TELECOM CHURN PREDICTION

1. **Introduction**
    - **Customer churn**
    - **Problem statement**
    - **Data set**
    - **Mapping to ML problem**
2. **Data Analysis**
    a. **Loading Libraries**
    b. **Load dataset**
    c. **Preprocessing data**
    d. **Define churn variable**
3. **Exploratory Data Analysis**
    a. **Histogram**
    b. **Heatmap**
    c. **Univariate Analysis**
    d. **Bivariate Analysis**
    e. **Multivariate Analysis**
4. **Feature Engineering**
5. **DataStandardisation**
6. **Splitting of data**
7. **Balancing data**
8. **Metrics Used**
    a. **ConfusionMatrix,**
    b. **ROC & AUROC**
9. **Architecture Diagram**

10. ModelBuilding
    a. Baseline Model(Default & Hyperparameters)
    b. ComplexModels(Default & Hyperparameters)
        i. Logistic Regression
        ii. DeciscionTree
        iii. XGBoost
        iv. Randomforest
11. Advanced modeling using voting classifier
12. Model Interpretability
13. Comparison of models
14. Pros & cons of Model
15. FeatureImportance
16. Save the model
17. Load the trained model
18. Evaluate the loaded model
19. Evaluation of trained model object
20. References

1. Customer churn :

Customer churn means the customer attrition rate in an enterprise. i.e customers ending their contract or canceling their subscription service.

Problem statement :

To reduce customer churn, telecom companies need to predict which customers are at high risk of churn.

Given three months of customer's telephonic data. We need to predict the customer churn in the next month.

- It is important to predict churn because Customer churn directly affects the revenues of companies especially in the telecom industry. The service providers strive very hard to sustain in this competition. It is much less expensive to retain existing customers than it is to acquire new customers.
- This analysis helps companies to identify the cause of churn and implement effective strategies for customer retention.
- if we predict churn in early stages we can reduce the number of churned customers by giving some incentives like special packages etc..Customers are an asset to an enterprise. So predicting customer churn in the telecom industry is crucial.

Dataset:

The Data set can be downloaded from
https://www.kaggle.com/vijaysrikanth/telecom-churn-data-set-for-the-south-asian-market

- File format: .csv
- File Size: 75.4 MB
- Number of Data Points: 99999
- Number of Columns: 226

The Data is collected for four months i.e. June, July, August and September. Out of these four months of data, we need to use the first three months Data to Predict the Customers who are at High Risk of Getting into the Churn Category and the September month data can be used to Label the Data set.

Features in data set: some features in data set are:

mobile_number --------- mobile number of user
arpu -------- average revenue per user
last_date_of_month --------- last date in that month
fb_user --------- facebook user
max_rech_amt ---------- maximum recharge amount
total_rech_data --------- total number of recharges per data
loc_ic_mou --------- minutes of usage of local incoming calls
loc_og_mou --------- minutes of usage of local outgoing calls
total_ic_mou --------- minutes of usage of total incoming calls
total_og_mou --------- minutes of usage of total outgoing calls
onnet_mou --------- minutes of usage of all calls on same network
offnet_mou ------- minutes of usage of all calls on other network
vol_2g_mb -------- amount of 2g internet usage
vol_3g_mb --------- amount of 3g internet usage
aon --------- age on network

Mapping the problem to ML problem:

It is a two-class classification problem.

Here the output variable is categorical , is the customer “churn” or “not”. So it is considered a classification problem.

Solution approaches:

- Understand the columns in data. Remove columns that are not useful for our analysis.
- Missing value treatment by dropping the rows or replacing them by ‘0’ or ‘1’ based on remaining data.
- Remove outliers of data.

- Filter high value customers.Labeling the data.
- The cleaned form of data is used for analysis and model building.

Real world challenges and constraints:

- Interpretability is important here.
- Misclassification costs will be high.
- No low latency constraint.
- Requirement that our solution must meet: To predict as many customers as possible in a churned state.

2. Data analysis:

Data Analysis is one of the most important steps in solving any Machine Learning problem. In exploratory data analysis we look into trends, patterns and relationships between various entities in data.As the very first step, let's import the required libraries to solve this problem.

Loading Libraries

In [195]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import numpy as np # linear algebra
import os # accessing directory structure
from scipy import stats
from scipy.stats import zscore
```

As the very first step, let's import the required libraries to solve this problem.

Loading the data:

Data set is in the format of a csv file. Load the data from "telecom_churn_data.csv" to a pandas dataframe.

```
# read the data

churn_data = pd.read_csv("downloads/telecom_churn_data.csv")
```

The top 5 rows of data , some features and dataset shape is as follows

```
In [6]: #print first five rows of data
        churn_data.head(5)
```

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_of_month_7	last_date_of_month_8	last_date_of
0	7000842753	109	0.00	0.00	0.00	6/30/2014	7/31/2014	8/31/2014	
1	7001865778	109	0.00	0.00	0.00	6/30/2014	7/31/2014	8/31/2014	
2	7001625959	109	0.00	0.00	0.00	6/30/2014	7/31/2014	8/31/2014	
3	7001204172	109	0.00	0.00	0.00	6/30/2014	7/31/2014	8/31/2014	
4	7000142493	109	0.00	0.00	0.00	6/30/2014	7/31/2014	8/31/2014	

```
# shape of data
churn_data.shape
```

```
(99999, 226)
```

There are 99999 data points and 226 columns in the dataset.

Among these 226, 214 are numeric(int, float) and 12 are non-numeric(object type) data types. These 12 non numerics are date time columns.so convert these columns into date time format.

Preprocessing data:

Preprocessing is a data mining technique that transforms raw data into an understandable format. Raw data(real world data) is always incomplete and that data cannot be sent through a model. That would cause certain errors. That is why we need to preprocess data before sending through a model. The techniques like

- Converting data types into acceptable ones

The date column fields in data are of type objects. Convert these into data format.

```
#converting these dateime cols to datetime format
for col in date.columns:
    churn_data[col] = pd.to_datetime(churn_data[col])
```

- checking missing values

```
21]: #print missing percentage in each column.
     percent_missing = churn_data.isna().sum()/(len(churn_data))*100
     percent_missing.sort_values(ascending = False)
```

```
Out[32]: date_of_last_rech_data_6   74.85
         arpu_3g_6                 74.85
         arpu_2g_6                 74.85
         arpu_2g_7                 74.43
         arpu_3g_7                 74.43
         date_of_last_rech_data_7  74.43
         date_of_last_rech_data_9  74.08
         arpu_3g_9                 74.08
         arpu_2g_9                 74.08
         arpu_2g_8                 73.66
         arpu_3g_8                 73.66
         date_of_last_rech_data_8  73.66
         og_others_9                7.75
         loc_og_t2c_mou_9           7.75
         std_ic_t2t_mou_9           7.75
         std_ic_t2m_mou_9           7.75
         std_ic_t2f_mou_9           7.75
         loc_ic_t2t_mou_9           7.75
         spl_ic_mou_9               7.75
```

By observing the date columns 74% of 'date_of_last_rech_data_', 'date_of_last_rech_' column values are missing. we can fill those with the first date of every month.

```
# Date of last recharge column fill with first date of month
churn_data['date_of_last_rech_data_6'].fillna('6/1/2014',inplace=True)
churn_data['date_of_last_rech_data_7'].fillna('7/1/2014',inplace=True)
churn_data['date_of_last_rech_data_8'].fillna('8/1/2014',inplace=True)
churn_data['date_of_last_rech_data_9'].fillna('9/1/2014',inplace=True)
```

- Searching for unique values

There are a lot of unique values and missing values in data. These values can be filled by using some missing value imputation techniques like, Remove data points which

have more than 70% of missing values. Datecolumns will be filled by dates, Boolean columns filled by 1 or 0, Numerical columns filled by numbers.

- Drop unique value columns
- Filling with 0's and 1's based on data in those columns.
- Date columns filled by first date of month

```
In [14]: # Drop the unique value columns
         churn_data.drop(unique_col, axis=1, inplace = True)
```

```
In [15]: # shape of data after dropimg  unique columns
         churn_data.shape

Out[15]: (99999, 210)
```

- Checking for highly correlated columns

```
In [37]: # Checking the correlation between the above mentioned columns in tabular for months 6,7

         print("Correlation  for month 6\n\n")
         print( churn_data[['arpu_3g_6','arpu_2g_6','av_rech_amt_data_6']].corr())
         print("\nCorrelation  for month 7")
         print( churn_data[['arpu_3g_7','arpu_2g_7','av_rech_amt_data_7']].corr())
```

```
Correlation  for month 6


                   arpu_3g_6  arpu_2g_6  av_rech_amt_data_6
arpu_3g_6               1.00       0.93                0.81
arpu_2g_6               0.93       1.00                0.83
av_rech_amt_data_6      0.81       0.83                1.00
```

There are highly correlated columns in data. Dropping some correlated features doesn't affect our data. The correlation between columns shown by heat map also.

Output variable analysis:

We have to predict the customer churn in September as output by using previous months data.

Define churn variable:

We define the 'churn' variable as binary(churn = 1 and not churn = 0) by using September month features.

```
# Selecting the columns to define churn variable as  TARGET Variable

churn_col=['total_ic_mou_9','total_og_mou_9','vol_2g_mb_9','vol_3g_mb_9']

print(churn_col)
churn_data[churn_col].info()
```

After defining add churn column to database and find out churn and not churn perentage.

In [48]:
```
# Add churn column to data
churn_data['churn']=np.nan

# Imputing the churn values based on the condition
churn_data['churn'] = np.where(churn_data[churn_col].sum(axis=1) == 0, 1, 0)

churn_data['churn'].head()
```

In [49]:
```
# lets find out churn/non churn percentage
print((churn_data['churn'].value_counts()/len(churn_data))*100)
```

The September month columns which are used for labeling are dropped from data.

```
churn_cols = [col for col in churn_data.columns if '_9' in col]
print("The columns from churn phase are:\n",churn_cols)
```

```
The columns from churn phase are:
 ['arpu_9', 'onnet_mou_9', 'offnet_mou_9', 'roam_ic_mou_9', 'roam_og_mou_9', 'loc_og_t2t_mou_9', 'loc_og_t2m_mou_9', 'loc_og_t2
f_mou_9', 'loc_og_t2c_mou_9', 'loc_og_mou_9', 'std_og_t2t_mou_9', 'std_og_t2m_mou_9', 'std_og_t2f_mou_9', 'std_og_mou_9', 'isd_
og_mou_9', 'spl_og_mou_9', 'og_others_9', 'total_og_mou_9', 'loc_ic_t2t_mou_9', 'loc_ic_t2m_mou_9', 'loc_ic_t2f_mou_9', 'loc_ic
_mou_9', 'std_ic_t2t_mou_9', 'std_ic_t2m_mou_9', 'std_ic_t2f_mou_9', 'std_ic_mou_9', 'total_ic_mou_9', 'spl_ic_mou_9', 'isd_ic_
mou_9', 'ic_others_9', 'total_rech_num_9', 'total_rech_amt_9', 'max_rech_amt_9', 'date_of_last_rech_9', 'last_day_rch_amt_9',
'date_of_last_rech_data_9', 'max_rech_data_9', 'count_rech_2g_9', 'count_rech_3g_9', 'vol_2g_mb_9', 'vol_3g_mb_9', 'night_pck_u
ser_9', 'monthly_2g_9', 'sachet_2g_9', 'monthly_3g_9', 'sachet_3g_9', 'fb_user_9', 'total_rech_amt_data_9']
```

In [52]:
```
# Dropping the selected churn phase columns
churn_data.drop(churn_cols, axis=1, inplace=True)

# The curent dimension of the dataset after dropping the churn related columns
churn_data.shape
```

Out[52]: (99999, 154)

Feature engineering:

Create some new columns by merging related ones using addition, subtraction and multiplication of existing features.

:
```
# Merging the std_og calls
churn_data['std_og_mou_t_6']=churn_data['std_og_t2t_mou_6']+churn_data['std_og_t2m_mou_6']+churn_data['std_og_t2f_mou_6']
churn_data['std_og_mou_t_7']=churn_data['std_og_t2t_mou_7']+churn_data['std_og_t2m_mou_7']+churn_data['std_og_t2f_mou_7']
churn_data['std_og_mou_t_8']=churn_data['std_og_t2t_mou_8']+churn_data['std_og_t2m_mou_8']+churn_data['std_og_t2f_mou_8']
```

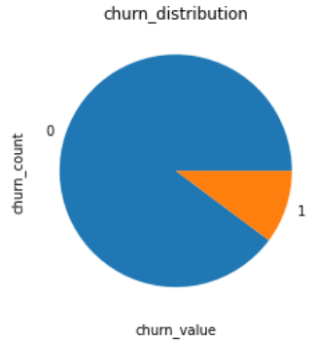
```
# Calculating the total recharge amount done for data alone in months 6,7,8 and 9
churn_data['total_rech_amt_data_6'] = churn_data['av_rech_amt_data_6'] * churn_data['total_rech_data_6']
churn_data['total_rech_amt_data_7'] = churn_data['av_rech_amt_data_7'] * churn_data['total_rech_data_7']
churn_data['total_rech_amt_data_8'] = churn_data['av_rech_amt_data_8'] * churn_data['total_rech_data_8']
churn_data['total_rech_amt_data_9'] = churn_data['av_rech_amt_data_9'] * churn_data['total_rech_data_9']
```

3. Exploratory Data Analysis:

Exploratory Data Analysis(EDA) is understanding the data sets by summarizing their main characteristics, often plotting them visually. Plotting in EDA consists of Histograms, Boxplots,Scatter Plots and many more.

In [49]:
```
# lets find out churn/non churn percentage
print((churn_data['churn'].value_counts()/len(churn_data))*100)
((churn_data['churn'].value_counts()/len(churn_data))*100).plot(kind="pie")
plt.title("churn_distribution")
plt.xlabel("churn_value")
plt.ylabel("churn_count")
plt.show()
```

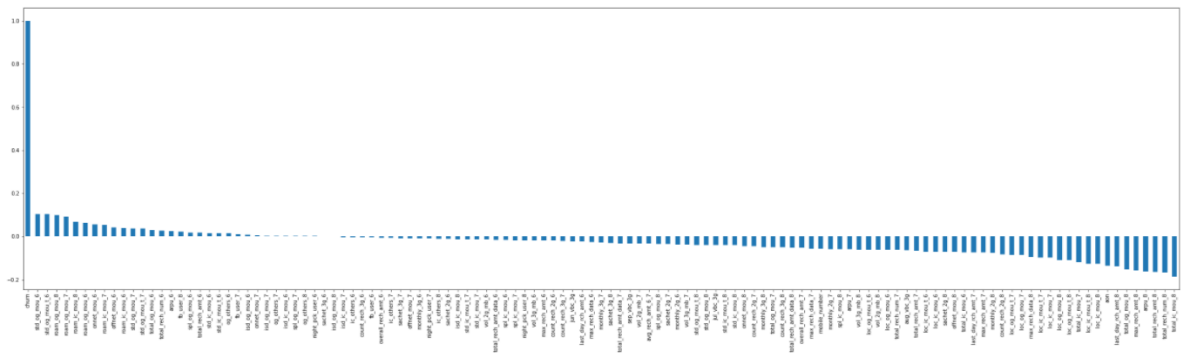
```
0   89.81
1   10.19
Name: churn, dtype: float64
```



By observing the above graph almost 90% of customers are not churn. Only 10% of customers are in a churned state.

```
In [61]: # Correlation of "Churn" with other variables:
         plt.figure(figsize=(40,10))
         churn_data.corr()['churn'].sort_values(ascending = False).plot(kind='bar')
```

```
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x18bcb4ac6d0>
```

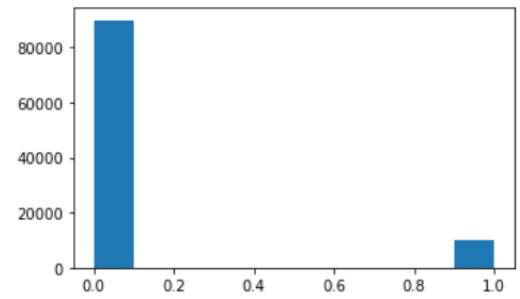


By observing the above plot the features total_ic_mou, total_rech_num, max_rech_amt ,total_og_mou, last_day_rech_amt of month 8 are negatively correlated with the churn variable. std_og_mou,onnet_mou,roam_og_mou,offnet_mou features of moth 6 are positively correlated with churn.

Histogram:

A histogram is a great tool for quickly assessing a probability distribution that is easy for interpretation by almost any audience.

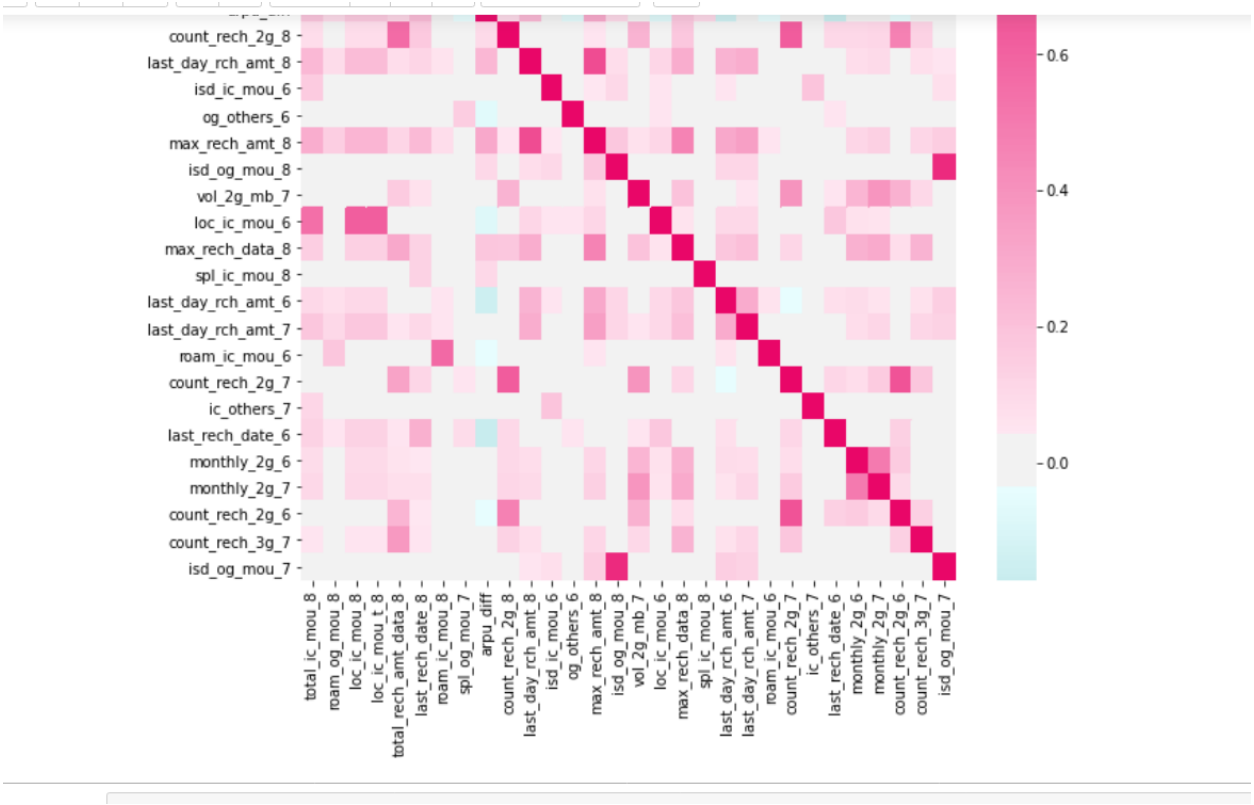
In [261]:
```
# Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(8, 7),
                        tight_layout = True)
axs.hist(churn_data['churn'], bins = 10)
# Show plot
plt.show()
```



Heatmap:

The Heat Map procedure shows the distribution of a quantitative variable over all combinations of 2 categorical factors. If one of the 2 factors represents time, then the evolution of the variable can be easily viewed using the map. A gradient color scale is used to represent the values of the quantitative variable. The correlation between two random variables is a number that runs from -1 through 0 to +1 and indicates a strong inverse relationship, no relationship, and a strong direct relationship, respectively.

In [227]:
```
# plot feature correlation
import seaborn as sns
plt.rcParams["figure.figsize"] =(10,10)
mycmap = sns.diverging_palette(199, 359, s=99, center="light", as_cmap=True)
sns.heatmap(data=x_tr[top_features].corr(), center=0.0, cmap=mycmap)
```



Univariate analysis:-

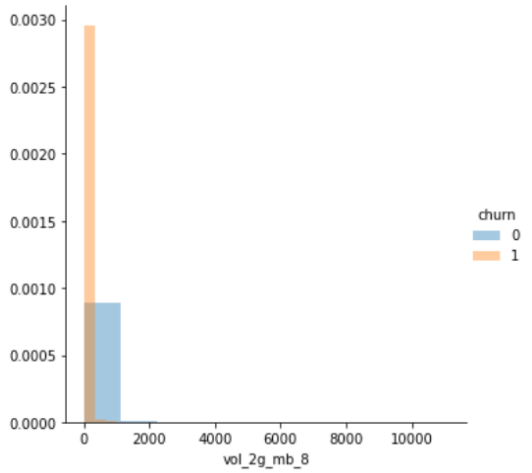
This analysis is done based on one variable. Main objective of this is to describe data.The analysis will take data,summarize it and identify the patterns present in the data. It doesn't find cause and relationship between variables.

Univariate analysis is done by using the following plots.

- Distribution Tables
- Barcharts
- Pie Charts
- Histograms

In our dataset we use seaborn for plotting.

[65]:
```
# distribution of 3g net usage

sns.FacetGrid(churn_data, hue = 'churn', height = 5)\
    .map(sns.distplot,'vol_2g_mb_8',bins = 10)\
    .add_legend()
plt.show()
```

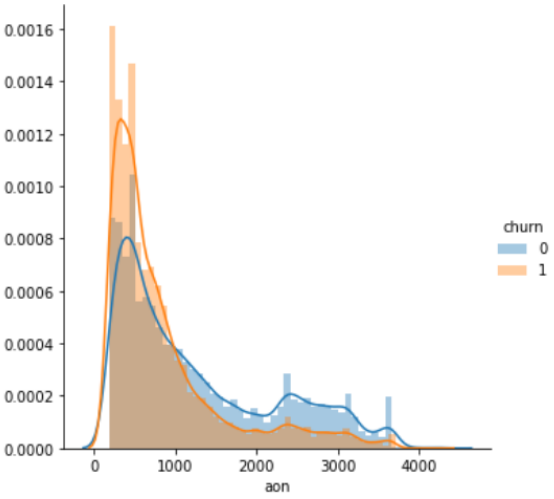


---

observation:

churned customers use less 2g _mobile internet usage in month 8, almost close to zero.

In [68]:
```
# distribution of aon
sns.FacetGrid(churn_data, hue = 'churn', height = 5)\
    .map(sns.distplot,'aon')\
    .add_legend()
plt.show()
```



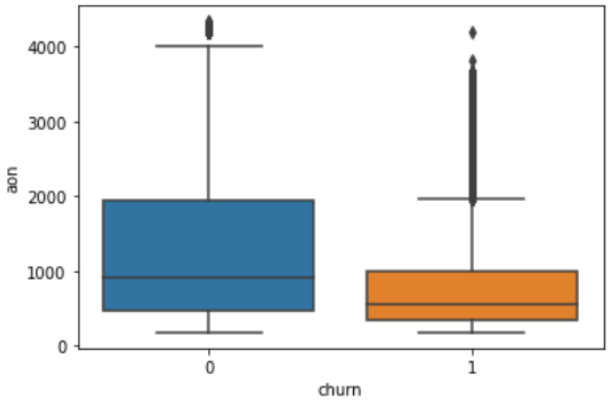
By observing the above plot, there is a lot of overlap between churned and non churned by using aon.

Bivariate analysis:-

It is performed to find the relationship between each variable in the dataset and the target variable of interest (or) using 2 variables and finding the relationship between them .
Ex:-Box plot, Violin plots

Boxplots are a measure of how well distributed the data in a data set is. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.
It allows users to quickly get the median, quartiles and outliers but also hides the dataset's individual data points.

```
In [76]: # plot of aon and churn
         sns.boxplot(y = 'aon', x = 'churn', data = churn_data)

         plt.show()
```

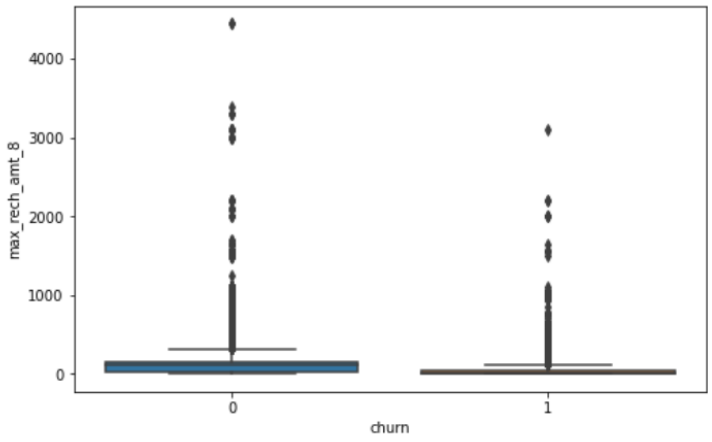


observation:

From the above plot , its clear that tenured customers(who stay with the network for a long time) do not churn much.

The relationship between churn and max_rech_amt_8 is shown as follows.

```
In [70]: # plot of max_rech_amt_8
         plt.figure(figsize=(8,5))
         sns.boxplot( x = 'churn', y = 'max_rech_amt_8',data = churn_data)
         plt.show()
```



observation:

churned customers make less than the maximum recharge amount in month 8.

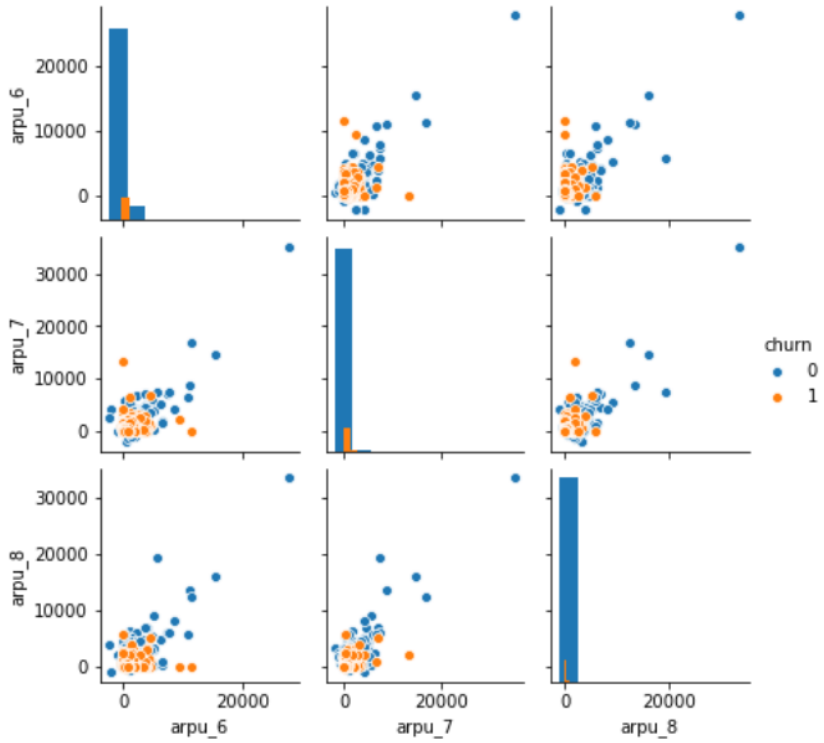
Multivariate analysis:

It is performed to understand interactions between fields in the dataset (or) finding interactions between variables more than two.
Ex: Pair plot, 3D Scatterplot

```
In [73]: # bivariate distributions in a dataset, use pair plots


         import seaborn as sns
         #plt.figure(figsize = (8,5))
         sns.pairplot(arpu,hue = 'churn', height=3,diag_kind='hist')
         plt.show()
```

The pair plot builds on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the

scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables. For example, the left-most plot in the second row shows the scatter plot of arpu_7 versus arpu_8



observation:

In the above pair plots, there is a lot of overlapping.The churned customers have made a good revenue in one month and not in the other month.The diagonal histogram shows that most of the churned customers have values close to 0.

Observations of EDA:

- For features like incoming,outgoing,recharge, most of the churned customers have low values. Especially in the eighth month
- Recharge amounts distinguish churned customers better than the other features.
- The pair plots show that most of the churned customers have close to 0 value.
- The total outgoing minutes of usage for churned customers is almost equal to zero.

- The Tenured customers(who stay with the network for a long time) do not churn much.
- The churned customers have good revenue at one month and not in another month.

4. Feature engineering:

creating some new features based on existing ones for better analysis of data.

creating new features using the difference between arpu_8 and avg of 6and 7 th months

```
In [86]: # creating new feature using difference between arpu_8 and avg of 6and 7 th months

         churn_data['arpu_diff'] = churn_data['arpu_8']- (churn_data[['arpu_6','arpu_7']].mean(axis=1))
```

arpu_diff column will calculate the difference between arpu of 8 th month and avg arpu of 6 and 7 months. This arpu_diff shows the customer behavior.

```
In [87]: churn_data.shape

Out[87]: (99999, 128)
```

The final shape of our data after EDA is (99999,128). This will be used for model building.

Drop the target column ‘churn’ from dataset:

Let’s Segregate our target variable from other columns.

```
In [89]: # Drop the column mobile number
         df_model.drop('mobile_number', axis=1, inplace=True)

         # fill the remaining fields
         df_model.fillna(0, inplace = True)
         df_model.head()
```

Drop the column mobile_number from data. It is not required for prediction.

```
In [90]: # Drop the churn column for prediction

         X = df_model.drop(['churn'], axis=1)
         y = df_model['churn']

         df_model.drop('churn', axis=1, inplace=True)
```

Drop the column churn from data. We predict the churn column from the model.

```
In [92]: df_model.shape

Out[92]: (99999, 126)
```

The final data after removing the churn column has 99999 rows and 126 columns.
This data is used for predicting the churn.

5. Data standardization:

It is the process of converting the data into uniform format to enable users to process and analyze it.
when features of an input data set have large differences between their ranges, or simply when they are measured in different measurement units (e.g., Pounds, Meters, Miles … etc).These differences in the ranges of initial features cause trouble for many machine learning models.

Here we use sklearn standardscaler for standardizing data.

The standard score of a sample `x` is calculated as:

$$z = (x - u) / s$$

where ‘u’ is the mean of the training samples or zero and `s` is the standard deviation of the training samples or one .

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training

In [94]:
```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

6. Splitting the data:

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model The train-test split procedure is appropriate when you have a very large dataset, a costly model to train, or require a good estimate of model performance quickly.

- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.

In [95]:
```
# splitting the data into test and train

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size = 0.3, random_state = 0)
```

7. Balancing the data:

Imbalanced class distribution is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. Most machine learning algorithms work best when the number of

samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce errors.
Many techniques were used to balance the data.Here we use SMOTE for balancing the data set.

SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.
Sampling techniques should be applied only on **train data not on test data.**

```
]: # Balancing DataSet
   #from imblearn.over_sampling import SMOTE
   from imblearn.over_sampling import SMOTE


   sm = SMOTE(random_state=42)
   x_tr,y_tr = sm.fit_resample(x_train,y_train)
```

```
In [103]: print(x_tr.shape)
          print(y_tr.shape)
          print("count label 0:",sum(y_tr==0))
          print("count label 1:",sum(y_tr==1))

          (125872, 126)
          (125872,)
          count label 0: 62936
          count label 1: 62936
```

This balanced data is used for model building .Now the no.of observations in both classes are the same.

Encoding of features:

It is essential to convert categorical features into numerical features. There are different ways for this encoding like labelencoder , onehot encoder,pandas get_dummies etc.
Here we use a label encoder for churn variables as,

- Customer churn = 1
- Not churn = 0

8. Metrics used :

Metrics are used to monitor and evaluate the performance of a model during training and testing. These are used by companies to track it's progress towards performance goals. Several metrics are used to determine the performance of the model.

Confusion matrix

A good and yet simple metric that should always be used when dealing with classification problems is the confusion matrix.

Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

The dataset has a binary classification task. The confusion matrix is used to determine the performance of the classification models for a given set of test data.

It is a tabular visualization of the truth labels versus the model's predictions. Each row of the confusion matrix represents instances in a predicted class and each column represents instances in an actual class.

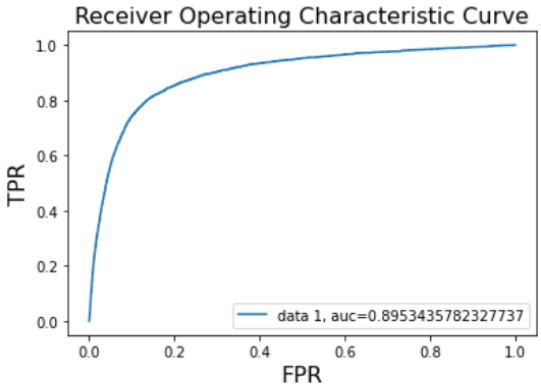
ROC and AUROC:

The ROC curve plot is between FPR and TPR.This ROC curve starts at point (0,0), ends at point (1,1) and is increasing. A good model will have a curve that increases quickly from 0 to 1 (meaning that only a little precision has to be sacrificed to get a high recall)

The Dataset is Imbalanced . AUROC has better Sensitivity to the Imbalanced Dataset. Whereas, the Accuracy Fails in this Case. AUROC measures the entire two

dimensional area present underneath the ROC curve. The ROC graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis). Higher the AUROC assumes the better performance of the model. AUROC measures how well our predictions ranked rather than their absolute values.

The AUROC tends towards 1.0 for the best case and 0.5 for the worst case.



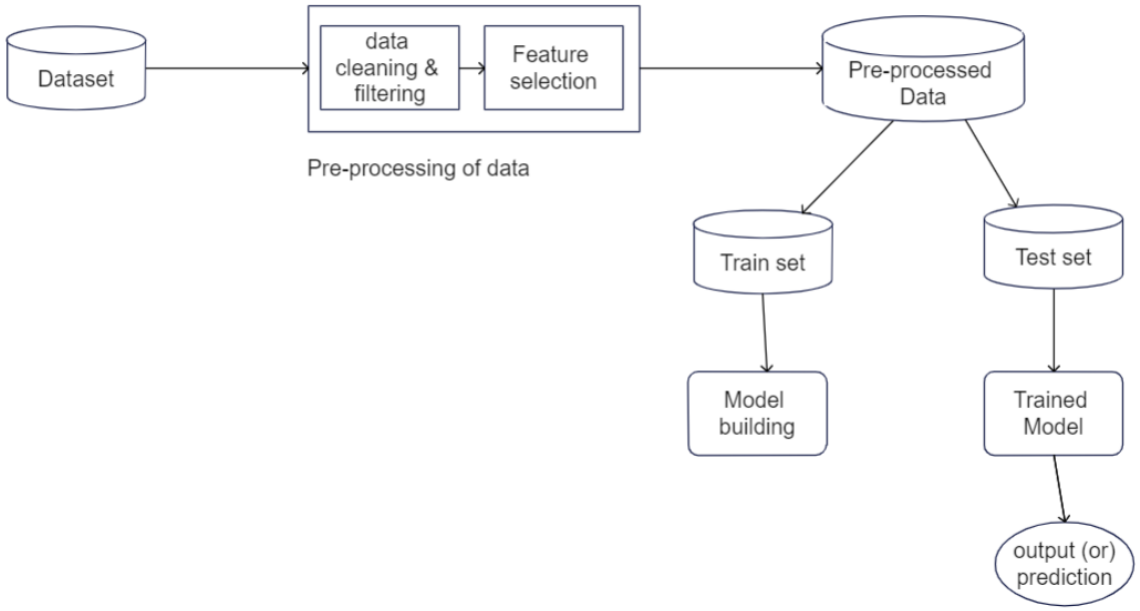

```
# Print AUC
```

Failures of AUROC:

Sometimes we need well calibrated probability outputs. AUC won't tell us about probabilities.

Wide disparities between false negatives and false positives,it is difficult to minimize one type of error. AUROC is not useful for this type of optimization.

9 . Architecture Diagram:



10. Model Building:

After splitting the preprocessed data into Train and Test,

we use

- Trained data —---> For training the model
- Test data —----> For testing our trained model

Apply a Dummy model:

Apply a dummy classifier using sklearn on upsampled data. The dummy classifier gives us a baseline performance. We expect all other future models will give better performance than this dummy classifier. The dummy classifier gives 49% on train data and 50% on test data. after SMOTE.

In [268]:
```
from sklearn.dummy import DummyClassifier

# Initialize Estimator
dummy_clf = DummyClassifier(strategy='stratified')
dummy_clf.fit(x_tr,y_tr)

# Check for Model Accuracy
dummy_clf.score(x_tr,y_tr)
y_pred = dummy_clf.predict(x_test)
print("AUROC: ", roc_auc_score(y_test, y_pred))
```

```
AUROC:  0.5091856801097026
```

Baseline model: Logistic Regression

Apply a baseline model like Logistic regression with default parameters.It always gives poor predictions, so your other models will look good by comparison.
This gives AUC of 0.82.

Complex models:

Next, apply complex models like **Decision tree ,Randomforest and Xgboost** models with default parameters. These give better performance than the baseline model as
Decision tree : AUROC –0.8465
Random forest : AUROC–0.852
Xgboost : AUROC– 0.857

Hyperparameter Tuning:

This is the process of determining the right combination of hyperparameters that allows the model to maximize model performance.

Setting the correct combination of hyperparameters is the only way to extract the maximum performance out of models.

We can Improve the performance of these models by applying better hyperparameter tuning.
There are different methods used for hyperparameter tuning like

- Random search
- Gridsearch
- Bayesian

Gridsearch is a simple method used here.

Logistic regression:

With hyperparameter tuning, LR gives an AUC of 89%.

DT model:

Apply gridsearch on a different set of parameters. This will result in the best hyperparameters.
Apply model classifier on those best hyperparameters and trin the model.

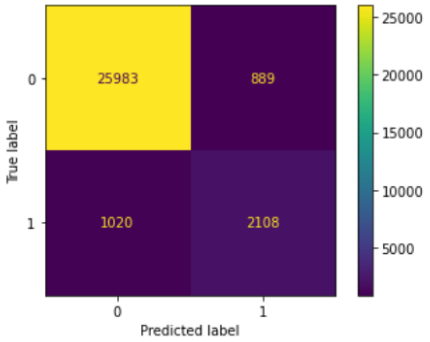
```
# model with optimal hyperparameters
import time
start_time = time.time()

clf_gini = DecisionTreeClassifier(criterion = 'gini',
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=25,
                                  min_samples_split=50)
print(clf_gini.fit(x_tr, y_tr))
end_time = time.time()
print("total time:",end_time - start_time)
```

XGBoost:

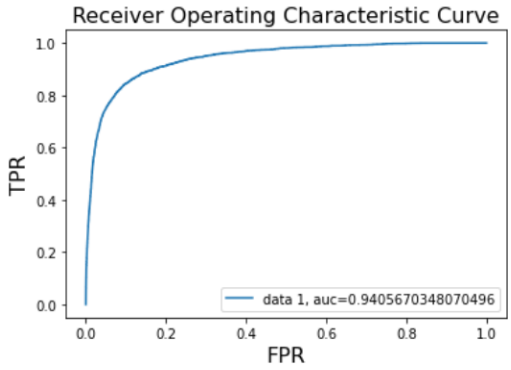
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.

In [143]:
```
# model with optimal hyperparameters
import time
start_time = time.time()

clf = XGBClassifier( max_depth=2,
                     learning_rate = 0.3,
                    n_estimators = 200,
                   sub_sample = 0.2)
print(clf.fit(x_tr, y_tr))
end_time = time.time()
print("total time:",end_time - start_time)
```

In [147]:
```
cm = confusion_matrix(y_test, y_predict, labels=search.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=search.classes_)
disp.plot()

plt.show()
```



In [148]:
```
import sklearn.metrics as metrics

y_pred_proba = clf.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))

plt.title('Receiver Operating Characteristic Curve', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlabel('FPR', fontsize=16)

plt.legend(loc=4)
plt.show()
```



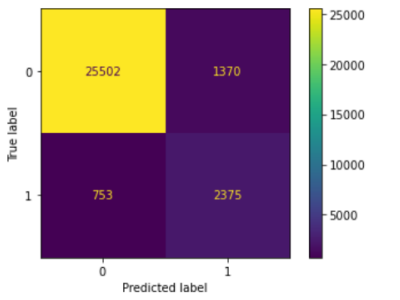
By observing the above confusion matrix and roc curve, xgboost gives best performance.

RF Model:

Random forest (RF) models that make output predictions by combining outcomes from a sequence of regression decision trees. Each tree is constructed independently and depends on a random vector sampled from the input data, with all the trees in the forest having the same distribution.

```
clf = RandomForestClassifier(max_depth=32, max_features=30,
                             n_estimators=300)



print(clf.fit(x_tr, y_tr))
```

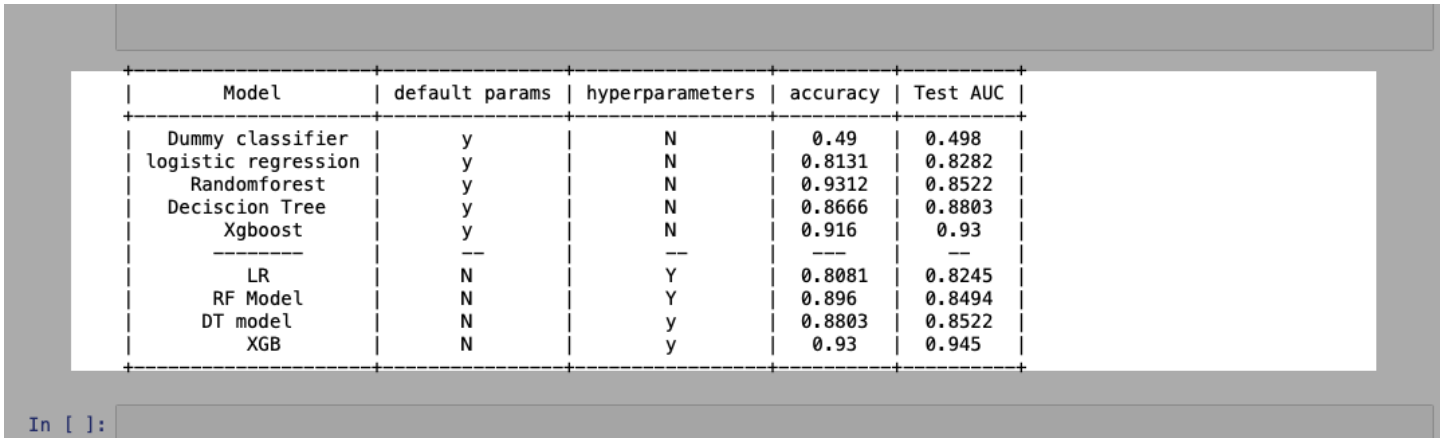
```
In [154]: cm = confusion_matrix(y_test, y_pred, labels=search.classes_)
          disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                        display_labels=search.classes_)
          disp.plot()

          plt.show()
```



XGBoost and RF models with hyperparameter tuning gives almost same AUCROC of 94%. But by observing the confusion matrix , xgboost gives the best performance.

11. Comparison of models:

The performance of models with default parameters and with fine tune hyperparameters is shown below.



Model	default params	hyperparameters	accuracy	Test AUC
Dummy classifier	y	N	0.49	0.498
logistic regression	y	N	0.8131	0.8282
Randomforest	y	N	0.9312	0.8522
Deciscion Tree	y	N	0.8666	0.8803
Xgboost	y	N	0.916	0.93
--------	--	--	---	--
LR	N	Y	0.8081	0.8245
RF Model	N	Y	0.896	0.8494
DT model	N	y	0.8803	0.8522
XGB	N	y	0.93	0.945

By observing all the models with default and hyperparameters, the xgboost model with hyperparameter tuning gives best results.

12. Advanced Modeling and Feature Engineering

In the Advanced modeling concept we combine various models through a voting classifier for better performance. Mix one or two weaker machine learning models to solve a problem and get better outcomes. Here we combine different models like SVC and RF, XGBoost and Logistic Regression, XGBoost and RandomForest, SVC and Decision Tree models.

Voting classifier is a powerful ensemble classifier. It was built by combining different models, which turns out to be a stronger meta-classifier that balances out the individual classifier's weakness on a particular dataset. Voting classifier takes majority voting based on weights applied to the class.

SVC and RF： we combine svc and RfF models for better performance.

```
: from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC


# group / ensemble of models
estimator = []
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))
estimator.append(('RF', RandomForestClassifier(max_depth=32, max_features=30,
                            n_estimators=300)))
```

```
In [158]: # Voting Classifier with soft voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(x_tr, y_tr)
y_pred = vot_soft.predict(x_test)

# using accuracy_score
score = accuracy_score(y_test, y_pred)
print("Soft Voting Score % d" % score)
```

```
Soft Voting Score  0
```

```
159]: print("AUC: ", roc_auc_score(y_test, y_pred))
```

```
AUC:  0.8601532109407182
```

By combining SVC and RF models , it gives a roc_auc_score of 0.860.

XGboost and logistic regression give a roc_auc_score of 0.848.

XGboost and RF classifier with soft voting classifier gives roc_auc_scor of 0.841.

SVC and DT models

In [168]:
```
# group / ensemble of models
estimator = []
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))

estimator.append(("DT", DecisionTreeClassifier(criterion = 'gini',
                                    random_state = 100,
                                    max_depth=10,
                                    min_samples_leaf=25,
                                    min_samples_split=50)))


# Voting Classifier with hard voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(x_tr, y_tr)
y_pred = vot_soft.predict(x_test)

# using accuracy_score metric to predict accuracy
score = accuracy_score(y_test, y_pred)
print("soft Voting Score % d" % score)
```

```
soft Voting Score  0
```

In [169]:
```
print("AUC: ", roc_auc_score(y_test, y_pred))
```

```
AUC:  0.8621457012461845
```

combining SVC and DT models , it gives a roc_auc_score of 0.862.

13. Model interpretability:

Interpretability is the degree to which a model can be understood in human terms. We say a model is an interpretable model if it can be understood without any other aids/techniques. Linear models are more interpretable than others.

We tried different models with hyperparameters and different combinations of models. Out of them, xgboost only gives higher performance. But interpretability of xgboost is harder than simple linear models.

In the combination of SVC and RF, SVC and DT models give higher performance. The interpretability of SVC and RF is harder than SVC and DT. In Decision Trees simple if-else conditions are used to make decisions. So it is easy to interpret.

Interpretable models are

- Easier to explain
- Easy to check and fix errors
- Easy to learn from model
- Easy to determine future importance

Model	default params	hyperparameters	accuracy	Test AUC
Dummy classifier	y	N	0.49	0.498
logistic regression	y	N	0.8131	0.8282
Randomforest	y	N	0.9312	0.8522
Deciscion Tree	y	N	0.8666	0.8803
Xgboost	y	N	0.916	0.93
--------	--	--	---	--
LR	N	Y	0.8081	0.8245
RF Model	N	Y	0.92	0.85
DT model	N	y	0.8803	0.8522
XGB	N	y	0.93	0.945
----------	---	-----	----	----
SVC+RF	N	Y	--	0.944
XGB+LR	N	Y	--	0.944
XGB+RF	N	Y	--	0.946
SVCF+DT	N	Y	--	0.94

By observing the above models, xgboost with hyperparameters and XGB+RF with hyperparameters give the best AUC. But interpretability of RF is very hard. By observing the confusion matrix XGBoost predicts more number of non-churned customers than XGB+RF. So we choose xgboost with hyperparameters. This gives better performance than others.

14 . Pros and cons of XGBoost model:

- Less feature engineering required,
- Handles large data sets
- Less execution time
- Good model performance
- Less prone to overfitting.
- But it is difficult to interpret.
- Hard to tune a lot of parameters.
- Leads to overfitting if parameters are not tuned properly.
- Better to use for classification problems with large datasets with more features.

Ablation analysis:

In this analysis remove some features from the dataset and apply model on remaining features and compare the results.

In ablation analysis by removing the last last few rows the model performance is decreased. This means that the removed features have some importance in predicting the churned customers.

15. Feature importance:

After choosing xgboost as our model,we can calculate the model - based on feature importance.

```
: # feature_importance
  importance = model.feature_importances_

  # create dataframe
  feature_importance = pd.DataFrame({'variables': X.columns, 'importance_percentage': importance*100})
  feature_importance = feature_importance[['variables', 'importance_percentage']]

  # sort features
  feature_importance = feature_importance.sort_values('importance_percentage', ascending=False).reset_index(drop=True)
  print("Sum of importance=", feature_importance.importance_percentage.sum())
  feature_importance
```

```
Sum of importance= 100.0
```

Out[113]:

	variables	importance_percentage
0	total_ic_mou_8	21.35
1	roam_og_mou_8	9.80
2	loc_ic_mou_8	7.27
3	loc_ic_mou_t_8	5.49
4	total_rech_amt_data_8	4.21
5	last_rech_date_8	3.50
6	roam_ic_mou_8	2.78
7	spl_og_mou_7	2.27
8	arpu_diff	2.12
9	count_rech_2g_8	2.04

Extracting top features based on model-based feature importance.

```
In [114]: # extract top 'n' features
          top_n = 30
          top_features = feature_importance.variables[0:top_n]
          top_features.values
```

```
Out[114]: array(['total_ic_mou_8', 'roam_og_mou_8', 'loc_ic_mou_8',
                 'loc_ic_mou_t_8', 'total_rech_amt_data_8', 'last_rech_date_8',
                 'roam_ic_mou_8', 'spl_og_mou_7', 'arpu_diff', 'count_rech_2g_8',
                 'last_day_rch_amt_8', 'isd_ic_mou_6', 'og_others_6',
                 'max_rech_amt_8', 'isd_og_mou_8', 'vol_2g_mb_7', 'loc_ic_mou_6',
                 'max_rech_data_8', 'spl_ic_mou_8', 'last_day_rch_amt_6',
                 'last_day_rch_amt_7', 'roam_ic_mou_6', 'count_rech_2g_7',
                 'ic_others_7', 'last_rech_date_6', 'monthly_2g_6', 'monthly_2g_7',
                 'count_rech_2g_6', 'count_rech_3g_7', 'isd_og_mou_7'], dtype=object)
```

We train our model on important features which were extracted from model-based feature importance.

```
In [125]: #model with top most features
          import time
          start_time = time.time()
          model = XGBClassifier( max_depth=2,
                                 learning_rate = 0.3,
                                n_estimators = 200,
                               sub_sample = 1)
          model.fit(x_out, y_tr)

          end_time = time.time()
          print("total time:",end_time - start_time)
```

16. Save the Model:

After the train we save our model . There are two ways to save models.

Using pickle and joblib

```
134]: import joblib
       joblib.dump(model,'r_model.pkl')
```

```
134]: ['r_model.pkl']
```

17 . Load the train model:

After saving the model, again load the trained model.

In [ ]:
```
import joblib
# Load the model
loaded_model = joblib.load('r_model.pkl')
```

Now we make predictions on this loaded model.

In [ ]:
```
# Use the loaded model to make predictions
response_model = loaded_model.predict(x_test)
response_model.shape
```

18. Evaluate the loaded model:

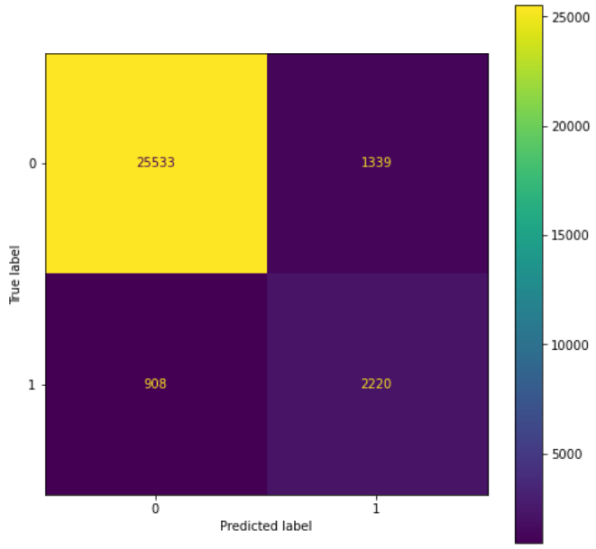
Make the predictions on the loaded model and compute the confusion matrix, roc_auc_score.

```
import time
start_time = time.time()

# predict churn on test data
y_pred_load = loaded_model.predict(x_test)
end_time = time.time()
print("total time:",end_time - start_time)
print(y_pred_load)
```

Confusion matrix:

Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.

```
# create onfusion matrix
cm_matrix = confusion_matrix(y_test, y_pred_load)
disp = ConfusionMatrixDisplay(confusion_matrix=cm_matrix,
                              display_labels=model.classes_)
disp.plot()

plt.show()
```



ROC: 0.93

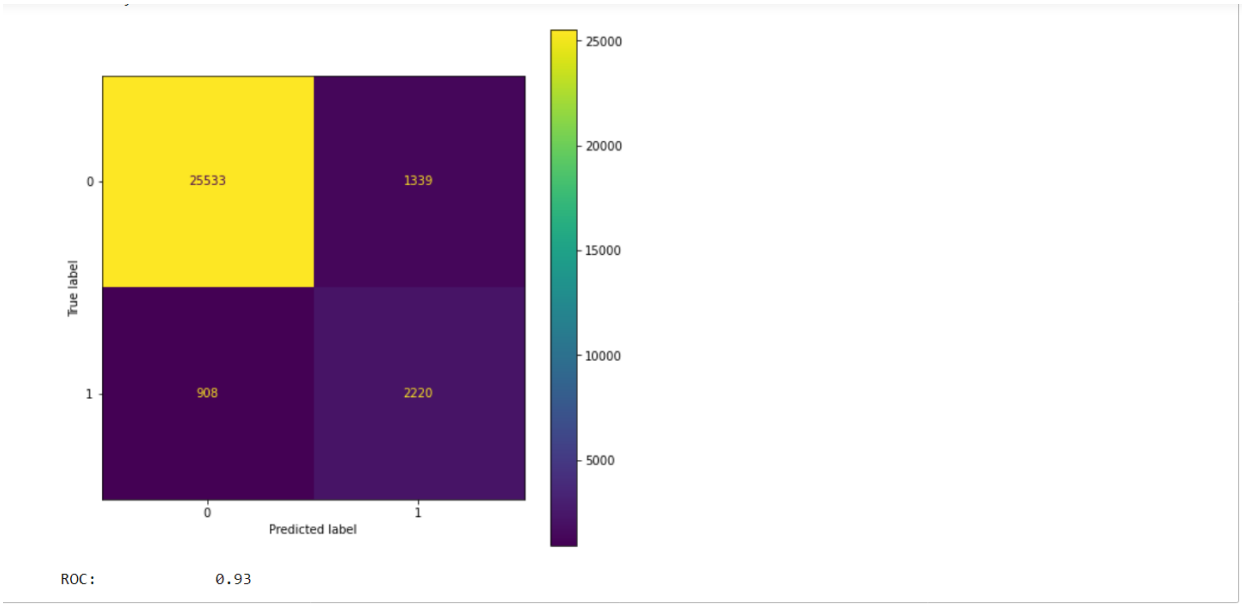
19. Evaluation of trained model object:

```
In [139]: #predict churn
          y_predict = model.predict(x_test)
          print('Accuracy: {}'.format(accuracy_score(y_test, y_predict)))
```

Make predictions on a trained model and also compute the confusion matrix , roc_auc_score. Compare the results of both to show our model accuracy.

Confusion matrix:

```
print(confusion_matrix(y_test,y_predict))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=model.classes_)
disp.plot()

plt.show()

# check area under curve
y_predict = model.predict(x_test)
print("ROC:    \t", round(roc_auc_score(y_test, y_pred_prob_load),2))
```



conclusion:

To reduce customer churn, telecom companies need to predict the customers who are at high risk of churn.

In this we analyzed customer-level data , built predictive models to identify customers at high risk of churn and identified the main factors which were the cause of churn. If Telecom companies focus on those factors,they can easily retain the customers.

20. References :

The Data set can be downloaded from
https://www.kaggle.com/vijaysrikanth/telecom-churn-data-set-for-the-south-asian-market

http://appliedroots.com