Telecom Churn Case Study

BY RISHABH, ARUNIMA, NAREN

Problem Statement

Business problem overview

In the telecommunications sector, customers have the option to select from various service providers and frequently switch between operators. In this fiercely competitive market, the industry faces an annual churn rate averaging between 15-25%.

Considering the substantial cost, which is 5-10 times higher, associated with acquiring a new customer compared to retaining an existing one, the focus has shifted towards prioritizing customer retention over acquisition.

For many established operators, the primary business objective is retaining highly profitable customers. To mitigate customer churn, telecom companies must proactively anticipate which customers are prone to high churn risk.

This project involves analyzing customer-level data from a prominent telecom firm, constructing predictive models to pinpoint customers with a high risk of churn, and identifying the key indicators influencing churn.

Understanding and defining churn

The telecom industry operates on two primary payment models: postpaid, where customers settle a monthly or annual bill after utilizing services, and prepaid, where customers pay or recharge with a predetermined amount in advance and subsequently consume services.

In the postpaid model, instances of churn are relatively straightforward to identify as customers typically notify the existing operator when switching to another provider, facilitating direct recognition of churn. Conversely, in the prepaid model, customers wishing to switch networks can cease service usage without notice. Distinguishing whether this constitutes actual churn or temporary service suspension (e.g., due to travel abroad) poses challenges.

Churn prediction is notably more critical and intricate for prepaid customers, necessitating a careful definition of the term 'churn.' Additionally, it's worth noting that the prepaid model is predominant in India and Southeast Asia, whereas the postpaid model is more prevalent in Europe and North America.

This project is focused on the Indian and Southeast Asian markets.

Churn can be defined in various ways, including:

Revenue-Based Churn: Identifying customers who haven't utilized revenue-generating services like mobile internet, outgoing calls, SMS, etc., within a specified timeframe. Metrics such as 'customers generating less than INR 4 per month in total/average/median revenue' can also be employed. However, this definition may fall short as it doesn't account for customers who receive calls/SMS from earning counterparts without generating revenue, especially common in rural areas.

Usage-Based Churn: Spotting customers who haven't engaged in any usage, whether incoming or outgoing, such as calls or internet usage, over a specific period. A drawback of this definition is that, by the time a customer has ceased using services for a while, it might be too late to take corrective actions to retain them. For instance, predicting churn based on a 'two-months zero usage' period could be ineffective as the customer might have already switched to another operator.

In this project, the churn definition will be based on usage patterns.

High-value churn

In the markets of India and Southeast Asia, a substantial 80% of the revenue is contributed by the top 20% of customers, commonly referred to as high-value customers. Consequently, mitigating churn among these high-value customers holds the potential to significantly curtail revenue losses. This project focuses on categorizing high-value customers using a specific metric (outlined below) and exclusively forecasting churn within this segment..

Understanding the business objective and the data

The dataset encompasses individual customer details spanning four sequential months: June (6), July (7), August (8), and September (9). The primary goal is to forecast churn in the final month (September) by utilizing data (features) extracted from the initial three months. A comprehensive grasp of typical customer behavior during churn proves invaluable for effectively addressing this task.

Understanding customer behaviour during churn

Customers usually do not decide to switch to another competitor instantly, but rather over a period of time (this is especially applicable to high-value customers). In churn prediction, we assume that there are three phases of customer lifecycle:

The ‘good’ phase: In this phase, the customer is happy with the service and behaves as usual.

The ‘action’ phase: The customer experience starts to sore in this phase, for e.g. he/she gets a compelling offer from a competitor, faces unjust charges, becomes unhappy with service quality etc. In this phase, the customer usually shows different behaviour than the ‘good’ months. Also, it is crucial to identify high-churn-risk customers in this phase, since some corrective actions can be taken at this point (such as matching the competitor’s offer/improving the service quality etc.)

The ‘churn’ phase: In this phase, the customer is said to have churned. You define churn based on this phase. Also, it is important to note that at the time of prediction (i.e. the action months), this data is not available to you for prediction. Thus, after tagging churn as 1/0 based on this phase, you discard all data corresponding to this phase

In this case, since you are working over a four-month window, the first two months are the ‘good’ phase, the third month is the ‘action’ phase, while the fourth month is the ‘churn’ phase.

Importing the necessary libraries

In other to accomplish the process explained in this article, we will need to import some Python libraries. In our Jupyter notebook, we insert the following lines:

```
# Basic libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import time

# Supressing the warnings generated
import warnings
warnings.filterwarnings('ignore')

# Importing Pandas EDA tool
import pandas_profiling as pp
from pandas_profiling import ProfileReport

# Displaying all Columns without restrictions
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
```

Loading the Dataset

In order to load the dataset we make use of the pandas library that we imported in the previous section:

```python
# Reading the csv data file.
telecom_data = pd.read_csv("telecom_churn_data.csv")
```

We use pandas library to get an extract of the dataset, thanks to the head function, that looks as follows:

```python
# Displaying the first 10 field with all columns in the dataset
telecom_data.head(10)
```

	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	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
1	7001865778	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
2	7001625959	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
3	7001204172	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
4	7000142493	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
5	7000286308	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
6	7001051193	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
7	7000701601	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
8	7001524846	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
9	7001864400	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	

Checking the dimensions of the dataset

```
# Checking the dimensions of the dataset
telecom_data.shape
```

```
(99999, 226)
```

Checking the information regarding the dataset

```
telecom_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 226 columns):
 #     Column                    Dtype
---    ------                    -----
 0     mobile_number             int64
 1     circle_id                 int64
 2     loc_og_t2o_mou            float64
 3     std_og_t2o_mou            float64
 4     loc_ic_t2o_mou            float64
 5     last_date_of_month_6      object
 6     last_date_of_month_7      object
 7     last_date_of_month_8      object
 8     last_date_of_month_9      object
 9     arpu_6                    float64
 10    arpu_7                    float64
 11    arpu_8                    float64
 12    arpu_9                    float64
 13    onnet_mou_6               float64
```

This telecom dataset has 99999 rows and 226 columns

Initial Statistical Analysis of the Data

```
# Statistical analysis of the numercial features
telecom_data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
mobile_number	99999.0	7.001207e+09	695669.386290	7.000000e+09	7.000606e+09	7.001205e+09	7.001812e+09	7.002411e+09
circle_id	99999.0	1.090000e+02	0.000000	1.090000e+02	1.090000e+02	1.090000e+02	1.090000e+02	1.090000e+02
loc_og_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
std_og_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
loc_ic_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
arpu_6	99999.0	2.829874e+02	328.439770	-2.258709e+03	9.341150e+01	1.977040e+02	3.710600e+02	2.773109e+04
arpu_7	99999.0	2.785366e+02	338.156291	-2.014045e+03	8.698050e+01	1.916400e+02	3.653445e+02	3.514583e+04
arpu_8	99999.0	2.791547e+02	344.474791	-9.458080e+02	8.412600e+01	1.920800e+02	3.693705e+02	3.354362e+04
arpu_9	99999.0	2.616451e+02	341.998630	-1.899505e+03	6.268500e+01	1.768490e+02	3.534665e+02	3.880562e+04
onnet_mou_6	96062.0	1.323959e+02	297.207406	0.000000e+00	7.380000e+00	3.431000e+01	1.187400e+02	7.376710e+03
onnet_mou_7	96140.0	1.336708e+02	308.794148	0.000000e+00	6.660000e+00	3.233000e+01	1.155950e+02	8.157780e+03
onnet_mou_8	94621.0	1.330181e+02	308.951589	0.000000e+00	6.460000e+00	3.236000e+01	1.158600e+02	1.075256e+04

Validating unique values

lets check the columns unique values and drop such columns with its value as 1

This telecom dataset has 99999 rows and 226 columns

```
unique_1_col=[]
for i in telecom_data.columns:
    if telecom_data[i].nunique() == 1:
        unique_1_col.append(i)
    else:
        pass

telecom_data.drop(unique_1_col, axis=1, inplace = True)
print("\n The following Columns are dropped from the dataset as their unique value is 1. (i.e.)It has no variance in the model\n'
      unique_1_col)
```

```
 The following Columns are dropped from the dataset as their unique value is 1. (i.e.)It has no variance in the model
 ['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_month_9', 'std_og_t2c_mou_6', 'std_og_t2c_mou_7', 'std_og_t2c_mou_8', 'std_og_t2c_mou_9', 'std_ic_t2o_mou_6', 'std_ic_t2o_mou_7', 'std_ic_t2o_mou_8', 'std_ic_t2o_mou_9']
```

Missing values:

As we can see that the columns with datetime values represented as object, they can be converted into datetime format

```
max_rech_data_6             74.85
fb_user_6                   74.85
count_rech_3g_6             74.85
count_rech_2g_6             74.85
night_pck_user_6            74.85
arpu_3g_6                   74.85
total_rech_data_6           74.85
av_rech_amt_data_6          74.85
arpu_2g_6                   74.85
date_of_last_rech_data_6    74.85
arpu_3g_7                   74.43
night_pck_user_7            74.43
total_rech_data_7           74.43
date_of_last_rech_data_7    74.43
av_rech_amt_data_7          74.43
max_rech_data_7             74.43
fb_user_7                   74.43
count_rech_3g_7             74.43
arpu_2g_7                   74.43
```

Columns which represents as object

```
# selecting all the columns with datetime format
date_col= telecom_data.select_dtypes(include=['object'])
print("\nThese are the columns available with datetime format represented as object\n",date_col.columns)

# Converting the selected columns to datetime format
for i in date_col.columns:
    telecom_data[i] = pd.to_datetime(telecom_data[i])

# Current dimension of the dataset
telecom_data.shape
```

```
These are the columns available with datetime format represented as object
 Index(['date_of_last_rech_6', 'date_of_last_rech_7', 'date_of_last_rech_8',
       'date_of_last_rech_9', 'date_of_last_rech_data_6',
       'date_of_last_rech_data_7', 'date_of_last_rech_data_8',
       'date_of_last_rech_data_9'],
      dtype='object')

(99999, 210)
```

Handling missing values

Consider the "date_of_last_rech_data" column, denoting the date of the last recharge for mobile internet in a given month. If both "total_rech_data" and "max_rech_data" also contain missing values, these gaps in the mentioned columns are deemed meaningful. Consequently, filling these missing values with zeros is appropriate.

In this context, meaningful missing implies that the customer has not performed any recharge for mobile internet.

```
# Handling missing values with respect to `data recharge` attributes
telecom_data[['date_of_last_rech_data_6','total_rech_data_6','max_rech_data_6']].head(10)
```

	date_of_last_rech_data_6	total_rech_data_6	max_rech_data_6
0	2014-06-21	1.0	252.0
1	NaT	NaN	NaN
2	NaT	NaN	NaN
3	NaT	NaN	NaN
4	2014-06-04	1.0	56.0
5	NaT	NaN	NaN
6	NaT	NaN	NaN
7	NaT	NaN	NaN
8	NaT	NaN	NaN
9	NaT	NaN	NaN

Correlation Validation

From the below correlation table between attributes arpu_2g_* and arpu_3g_* for each month from 6 to 9 respectively is highly correlated to the attribute av_rech_amt_data_* for each month from 6 to 9 respectively.

Considering the high correlation between them, it is safer to drop the attributes arpu_2g_* and arpu_3g_*.

```
print("Correlation table for month 6\n\n", telecom_data[['arpu_3g_6','arpu_2g_6','av_rech_amt_data_6']].corr())
print("\nCorrelation table for month 7\n\n", telecom_data[['arpu_3g_7','arpu_2g_7','av_rech_amt_data_7']].corr())
print("\nCorrelation table for month 8\n\n", telecom_data[['arpu_3g_8','arpu_2g_8','av_rech_amt_data_8']].corr())
print("\nCorrelation table for month 9\n\n", telecom_data[['arpu_3g_9','arpu_2g_9','av_rech_amt_data_9']].corr())
```

```
Correlation table for month 6

                    arpu_3g_6  arpu_2g_6  av_rech_amt_data_6
arpu_3g_6            1.000000   0.932232   0.809695
arpu_2g_6            0.932232   1.000000   0.834065
av_rech_amt_data_6   0.809695   0.834065   1.000000

Correlation table for month 7

                    arpu_3g_7  arpu_2g_7  av_rech_amt_data_7
arpu_3g_7            1.000000   0.930366   0.796131
arpu_2g_7            0.930366   1.000000   0.815933
av_rech_amt_data_7   0.796131   0.815933   1.000000

Correlation table for month 8

                    arpu_3g_8  arpu_2g_8  av_rech_amt_data_8
arpu_3g_8            1.000000   0.924925   0.787165
arpu_2g_8            0.924925   1.000000   0.805482
av_rech_amt_data_8   0.787165   0.805482   1.000000

Correlation table for month 9

                    arpu_3g_9  arpu_2g_9  av_rech_amt_data_9
arpu_3g_9            1.000000   0.852253   0.722932
arpu_2g_9            0.852253   1.000000   0.817815
av_rech_amt_data_9   0.722932   0.817815   1.000000
```

Handling the other attributes with higher missing value percentage

The column fb_user_* and night_pck_user_* for each month from 6 to 9 respectively has a missing values above 50% and does not seem to add any information to understand the data. Hence we can drop these columns for further analysis.

```
telecom_data.drop(['fb_user_6','fb_user_7','fb_user_8','fb_user_9',
                   'night_pck_user_6','night_pck_user_7','night_pck_user_8','night_pck_user_9'],
                   axis=1, inplace=True)
print("\nThe columns 'fb_user_6','fb_user_7','fb_user_8','fb_user_9','night_pck_user_6','night_pck_user_7','night_pck_user_8','ni
```

```
The columns 'fb_user_6','fb_user_7','fb_user_8','fb_user_9','night_pck_user_6','night_pck_user_7','night_pck_user_8','night_pck
_user_9' are dropped from the dataset as it has no meaning to the data snd has high missing values above 50%
```

```
# The curent dimensions of the dataset
telecom_data.shape
```

```
(99999, 186)
```

missing values for the attributes

From the above tabular it is deduced that the missing values for the column **av_rech_amt_data_*** for each month from 6 to 9 can be replaced as 0 if the **total_rech_data_*** for each month from 6 to 9 respectively is 0. i.e. if the total recharge done is 0 then the average recharge amount shall also be 0.

```
# Checking the related columns values
telecom_data[['av_rech_amt_data_7','max_rech_data_7','total_rech_data_7']].head(10)
```

	av_rech_amt_data_7	max_rech_data_7	total_rech_data_7
0	252.0	252.0	1.0
1	154.0	154.0	1.0
2	NaN	0.0	0.0
3	NaN	0.0	0.0
4	NaN	0.0	0.0
5	NaN	0.0	0.0
6	NaN	0.0	0.0
7	NaN	0.0	0.0
8	177.0	154.0	2.0
9	154.0	154.0	1.0

conditional imputation

Execution Time = 189.69 seconds

The columns 'av_rech_amt_data_6','av_rech_amt_data_7','av_rech_amt_data_8' and 'av_rech_amt_data_9' are imputed with 0 based on the condition explained below

```
# Code for conditional imputation
start_time = time.time()
for i in range(len(telecom_data)):
  # Handling `av_rech_amt_data`  for month 6
    if (pd.isnull(telecom_data['av_rech_amt_data_6'][i]) and (telecom_data['total_rech_data_6'][i]==0)):
        telecom_data['av_rech_amt_data_6'][i] = 0

  # Handling `av_rech_amt_data`  for month 7
    if (pd.isnull(telecom_data['av_rech_amt_data_7'][i]) and (telecom_data['total_rech_data_7'][i]==0)):
        telecom_data['av_rech_amt_data_7'][i] = 0

  # Handling `av_rech_amt_data`  for month 8
    if (pd.isnull(telecom_data['av_rech_amt_data_8'][i]) and (telecom_data['total_rech_data_8'][i]==0)):
        telecom_data['av_rech_amt_data_8'][i] = 0

  # Handling `av_rech_amt_data`  for month 9
    if (pd.isnull(telecom_data['av_rech_amt_data_9'][i]) and (telecom_data['total_rech_data_9'][i]==0)):
        telecom_data['av_rech_amt_data_9'][i] = 0


end_time=time.time()
print("\nExecution Time = ", round(end_time-start_time,2),"seconds")
print("\nThe columns 'av_rech_amt_data_6','av_rech_amt_data_7','av_rech_amt_data_8' and 'av_rech_amt_data_9' are imputed with 0 b
```

overall missing values

From the above results, we can conclude, the **date_of_last_rech_data_*** corresponding to months 6,7,8 and 9 are of no value after the conditional imputation of of columns **total_rech_data_*,max_rech_data_***are completes.
Also the missing value percentage is high for these columns and can be dropped from the dataset.

```
# Checkng the overall missing values in the dataset
((telecom_data.isnull().sum()/telecom_data.shape[0])*100).round(2).sort_values(ascending=False)
```

```
total_rech_amt_6          0.00
max_rech_data_6           0.00
last_day_rch_amt_7        0.00
total_rech_data_9         0.00
total_rech_data_8         0.00
total_rech_data_7         0.00
total_rech_data_6         0.00
last_day_rch_amt_9        0.00
last_day_rch_amt_8        0.00
last_day_rch_amt_6        0.00
total_rech_amt_7          0.00
max_rech_amt_9            0.00
max_rech_amt_8            0.00
max_rech_amt_7            0.00
max_rech_amt_6            0.00
total_rech_amt_9          0.00
total_rech_amt_8          0.00
sep_vbc_3g                0.00
dtype: float64
```

```
telecom_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Columns: 186 entries, mobile_number to sep_vbc_3g
dtypes: datetime64[ns](8), float64(144), int64(34)
memory usage: 141.9 MB
```

Filtering the High Value Customer from Good Phase

We are filtering the data in accordance to total revenue generated per customer.

first we need the total amount recharge amount done for data alone, we have average recharge amount done.

```
 # Calculating the total recharge amount done for data alone in months 6,7,8 and 9
telecom_data['total_rech_amt_data_6']=telecom_data['av_rech_amt_data_6'] * telecom_data['total_rech_data_6']
telecom_data['total_rech_amt_data_7']=telecom_data['av_rech_amt_data_7'] * telecom_data['total_rech_data_7']

# Calculating the overall recharge amount for the months 6,7,8 and 9
telecom_data['overall_rech_amt_6'] = telecom_data['total_rech_amt_data_6'] + telecom_data['total_rech_amt_6']
telecom_data['overall_rech_amt_7'] = telecom_data['total_rech_amt_data_7'] + telecom_data['total_rech_amt_7']

# Calculating the average recharge done by customer in months June and July(i.e. 6th and 7th month)
telecom_data['avg_rech_amt_6_7'] = (telecom_data['overall_rech_amt_6'] + telecom_data['overall_rech_amt_7'])/2

# Finding the value of 70th percentage in the overall revenues defining the high value customer creteria for the company
cut_off = telecom_data['avg_rech_amt_6_7'].quantile(0.70)
print("\nThe 70th quantile value to determine the High Value Customer is: ",cut_off,"\n")

# Filtering the data to the top 30% considered as High Value Customer
telecom_data = telecom_data[telecom_data['avg_rech_amt_6_7'] >= cut_off]
```

```
The 70th quantile value to determine the High Value Customer is:  478.0
```

Defining Churn variable

As explained above in the introduction, we are deriving based on usage based for this model.

For that, we need to find the derive churn variable using total_ic_mou_9,total_og_mou_9,vol_2g_mb_9 and vol_3g_mb_9 attributes

```python
# Initializing the churn variable.
telecom_data['churn']=0

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

```python
# Checking the top 10 data
telecom_data.head(10)
```

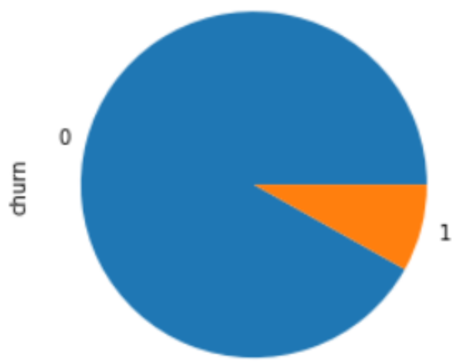
	mobile_number	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_mou_8	onnet_mou_9	offnet_mou_6	offnet_mou_7	offnet_mou_
0	7.000843e+09	197.385	214.816	213.803	21.100	53.27	24.613333	0.00	33.590000	84.23	23.993333	0.(
7	7.000702e+09	1069.180	1349.850	3171.480	500.000	57.84	54.680000	52.29	65.276667	453.43	567.160000	325.!
8	7.001525e+09	378.721	492.223	137.362	166.787	413.69	351.030000	35.08	33.460000	94.66	80.630000	136..
21	7.002124e+09	514.453	597.753	637.760	578.596	102.41	132.110000	85.14	161.630000	757.93	896.680000	983.:
23	7.000887e+09	74.350	193.897	366.966	811.480	48.96	50.660000	33.58	15.740000	85.41	89.360000	205.!
33	7.000150e+09	977.020	2362.833	409.230	799.356	0.00	0.000000	0.00	0.000000	0.00	0.000000	0.(
38	7.000815e+09	363.987	486.558	393.909	391.709	248.99	619.960000	666.38	494.790000	88.86	50.580000	97.!
41	7.000721e+09	482.832	425.764	229.769	143.596	86.39	118.880000	80.44	40.060000	232.36	280.780000	136.(
48	7.000294e+09	1873.271	575.927	179.218	1189.744	2061.69	881.430000	156.91	1589.230000	1087.76	258.290000	68.'
53	7.002189e+09	978.077	1141.296	706.020	1076.247	135.14	119.590000	102.69	99.830000	479.31	543.180000	261.:

Churn/non churn percentage

As we can see that 91% of the customers do not churn, there is a possibility of class imbalance
Since this variable churn is the target variable, all the columns relating to this variable
(i.e. all columns with suffix _9) can be dropped forn the dataset.

```
]: # lets find out churn/non churn percentage
print((telecom_data['churn'].value_counts()/len(telecom_data))*100)
((telecom_data['churn'].value_counts()/len(telecom_data))*100).plot(kind="pie")
plt.show()
```

```
0    91.863605
1     8.136395
Name: churn, dtype: float64
```



Collineartity of the indepedent variables

```
# creating a list of column names for each month
mon_6_cols = [col for col in telecom_data.columns if '_6' in col]
mon_7_cols = [col for col in telecom_data.columns if '_7' in col]
mon_8_cols = [col for col in telecom_data.columns if '_8' in col]
```

```
# lets check the correlation amongst the independent variables, drop the highly correlated ones
telecom_data_corr = telecom_data.corr()
telecom_data_corr.loc[:,:] = np.tril(telecom_data_corr, k=-1)
telecom_data_corr = telecom_data_corr.stack()
telecom_data_corr
telecom_data_corr[(telecom_data_corr > 0.80) | (telecom_data_corr < -0.80)].sort_values(ascending=False)
```

```
col_to_drop=['total_rech_amt_8','isd_og_mou_8','isd_og_mou_7','sachet_2g_8','total_ic_mou_6',
             'total_ic_mou_8','total_ic_mou_7','std_og_t2t_mou_6','std_og_t2t_mou_8','std_og_t2t_mou_7',
             'std_og_t2m_mou_7','std_og_t2m_mou_8',]

# These columns can be dropped as they are highly collinered with other predictor variables.
# criteria set is for collinearity of 85%

#  dropping these column
telecom_data.drop(col_to_drop, axis=1, inplace=True)
```

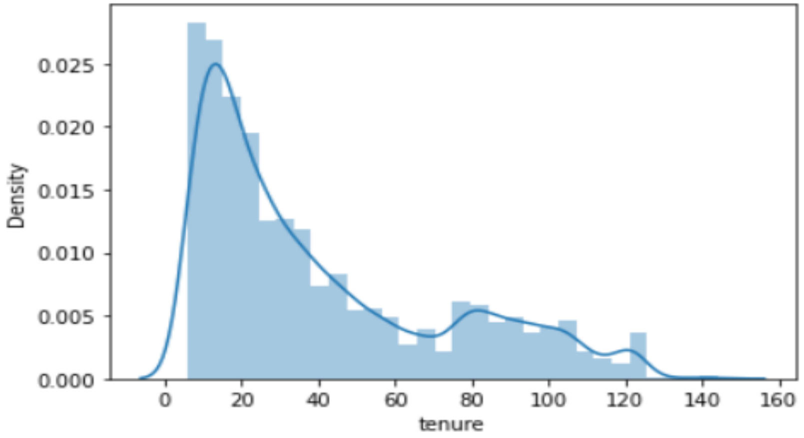
```
# The curent dimension of the dataset after dropping few unwanted columns
telecom_data.shape
```

```
(30001, 121)
```

Deriving new variables to understand the data

```
# We have a column called 'aon'

# we can derive new variables from this to explain the data w.r.t churn.

# creating a new variable 'tenure'
telecom_data['tenure'] = (telecom_data['aon']/30).round(0)

# Since we derived a new column from 'aon', we can drop it
telecom_data.drop('aon',axis=1, inplace=True)
```

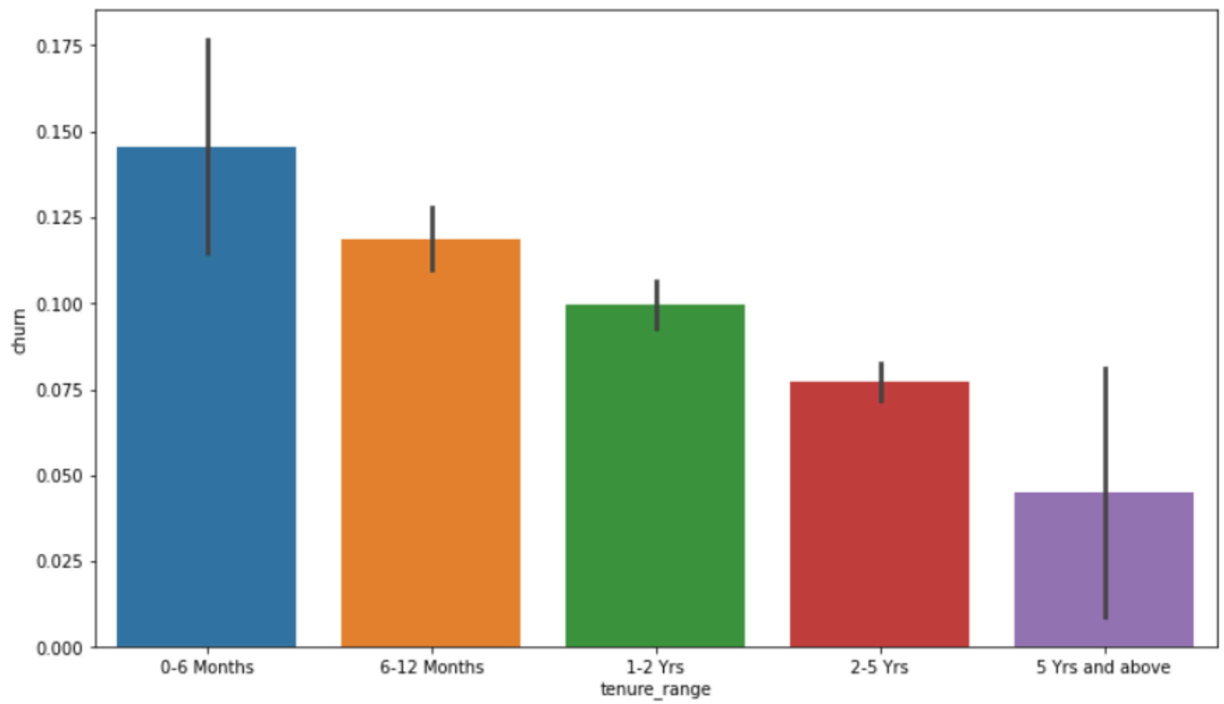
```
# Checking the distribution of he tenure variable

sns.distplot(telecom_data['tenure'],bins=30)
plt.show()
```



Tenure range

It can be seen that the maximum churn rate happens within 0-6 month, but it gradually decreases as the customer retains in the network.

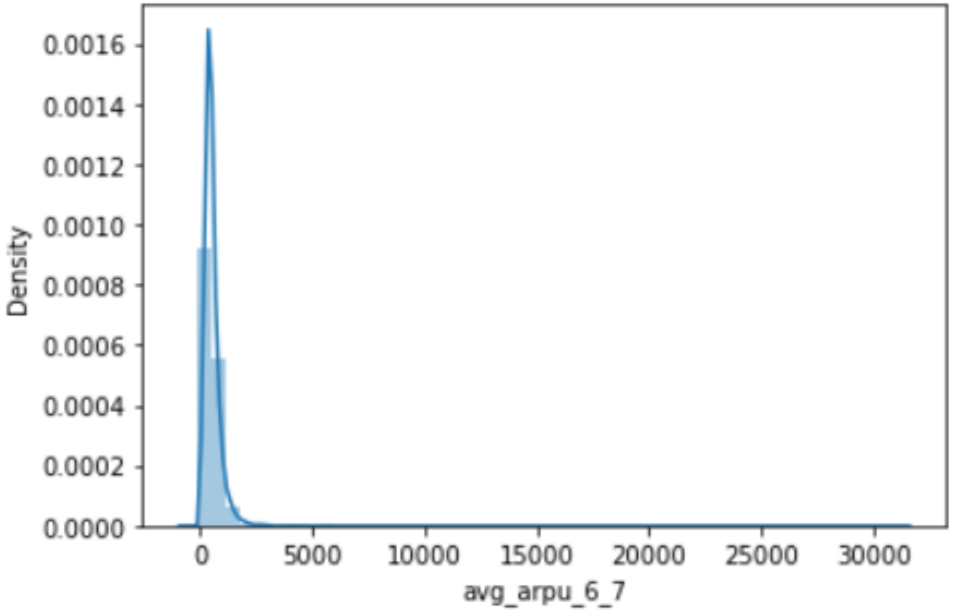
plt.figure(figsize=[12,7])
sns.barplot(x='tenure_range',y='churn', data=telecom_data)
plt.show()



The average revenue per user

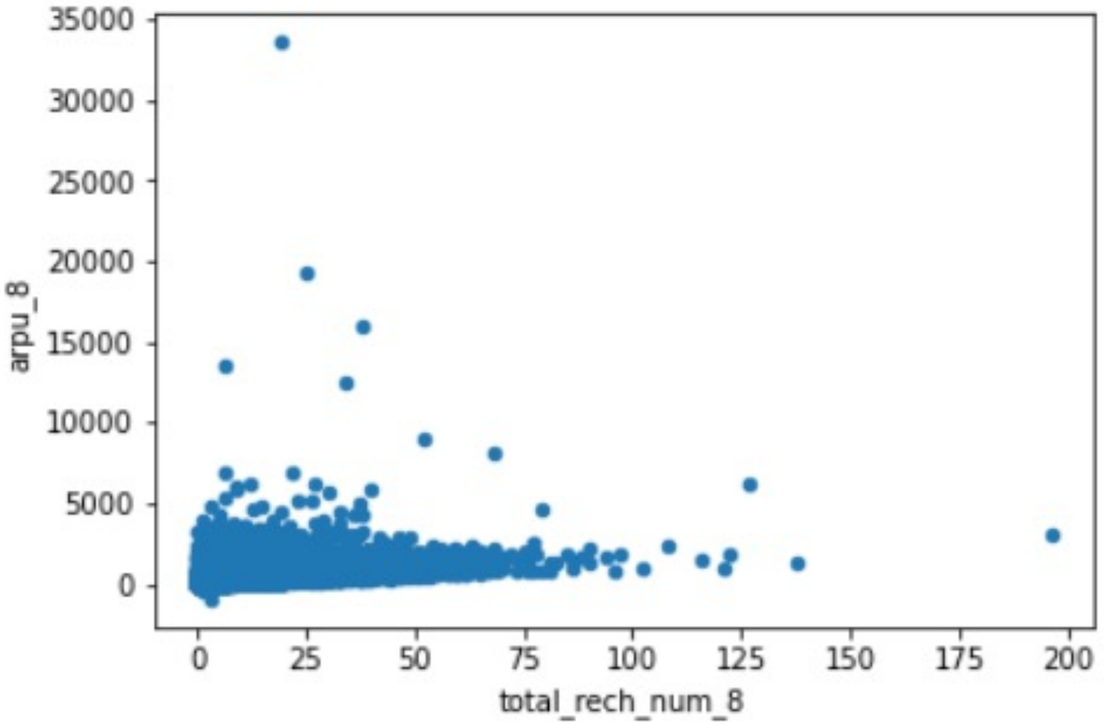
it is good phase of customer is given by arpu_6 and arpu_7. since we have two seperate averages, lets take an average to these two and drop the other columns.

```
# Visualizing the column created
sns.distplot(telecom_data['avg_arpu_6_7'])
plt.show()
```

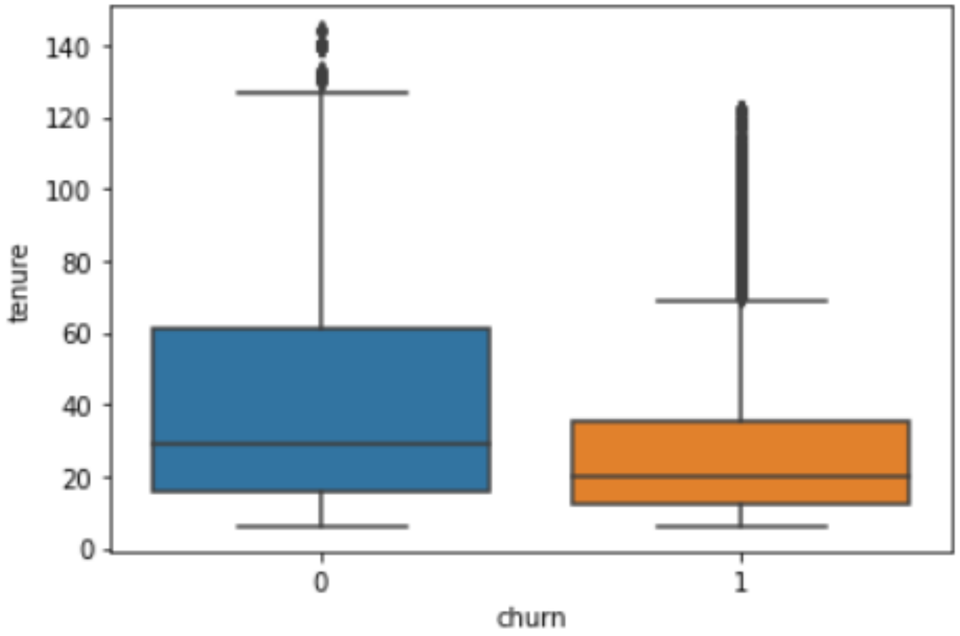


Sale Price

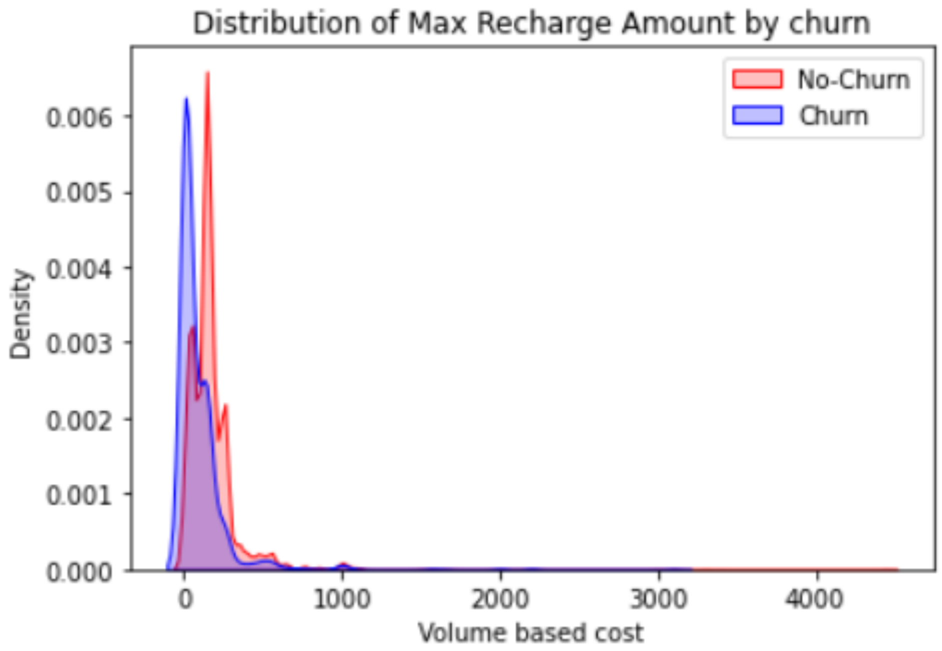
- Avg Outgoing Calls & calls on romaning for 6 & 7th months are positively correlated with churn.
- Avg Revenue, No. Of Recharge for 8th month has negative correlation with churn.



From the below plot , its clear tenured customers do no churn and they keep availing telecom services

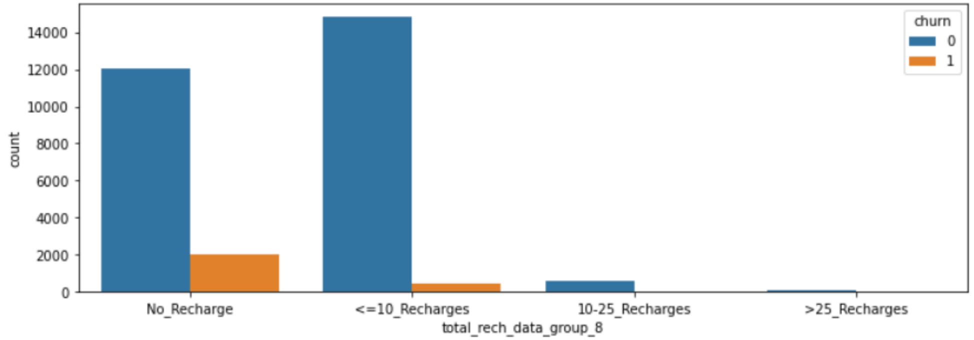


max rechare amount



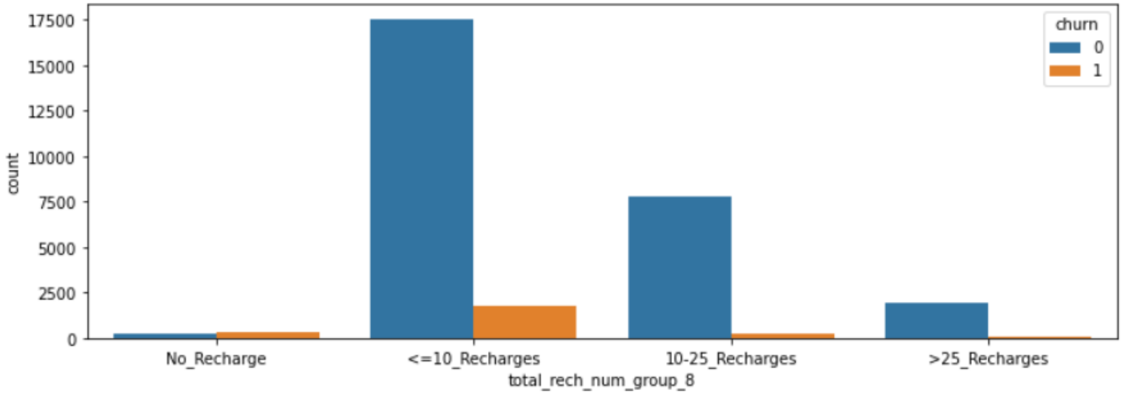
Number of recharge rate increases, the churn rate decreases clearly.

```
Distribution of total_rech_data_8 variable
 <=10_Recharges     15307
No_Recharge        14048
10-25_Recharges    608
>25_Recharges      38
Name: total_rech_data_group_8, dtype: int64
```




```
Distribution of total_rech_num_8 variable
```

```
Distribution of total_rech_num_8 variable
 <=10_Recharges     19349
10-25_Recharges    8073
>25_Recharges      1996
No_Recharge        583
Name: total_rech_num_group_8, dtype: int64
```



Data Imbalance

Using SMOTE method, we can balance the data w.r.t. churn variable and proceed further

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_train_sm,y_train_sm = sm.fit_resample(X_train,y_train)
```

```
print("Dimension of X_train_sm Shape:", X_train_sm.shape)
print("Dimension of y_train_sm Shape:", y_train_sm.shape)
```

```
Dimension of X_train_sm Shape: (38576, 126)
Dimension of y_train_sm Shape: (38576,)
```

Logistic Regression

using Feature Selection (RFE method

Generalized Linear Model Regression Results

Dep. Variable:	churn	**No. Observations:**	38576
Model:	GLM	**Df Residuals:**	38450
Model Family:	Binomial	**Df Model:**	125
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	nan
Date:	Mon, 01 Mar 2021	**Deviance:**	nan
Time:	15:17:56	**Pearson chi2:**	2.47e+14
No. Iterations:	100		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.0696	0.152	7.047	0.000	0.772	1.367

Churn flag and the predicted probabilities

```
y_train_sm_pred_final = pd.DataFrame({'Converted':y_train_sm.values, 'Converted_prob':y_train_sm_pred})
y_train_sm_pred_final.head()
```

	Converted	Converted_prob
0	0	0.138574
1	0	0.401122
2	0	0.324276
3	0	0.414619
4	0	0.508730

```
y_train_sm_pred_final['churn_pred'] = y_train_sm_pred_final.Converted_prob.map(lambda x: 1 if x > 0.5 else 0)

# Viewing the prediction results
y_train_sm_pred_final.head()
```

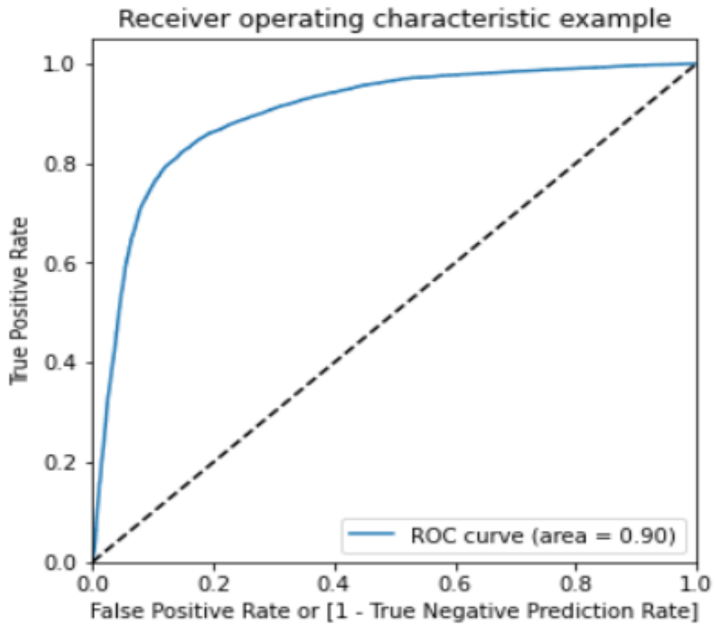
	Converted	Converted_prob	churn_pred
0	0	0.138574	0
1	0	0.401122	0
2	0	0.324276	0
3	0	0.414619	0
4	0	0.508730	1

Plotting the ROC Curve

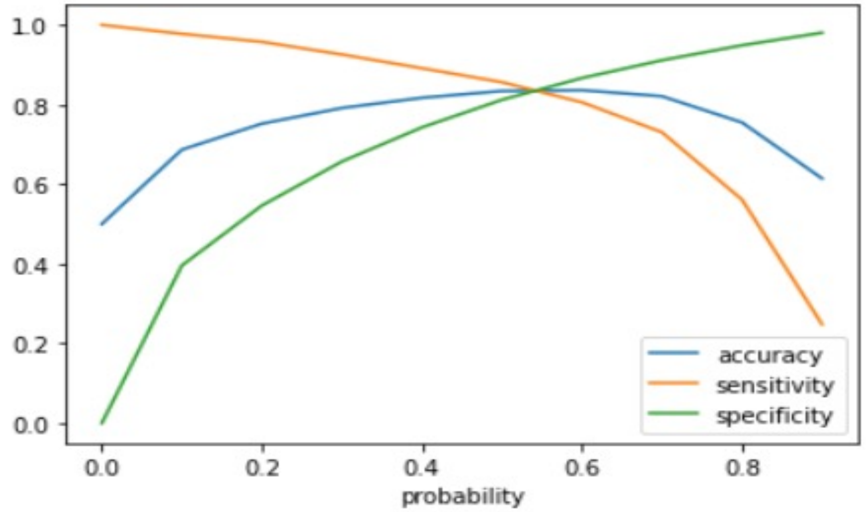
Plotting the curve for the obtained metrics

Initially we selected the optimum point of classification as **0.5.**

From the above graph, we can see the optimum cutoff is slightly higher than 0.5 but lies lower than **0.6**. So lets tweak a little more within this range



<Figure size 1080x1080 with 0 Axes>



sensitivity and specificity for various probabilities

Metrics Evaluation & validation parameters

```
confusion2_test = metrics.confusion_matrix(y_pred_final.churn, y_pred_final.test_churn_pred)
print("Confusion Matrix\n",confusion2_test)
```

Confusion Matrix
 [[6860 1412]
 [145 584]]

```
# Let's see the sensitivity of our logistic regression model
print("Sensitivity = ",TP3 / float(TP3+FN3))

# Let us calculate specificity
print("Specificity = ",TN3 / float(TN3+FP3))

# Calculate false postive rate - predicting churn when customer does not have churned
print("False Positive Rate = ",FP3/ float(TN3+FP3))

# positive predictive value
print ("Precision = ",TP3 / float(TP3+FP3))

# Negative predictive value
print ("True Negative Prediction Rate = ",TN3 / float(TN3+FN3))
```

Sensitivity = 0.8010973936899863
Specificity = 0.8293036750483559
False Positive Rate = 0.1706963249516441
Precision = 0.2925851703406814
True Negative Prediction Rate = 0.979300499643112

Performing Logistic Regression

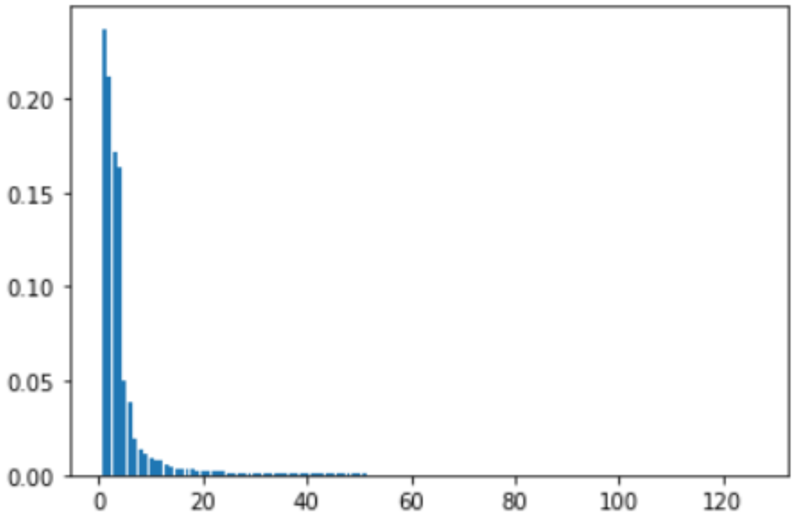
Accuracy of the logistic regression model with PCA: 0.818131318742362

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg_pca = LogisticRegression()
logreg_pca.fit(X_train_sm_pca, y_train_sm)

# making the predictions
y_pred = logreg_pca.predict(X_test_pca)

# converting the prediction into a dataframe
y_pred_df = pd.DataFrame(y_pred)
print("Dimension of y_pred_df:", y_pred_df.shape)
```

```
Dimension of y_pred_df: (9001, 1)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Checking the Confusion matrix
print("Confusion Matirx for y_test & y_pred\n",confusion_matrix(y_test,y_pred),"\n")

# Checking the Accuracy of the Predicted model.
print("Accuracy of the logistic regression model with PCA: ",accuracy_score(y_test,y_pred))
```

```
Confusion Matirx for y_test & y_pred
 [[6761 1511]
 [ 126  603]]
```



Conclusion

The Random Forest model highlights the top 5 crucial features, with a focus on data from month 8, the active month. These features include:

1. Last day recharge amount for month 8
2. Local incoming minutes over call for month 8
3. Average recharge amount data for month 8
4. Average roaming for outgoing minutes over call
5. Local T2M incoming minutes over call for month 8

Additionally, several other features, primarily associated with month 8, contribute to the model's insights.

Recommendations

Drawing insights from the modeling exercise, effective strategies to address customer churn among High-Value Prepaid customers include:

1. Prioritize attention on features related to the 'active' month, such as local calls, STD calls, incoming calls, outgoing calls, Operator T to T (mobile to mobile), Operator T to other operator mobile, average revenue per user, roaming calls, etc.
2. Vigilantly monitor any decrease in Minutes of Usage (MOU) or recharge amount compared to the previous month for these key features.
3. Implement competitive schemes that surpass offerings from other operators for high-value customers.
4. Utilize Customer Relationship Management (CRM) notifications, emails, or messages to engage and retain these customers.
5. Introduce enticing offers or plans tailored to the preferences of these high-value customers.
6. Establish trust by creating irresistible propositions for these consumers, making it challenging for them to consider switching to another operator.


THANK
YOU