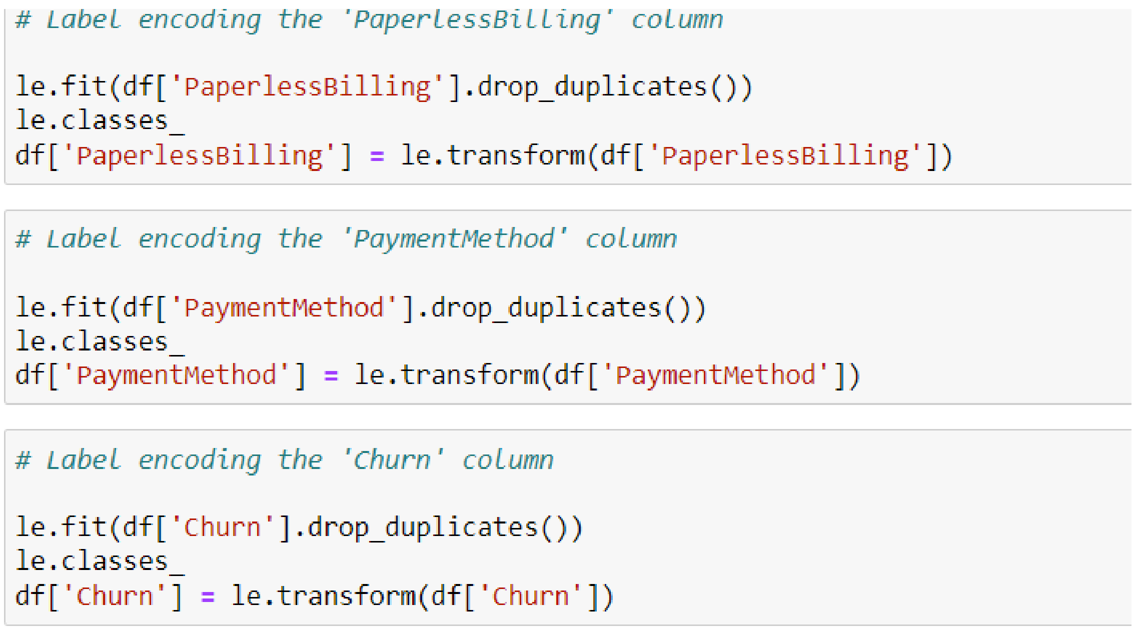
```
# Label encoding the 'MultipleLines' column

le.fit(df['MultipleLines'].drop_duplicates())
le.classes_
df['MultipleLines'] = le.transform(df['MultipleLines'])
```

```
# Label encoding the 'InternetService' column

le.fit(df['InternetService'].drop_duplicates())
le.classes_
df['InternetService'] = le.transform(df['InternetService'])
```

```
# label encoding the 'OnlineSecurity' column

le.fit(df['OnlineSecurity'].drop_duplicates())
le.classes_
df['OnlineSecurity'] = le.transform(df['OnlineSecurity'])
```

```
# Label encoding the 'OnlineBackup' column

le.fit(df['OnlineBackup'].drop_duplicates())
le.classes_
df['OnlineBackup'] = le.transform(df['OnlineBackup'])
```

```
# Label encoding the 'DeviceProtection' column

le.fit(df['DeviceProtection'].drop_duplicates())
le.classes_
df['DeviceProtection'] = le.transform(df['DeviceProtection'])
```

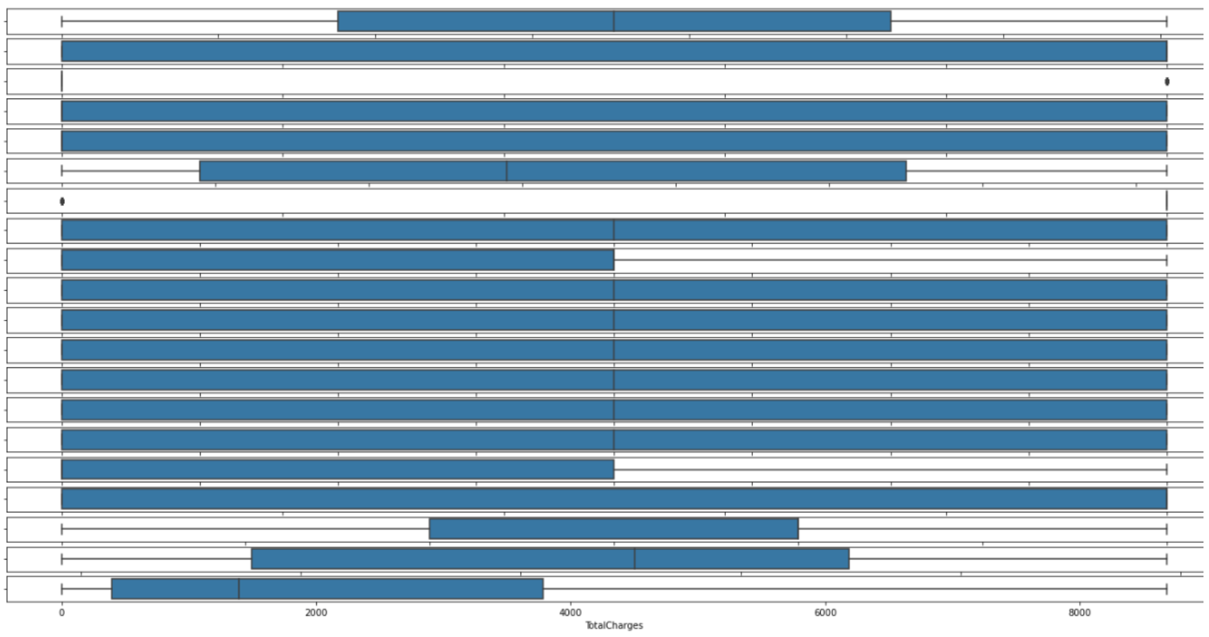


```
# Label encoding the 'PaperlessBilling' column

le.fit(df['PaperlessBilling'].drop_duplicates())
le.classes_
df['PaperlessBilling'] = le.transform(df['PaperlessBilling'])
```

```
# Label encoding the 'PaymentMethod' column

le.fit(df['PaymentMethod'].drop_duplicates())
le.classes_
df['PaymentMethod'] = le.transform(df['PaymentMethod'])
```

```
# Label encoding the 'Churn' column

le.fit(df['Churn'].drop_duplicates())
le.classes_
df['Churn'] = le.transform(df['Churn'])
```

Outliers: We can define outliers as the data point which is distant from the other similar points. They may be due to variability in the measurement or may indicate experimental errors. If possible, outliers should be excluded from the data set. However, detecting that anomalous instance might be very difficult, and is not always possible.



No outliers are present

SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors of this point. The synthetic points are added between the chosen point and its neighbors.

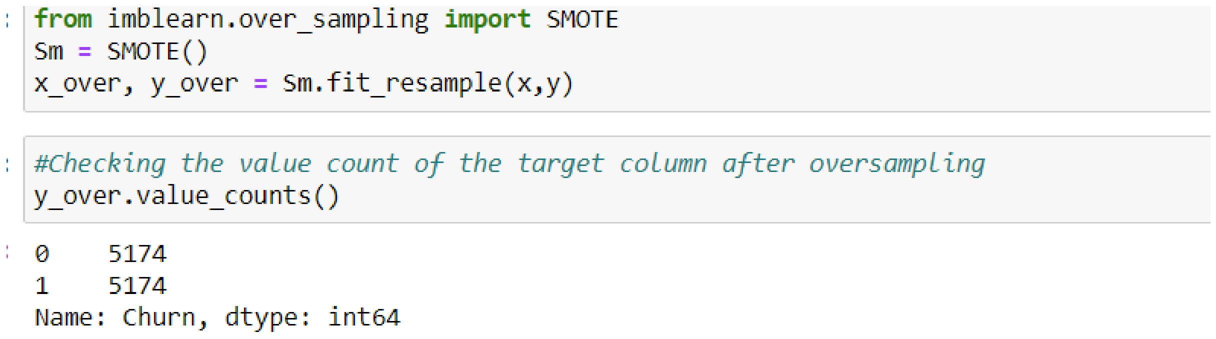
SMOTE algorithm works in 4 simple steps:-

I. Choose a minority class as input vector.
II. Find its k-nearest neighbors.
III. Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbors.

Repeat the step until the data is balanced.

Balancing the imbalanced dataset:

We have seen earlier that, our dataset was imbalance. There are different methods available to make the data balanced. We have used SMOTE method.



```
from imblearn.over_sampling import SMOTE
Sm = SMOTE()
x_over, y_over = Sm.fit_resample(x,y)
```

```
#Checking the value count of the target column after oversampling
y_over.value_counts()
```

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

We can see the target column is balanced

Machine Learning Model Building:

We can find many types of methods for model building in the skLearn library.

There are two types of models which are present in the skLearn library: 1. Regression and 2. Classification

In the case of our dataset, we have to make machine learning model to predict whether the claim made by the customers are fraud or not. Since we have only two values, so we will use classification models to build the machine learning model.

Before fitting the data to the model we will need to first separate the dependent variable and independent variables, then we will pass these values to train_test_split to generate random training and testing subset of the data.

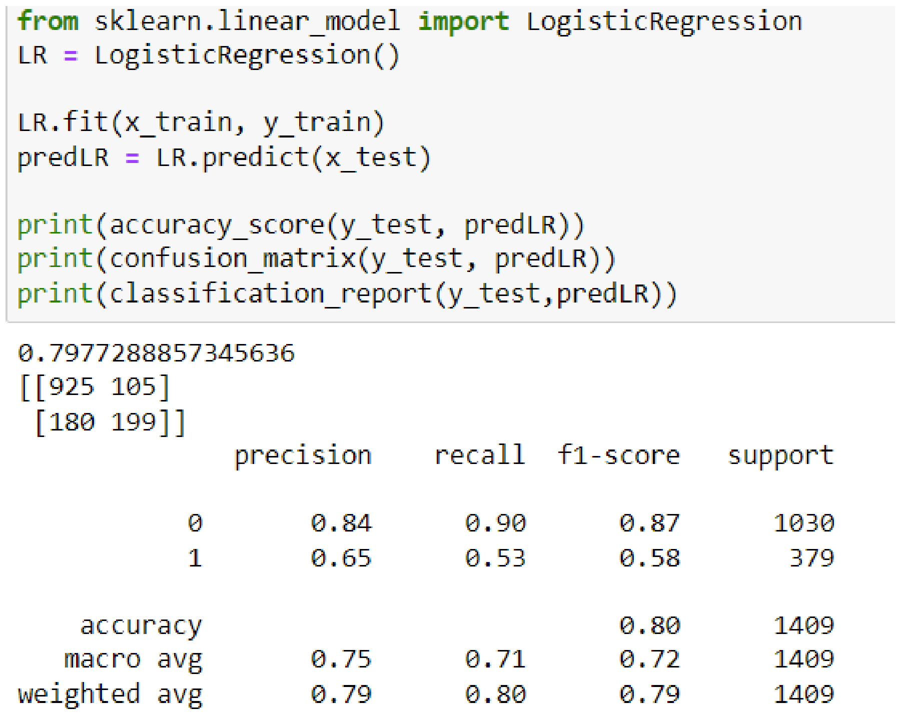
About train_test_split method: It is a function, which we can find in skLearn model selection library. We use this function for splitting data arrays into two subsets for training data and testing data. By using this function we can avoid the manual task to separate the dataset. By default, sklearn train_test_split will make random partitions for the two subsets. However, we can also specify a random state for the operation. It gives four output x_train, x_test, y_train and y_test. The x_train and x_test contains the training and testing predictor variables while y_train and y_test contains the training and testing target variable.

After performing train_test_split we have to choose the models to pass the training variable.

We can build as many models as we want to compare the accuracy given by these models and to select best model among them.

We have made 4 models to get the closest possible accuracy score to predict the churn analysis.

LOGISTIC REGRESSION:



```python
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()

LR.fit(x_train, y_train)
predLR = LR.predict(x_test)

print(accuracy_score(y_test, predLR))
print(confusion_matrix(y_test, predLR))
print(classification_report(y_test,predLR))
```

```
0.7977288857345636
[[925 105]
 [180 199]]
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1030
           1       0.65      0.53      0.58       379

    accuracy                           0.80      1409
   macro avg       0.75      0.71      0.72      1409
weighted avg       0.79      0.80      0.79      1409
```

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is binary, which means there would be only two possible classes 1 (stands for success/yes) or 0 (stands for failure/no). Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems.

1. <u>Random Forest Classifier:</u>

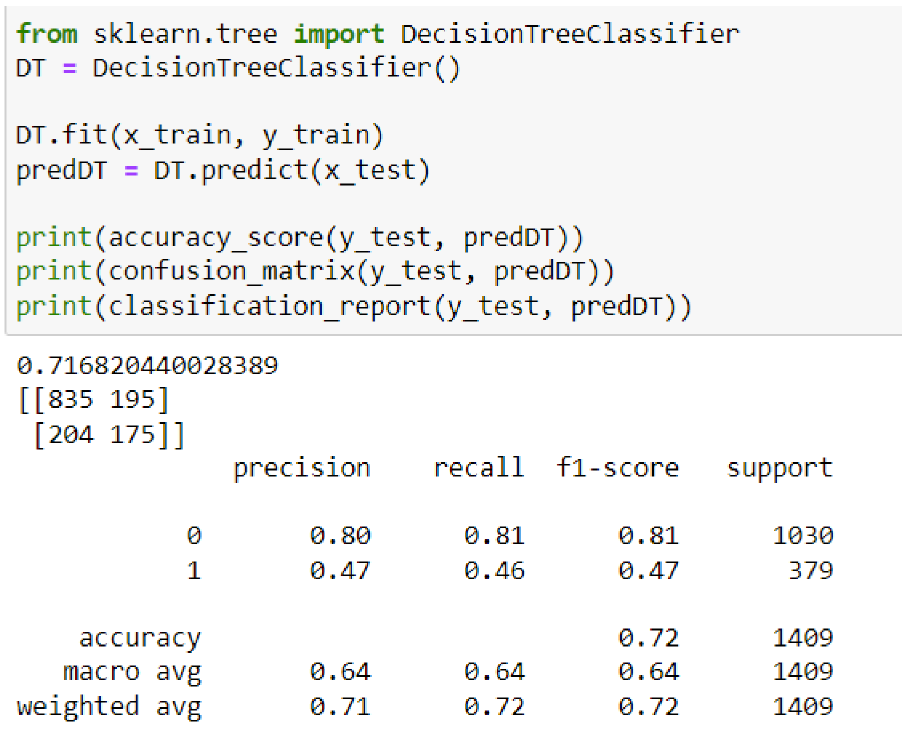
As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decisiontree on datasamples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
predRF = RF.predict(x_test)

print(accuracy_score(y_test, predRF))
print(confusion_matrix(y_test, predRF))
print(classification_report(y_test, predRF))
```

```
0.7984386089425124
[[941  89]
 [195 184]]
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1030
           1       0.67      0.49      0.56       379

    accuracy                           0.80      1409
   macro avg       0.75      0.70      0.72      1409
weighted avg       0.79      0.80      0.79      1409
```

2. Decision Tree Classifier:

Decision Tree Classifier is a class capable of performing multi-class classification on a dataset. As with other classifiers, DecisionTreeClassifier takes as input two arrays: an array X, sparse or dense, of shape (n_samples, n_features) holding the training samples, and an array Y of integer values, shape (n_samples,), holding the class labels for the training samples. It is capable of both binary (where the labels are [-1, 1]) classification and multi-class (where the labels are [0, ..., K-1]) classification.

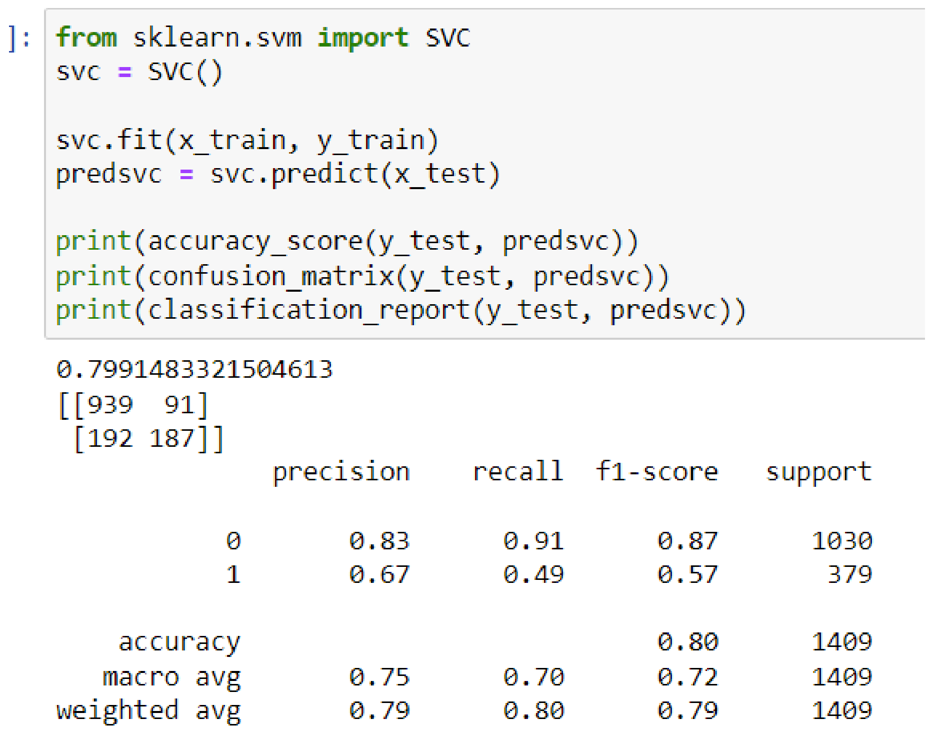


```python
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
predDT = DT.predict(x_test)

print(accuracy_score(y_test, predDT))
print(confusion_matrix(y_test, predDT))
print(classification_report(y_test, predDT))
```

```
0.716820440028389
[[835 195]
 [204 175]]
              precision    recall  f1-score   support

           0       0.80      0.81      0.81      1030
           1       0.47      0.46      0.47       379

    accuracy                           0.72      1409
   macro avg       0.64      0.64      0.64      1409
weighted avg       0.71      0.72      0.72      1409
```

3. SVC: (Support Vector Classifier)

SVC is a non parametric clustering algorithm that does not make any assumption on the number or shape of the clusters in the data. In SVC data points are mapped from data space to a high dimensional feature space using a kernel function. In the kernel's feature space the algorithm searches for the smallest sphere that encloses the image of the data using the Support Vector Domain Description algorithm. This sphere, when mapped back to data space, forms a set of contours which enclose the data points. Those contours are then interpreted as cluster boundaries, and points enclosed by each contour are associated by SVC to the same cluster



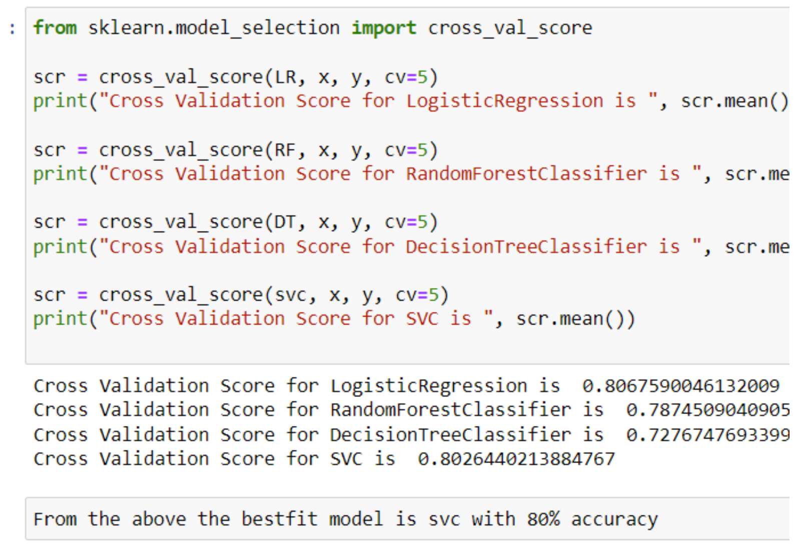
```
]: from sklearn.svm import SVC
   svc = SVC()

   svc.fit(x_train, y_train)
   predsvc = svc.predict(x_test)

   print(accuracy_score(y_test, predsvc))
   print(confusion_matrix(y_test, predsvc))
   print(classification_report(y_test, predsvc))
```

```
0.7991483321504613
[[939  91]
 [192 187]]
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1030
           1       0.67      0.49      0.57       379

    accuracy                           0.80      1409
   macro avg       0.75      0.70      0.72      1409
weighted avg       0.79      0.80      0.79      1409
```

If we compare all the models we found that we are getting maximum accuracy score from Support Vector Classifier. But we can not decide the best model on the basis of accuracy score only, since this might be possible that our data is over-fitted.

So, to decide the best fit model we will check for the cross validation score of each model. We can import cross_val_score from the skLearn library (sklearn.model_selection).

CROSS VALIDATION:



```
: from sklearn.model_selection import cross_val_score

  scr = cross_val_score(LR, x, y, cv=5)
  print("Cross Validation Score for LogisticRegression is ", scr.mean()

  scr = cross_val_score(RF, x, y, cv=5)
  print("Cross Validation Score for RandomForestClassifier is ", scr.me

  scr = cross_val_score(DT, x, y, cv=5)
  print("Cross Validation Score for DecisionTreeClassifier is ", scr.me

  scr = cross_val_score(svc, x, y, cv=5)
  print("Cross Validation Score for SVC is ", scr.mean())
```

```
Cross Validation Score for LogisticRegression is  0.8067590046132009
Cross Validation Score for RandomForestClassifier is  0.7874509040905
Cross Validation Score for DecisionTreeClassifier is  0.7276747693399
Cross Validation Score for SVC is  0.8026440213884767
```

```
From the above the bestfit model is svc with 80% accuracy
```

The model having the lowest difference between the accuracy score and cross validation score will be the best model for our machine learning algorithm.so ,from the above support vector classifier is the best model for customer churn analysis dataset.

CONCLUSION:

From the above table we can say that the best fit model is Support Vector Classifier since it is showing minimum difference between the accuracy score and cross validation score.

- ✓ Our model is showing approx 80% accuracy score.
- ✓ It have given the f1 score of approx 80%.

Let's understand what does precision recall and f1 score and accuracy means.

- ✧ F1 score: It is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy matrix.

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision}*\text{Recall})}{(\text{Precision}+\text{Recall})}$$

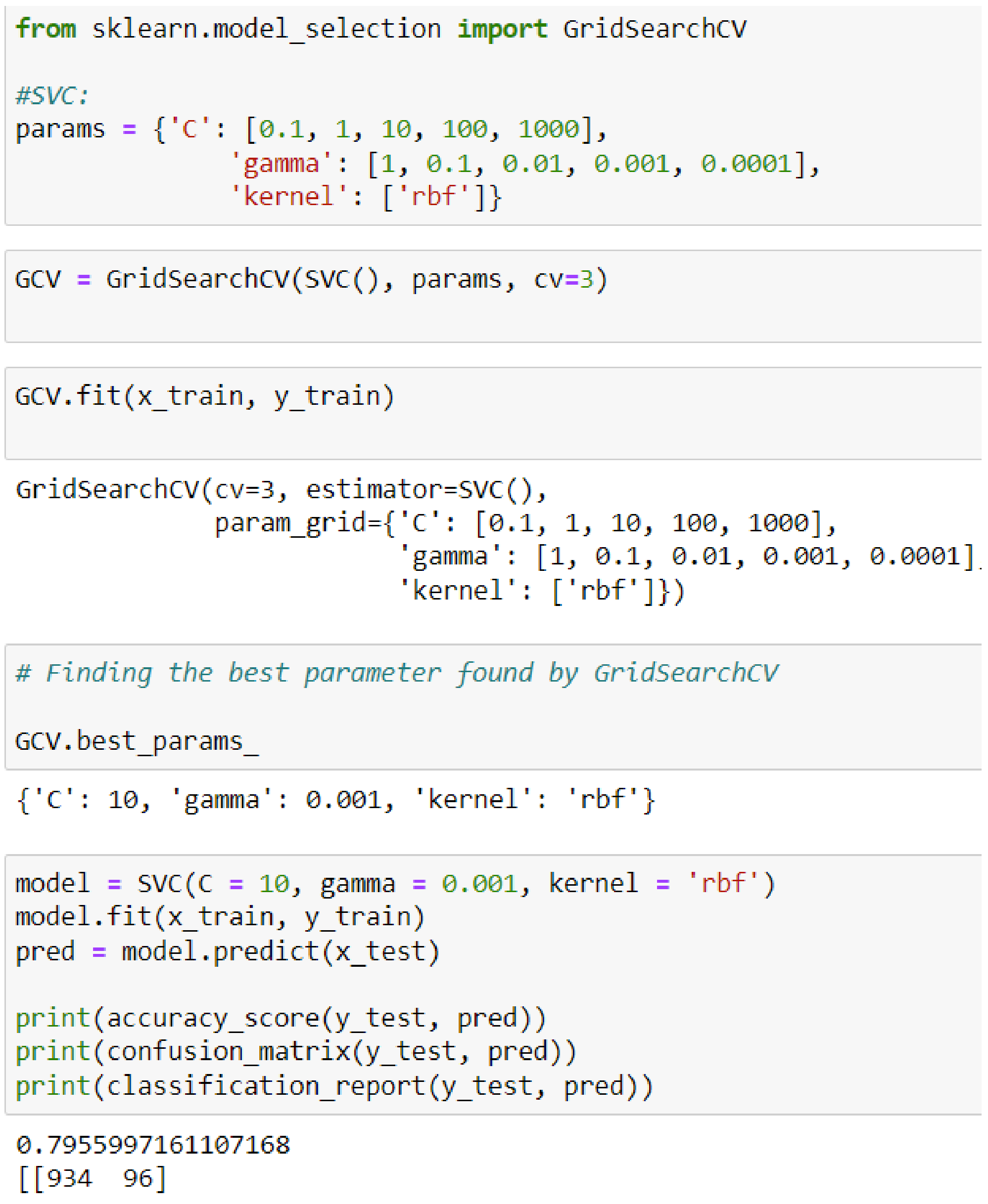
- ✧ Precision: It is implied as the measure of the correctly identified positive cases from all the predicted positive cases. Thus, it is useful when the costs of False Positives are high.

Hyper parameter tuning:

Hyper parameter optimization in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyper parameters, in contrast to model parameters, are set by the machine learning engineer before training. The number of trees in a random forest is a hyper parameter while the weights in a neural network are model parameters learned during training. I like to think of hyper parameters as the model settings to be tuned so that the model can optimally solve the machine learning problem. We will use GridSearchCV for the hyper parameter tuning.

GridSeatchCV:

In GridSearchCV approach, machine learning model is evaluated for a range of hyper parameter values. This approach is called GridSearchCV, because it searches for best set of hyper parameters from a grid of hyper parameters values.



```
from sklearn.model_selection import GridSearchCV

#SVC:
params = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
```

```
GCV = GridSearchCV(SVC(), params, cv=3)
```

```
GCV.fit(x_train, y_train)
```

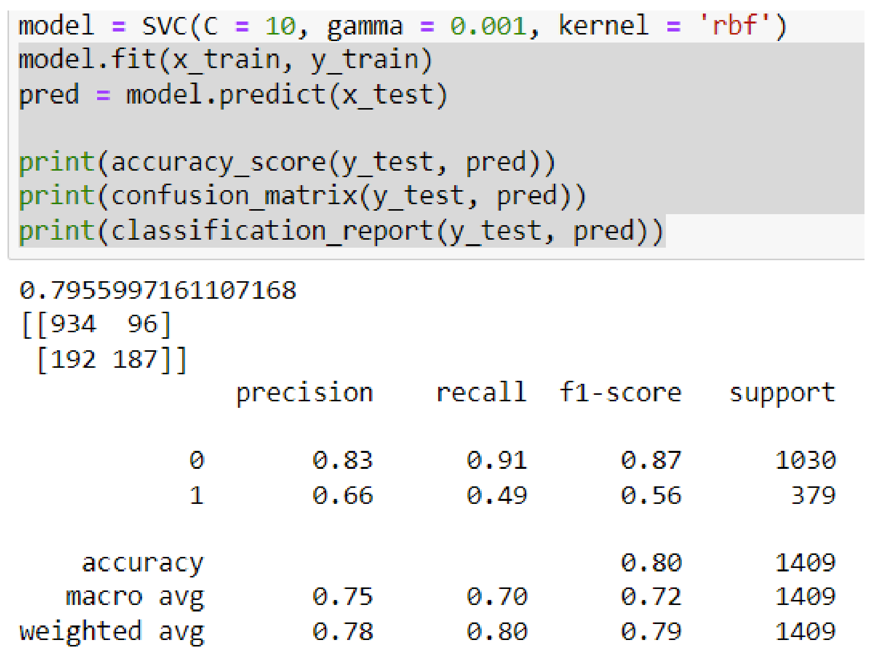
```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
                         'kernel': ['rbf']})
```

```
# Finding the best parameter found by GridSearchCV

GCV.best_params_
```

```
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
model = SVC(C = 10, gamma = 0.001, kernel = 'rbf')
model.fit(x_train, y_train)
pred = model.predict(x_test)

print(accuracy_score(y_test, pred))
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
0.7955997161107168
[[934  96]
```



```
model = SVC(C = 10, gamma = 0.001, kernel = 'rbf')
model.fit(x_train, y_train)
pred = model.predict(x_test)

print(accuracy_score(y_test, pred))
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
0.7955997161107168
[[934  96]
 [192 187]]
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1030
           1       0.66      0.49      0.56       379

    accuracy                           0.80      1409
   macro avg       0.75      0.70      0.72      1409
weighted avg       0.78      0.80      0.79      1409
```

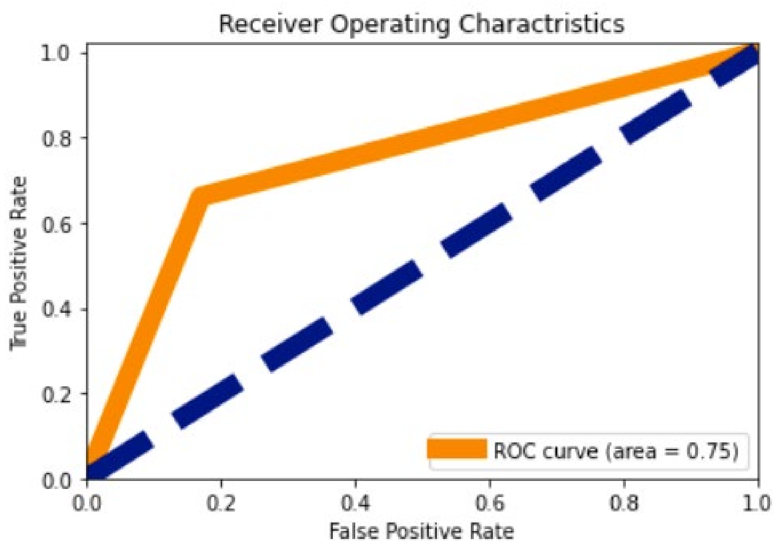
We can see that after hyper parameter tuning our accuracy model score is 80%.

AUC ROC Curve

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

We are plotting AUC ROC curve for the final model



REMARKS:

We have successfully built a model to predict the CUSTOMER CHURN made in the big companies. Our model can help companies to increase their profit level by identifying the churn made by the customers. The most challenging job for machine learning model to predict the churn.

References:

https://scikit-learn.org/

https://towardsdatascience.com/