Telecom Churn Case Study

Problem Statement

Business problem overview

In the telecom industry, customers are able to choose from multiple service providers and actively switch from one operator to another. In this highly competitive market, the telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, **customer retention** has now become even more important than customer acquisition.

For many incumbent operators, retaining high profitable customers is the number one business goal.

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

In this project, you will analyse customer-level data of a leading telecom firm, build predictive models to identify customers at high risk of churn and identify the main indicators of churn.

Understanding and defining churn

There are two main models of payment in the telecom industry - `postpaid` (customers pay a monthly/annual bill after using the services) and `prepaid` (customers pay/recharge with a certain amount in advance and then use the services).

In the postpaid model, when customers want to switch to another operator, they usually inform the existing operator to terminate the services, and you directly know that this is an instance of churn.

However, in the prepaid model, customers who want to switch to another network can simply stop using the services without any notice, and it is hard to know whether someone has actually churned or is simply not using the services temporarily (e.g. someone may be on a trip abroad for a month or two and then intend to resume using the services again).

Thus, churn prediction is usually more critical (and non-trivial) for prepaid customers, and the term 'churn' should be defined carefully. Also, prepaid is the most common model in India and Southeast Asia, while postpaid is more common in Europe in North America.

This project is based on the Indian and Southeast Asian market.

Definitions of churn

There are various ways to define churn, such as:

`Revenue-based churn` : Customers who have not utilised any revenue-generating facilities such as mobile internet, outgoing calls, SMS etc. over a given period of time. One could also use aggregate metrics such as 'customers who have generated less than INR 4 per month in total/average/median revenue'.

The main shortcoming of this definition is that there are customers who only receive calls/SMSes from their wage-earning counterparts, i.e. they don't generate revenue but use the services. For example, many users in rural areas only receive calls from their wage-earning siblings in urban areas.

`Usage-based churn` : Customers who have not done any usage, either incoming or outgoing - in terms of calls, internet etc. over a period of time.

A potential shortcoming of this definition is that when the customer has stopped using the services for a while, it may be too late to take any corrective actions to retain them. For e.g., if you define churn based on a 'two-months zero usage' period, predicting churn could be useless since by that time the customer would have already switched to another operator.

In this project, you will use the `usage-based definition` to define churn.

High-value churn

In the Indian and the Southeast Asian market, approximately 80% of revenue comes from the top 20% customers (called high-value customers). Thus, if we can reduce churn of the high-value customers, we will be able to reduce significant revenue leakage.

In this project, you will define high-value customers based on a certain metric (mentioned later below) and predict churn only on high-value customers.

Understanding the business objective and the data

The dataset contains customer-level information for a span of four consecutive months - June, July, August and September. The months are encoded as 6, 7, 8 and 9, respectively.

The business objective is to predict the churn in the last (i.e. the ninth) month using the data (features) from the first three months. To do this task well, understanding the typical customer behaviour during churn will be helpful.

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 Libraries

```python
# 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)
```

Importing the Dataset

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

```
# 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_montl
0	7000842753	109	0.0	0.0	0.0	6/30/2014	7/31/2C
1	7001865778	109	0.0	0.0	0.0	6/30/2014	7/31/2C
2	7001625959	109	0.0	0.0	0.0	6/30/2014	7/31/2C
3	7001204172	109	0.0	0.0	0.0	6/30/2014	7/31/2C
4	7000142493	109	0.0	0.0	0.0	6/30/2014	7/31/2C
5	7000286308	109	0.0	0.0	0.0	6/30/2014	7/31/2C
6	7001051193	109	0.0	0.0	0.0	6/30/2014	7/31/2C
7	7000701601	109	0.0	0.0	0.0	6/30/2014	7/31/2C
8	7001524846	109	0.0	0.0	0.0	6/30/2014	7/31/2C
9	7001864400	109	0.0	0.0	0.0	6/30/2014	7/31/2C

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

```
(99999, 226)
```

```
# Checking the informations 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
 14   onnet_mou_7               float64
 15   onnet_mou_8               float64
 16   onnet_mou_9               float64
 17   offnet_mou_6              float64
 18   offnet_mou_7              float64
 19   offnet_mou_8              float64
 20   offnet_mou_9              float64
 21   roam_ic_mou_6             float64
 22   roam_ic_mou_7             float64
 23   roam_ic_mou_8             float64
 24   roam_ic_mou_9             float64
 25   roam_og_mou_6             float64
 26   roam_og_mou_7             float64
 27   roam_og_mou_8             float64
 28   roam_og_mou_9             float64
 29   loc_og_t2t_mou_6          float64
 30   loc_og_t2t_mou_7          float64
 31   loc_og_t2t_mou_8          float64
 32   loc_og_t2t_mou_9          float64
 33   loc_og_t2m_mou_6          float64
 34   loc_og_t2m_mou_7          float64
 35   loc_og_t2m_mou_8          float64
 36   loc_og_t2m_mou_9          float64
 37   loc_og_t2f_mou_6          float64
 38   loc_og_t2f_mou_7          float64
 39   loc_og_t2f_mou_8          float64
```

```
 40  loc_og_t2f_mou_9     float64
 41  loc_og_t2c_mou_6     float64
 42  loc_og_t2c_mou_7     float64
 43  loc_og_t2c_mou_8     float64
 44  loc_og_t2c_mou_9     float64
 45  loc_og_mou_6         float64
 46  loc_og_mou_7         float64
 47  loc_og_mou_8         float64
 48  loc_og_mou_9         float64
 49  std_og_t2t_mou_6     float64
 50  std_og_t2t_mou_7     float64
 51  std_og_t2t_mou_8     float64
 52  std_og_t2t_mou_9     float64
 53  std_og_t2m_mou_6     float64
 54  std_og_t2m_mou_7     float64
 55  std_og_t2m_mou_8     float64
 56  std_og_t2m_mou_9     float64
 57  std_og_t2f_mou_6     float64
 58  std_og_t2f_mou_7     float64
 59  std_og_t2f_mou_8     float64
 60  std_og_t2f_mou_9     float64
 61  std_og_t2c_mou_6     float64
 62  std_og_t2c_mou_7     float64
 63  std_og_t2c_mou_8     float64
 64  std_og_t2c_mou_9     float64
 65  std_og_mou_6         float64
 66  std_og_mou_7         float64
 67  std_og_mou_8         float64
 68  std_og_mou_9         float64
 69  isd_og_mou_6         float64
 70  isd_og_mou_7         float64
 71  isd_og_mou_8         float64
 72  isd_og_mou_9         float64
 73  spl_og_mou_6         float64
 74  spl_og_mou_7         float64
 75  spl_og_mou_8         float64
 76  spl_og_mou_9         float64
 77  og_others_6          float64
 78  og_others_7          float64
 79  og_others_8          float64
 80  og_others_9          float64
 81  total_og_mou_6       float64
 82  total_og_mou_7       float64
```

```
83   total_og_mou_8         float64
84   total_og_mou_9         float64
85   loc_ic_t2t_mou_6       float64
86   loc_ic_t2t_mou_7       float64
87   loc_ic_t2t_mou_8       float64
88   loc_ic_t2t_mou_9       float64
89   loc_ic_t2m_mou_6       float64
90   loc_ic_t2m_mou_7       float64
91   loc_ic_t2m_mou_8       float64
92   loc_ic_t2m_mou_9       float64
93   loc_ic_t2f_mou_6       float64
94   loc_ic_t2f_mou_7       float64
95   loc_ic_t2f_mou_8       float64
96   loc_ic_t2f_mou_9       float64
97   loc_ic_mou_6           float64
98   loc_ic_mou_7           float64
99   loc_ic_mou_8           float64
100  loc_ic_mou_9           float64
101  std_ic_t2t_mou_6       float64
102  std_ic_t2t_mou_7       float64
103  std_ic_t2t_mou_8       float64
104  std_ic_t2t_mou_9       float64
105  std_ic_t2m_mou_6       float64
106  std_ic_t2m_mou_7       float64
107  std_ic_t2m_mou_8       float64
108  std_ic_t2m_mou_9       float64
109  std_ic_t2f_mou_6       float64
110  std_ic_t2f_mou_7       float64
111  std_ic_t2f_mou_8       float64
112  std_ic_t2f_mou_9       float64
113  std_ic_t2o_mou_6       float64
114  std_ic_t2o_mou_7       float64
115  std_ic_t2o_mou_8       float64
116  std_ic_t2o_mou_9       float64
117  std_ic_mou_6           float64
118  std_ic_mou_7           float64
119  std_ic_mou_8           float64
120  std_ic_mou_9           float64
121  total_ic_mou_6         float64
122  total_ic_mou_7         float64
123  total_ic_mou_8         float64
124  total_ic_mou_9         float64
125  spl_ic_mou_6           float64
```

```
126  spl_ic_mou_7             float64
127  spl_ic_mou_8             float64
128  spl_ic_mou_9             float64
129  isd_ic_mou_6             float64
130  isd_ic_mou_7             float64
131  isd_ic_mou_8             float64
132  isd_ic_mou_9             float64
133  ic_others_6              float64
134  ic_others_7              float64
135  ic_others_8              float64
136  ic_others_9              float64
137  total_rech_num_6         int64
138  total_rech_num_7         int64
139  total_rech_num_8         int64
140  total_rech_num_9         int64
141  total_rech_amt_6         int64
142  total_rech_amt_7         int64
143  total_rech_amt_8         int64
144  total_rech_amt_9         int64
145  max_rech_amt_6           int64
146  max_rech_amt_7           int64
147  max_rech_amt_8           int64
148  max_rech_amt_9           int64
149  date_of_last_rech_6      object
150  date_of_last_rech_7      object
151  date_of_last_rech_8      object
152  date_of_last_rech_9      object
153  last_day_rch_amt_6       int64
154  last_day_rch_amt_7       int64
155  last_day_rch_amt_8       int64
156  last_day_rch_amt_9       int64
157  date_of_last_rech_data_6 object
158  date_of_last_rech_data_7 object
159  date_of_last_rech_data_8 object
160  date_of_last_rech_data_9 object
161  total_rech_data_6        float64
162  total_rech_data_7        float64
163  total_rech_data_8        float64
164  total_rech_data_9        float64
165  max_rech_data_6          float64
166  max_rech_data_7          float64
167  max_rech_data_8          float64
168  max_rech_data_9          float64
```

```
169  count_rech_2g_6          float64
170  count_rech_2g_7          float64
171  count_rech_2g_8          float64
172  count_rech_2g_9          float64
173  count_rech_3g_6          float64
174  count_rech_3g_7          float64
175  count_rech_3g_8          float64
176  count_rech_3g_9          float64
177  av_rech_amt_data_6       float64
178  av_rech_amt_data_7       float64
179  av_rech_amt_data_8       float64
180  av_rech_amt_data_9       float64
181  vol_2g_mb_6              float64
182  vol_2g_mb_7              float64
183  vol_2g_mb_8              float64
184  vol_2g_mb_9              float64
185  vol_3g_mb_6              float64
186  vol_3g_mb_7              float64
187  vol_3g_mb_8              float64
188  vol_3g_mb_9              float64
189  arpu_3g_6                float64
190  arpu_3g_7                float64
191  arpu_3g_8                float64
192  arpu_3g_9                float64
193  arpu_2g_6                float64
194  arpu_2g_7                float64
195  arpu_2g_8                float64
196  arpu_2g_9                float64
197  night_pck_user_6         float64
198  night_pck_user_7         float64
199  night_pck_user_8         float64
200  night_pck_user_9         float64
201  monthly_2g_6             int64
202  monthly_2g_7             int64
203  monthly_2g_8             int64
204  monthly_2g_9             int64
205  sachet_2g_6              int64
206  sachet_2g_7              int64
207  sachet_2g_8              int64
208  sachet_2g_9              int64
209  monthly_3g_6             int64
210  monthly_3g_7             int64
211  monthly_3g_8             int64
```

```
 212   monthly_3g_9              int64
 213   sachet_3g_6               int64
 214   sachet_3g_7               int64
 215   sachet_3g_8               int64
 216   sachet_3g_9               int64
 217   fb_user_6                 float64
 218   fb_user_7                 float64
 219   fb_user_8                 float64
 220   fb_user_9                 float64
 221   aon                       int64
 222   aug_vbc_3g                float64
 223   jul_vbc_3g                float64
 224   jun_vbc_3g                float64
 225   sep_vbc_3g                float64
dtypes: float64(179), int64(35), object(12)
memory usage: 172.4+ MB
```

This telecom dataset has 99999 rows and 226 columns

Checking the terms used in the data from data dictionary provided.

```
# Importing the excel file of the dictionary.
telecom_data_dict = pd.read_excel("Data+Dictionary-+Telecom+Churn+Case+Study.xlsx")
```

```
# Displaying the dictionary items
telecom_data_dict
```

	Acronyms	Descriptions
0	MOBILE_NUMBER	Customer phone number
1	CIRCLE_ID	Telecom circle area to which the customer belongs to
2	LOC	Local calls - within same telecom circle
3	STD	STD calls - outside the calling circle
4	IC	Incoming calls
5	OG	Outgoing calls
6	T2T	Operator T to T, i.e. within same operator (mobile to mobile)
7	T2M	Operator T to other operator mobile
8	T2O	Operator T to other operator fixed line
9	T2F	Operator T to fixed lines of T
10	T2C	Operator T to it’s own call center
11	ARPU	Average revenue per user
12	MOU	Minutes of usage - voice calls
13	AON	Age on network - number of days the customer is using the operator T network
14	ONNET	All kind of calls within the same operator network

	Acronyms	Descriptions
15	OFFNET	All kind of calls outside the operator T network
16	ROAM	Indicates that customer is in roaming zone during the call
17	SPL	Special calls
18	ISD	ISD calls
19	RECH	Recharge
20	NUM	Number
21	AMT	Amount in local currency
22	MAX	Maximum
23	DATA	Mobile internet
24	3G	3G network
25	AV	Average
26	VOL	Mobile internet usage volume (in MB)
27	2G	2G network
28	PCK	Prepaid service schemes called - PACKS
29	NIGHT	Scheme to use during specific night hours only
30	MONTHLY	Service schemes with validity equivalent to a month
31	SACHET	Service schemes with validity smaller than a month
32	*.6	KPI for the month of June
33	*.7	KPI for the month of July
34	*.8	KPI for the month of August
35	*.9	KPI for the month of September
36	FB_USER	Service scheme to avail services of Facebook and similar social networking sites
37	VBC	Volume based cost - when no specific scheme is not purchased and paid as per usage

Initial Statistical Analysis of the Data

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

	count	mean	std	min	25%	50%	
mobile_number	99999.0	7.001207e+09	695669.386290	7.000000e+09	7.000606e+09	7.001205e+09	7.00181
circle_id	99999.0	1.090000e+02	0.000000	1.090000e+02	1.090000e+02	1.090000e+02	1.09000
loc_og_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_ic_t2o_mou	98981.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
arpu_6	99999.0	2.829874e+02	328.439770	-2.258709e+03	9.341150e+01	1.977040e+02	3.71060
arpu_7	99999.0	2.785366e+02	338.156291	-2.014045e+03	8.698050e+01	1.916400e+02	3.65344
arpu_8	99999.0	2.791547e+02	344.474791	-9.458080e+02	8.412600e+01	1.920800e+02	3.69370
arpu_9	99999.0	2.616451e+02	341.998630	-1.899505e+03	6.268500e+01	1.768490e+02	3.53466
onnet_mou_6	96062.0	1.323959e+02	297.207406	0.000000e+00	7.380000e+00	3.431000e+01	1.18740

	count	mean	std	min	25%	50%	
onnet_mou_7	96140.0	1.336708e+02	308.794148	0.000000e+00	6.660000e+00	3.233000e+01	1.15595
onnet_mou_8	94621.0	1.330181e+02	308.951589	0.000000e+00	6.460000e+00	3.236000e+01	1.15860
onnet_mou_9	92254.0	1.303023e+02	308.477668	0.000000e+00	5.330000e+00	2.984000e+01	1.12130
offnet_mou_6	96062.0	1.979356e+02	316.851613	0.000000e+00	3.473000e+01	9.631000e+01	2.31860
offnet_mou_7	96140.0	1.970451e+02	325.862803	0.000000e+00	3.219000e+01	9.173500e+01	2.26815
offnet_mou_8	94621.0	1.965748e+02	327.170662	0.000000e+00	3.163000e+01	9.214000e+01	2.28260
offnet_mou_9	92254.0	1.903372e+02	319.396092	0.000000e+00	2.713000e+01	8.729000e+01	2.20505
roam_ic_mou_6	96062.0	9.950013e+00	72.825411	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_ic_mou_7	96140.0	7.149898e+00	73.447948	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_ic_mou_8	94621.0	7.292981e+00	68.402466	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_ic_mou_9	92254.0	6.343841e+00	57.137537	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_og_mou_6	96062.0	1.391134e+01	71.443196	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_og_mou_7	96140.0	9.818732e+00	58.455762	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_og_mou_8	94621.0	9.971890e+00	64.713221	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
roam_og_mou_9	92254.0	8.555519e+00	58.438186	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_og_t2t_mou_6	96062.0	4.710076e+01	150.856393	0.000000e+00	1.660000e+00	1.191000e+01	4.09600
loc_og_t2t_mou_7	96140.0	4.647301e+01	155.318705	0.000000e+00	1.630000e+00	1.161000e+01	3.99100
loc_og_t2t_mou_8	94621.0	4.588781e+01	151.184830	0.000000e+00	1.600000e+00	1.173000e+01	4.01100
loc_og_t2t_mou_9	92254.0	4.458445e+01	147.995390	0.000000e+00	1.360000e+00	1.126000e+01	3.92800
loc_og_t2m_mou_6	96062.0	9.334209e+01	162.780544	0.000000e+00	9.880000e+00	4.103000e+01	1.10390
loc_og_t2m_mou_7	96140.0	9.139713e+01	157.492308	0.000000e+00	1.002500e+01	4.043000e+01	1.07560
loc_og_t2m_mou_8	94621.0	9.175513e+01	156.537048	0.000000e+00	9.810000e+00	4.036000e+01	1.09090
loc_og_t2m_mou_9	92254.0	9.046319e+01	158.681454	0.000000e+00	8.810000e+00	3.912000e+01	1.06810
loc_og_t2f_mou_6	96062.0	3.751013e+00	14.230438	0.000000e+00	0.000000e+00	0.000000e+00	2.08000
loc_og_t2f_mou_7	96140.0	3.792985e+00	14.264986	0.000000e+00	0.000000e+00	0.000000e+00	2.09000
loc_og_t2f_mou_8	94621.0	3.677991e+00	13.270996	0.000000e+00	0.000000e+00	0.000000e+00	2.04000
loc_og_t2f_mou_9	92254.0	3.655123e+00	13.457549	0.000000e+00	0.000000e+00	0.000000e+00	1.94000
loc_og_t2c_mou_6	96062.0	1.123056e+00	5.448946	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_og_t2c_mou_7	96140.0	1.368500e+00	7.533445	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_og_t2c_mou_8	94621.0	1.433821e+00	6.783335	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_og_t2c_mou_9	92254.0	1.232726e+00	5.619021	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
loc_og_mou_6	96062.0	1.442012e+02	251.751489	0.000000e+00	1.711000e+01	6.511000e+01	1.68270
loc_og_mou_7	96140.0	1.416705e+02	248.731086	0.000000e+00	1.748000e+01	6.368500e+01	1.64382
loc_og_mou_8	94621.0	1.413282e+02	245.914311	0.000000e+00	1.711000e+01	6.373000e+01	1.66110
loc_og_mou_9	92254.0	1.387100e+02	245.934517	0.000000e+00	1.556000e+01	6.184000e+01	1.62225
std_og_t2t_mou_6	96062.0	7.982987e+01	252.476533	0.000000e+00	0.000000e+00	0.000000e+00	3.08075
std_og_t2t_mou_7	96140.0	8.329960e+01	263.631042	0.000000e+00	0.000000e+00	0.000000e+00	3.11325
std_og_t2t_mou_8	94621.0	8.328267e+01	265.486090	0.000000e+00	0.000000e+00	0.000000e+00	3.05800
std_og_t2t_mou_9	92254.0	8.234292e+01	267.184991	0.000000e+00	0.000000e+00	0.000000e+00	2.82300

	count	mean	std	min	25%	50%	
std_og_t2m_mou_6	96062.0	8.729962e+01	255.617850	0.000000e+00	0.000000e+00	3.950000e+00	5.32900
std_og_t2m_mou_7	96140.0	9.080414e+01	269.347911	0.000000e+00	0.000000e+00	3.635000e+00	5.40400
std_og_t2m_mou_8	94621.0	8.983839e+01	271.757783	0.000000e+00	0.000000e+00	3.310000e+00	5.24900
std_og_t2m_mou_9	92254.0	8.627662e+01	261.407396	0.000000e+00	0.000000e+00	2.500000e+00	4.85600
std_og_t2f_mou_6	96062.0	1.129011e+00	7.984970	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2f_mou_7	96140.0	1.115010e+00	8.599406	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2f_mou_8	94621.0	1.067792e+00	7.905971	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2f_mou_9	92254.0	1.042362e+00	8.261770	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2c_mou_6	96062.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2c_mou_7	96140.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2c_mou_8	94621.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_t2c_mou_9	92254.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_og_mou_6	96062.0	1.682612e+02	389.948499	0.000000e+00	0.000000e+00	1.164000e+01	1.44837
std_og_mou_7	96140.0	1.752214e+02	408.922934	0.000000e+00	0.000000e+00	1.109000e+01	1.50615
std_og_mou_8	94621.0	1.741915e+02	411.633049	0.000000e+00	0.000000e+00	1.041000e+01	1.47940
std_og_mou_9	92254.0	1.696645e+02	405.138658	0.000000e+00	0.000000e+00	8.410000e+00	1.42105
isd_og_mou_6	96062.0	7.982775e-01	25.765248	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
isd_og_mou_7	96140.0	7.765721e-01	25.603052	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
isd_og_mou_8	94621.0	7.912471e-01	25.544471	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
isd_og_mou_9	92254.0	7.238921e-01	21.310751	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
spl_og_mou_6	96062.0	3.916811e+00	14.936449	0.000000e+00	0.000000e+00	0.000000e+00	2.43000
spl_og_mou_7	96140.0	4.978279e+00	20.661570	0.000000e+00	0.000000e+00	0.000000e+00	3.71000
spl_og_mou_8	94621.0	5.053769e+00	17.855111	0.000000e+00	0.000000e+00	0.000000e+00	3.99000
spl_og_mou_9	92254.0	4.412767e+00	16.328227	0.000000e+00	0.000000e+00	0.000000e+00	3.23000
og_others_6	96062.0	4.541571e-01	4.125911	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
og_others_7	96140.0	3.023539e-02	2.161717	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
og_others_8	94621.0	3.337198e-02	2.323464	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
og_others_9	92254.0	4.745572e-02	3.635466	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
total_og_mou_6	99999.0	3.051334e+02	463.419481	0.000000e+00	4.474000e+01	1.451400e+02	3.72860
total_og_mou_7	99999.0	3.102312e+02	480.031178	0.000000e+00	4.301000e+01	1.415300e+02	3.78570
total_og_mou_8	99999.0	3.041195e+02	478.150031	0.000000e+00	3.858000e+01	1.386100e+02	3.69900
total_og_mou_9	99999.0	2.892792e+02	468.980002	0.000000e+00	2.551000e+01	1.254600e+02	3.53480
loc_ic_t2t_mou_6	96062.0	4.792237e+01	140.258485	0.000000e+00	2.990000e+00	1.569000e+01	4.68400
loc_ic_t2t_mou_7	96140.0	4.799052e+01	145.795055	0.000000e+00	3.230000e+00	1.574000e+01	4.58100
loc_ic_t2t_mou_8	94621.0	4.721136e+01	137.239552	0.000000e+00	3.280000e+00	1.603000e+01	4.62900
loc_ic_t2t_mou_9	92254.0	4.628179e+01	140.130610	0.000000e+00	3.290000e+00	1.566000e+01	4.51800
loc_ic_t2m_mou_6	96062.0	1.074757e+02	171.713903	0.000000e+00	1.729000e+01	5.649000e+01	1.32387
loc_ic_t2m_mou_7	96140.0	1.071205e+02	169.423620	0.000000e+00	1.859000e+01	5.708000e+01	1.30960
loc_ic_t2m_mou_8	94621.0	1.084605e+02	169.723759	0.000000e+00	1.893000e+01	5.824000e+01	1.33930

	count	mean	std	min	25%	50%	
loc_ic_t2m_mou_9	92254.0	1.061555e+02	165.492803	0.000000e+00	1.856000e+01	5.661000e+01	1.30490
loc_ic_t2f_mou_6	96062.0	1.208430e+01	40.140895	0.000000e+00	0.000000e+00	8.800000e-01	8.14000
loc_ic_t2f_mou_7	96140.0	1.259970e+01	42.977442	0.000000e+00	0.000000e+00	9.300000e-01	8.28250
loc_ic_t2f_mou_8	94621.0	1.175183e+01	39.125379	0.000000e+00	0.000000e+00	9.300000e-01	8.11000
loc_ic_t2f_mou_9	92254.0	1.217310e+01	43.840776	0.000000e+00	0.000000e+00	9.600000e-01	8.14000
loc_ic_mou_6	96062.0	1.674911e+02	254.124029	0.000000e+00	3.039000e+01	9.216000e+01	2.08075
loc_ic_mou_7	96140.0	1.677195e+02	256.242707	0.000000e+00	3.246000e+01	9.255000e+01	2.05837
loc_ic_mou_8	94621.0	1.674326e+02	250.025523	0.000000e+00	3.274000e+01	9.383000e+01	2.07280
loc_ic_mou_9	92254.0	1.646193e+02	249.845070	0.000000e+00	3.229000e+01	9.164000e+01	2.02737
std_ic_t2t_mou_6	96062.0	9.575993e+00	54.330607	0.000000e+00	0.000000e+00	0.000000e+00	4.06000
std_ic_t2t_mou_7	96140.0	1.001190e+01	57.411971	0.000000e+00	0.000000e+00	0.000000e+00	4.23000
std_ic_t2t_mou_8	94621.0	9.883921e+00	55.073186	0.000000e+00	0.000000e+00	0.000000e+00	4.08000
std_ic_t2t_mou_9	92254.0	9.432479e+00	53.376273	0.000000e+00	0.000000e+00	0.000000e+00	3.51000
std_ic_t2m_mou_6	96062.0	2.072224e+01	80.793414	0.000000e+00	0.000000e+00	2.030000e+00	1.50300
std_ic_t2m_mou_7	96140.0	2.165641e+01	86.521393	0.000000e+00	0.000000e+00	2.040000e+00	1.57400
std_ic_t2m_mou_8	94621.0	2.118321e+01	83.683565	0.000000e+00	0.000000e+00	2.030000e+00	1.53600
std_ic_t2m_mou_9	92254.0	1.962091e+01	74.913050	0.000000e+00	0.000000e+00	1.740000e+00	1.42600
std_ic_t2f_mou_6	96062.0	2.156397e+00	16.495594	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2f_mou_7	96140.0	2.216923e+00	16.454061	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2f_mou_8	94621.0	2.085004e+00	15.812580	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2f_mou_9	92254.0	2.173419e+00	15.978601	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2o_mou_6	96062.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2o_mou_7	96140.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2o_mou_8	94621.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_t2o_mou_9	92254.0	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
std_ic_mou_6	96062.0	3.245718e+01	106.283386	0.000000e+00	0.000000e+00	5.890000e+00	2.69300
std_ic_mou_7	96140.0	3.388783e+01	113.720168	0.000000e+00	0.000000e+00	5.960000e+00	2.83100
std_ic_mou_8	94621.0	3.315474e+01	110.127008	0.000000e+00	1.000000e-02	5.880000e+00	2.77100
std_ic_mou_9	92254.0	3.122934e+01	101.982303	0.000000e+00	0.000000e+00	5.380000e+00	2.56900
total_ic_mou_6	99999.0	2.001300e+02	291.651671	0.000000e+00	3.853000e+01	1.147400e+02	2.51670
total_ic_mou_7	99999.0	2.028531e+02	298.124954	0.000000e+00	4.119000e+01	1.163400e+02	2.50660
total_ic_mou_8	99999.0	1.987508e+02	289.321094	0.000000e+00	3.829000e+01	1.146600e+02	2.48990
total_ic_mou_9	99999.0	1.892143e+02	284.823024	0.000000e+00	3.237000e+01	1.058900e+02	2.36320
spl_ic_mou_6	96062.0	6.155660e-02	0.160920	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
spl_ic_mou_7	96140.0	3.358477e-02	0.155725	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
spl_ic_mou_8	94621.0	4.036134e-02	0.146147	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
spl_ic_mou_9	92254.0	1.631370e-01	0.527860	0.000000e+00	0.000000e+00	0.000000e+00	6.00000
isd_ic_mou_6	96062.0	7.460608e+00	59.722948	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
isd_ic_mou_7	96140.0	8.334936e+00	65.219829	0.000000e+00	0.000000e+00	0.000000e+00	0.00000

	count	mean	std	min	25%	50%	
isd_ic_mou_8	94621.0	8.442001e+00	63.813098	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
isd_ic_mou_9	92254.0	8.063003e+00	63.505379	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
ic_others_6	96062.0	8.546555e-01	11.955164	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
ic_others_7	96140.0	1.012960e+00	12.673099	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
ic_others_8	94621.0	9.708005e-01	13.284348	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
ic_others_9	92254.0	1.017162e+00	12.381172	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
total_rech_num_6	99999.0	7.558806e+00	7.078405	0.000000e+00	3.000000e+00	6.000000e+00	9.00000
total_rech_num_7	99999.0	7.700367e+00	7.070422	0.000000e+00	3.000000e+00	6.000000e+00	1.00000
total_rech_num_8	99999.0	7.212912e+00	7.203753	0.000000e+00	3.000000e+00	5.000000e+00	9.00000
total_rech_num_9	99999.0	6.893019e+00	7.096261	0.000000e+00	3.000000e+00	5.000000e+00	9.00000
total_rech_amt_6	99999.0	3.275146e+02	398.019701	0.000000e+00	1.090000e+02	2.300000e+02	4.37500
total_rech_amt_7	99999.0	3.229630e+02	408.114237	0.000000e+00	1.000000e+02	2.200000e+02	4.28000
total_rech_amt_8	99999.0	3.241571e+02	416.540455	0.000000e+00	9.000000e+01	2.250000e+02	4.34500
total_rech_amt_9	99999.0	3.033457e+02	404.588583	0.000000e+00	5.200000e+01	2.000000e+02	4.15000
max_rech_amt_6	99999.0	1.046375e+02	120.614894	0.000000e+00	3.000000e+01	1.100000e+02	1.20000
max_rech_amt_7	99999.0	1.047524e+02	124.523970	0.000000e+00	3.000000e+01	1.100000e+02	1.28000
max_rech_amt_8	99999.0	1.077282e+02	126.902505	0.000000e+00	3.000000e+01	9.800000e+01	1.44000
max_rech_amt_9	99999.0	1.019439e+02	125.375109	0.000000e+00	2.800000e+01	6.100000e+01	1.44000
last_day_rch_amt_6	99999.0	6.315625e+01	97.356649	0.000000e+00	0.000000e+00	3.000000e+01	1.10000
last_day_rch_amt_7	99999.0	5.938580e+01	95.915385	0.000000e+00	0.000000e+00	3.000000e+01	1.10000
last_day_rch_amt_8	99999.0	6.264172e+01	104.431816	0.000000e+00	0.000000e+00	3.000000e+01	1.30000
last_day_rch_amt_9	99999.0	4.390125e+01	90.809712	0.000000e+00	0.000000e+00	0.000000e+00	5.00000
total_rech_data_6	25153.0	2.463802e+00	2.789128	1.000000e+00	1.000000e+00	1.000000e+00	3.00000
total_rech_data_7	25571.0	2.666419e+00	3.031593	1.000000e+00	1.000000e+00	1.000000e+00	3.00000
total_rech_data_8	26339.0	2.651999e+00	3.074987	1.000000e+00	1.000000e+00	1.000000e+00	3.00000
total_rech_data_9	25922.0	2.441170e+00	2.516339	1.000000e+00	1.000000e+00	2.000000e+00	3.00000
max_rech_data_6	25153.0	1.263934e+02	108.477235	1.000000e+00	2.500000e+01	1.450000e+02	1.77000
max_rech_data_7	25571.0	1.267295e+02	109.765267	1.000000e+00	2.500000e+01	1.450000e+02	1.77000
max_rech_data_8	26339.0	1.257173e+02	109.437851	1.000000e+00	2.500000e+01	1.450000e+02	1.79000
max_rech_data_9	25922.0	1.249414e+02	111.363760	1.000000e+00	2.500000e+01	1.450000e+02	1.79000
count_rech_2g_6	25153.0	1.864668e+00	2.570254	0.000000e+00	1.000000e+00	1.000000e+00	2.00000
count_rech_2g_7	25571.0	2.044699e+00	2.768332	0.000000e+00	1.000000e+00	1.000000e+00	2.00000
count_rech_2g_8	26339.0	2.016288e+00	2.720132	0.000000e+00	1.000000e+00	1.000000e+00	2.00000
count_rech_2g_9	25922.0	1.781807e+00	2.214701	0.000000e+00	1.000000e+00	1.000000e+00	2.00000
count_rech_3g_6	25153.0	5.991333e-01	1.274428	0.000000e+00	0.000000e+00	0.000000e+00	1.00000
count_rech_3g_7	25571.0	6.217199e-01	1.394524	0.000000e+00	0.000000e+00	0.000000e+00	1.00000
count_rech_3g_8	26339.0	6.357113e-01	1.422827	0.000000e+00	0.000000e+00	0.000000e+00	1.00000
count_rech_3g_9	25922.0	6.593627e-01	1.411513	0.000000e+00	0.000000e+00	0.000000e+00	1.00000
av_rech_amt_data_6	25153.0	1.926010e+02	192.646318	1.000000e+00	8.200000e+01	1.540000e+02	2.52000

	count	mean	std	min	25%	50%	
av_rech_amt_data_7	25571.0	2.009813e+02	196.791224	5.000000e-01	9.200000e+01	1.540000e+02	2.52000
av_rech_amt_data_8	26339.0	1.975265e+02	191.301305	5.000000e-01	8.700000e+01	1.540000e+02	2.52000
av_rech_amt_data_9	25922.0	1.927343e+02	188.400286	1.000000e+00	6.900000e+01	1.640000e+02	2.52000
vol_2g_mb_6	99999.0	5.190496e+01	213.356445	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_2g_mb_7	99999.0	5.122994e+01	212.302217	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_2g_mb_8	99999.0	5.017015e+01	212.347892	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_2g_mb_9	99999.0	4.471970e+01	198.653570	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_3g_mb_6	99999.0	1.213962e+02	544.247227	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_3g_mb_7	99999.0	1.289958e+02	541.494013	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_3g_mb_8	99999.0	1.354107e+02	558.775335	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
vol_3g_mb_9	99999.0	1.360566e+02	577.394194	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
arpu_3g_6	25153.0	8.955506e+01	193.124653	-3.082000e+01	0.000000e+00	4.800000e-01	1.22070
arpu_3g_7	25571.0	8.938412e+01	195.893924	-2.604000e+01	0.000000e+00	4.200000e-01	1.19560
arpu_3g_8	26339.0	9.117385e+01	188.180936	-2.449000e+01	0.000000e+00	8.800000e-01	1.22070
arpu_3g_9	25922.0	1.002641e+02	216.291992	-7.109000e+01	0.000000e+00	2.605000e+00	1.40010
arpu_2g_6	25153.0	8.639800e+01	172.767523	-3.583000e+01	0.000000e+00	1.083000e+01	1.22070
arpu_2g_7	25571.0	8.591445e+01	176.379871	-1.548000e+01	0.000000e+00	8.810000e+00	1.22070
arpu_2g_8	26339.0	8.659948e+01	168.247852	-5.583000e+01	0.000000e+00	9.270000e+00	1.22070
arpu_2g_9	25922.0	9.371203e+01	171.384224	-4.574000e+01	0.000000e+00	1.480000e+01	1.40010
night_pck_user_6	25153.0	2.508647e-02	0.156391	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
night_pck_user_7	25571.0	2.303391e-02	0.150014	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
night_pck_user_8	26339.0	2.084362e-02	0.142863	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
night_pck_user_9	25922.0	1.597099e-02	0.125366	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_2g_6	99999.0	7.964080e-02	0.295058	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_2g_7	99999.0	8.322083e-02	0.304395	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_2g_8	99999.0	8.100081e-02	0.299568	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_2g_9	99999.0	6.878069e-02	0.278120	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_2g_6	99999.0	3.893839e-01	1.497320	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_2g_7	99999.0	4.396344e-01	1.636230	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_2g_8	99999.0	4.500745e-01	1.630263	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_2g_9	99999.0	3.931039e-01	1.347140	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_3g_6	99999.0	7.592076e-02	0.363371	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_3g_7	99999.0	7.858079e-02	0.387231	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_3g_8	99999.0	8.294083e-02	0.384947	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
monthly_3g_9	99999.0	8.634086e-02	0.384978	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_3g_6	99999.0	7.478075e-02	0.568344	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_3g_7	99999.0	8.040080e-02	0.628334	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_3g_8	99999.0	8.450085e-02	0.660234	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sachet_3g_9	99999.0	8.458085e-02	0.650457	0.000000e+00	0.000000e+00	0.000000e+00	0.00000

	count	mean	std	min	25%	50%	
fb_user_6	25153.0	9.144038e-01	0.279772	0.000000e+00	1.000000e+00	1.000000e+00	1.00000
fb_user_7	25571.0	9.087638e-01	0.287950	0.000000e+00	1.000000e+00	1.000000e+00	1.00000
fb_user_8	26339.0	8.908083e-01	0.311885	0.000000e+00	1.000000e+00	1.000000e+00	1.00000
fb_user_9	25922.0	8.609675e-01	0.345987	0.000000e+00	1.000000e+00	1.000000e+00	1.00000
aon	99999.0	1.219855e+03	954.733842	1.800000e+02	4.670000e+02	8.630000e+02	1.80750
aug_vbc_3g	99999.0	6.817025e+01	267.580450	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
jul_vbc_3g	99999.0	6.683906e+01	271.201856	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
jun_vbc_3g	99999.0	6.002120e+01	253.938223	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
sep_vbc_3g	99999.0	3.299373e+00	32.408353	0.000000e+00	0.000000e+00	0.000000e+00	0.00000

```python
# lets check the columns unique values and drop such columns with its value as 1
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
      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']
```

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

```
(99999, 210)
```

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

```
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
count_rech_2g_7	74.43
count_rech_3g_9	74.08
date_of_last_rech_data_9	74.08
count_rech_2g_9	74.08
fb_user_9	74.08
total_rech_data_9	74.08
max_rech_data_9	74.08
night_pck_user_9	74.08
arpu_2g_9	74.08
av_rech_amt_data_9	74.08
arpu_3g_9	74.08
arpu_3g_8	73.66
fb_user_8	73.66
total_rech_data_8	73.66
count_rech_2g_8	73.66
arpu_2g_8	73.66
date_of_last_rech_data_8	73.66
count_rech_3g_8	73.66
max_rech_data_8	73.66
av_rech_amt_data_8	73.66
night_pck_user_8	73.66
loc_og_t2t_mou_9	7.75
std_ic_t2m_mou_9	7.75
isd_og_mou_9	7.75
roam_og_mou_9	7.75
std_ic_t2t_mou_9	7.75
spl_og_mou_9	7.75
loc_ic_mou_9	7.75
og_others_9	7.75
roam_ic_mou_9	7.75
ic_others_9	7.75
offnet_mou_9	7.75
loc_ic_t2f_mou_9	7.75
loc_og_t2m_mou_9	7.75
loc_ic_t2t_mou_9	7.75
loc_ic_t2m_mou_9	7.75
spl_ic_mou_9	7.75

std_ic_t2f_mou_9	7.75
std_og_mou_9	7.75
std_og_t2m_mou_9	7.75
loc_og_mou_9	7.75
loc_og_t2c_mou_9	7.75
std_og_t2t_mou_9	7.75
isd_ic_mou_9	7.75
loc_og_t2f_mou_9	7.75
onnet_mou_9	7.75
std_ic_mou_9	7.75
std_og_t2f_mou_9	7.75
std_ic_t2t_mou_8	5.38
offnet_mou_8	5.38
std_ic_mou_8	5.38
loc_ic_mou_8	5.38
onnet_mou_8	5.38
loc_ic_t2m_mou_8	5.38
isd_ic_mou_8	5.38
std_ic_t2f_mou_8	5.38
loc_ic_t2f_mou_8	5.38
spl_ic_mou_8	5.38
std_ic_t2m_mou_8	5.38
ic_others_8	5.38
loc_og_t2m_mou_8	5.38
std_og_t2m_mou_8	5.38
roam_og_mou_8	5.38
loc_og_mou_8	5.38
std_og_t2t_mou_8	5.38
isd_og_mou_8	5.38
loc_og_t2t_mou_8	5.38
spl_og_mou_8	5.38
loc_og_t2c_mou_8	5.38
std_og_mou_8	5.38
og_others_8	5.38
roam_ic_mou_8	5.38
std_og_t2f_mou_8	5.38
loc_og_t2f_mou_8	5.38
loc_ic_t2t_mou_8	5.38
date_of_last_rech_9	4.76
std_og_t2t_mou_6	3.94
onnet_mou_6	3.94
std_og_t2m_mou_6	3.94
spl_ic_mou_6	3.94
loc_ic_t2m_mou_6	3.94
isd_ic_mou_6	3.94
loc_og_t2m_mou_6	3.94
ic_others_6	3.94
loc_og_t2c_mou_6	3.94
loc_og_t2f_mou_6	3.94

loc_og_mou_6	3.94
std_ic_mou_6	3.94
std_og_t2f_mou_6	3.94
offnet_mou_6	3.94
loc_ic_t2f_mou_6	3.94
std_og_mou_6	3.94
loc_og_t2t_mou_6	3.94
std_ic_t2f_mou_6	3.94
isd_og_mou_6	3.94
std_ic_t2m_mou_6	3.94
og_others_6	3.94
std_ic_t2t_mou_6	3.94
roam_og_mou_6	3.94
loc_ic_mou_6	3.94
loc_ic_t2t_mou_6	3.94
roam_ic_mou_6	3.94
spl_og_mou_6	3.94
loc_ic_mou_7	3.86
std_ic_t2t_mou_7	3.86
isd_og_mou_7	3.86
og_others_7	3.86
std_og_mou_7	3.86
loc_ic_t2t_mou_7	3.86
loc_ic_t2m_mou_7	3.86
loc_ic_t2f_mou_7	3.86
std_og_t2f_mou_7	3.86
std_ic_t2m_mou_7	3.86
std_ic_t2f_mou_7	3.86
std_ic_mou_7	3.86
std_og_t2m_mou_7	3.86
std_og_t2t_mou_7	3.86
loc_og_mou_7	3.86
spl_ic_mou_7	3.86
isd_ic_mou_7	3.86
ic_others_7	3.86
loc_og_t2c_mou_7	3.86
loc_og_t2f_mou_7	3.86
loc_og_t2m_mou_7	3.86
loc_og_t2t_mou_7	3.86
roam_og_mou_7	3.86
roam_ic_mou_7	3.86
offnet_mou_7	3.86
onnet_mou_7	3.86
spl_og_mou_7	3.86
date_of_last_rech_8	3.62
date_of_last_rech_7	1.77
date_of_last_rech_6	1.61
aug_vbc_3g	0.00
jul_vbc_3g	0.00

jun_vbc_3g	0.00
monthly_3g_8	0.00
aon	0.00
monthly_2g_8	0.00
monthly_3g_6	0.00
sachet_2g_9	0.00
sachet_2g_8	0.00
sachet_2g_7	0.00
sachet_2g_6	0.00
monthly_2g_9	0.00
monthly_2g_7	0.00
monthly_3g_7	0.00
monthly_3g_9	0.00
monthly_2g_6	0.00
sachet_3g_6	0.00
sachet_3g_7	0.00
sachet_3g_8	0.00
sachet_3g_9	0.00
mobile_number	0.00
total_ic_mou_6	0.00
vol_3g_mb_9	0.00
vol_3g_mb_8	0.00
total_rech_num_9	0.00
total_rech_num_8	0.00
total_rech_num_7	0.00
total_rech_num_6	0.00
total_ic_mou_9	0.00
total_ic_mou_8	0.00
total_ic_mou_7	0.00
arpu_6	0.00
total_og_mou_9	0.00
total_og_mou_8	0.00
total_og_mou_7	0.00
total_og_mou_6	0.00
arpu_9	0.00
arpu_8	0.00
arpu_7	0.00
total_rech_amt_6	0.00
total_rech_amt_7	0.00
total_rech_amt_8	0.00
last_day_rch_amt_9	0.00
vol_3g_mb_7	0.00
vol_3g_mb_6	0.00
vol_2g_mb_9	0.00
vol_2g_mb_8	0.00
vol_2g_mb_7	0.00
vol_2g_mb_6	0.00
last_day_rch_amt_8	0.00
total_rech_amt_9	0.00

```
last_day_rch_amt_7          0.00
last_day_rch_amt_6          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
sep_vbc_3g                  0.00
dtype: float64
```

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

```
# 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",

# 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)
```

```
# confirming the conversion of dtype
telecom_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 210 columns):
 #    Column                     Dtype
---   ------                     -----
 0    mobile_number              int64
 1    arpu_6                     float64
 2    arpu_7                     float64
 3    arpu_8                     float64
 4    arpu_9                     float64
 5    onnet_mou_6                float64
 6    onnet_mou_7                float64
```

7	onnet_mou_8	float64
8	onnet_mou_9	float64
9	offnet_mou_6	float64
10	offnet_mou_7	float64
11	offnet_mou_8	float64
12	offnet_mou_9	float64
13	roam_ic_mou_6	float64
14	roam_ic_mou_7	float64
15	roam_ic_mou_8	float64
16	roam_ic_mou_9	float64
17	roam_og_mou_6	float64
18	roam_og_mou_7	float64
19	roam_og_mou_8	float64
20	roam_og_mou_9	float64
21	loc_og_t2t_mou_6	float64
22	loc_og_t2t_mou_7	float64
23	loc_og_t2t_mou_8	float64
24	loc_og_t2t_mou_9	float64
25	loc_og_t2m_mou_6	float64
26	loc_og_t2m_mou_7	float64
27	loc_og_t2m_mou_8	float64
28	loc_og_t2m_mou_9	float64
29	loc_og_t2f_mou_6	float64
30	loc_og_t2f_mou_7	float64
31	loc_og_t2f_mou_8	float64
32	loc_og_t2f_mou_9	float64
33	loc_og_t2c_mou_6	float64
34	loc_og_t2c_mou_7	float64
35	loc_og_t2c_mou_8	float64
36	loc_og_t2c_mou_9	float64
37	loc_og_mou_6	float64
38	loc_og_mou_7	float64
39	loc_og_mou_8	float64
40	loc_og_mou_9	float64
41	std_og_t2t_mou_6	float64
42	std_og_t2t_mou_7	float64
43	std_og_t2t_mou_8	float64
44	std_og_t2t_mou_9	float64
45	std_og_t2m_mou_6	float64
46	std_og_t2m_mou_7	float64
47	std_og_t2m_mou_8	float64
48	std_og_t2m_mou_9	float64
49	std_og_t2f_mou_6	float64

```
50   std_og_t2f_mou_7          float64
51   std_og_t2f_mou_8          float64
52   std_og_t2f_mou_9          float64
53   std_og_mou_6              float64
54   std_og_mou_7              float64
55   std_og_mou_8              float64
56   std_og_mou_9              float64
57   isd_og_mou_6              float64
58   isd_og_mou_7              float64
59   isd_og_mou_8              float64
60   isd_og_mou_9              float64
61   spl_og_mou_6              float64
62   spl_og_mou_7              float64
63   spl_og_mou_8              float64
64   spl_og_mou_9              float64
65   og_others_6               float64
66   og_others_7               float64
67   og_others_8               float64
68   og_others_9               float64
69   total_og_mou_6            float64
70   total_og_mou_7            float64
71   total_og_mou_8            float64
72   total_og_mou_9            float64
73   loc_ic_t2t_mou_6          float64
74   loc_ic_t2t_mou_7          float64
75   loc_ic_t2t_mou_8          float64
76   loc_ic_t2t_mou_9          float64
77   loc_ic_t2m_mou_6          float64
78   loc_ic_t2m_mou_7          float64
79   loc_ic_t2m_mou_8          float64
80   loc_ic_t2m_mou_9          float64
81   loc_ic_t2f_mou_6          float64
82   loc_ic_t2f_mou_7          float64
83   loc_ic_t2f_mou_8          float64
84   loc_ic_t2f_mou_9          float64
85   loc_ic_mou_6              float64
86   loc_ic_mou_7              float64
87   loc_ic_mou_8              float64
88   loc_ic_mou_9              float64
89   std_ic_t2t_mou_6          float64
90   std_ic_t2t_mou_7          float64
91   std_ic_t2t_mou_8          float64
92   std_ic_t2t_mou_9          float64
```

93	std_ic_t2m_mou_6	float64
94	std_ic_t2m_mou_7	float64
95	std_ic_t2m_mou_8	float64
96	std_ic_t2m_mou_9	float64
97	std_ic_t2f_mou_6	float64
98	std_ic_t2f_mou_7	float64
99	std_ic_t2f_mou_8	float64
100	std_ic_t2f_mou_9	float64
101	std_ic_mou_6	float64
102	std_ic_mou_7	float64
103	std_ic_mou_8	float64
104	std_ic_mou_9	float64
105	total_ic_mou_6	float64
106	total_ic_mou_7	float64
107	total_ic_mou_8	float64
108	total_ic_mou_9	float64
109	spl_ic_mou_6	float64
110	spl_ic_mou_7	float64
111	spl_ic_mou_8	float64
112	spl_ic_mou_9	float64
113	isd_ic_mou_6	float64
114	isd_ic_mou_7	float64
115	isd_ic_mou_8	float64
116	isd_ic_mou_9	float64
117	ic_others_6	float64
118	ic_others_7	float64
119	ic_others_8	float64
120	ic_others_9	float64
121	total_rech_num_6	int64
122	total_rech_num_7	int64
123	total_rech_num_8	int64
124	total_rech_num_9	int64
125	total_rech_amt_6	int64
126	total_rech_amt_7	int64
127	total_rech_amt_8	int64
128	total_rech_amt_9	int64
129	max_rech_amt_6	int64
130	max_rech_amt_7	int64
131	max_rech_amt_8	int64
132	max_rech_amt_9	int64
133	date_of_last_rech_6	datetime64[ns]
134	date_of_last_rech_7	datetime64[ns]
135	date_of_last_rech_8	datetime64[ns]

```
136  date_of_last_rech_9       datetime64[ns]
137  last_day_rch_amt_6        int64
138  last_day_rch_amt_7        int64
139  last_day_rch_amt_8        int64
140  last_day_rch_amt_9        int64
141  date_of_last_rech_data_6  datetime64[ns]
142  date_of_last_rech_data_7  datetime64[ns]
143  date_of_last_rech_data_8  datetime64[ns]
144  date_of_last_rech_data_9  datetime64[ns]
145  total_rech_data_6         float64
146  total_rech_data_7         float64
147  total_rech_data_8         float64
148  total_rech_data_9         float64
149  max_rech_data_6           float64
150  max_rech_data_7           float64
151  max_rech_data_8           float64
152  max_rech_data_9           float64
153  count_rech_2g_6           float64
154  count_rech_2g_7           float64
155  count_rech_2g_8           float64
156  count_rech_2g_9           float64
157  count_rech_3g_6           float64
158  count_rech_3g_7           float64
159  count_rech_3g_8           float64
160  count_rech_3g_9           float64
161  av_rech_amt_data_6        float64
162  av_rech_amt_data_7        float64
163  av_rech_amt_data_8        float64
164  av_rech_amt_data_9        float64
165  vol_2g_mb_6               float64
166  vol_2g_mb_7               float64
167  vol_2g_mb_8               float64
168  vol_2g_mb_9               float64
169  vol_3g_mb_6               float64
170  vol_3g_mb_7               float64
171  vol_3g_mb_8               float64
172  vol_3g_mb_9               float64
173  arpu_3g_6                 float64
174  arpu_3g_7                 float64
175  arpu_3g_8                 float64
176  arpu_3g_9                 float64
177  arpu_2g_6                 float64
178  arpu_2g_7                 float64
```

```
 179  arpu_2g_8                  float64
 180  arpu_2g_9                  float64
 181  night_pck_user_6           float64
 182  night_pck_user_7           float64
 183  night_pck_user_8           float64
 184  night_pck_user_9           float64
 185  monthly_2g_6               int64
 186  monthly_2g_7               int64
 187  monthly_2g_8               int64
 188  monthly_2g_9               int64
 189  sachet_2g_6                int64
 190  sachet_2g_7                int64
 191  sachet_2g_8                int64
 192  sachet_2g_9                int64
 193  monthly_3g_6               int64
 194  monthly_3g_7               int64
 195  monthly_3g_8               int64
 196  monthly_3g_9               int64
 197  sachet_3g_6                int64
 198  sachet_3g_7                int64
 199  sachet_3g_8                int64
 200  sachet_3g_9                int64
 201  fb_user_6                  float64
 202  fb_user_7                  float64
 203  fb_user_8                  float64
 204  fb_user_9                  float64
 205  aon                        int64
 206  aug_vbc_3g                 float64
 207  jul_vbc_3g                 float64
 208  jun_vbc_3g                 float64
 209  sep_vbc_3g                 float64
dtypes: datetime64[ns](8), float64(168), int64(34)
memory usage: 160.2 MB
```

Handling missing values

Handling missing values of meaningful attribute column

```
# 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(1
```

	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

	date_of_last_rech_data_6	total_rech_data_6	max_rech_data_6
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

- Let us conside the column `date_of_last_rech_data` indicating the date of the last recharge made in any given month for mobile internet. Here it can deduced if the `total_rech_data` and the `max_rech_data` also has missing values, the missing values in all the columns mentioned can be considered as meaningful missing.
- Hence imputing 0 as their values.
- Meaningfull missing in this case represents the the customer has not done any recharge for mobile interenet.

Handling the missing values for the attributes `total_rech_data_*`, `max_rech_data_*` and for month 6,7,8 and 9

```
# Code for conditional imputation
start_time=time.time()
for i in range(len(telecom_data)):
  # Handling 'total_rech_data', 'max_rech_data' and for month 6
    if pd.isnull((telecom_data['total_rech_data_6'][i]) and (telecom_data['max_rech_dat
        if pd.isnull(telecom_data['date_of_last_rech_data_6'][i]):
            telecom_data['total_rech_data_6'][i]=0
            telecom_data['max_rech_data_6'][i]=0

  # Handling 'total_rech_data', 'max_rech_data' and for month 7
    if pd.isnull((telecom_data['total_rech_data_7'][i]) and (telecom_data['max_rech_dat
        if pd.isnull(telecom_data['date_of_last_rech_data_7'][i]):
            telecom_data['total_rech_data_7'][i]=0
            telecom_data['max_rech_data_7'][i]=0

  # Handling 'total_rech_data', 'max_rech_data' and for month 8
    if pd.isnull((telecom_data['total_rech_data_8'][i]) and (telecom_data['max_rech_dat
        if pd.isnull(telecom_data['date_of_last_rech_data_8'][i]):
            telecom_data['total_rech_data_8'][i]=0
            telecom_data['max_rech_data_8'][i]=0

  # Handling 'total_rech_data', 'max_rech_data' and for month 9
    if pd.isnull((telecom_data['total_rech_data_9'][i]) and (telecom_data['max_rech_dat
        if pd.isnull(telecom_data['date_of_last_rech_data_9'][i]):
            telecom_data['total_rech_data_9'][i]=0
            telecom_data['max_rech_data_9'][i]=0
```

```
end_time = time.time()
print("\nExecution Time = ", round(end_time-start_time,2),"seconds")
print("The columns \n'total_rech_data_6','total_rech_data_7','total_rech_data_8','total
```

```
Execution Time =  382.04 seconds
The columns
'total_rech_data_6','total_rech_data_7','total_rech_data_8','total_rech_data_9'
'max_rech_data_6','max_rech_data_7','max_rech_data_8','max_rech_data_9' are imputed
with 0 based on the condition explained above
```

Handling the missing values for the attributes `count_rech_2g_*`,`count_rech_3g_*` for month 6,7,8 and 9

```
# Checking the related columns values
telecom_data[['count_rech_2g_6','count_rech_3g_6','total_rech_data_6']].head(10)
```

	count_rech_2g_6	count_rech_3g_6	total_rech_data_6
0	0.0	1.0	1.0
1	NaN	NaN	0.0
2	NaN	NaN	0.0
3	NaN	NaN	0.0
4	1.0	0.0	1.0
5	NaN	NaN	0.0
6	NaN	NaN	0.0
7	NaN	NaN	0.0
8	NaN	NaN	0.0
9	NaN	NaN	0.0

From the above tablular the column values of `total_rech_data` for each month from 6 to 9 respectively is the sum of the columns values of `count_rech_2g` for each month from 6 to 9 respectively and `count_rech_3g` for each month from 6 to 9 respectively, which derives to a multicollinearity issue. In order to reduce the multicollinearity, we can drop the columns `count_rech_2g` for each month from 6 to 9 respectively and `count_rech_3g` for each month from 6 to 9 respectively.

```
# Dropping the columns 'count_rech_2g_*' & 'count_rech_3g_*' for the months 6,7,8 and 9
telecom_data.drop(['count_rech_2g_6','count_rech_3g_6',
                   'count_rech_2g_7','count_rech_3g_7',
                   'count_rech_2g_8','count_rech_3g_8',
                   'count_rech_2g_9','count_rech_3g_9'],axis=1, inplace=True)

print("The 'count_rech_2g_6','count_rech_3g_6','count_rech_2g_7','count_rech_3g_7','cou
```

```
The
'count_rech_2g_6','count_rech_3g_6','count_rech_2g_7','count_rech_3g_7','count_rech_2g_8
columns are dropped as they can be explained from the 'total_rech_data'column
```

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

```
(99999, 202)
```

Handling the missing values for the attributes `arpu_3g_*`,`arpu_2g_*` for month 6,7,8 and 9

```
# Checking the related columns values
telecom_data[['arpu_3g_6','arpu_2g_6','av_rech_amt_data_6']].head(10)
```

	arpu_3g_6	arpu_2g_6	av_rech_amt_data_6
0	212.17	212.17	252.0
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	0.00	0.00	56.0
5	NaN	NaN	NaN
6	NaN	NaN	NaN
7	NaN	NaN	NaN
8	NaN	NaN	NaN
9	NaN	NaN	NaN

```
# Checking the correlation between the above mentioned columns in tabular for months 6,
print("Correlation table for month 6\n\n", telecom_data[['arpu_3g_6','arpu_2g_6','av_re
print("\nCorrelation table for month 7\n\n", telecom_data[['arpu_3g_7','arpu_2g_7','av_
print("\nCorrelation table for month 8\n\n", telecom_data[['arpu_3g_8','arpu_2g_8','av_
print("\nCorrelation table for month 9\n\n", telecom_data[['arpu_3g_9','arpu_2g_9','av_
```

```
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
```

From the above 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_*`.

```
# Dropping the columns 'arpu_3g_*'&'arpu_2g_*' in month 6,7,8 and 9 datafrom the datase
telecom_data.drop(['arpu_3g_6','arpu_2g_6',
                  'arpu_3g_7','arpu_2g_7',
                  'arpu_3g_8','arpu_2g_8',
                  'arpu_3g_9','arpu_2g_9'],axis=1, inplace=True)
print("\nThe columns'arpu_3g_6','arpu_2g_6','arpu_3g_7','arpu_2g_7','arpu_3g_8','arpu_2
```

```
The
columns'arpu_3g_6','arpu_2g_6','arpu_3g_7','arpu_2g_7','arpu_3g_8','arpu_2g_8','arpu_3g_
are dropped from the dataset due to high corellation between their respective arpu_* var
dataset
```

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

```
(99999, 194)
```

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_u
                  axis=1, inplace=True)
print("\nThe columns 'fb_user_6','fb_user_7','fb_user_8','fb_user_9','night_pck_user_6'
```

```
The columns
'fb_user_6','fb_user_7','fb_user_8','fb_user_9','night_pck_user_6','night_pck_user_7','r
are dropped from the dataset as it has no meaning to the data snd has high missing value
```

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

```
(99999, 186)
```

Handling the missing values for the attributes `av_rech_amt_data_*` for month 6,7,8 and 9

```
# 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

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.

```
# 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_
        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_
        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_
        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_
          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' anc
```

```
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 above
```

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

```
date_of_last_rech_data_6    74.85
date_of_last_rech_data_7    74.43
date_of_last_rech_data_9    74.08
date_of_last_rech_data_8    73.66
og_others_9                 7.75
loc_og_t2f_mou_9            7.75
loc_og_t2t_mou_9            7.75
loc_ic_t2f_mou_9            7.75
std_ic_mou_9                7.75
std_og_t2f_mou_9            7.75
loc_og_t2m_mou_9            7.75
loc_ic_mou_9                7.75
std_og_t2m_mou_9            7.75
std_ic_t2f_mou_9            7.75
std_ic_t2t_mou_9            7.75
loc_og_t2c_mou_9            7.75
std_ic_t2m_mou_9            7.75
std_og_t2t_mou_9            7.75
loc_og_mou_9                7.75
std_og_mou_9                7.75
spl_ic_mou_9                7.75
roam_og_mou_9               7.75
spl_og_mou_9                7.75
loc_ic_t2t_mou_9            7.75
isd_og_mou_9                7.75
roam_ic_mou_9               7.75
loc_ic_t2m_mou_9            7.75
isd_ic_mou_9                7.75
onnet_mou_9                 7.75
ic_others_9                 7.75
```

offnet_mou_9	7.75
og_others_8	5.38
std_ic_t2t_mou_8	5.38
std_og_t2m_mou_8	5.38
loc_ic_t2m_mou_8	5.38
spl_og_mou_8	5.38
loc_ic_t2f_mou_8	5.38
loc_ic_mou_8	5.38
std_og_t2f_mou_8	5.38
isd_og_mou_8	5.38
std_og_mou_8	5.38
std_og_t2t_mou_8	5.38
loc_ic_t2t_mou_8	5.38
std_ic_t2m_mou_8	5.38
loc_og_t2t_mou_8	5.38
onnet_mou_8	5.38
ic_others_8	5.38
offnet_mou_8	5.38
roam_ic_mou_8	5.38
isd_ic_mou_8	5.38
roam_og_mou_8	5.38
loc_og_mou_8	5.38
spl_ic_mou_8	5.38
loc_og_t2m_mou_8	5.38
std_ic_mou_8	5.38
loc_og_t2f_mou_8	5.38
loc_og_t2c_mou_8	5.38
std_ic_t2f_mou_8	5.38
date_of_last_rech_9	4.76
loc_ic_mou_6	3.94
spl_ic_mou_6	3.94
std_ic_mou_6	3.94
loc_ic_t2f_mou_6	3.94
isd_ic_mou_6	3.94
loc_ic_t2t_mou_6	3.94
ic_others_6	3.94
std_ic_t2t_mou_6	3.94
loc_ic_t2m_mou_6	3.94
std_ic_t2f_mou_6	3.94
std_ic_t2m_mou_6	3.94
loc_og_t2c_mou_6	3.94
spl_og_mou_6	3.94
std_og_t2t_mou_6	3.94
loc_og_t2f_mou_6	3.94
std_og_t2m_mou_6	3.94
onnet_mou_6	3.94
std_og_t2f_mou_6	3.94
loc_og_t2m_mou_6	3.94
std_og_mou_6	3.94

isd_og_mou_6	3.94
loc_og_t2t_mou_6	3.94
loc_og_mou_6	3.94
roam_og_mou_6	3.94
og_others_6	3.94
roam_ic_mou_6	3.94
offnet_mou_6	3.94
offnet_mou_7	3.86
loc_og_t2c_mou_7	3.86
onnet_mou_7	3.86
loc_og_t2f_mou_7	3.86
std_ic_mou_7	3.86
isd_ic_mou_7	3.86
loc_og_t2m_mou_7	3.86
roam_og_mou_7	3.86
loc_og_t2t_mou_7	3.86
roam_ic_mou_7	3.86
std_ic_t2f_mou_7	3.86
ic_others_7	3.86
spl_ic_mou_7	3.86
loc_og_mou_7	3.86
std_og_t2f_mou_7	3.86
loc_ic_t2t_mou_7	3.86
og_others_7	3.86
loc_ic_t2m_mou_7	3.86
spl_og_mou_7	3.86
loc_ic_t2f_mou_7	3.86
std_og_mou_7	3.86
loc_ic_mou_7	3.86
isd_og_mou_7	3.86
std_og_t2m_mou_7	3.86
std_ic_t2t_mou_7	3.86
std_og_t2t_mou_7	3.86
std_ic_t2m_mou_7	3.86
date_of_last_rech_8	3.62
date_of_last_rech_7	1.77
date_of_last_rech_6	1.61
jun_vbc_3g	0.00
vol_2g_mb_8	0.00
vol_3g_mb_6	0.00
av_rech_amt_data_6	0.00
vol_2g_mb_9	0.00
vol_2g_mb_7	0.00
vol_3g_mb_8	0.00
av_rech_amt_data_7	0.00
vol_2g_mb_6	0.00
av_rech_amt_data_8	0.00
av_rech_amt_data_9	0.00
aug_vbc_3g	0.00

jul_vbc_3g	0.00
vol_3g_mb_7	0.00
sachet_3g_9	0.00
vol_3g_mb_9	0.00
monthly_3g_6	0.00
sachet_3g_7	0.00
aon	0.00
max_rech_data_8	0.00
sachet_3g_6	0.00
monthly_3g_9	0.00
monthly_3g_8	0.00
monthly_3g_7	0.00
sachet_3g_8	0.00
monthly_2g_6	0.00
sachet_2g_9	0.00
sachet_2g_8	0.00
sachet_2g_7	0.00
sachet_2g_6	0.00
monthly_2g_9	0.00
monthly_2g_8	0.00
monthly_2g_7	0.00
max_rech_data_9	0.00
mobile_number	0.00
max_rech_data_7	0.00
arpu_6	0.00
total_rech_num_7	0.00
total_rech_num_6	0.00
total_ic_mou_9	0.00
total_ic_mou_8	0.00
total_ic_mou_7	0.00
total_ic_mou_6	0.00
total_og_mou_9	0.00
total_rech_num_9	0.00
total_og_mou_8	0.00
total_og_mou_7	0.00
total_og_mou_6	0.00
arpu_9	0.00
arpu_8	0.00
arpu_7	0.00
total_rech_num_8	0.00
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
```

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.

```
# Dropping the columns related to datetime dtype from the dataset
telecom_data.drop(["date_of_last_rech_data_6","date_of_last_rech_data_7",
                   "date_of_last_rech_data_8","date_of_last_rech_data_9"], axis=1, inpl
print("\nThe columns 'date_of_last_rech_data_6','date_of_last_rech_data_7','date_of_las
```

```
The columns
'date_of_last_rech_data_6','date_of_last_rech_data_7','date_of_last_rech_data_8','date_c
are dropped as it has no significance to the data
```

As we can no more utilise the datetime column, we can drop the `date_of_last_rech_data_*` column corresponding to months 6,7,8 and 9 respectively.

```
# Dropping the columns related to datetime dtype from the dataset
telecom_data.drop(["date_of_last_rech_6","date_of_last_rech_7",
                   "date_of_last_rech_8","date_of_last_rech_9"], axis=1, inplace=True)
print("\nThe columns 'date_of_last_rech_6','date_of_last_rech_7','date_of_last_rech_8',
```

```
The columns
'date_of_last_rech_6','date_of_last_rech_7','date_of_last_rech_8','date_of_last_rech_9'
```

are dropped as it has no significance to the data

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

(99999, 178)

Since the columns used to determine the High Value Customer is clear of null values, we can filter the overall data and then handle the remaining missing values for each column

Filtering the High Value Customer from Good Phase

```
# Filtering the data
 # 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

 # 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
telecom_data['total_rech_amt_data_7']=telecom_data['av_rech_amt_data_7'] * telecom_data

# 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_da
telecom_data['overall_rech_amt_7'] = telecom_data['total_rech_amt_data_7'] + telecom_da

# Calculating the average recharge done by customer in months June and July(i.e. 6th ar
telecom_data['avg_rech_amt_6_7'] = (telecom_data['overall_rech_amt_6'] + telecom_data['

# Finding the value of 70th percentage in the overall revenues defining the high value
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,"\r

# 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
```

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

(30001, 183)

The total number of customers is now limited to ~30k who lies under the High Value customer criteria basen upon which the model is built.

```
# Let us check the missing values percentages again for the HVC group
    # Checkng the overall missing values in the dataset
```

```
((telecom_data.isnull().sum()/telecom_data.shape[0])*100).round(2).sort_values(ascendir
```

```
loc_ic_t2f_mou_9        6.34
spl_og_mou_9            6.34
loc_og_t2m_mou_9        6.34
loc_og_t2f_mou_9        6.34
loc_ic_t2t_mou_9        6.34
isd_og_mou_9            6.34
loc_og_t2t_mou_9        6.34
loc_ic_t2m_mou_9        6.34
std_og_t2t_mou_9        6.34
roam_og_mou_9           6.34
std_og_mou_9            6.34
loc_ic_mou_9            6.34
std_ic_t2t_mou_9        6.34
roam_ic_mou_9           6.34
loc_og_t2c_mou_9        6.34
std_ic_t2m_mou_9        6.34
offnet_mou_9            6.34
std_ic_t2f_mou_9        6.34
std_og_t2f_mou_9        6.34
std_ic_mou_9            6.34
onnet_mou_9             6.34
spl_ic_mou_9            6.34
loc_og_mou_9            6.34
isd_ic_mou_9            6.34
std_og_t2m_mou_9        6.34
ic_others_9             6.34
og_others_9             6.34
std_og_mou_8            3.91
isd_og_mou_8            3.91
std_og_t2f_mou_8        3.91
std_ic_t2t_mou_8        3.91
og_others_8             3.91
loc_ic_t2t_mou_8        3.91
loc_ic_t2m_mou_8        3.91
loc_ic_t2f_mou_8        3.91
loc_ic_mou_8            3.91
std_ic_t2m_mou_8        3.91
std_ic_t2f_mou_8        3.91
std_ic_mou_8            3.91
spl_ic_mou_8            3.91
isd_ic_mou_8            3.91
ic_others_8             3.91
std_og_t2m_mou_8        3.91
spl_og_mou_8            3.91
std_og_t2t_mou_8        3.91
offnet_mou_8            3.91
loc_og_t2t_mou_8        3.91
loc_og_t2f_mou_8        3.91
```

roam_og_mou_8	3.91
roam_ic_mou_8	3.91
loc_og_t2c_mou_8	3.91
loc_og_t2m_mou_8	3.91
loc_og_mou_8	3.91
onnet_mou_8	3.91
offnet_mou_6	1.82
std_og_t2m_mou_6	1.82
loc_ic_t2m_mou_6	1.82
loc_og_t2m_mou_6	1.82
ic_others_6	1.82
loc_ic_t2f_mou_6	1.82
loc_og_t2t_mou_6	1.82
onnet_mou_6	1.82
std_ic_t2t_mou_6	1.82
isd_ic_mou_6	1.82
std_ic_mou_6	1.82
roam_og_mou_6	1.82
std_ic_t2m_mou_6	1.82
loc_ic_t2t_mou_6	1.82
spl_ic_mou_6	1.82
roam_ic_mou_6	1.82
std_ic_t2f_mou_6	1.82
loc_ic_mou_6	1.82
loc_og_mou_6	1.82
std_og_t2t_mou_6	1.82
loc_og_t2c_mou_6	1.82
std_og_t2f_mou_6	1.82
isd_og_mou_6	1.82
loc_og_t2f_mou_6	1.82
spl_og_mou_6	1.82
std_og_mou_6	1.82
og_others_6	1.82
std_ic_mou_7	1.79
roam_ic_mou_7	1.79
std_ic_t2f_mou_7	1.79
std_og_mou_7	1.79
offnet_mou_7	1.79
std_ic_t2m_mou_7	1.79
loc_og_mou_7	1.79
ic_others_7	1.79
std_og_t2m_mou_7	1.79
std_og_t2f_mou_7	1.79
spl_ic_mou_7	1.79
onnet_mou_7	1.79
isd_og_mou_7	1.79
loc_og_t2c_mou_7	1.79
std_og_t2t_mou_7	1.79
loc_og_t2t_mou_7	1.79

```
loc_ic_t2t_mou_7         1.79
loc_og_t2m_mou_7         1.79
loc_ic_t2m_mou_7         1.79
loc_og_t2f_mou_7         1.79
og_others_7              1.79
loc_ic_t2f_mou_7         1.79
isd_ic_mou_7             1.79
loc_ic_mou_7             1.79
spl_og_mou_7             1.79
roam_og_mou_7            1.79
std_ic_t2t_mou_7         1.79
monthly_2g_9             0.00
monthly_2g_8             0.00
vol_2g_mb_6              0.00
av_rech_amt_data_8       0.00
sachet_2g_6              0.00
sachet_2g_7              0.00
av_rech_amt_data_9       0.00
vol_3g_mb_6              0.00
monthly_2g_7             0.00
monthly_2g_6             0.00
vol_3g_mb_9              0.00
vol_3g_mb_8              0.00
vol_2g_mb_9              0.00
vol_3g_mb_7              0.00
sachet_2g_9              0.00
vol_2g_mb_7              0.00
vol_2g_mb_8              0.00
sachet_2g_8              0.00
jul_vbc_3g               0.00
monthly_3g_6             0.00
monthly_3g_7             0.00
overall_rech_amt_7       0.00
overall_rech_amt_6       0.00
total_rech_amt_data_7    0.00
total_rech_amt_data_6    0.00
sep_vbc_3g               0.00
jun_vbc_3g               0.00
av_rech_amt_data_6       0.00
aug_vbc_3g               0.00
aon                      0.00
sachet_3g_9              0.00
sachet_3g_8              0.00
sachet_3g_7              0.00
sachet_3g_6              0.00
monthly_3g_9             0.00
monthly_3g_8             0.00
av_rech_amt_data_7       0.00
mobile_number            0.00
```

```
max_rech_data_9          0.00
total_rech_amt_6         0.00
total_rech_num_8         0.00
total_rech_num_7         0.00
total_rech_num_6         0.00
total_ic_mou_9           0.00
total_ic_mou_8           0.00
total_ic_mou_7           0.00
total_ic_mou_6           0.00
arpu_6                   0.00
total_og_mou_9           0.00
total_og_mou_8           0.00
total_og_mou_7           0.00
total_og_mou_6           0.00
arpu_9                   0.00
arpu_8                   0.00
arpu_7                   0.00
total_rech_num_9         0.00
total_rech_amt_7         0.00
max_rech_data_8          0.00
total_rech_amt_8         0.00
max_rech_data_7          0.00
max_rech_data_6          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_7       0.00
last_day_rch_amt_6       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
avg_rech_amt_6_7         0.00
dtype: float64
```

*** The remaining attributes with missing value can be imputed using the advanced imputation technique like `KNNImputer` .***

```python
# Numerical columns available
num_col = telecom_data.select_dtypes(include = ['int64','float64']).columns.tolist()
```

```python
# Importing the libraries for Scaling and Imputation
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler
```

```
# Calling the Scaling function
scalar = MinMaxScaler()

# Scaling and transforming the data for the columns that are numerical
telecom_data[num_col]=scalar.fit_transform(telecom_data[num_col])

# Calling the KNN Imputer function
knn=KNNImputer(n_neighbors=3)

# Imputing the NaN values using KNN Imputer
start_time=time.time()

telecom_data_knn = pd.DataFrame(knn.fit_transform(telecom_data[num_col]))
telecom_data_knn.columns=telecom_data[num_col].columns

end_time=time.time()
print("\nExecution Time = ", round(end_time-start_time,2),"seconds\n")
```

```
Execution Time =  170.72 seconds
```

```
# check for any null values after imputation for numerical columns
telecom_data_knn.isnull().sum().sum()
```

```
0
```

The KNN Imputer has replaced all the null values in the numerical column usingK-means algorithm sucessfully

```
# Since we sclaed the numerical columns for the purpose of handling the null values,
    #we can restore the sclaed alues to its original form.

# Converting the scaled data back to the original data
telecom_data[num_col]=scalar.inverse_transform(telecom_data_knn)

# 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_
0	7.000843e+09	197.385	214.816	213.803	21.100	53.27	24.613333	0.00	33.5900(
7	7.000702e+09	1069.180	1349.850	3171.480	500.000	57.84	54.680000	52.29	65.2766(
8	7.001525e+09	378.721	492.223	137.362	166.787	413.69	351.030000	35.08	33.4600(
21	7.002124e+09	514.453	597.753	637.760	578.596	102.41	132.110000	85.14	161.6300(
23	7.000887e+09	74.350	193.897	366.966	811.480	48.96	50.660000	33.58	15.7400(
33	7.000150e+09	977.020	2362.833	409.230	799.356	0.00	0.000000	0.00	0.0000(
38	7.000815e+09	363.987	486.558	393.909	391.709	248.99	619.960000	666.38	494.7900(
41	7.000721e+09	482.832	425.764	229.769	143.596	86.39	118.880000	80.44	40.0600(
48	7.000294e+09	1873.271	575.927	179.218	1189.744	2061.69	881.430000	156.91	1589.2300(

	mobile_number	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_mou_8	onnet_mou_
53	7.002189e+09	978.077	1141.296	706.020	1076.247	135.14	119.590000	102.69	99.8300(

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

```
mobile_number         0.0
isd_ic_mou_8          0.0
ic_others_6           0.0
ic_others_7           0.0
ic_others_8           0.0
ic_others_9           0.0
total_rech_num_6      0.0
total_rech_num_7      0.0
total_rech_num_8      0.0
total_rech_num_9      0.0
total_rech_amt_6      0.0
total_rech_amt_7      0.0
total_rech_amt_8      0.0
total_rech_amt_9      0.0
max_rech_amt_6        0.0
max_rech_amt_7        0.0
max_rech_amt_8        0.0
max_rech_amt_9        0.0
last_day_rch_amt_6    0.0
last_day_rch_amt_7    0.0
last_day_rch_amt_8    0.0
isd_ic_mou_9          0.0
isd_ic_mou_7          0.0
total_rech_data_6     0.0
isd_ic_mou_6          0.0
std_ic_t2m_mou_7      0.0
std_ic_t2m_mou_8      0.0
std_ic_t2m_mou_9      0.0
std_ic_t2f_mou_6      0.0
std_ic_t2f_mou_7      0.0
std_ic_t2f_mou_8      0.0
std_ic_t2f_mou_9      0.0
std_ic_mou_6          0.0
std_ic_mou_7          0.0
std_ic_mou_8          0.0
std_ic_mou_9          0.0
total_ic_mou_6        0.0
total_ic_mou_7        0.0
total_ic_mou_8        0.0
total_ic_mou_9        0.0
spl_ic_mou_6          0.0
spl_ic_mou_7          0.0
spl_ic_mou_8          0.0
```

```
spl_ic_mou_9             0.0
last_day_rch_amt_9       0.0
total_rech_data_7        0.0
std_ic_t2t_mou_9         0.0
sachet_2g_6              0.0
sachet_2g_8              0.0
sachet_2g_9              0.0
monthly_3g_6             0.0
monthly_3g_7             0.0
monthly_3g_8             0.0
monthly_3g_9             0.0
sachet_3g_6              0.0
sachet_3g_7              0.0
sachet_3g_8              0.0
sachet_3g_9              0.0
aon                      0.0
aug_vbc_3g               0.0
jul_vbc_3g               0.0
jun_vbc_3g               0.0
sep_vbc_3g               0.0
total_rech_amt_data_6    0.0
total_rech_amt_data_7    0.0
overall_rech_amt_6       0.0
overall_rech_amt_7       0.0
sachet_2g_7              0.0
monthly_2g_9             0.0
total_rech_data_8        0.0
monthly_2g_8             0.0
total_rech_data_9        0.0
max_rech_data_6          0.0
max_rech_data_7          0.0
max_rech_data_8          0.0
max_rech_data_9          0.0
av_rech_amt_data_6       0.0
av_rech_amt_data_7       0.0
av_rech_amt_data_8       0.0
av_rech_amt_data_9       0.0
vol_2g_mb_6              0.0
vol_2g_mb_7              0.0
vol_2g_mb_8              0.0
vol_2g_mb_9              0.0
vol_3g_mb_6              0.0
vol_3g_mb_7              0.0
vol_3g_mb_8              0.0
vol_3g_mb_9              0.0
monthly_2g_6             0.0
monthly_2g_7             0.0
std_ic_t2m_mou_6         0.0
std_ic_t2t_mou_8         0.0
```

```
arpu_6               0.0
loc_og_t2t_mou_8     0.0
loc_og_t2m_mou_6     0.0
loc_og_t2m_mou_7     0.0
loc_og_t2m_mou_8     0.0
loc_og_t2m_mou_9     0.0
loc_og_t2f_mou_6     0.0
loc_og_t2f_mou_7     0.0
loc_og_t2f_mou_8     0.0
loc_og_t2f_mou_9     0.0
loc_og_t2c_mou_6     0.0
loc_og_t2c_mou_7     0.0
loc_og_t2c_mou_8     0.0
loc_og_t2c_mou_9     0.0
loc_og_mou_6         0.0
loc_og_mou_7         0.0
loc_og_mou_8         0.0
loc_og_mou_9         0.0
std_og_t2t_mou_6     0.0
std_og_t2t_mou_7     0.0
std_og_t2t_mou_8     0.0
loc_og_t2t_mou_9     0.0
loc_og_t2t_mou_7     0.0
std_og_t2m_mou_6     0.0
loc_og_t2t_mou_6     0.0
arpu_7               0.0
arpu_8               0.0
arpu_9               0.0
onnet_mou_6          0.0
onnet_mou_7          0.0
onnet_mou_8          0.0
onnet_mou_9          0.0
offnet_mou_6         0.0
offnet_mou_7         0.0
offnet_mou_8         0.0
offnet_mou_9         0.0
roam_ic_mou_6        0.0
roam_ic_mou_7        0.0
roam_ic_mou_8        0.0
roam_ic_mou_9        0.0
roam_og_mou_6        0.0
roam_og_mou_7        0.0
roam_og_mou_8        0.0
roam_og_mou_9        0.0
std_og_t2t_mou_9     0.0
std_og_t2m_mou_7     0.0
std_ic_t2t_mou_7     0.0
total_og_mou_6       0.0
total_og_mou_8       0.0
```

```
total_og_mou_9          0.0
loc_ic_t2t_mou_6        0.0
loc_ic_t2t_mou_7        0.0
loc_ic_t2t_mou_8        0.0
loc_ic_t2t_mou_9        0.0
loc_ic_t2m_mou_6        0.0
loc_ic_t2m_mou_7        0.0
loc_ic_t2m_mou_8        0.0
loc_ic_t2m_mou_9        0.0
loc_ic_t2f_mou_6        0.0
loc_ic_t2f_mou_7        0.0
loc_ic_t2f_mou_8        0.0
loc_ic_t2f_mou_9        0.0
loc_ic_mou_6            0.0
loc_ic_mou_7            0.0
loc_ic_mou_8            0.0
loc_ic_mou_9            0.0
std_ic_t2t_mou_6        0.0
total_og_mou_7          0.0
og_others_9             0.0
std_og_t2m_mou_8        0.0
og_others_8             0.0
std_og_t2m_mou_9        0.0
std_og_t2f_mou_6        0.0
std_og_t2f_mou_7        0.0
std_og_t2f_mou_8        0.0
std_og_t2f_mou_9        0.0
std_og_mou_6            0.0
std_og_mou_7            0.0
std_og_mou_8            0.0
std_og_mou_9            0.0
isd_og_mou_6            0.0
isd_og_mou_7            0.0
isd_og_mou_8            0.0
isd_og_mou_9            0.0
spl_og_mou_6            0.0
spl_og_mou_7            0.0
spl_og_mou_8            0.0
spl_og_mou_9            0.0
og_others_6             0.0
og_others_7             0.0
avg_rech_amt_6_7        0.0
dtype: float64
```

```python
# Reconfirming for missing values if any
telecom_data.isnull().sum().sum()
```

```
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

```
# Selecting the columns to define churn variable (i.e. TARGET Variable)
churn_col=['total_ic_mou_9','total_og_mou_9','vol_2g_mb_9','vol_3g_mb_9']
telecom_data[churn_col].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30001 entries, 0 to 99997
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   total_ic_mou_9  30001 non-null  float64
 1   total_og_mou_9  30001 non-null  float64
 2   vol_2g_mb_9     30001 non-null  float64
 3   vol_3g_mb_9     30001 non-null  float64
dtypes: float64(4)
memory usage: 1.1 MB
```

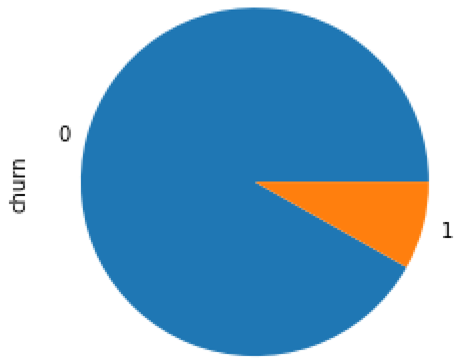
```
# Initializing the churn variable.
telecom_data['churn']=0

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

```
# 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_
0	7.000843e+09	197.385	214.816	213.803	21.100	53.27	24.613333	0.00	33.5900(
7	7.000702e+09	1069.180	1349.850	3171.480	500.000	57.84	54.680000	52.29	65.2766(
8	7.001525e+09	378.721	492.223	137.362	166.787	413.69	351.030000	35.08	33.4600(
21	7.002124e+09	514.453	597.753	637.760	578.596	102.41	132.110000	85.14	161.6300(
23	7.000887e+09	74.350	193.897	366.966	811.480	48.96	50.660000	33.58	15.7400(
33	7.000150e+09	977.020	2362.833	409.230	799.356	0.00	0.000000	0.00	0.0000(
38	7.000815e+09	363.987	486.558	393.909	391.709	248.99	619.960000	666.38	494.7900(
41	7.000721e+09	482.832	425.764	229.769	143.596	86.39	118.880000	80.44	40.0600(
48	7.000294e+09	1873.271	575.927	179.218	1189.744	2061.69	881.430000	156.91	1589.2300(
53	7.002189e+09	978.077	1141.296	706.020	1076.247	135.14	119.590000	102.69	99.8300(

```
# 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
```



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.

```
# Selecting all the churn phase columns in order to drop then

churn_phase_cols = [col for col in telecom_data.columns if '_9' in col]
print("The columns from churn phase are:\n",churn_phase_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_t2f_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', 'last_day_rch_amt_9', 'total_rech_data_9',
'max_rech_data_9', 'av_rech_amt_data_9', 'vol_2g_mb_9', 'vol_3g_mb_9', 'monthly_2g_9',
'sachet_2g_9', 'monthly_3g_9', 'sachet_3g_9']
```

```
# Dropping the selected churn phase columns
telecom_data.drop(churn_phase_cols, axis=1, inplace=True)
```

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

```
(30001, 141)
```

We can still clean the data by few possible columns relating to the good phase.

As we derived few columns in the good phase earlier, we can drop those related columns during creation.

```
# telecom_data['total_rech_amt_data_6']=telecom_data['av_rech_amt_data_6'] * telecom_da
# telecom_data['total_rech_amt_data_7']=telecom_data['av_rech_amt_data_7'] * telecom_da

# # 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_
# telecom_data['overall_rech_amt_7'] = telecom_data['total_rech_amt_data_7'] + telecom_

telecom_data.drop(['total_rech_amt_data_6','av_rech_amt_data_6',
                   'total_rech_data_6','total_rech_amt_6',
                  'total_rech_amt_data_7','av_rech_amt_data_7',
                   'total_rech_data_7','total_rech_amt_7'], axis=1, inplace=True)
```

We can also create new columns for the defining the good phase variables and drop the seperate 6th and 7 month variables.

Before proceding to check the remaining missing value handling, let us check the collineartity of the indepedent variables and try to understand their dependencies.

```
# 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 correla
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
```

```
total_rech_amt_8    arpu_8               0.955351
isd_og_mou_8        isd_og_mou_7         0.943433
                    isd_og_mou_6         0.919641
isd_og_mou_7        isd_og_mou_6         0.916237
sachet_2g_8         total_rech_data_8    0.900629
total_ic_mou_6      loc_ic_mou_6         0.895099
total_ic_mou_8      loc_ic_mou_8         0.893072
total_ic_mou_7      loc_ic_mou_7         0.883070
std_og_t2t_mou_8    onnet_mou_8          0.860483
std_og_t2t_mou_7    onnet_mou_7          0.860275
std_og_t2t_mou_6    onnet_mou_6          0.859593
```

```
avg_rech_amt_6_7    overall_rech_amt_7    0.856275
std_og_t2m_mou_7    offnet_mou_7          0.854685
std_og_t2m_mou_8    offnet_mou_8          0.851049
total_og_mou_8      std_og_mou_8          0.848858
total_og_mou_7      std_og_mou_7          0.848825
loc_ic_mou_8        loc_ic_t2m_mou_8      0.847512
std_ic_mou_8        std_ic_t2m_mou_8      0.845590
loc_ic_mou_6        loc_ic_t2m_mou_6      0.844418
loc_og_mou_8        loc_og_mou_7          0.844245
loc_ic_mou_8        loc_ic_mou_7          0.842908
avg_rech_amt_6_7    overall_rech_amt_6    0.842748
loc_og_t2t_mou_8    loc_og_t2t_mou_7      0.834612
loc_ic_mou_7        loc_ic_t2m_mou_7      0.834557
total_og_mou_6      std_og_mou_6          0.831720
std_og_t2m_mou_6    offnet_mou_6          0.830433
loc_og_t2m_mou_8    loc_og_t2m_mou_7      0.826720
loc_ic_mou_7        loc_ic_mou_6          0.821979
total_ic_mou_8      total_ic_mou_7        0.820529
std_ic_mou_7        std_ic_t2m_mou_7      0.819316
loc_ic_t2m_mou_8    loc_ic_t2m_mou_7      0.814748
std_ic_mou_6        std_ic_t2m_mou_6      0.814081
loc_og_t2f_mou_7    loc_og_t2f_mou_6      0.809471
onnet_mou_8         onnet_mou_7           0.808507
loc_ic_t2t_mou_8    loc_ic_t2t_mou_7      0.808102
loc_og_mou_7        loc_og_mou_6          0.807980
std_og_t2t_mou_8    std_og_t2t_mou_7      0.804607
loc_og_mou_6        loc_og_t2m_mou_6      0.803954
loc_ic_t2t_mou_7    loc_ic_t2t_mou_6      0.803421
total_ic_mou_7      total_ic_mou_6        0.803042
av_rech_amt_data_8  max_rech_data_8       0.801613
dtype: float64
```

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

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

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

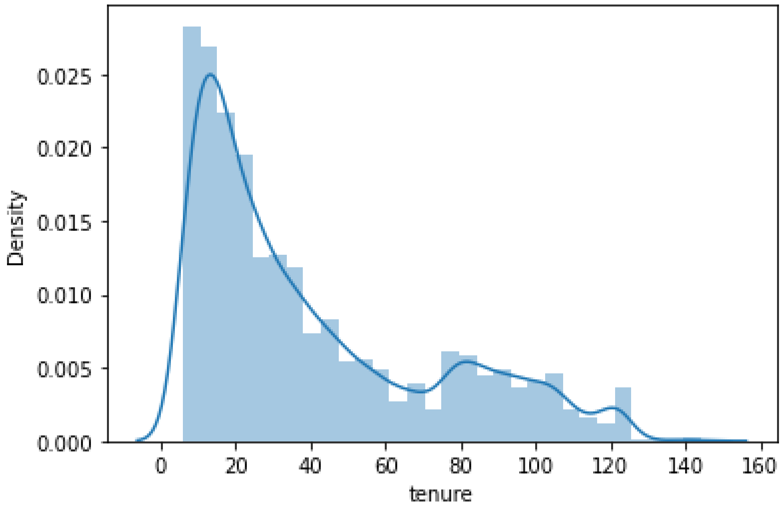
```
# The curent dimension of the dataset after dropping few unwanted columns
telecom_data.shape
```

```
(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)
```

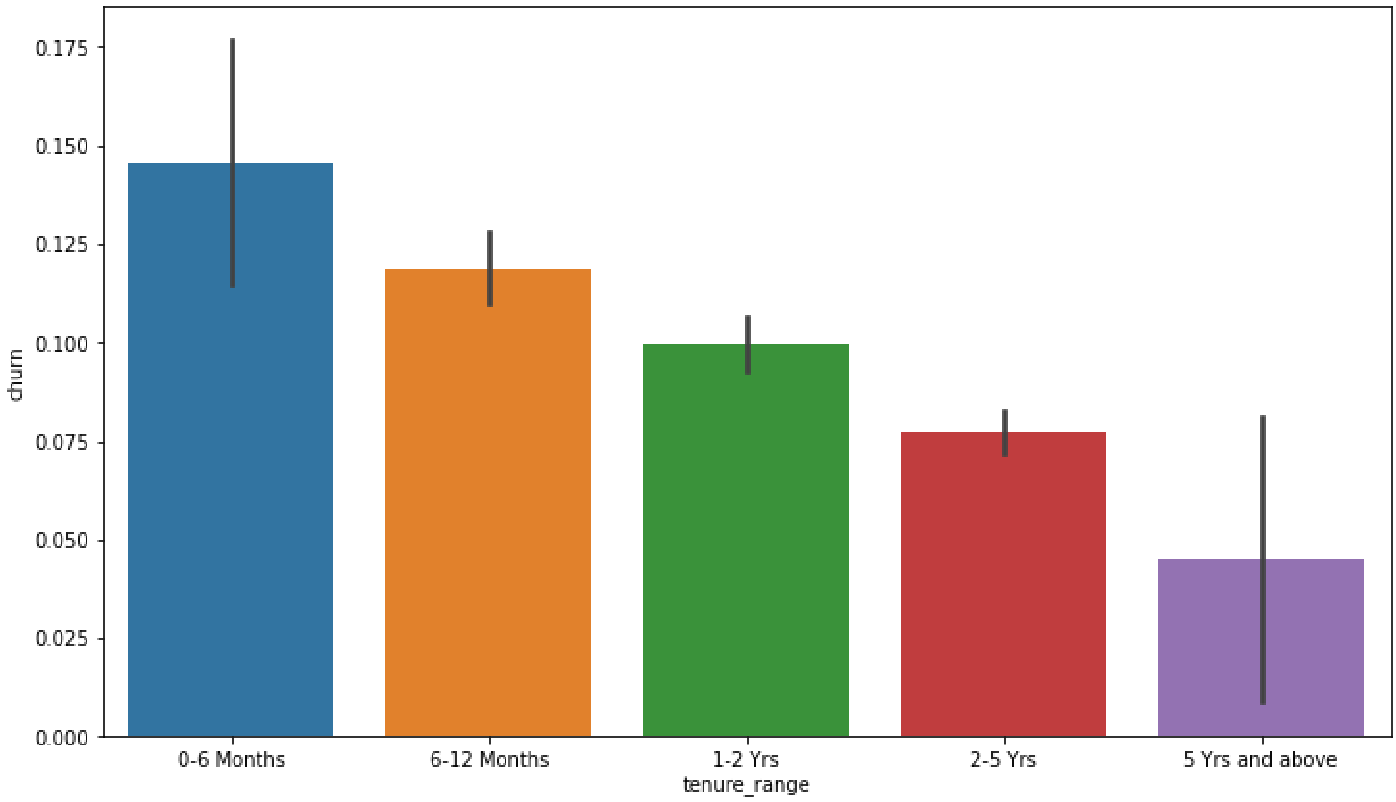
```
# Checking the distribution of he tenure variable

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




```
tn_range = [0, 6, 12, 24, 60, 61]
tn_label = [ '0-6 Months', '6-12 Months', '1-2 Yrs', '2-5 Yrs', '5 Yrs and above']
telecom_data['tenure_range'] = pd.cut(telecom_data['tenure'], tn_range, labels=tn_label
telecom_data['tenure_range'].head()
```

```
0     2-5 Yrs
7     2-5 Yrs
8     6-12 Months
21    1-2 Yrs
23    1-2 Yrs
Name: tenure_range, dtype: category
Categories (5, object): ['0-6 Months' < '6-12 Months' < '1-2 Yrs' < '2-5 Yrs' < '5 Yrs
and above']
```

```
# Plotting a bar plot for tenure range
plt.figure(figsize=[12,7])
sns.barplot(x='tenure_range',y='churn', data=telecom_data)
plt.show()
```



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.

The average revenue per user 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.

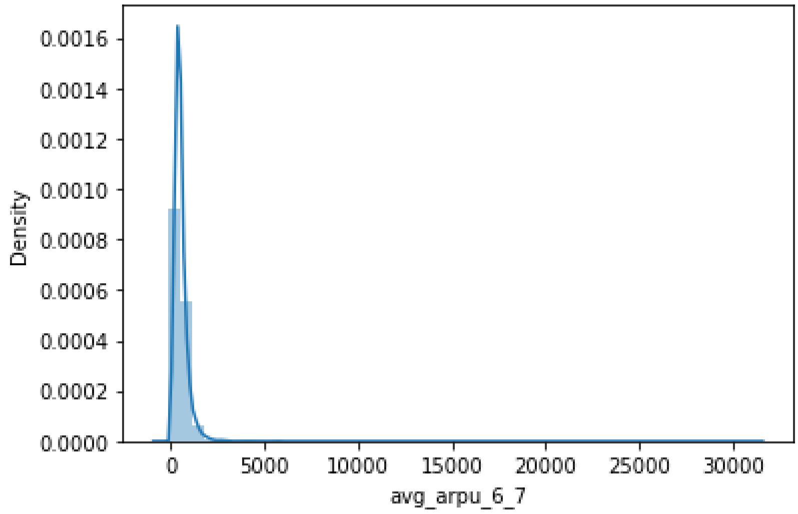
```
telecom_data["avg_arpu_6_7"]= (telecom_data['arpu_6']+telecom_data['arpu_7'])/2
telecom_data['avg_arpu_6_7'].head()
```

```
0      206.1005
7      1209.5150
8      435.4720
21     556.1030
23     134.1235
Name: avg_arpu_6_7, dtype: float64
```

```
# Lets drop the original columns as they are derived to a new column for better underst

telecom_data.drop(['arpu_6','arpu_7'], axis=1, inplace=True)


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

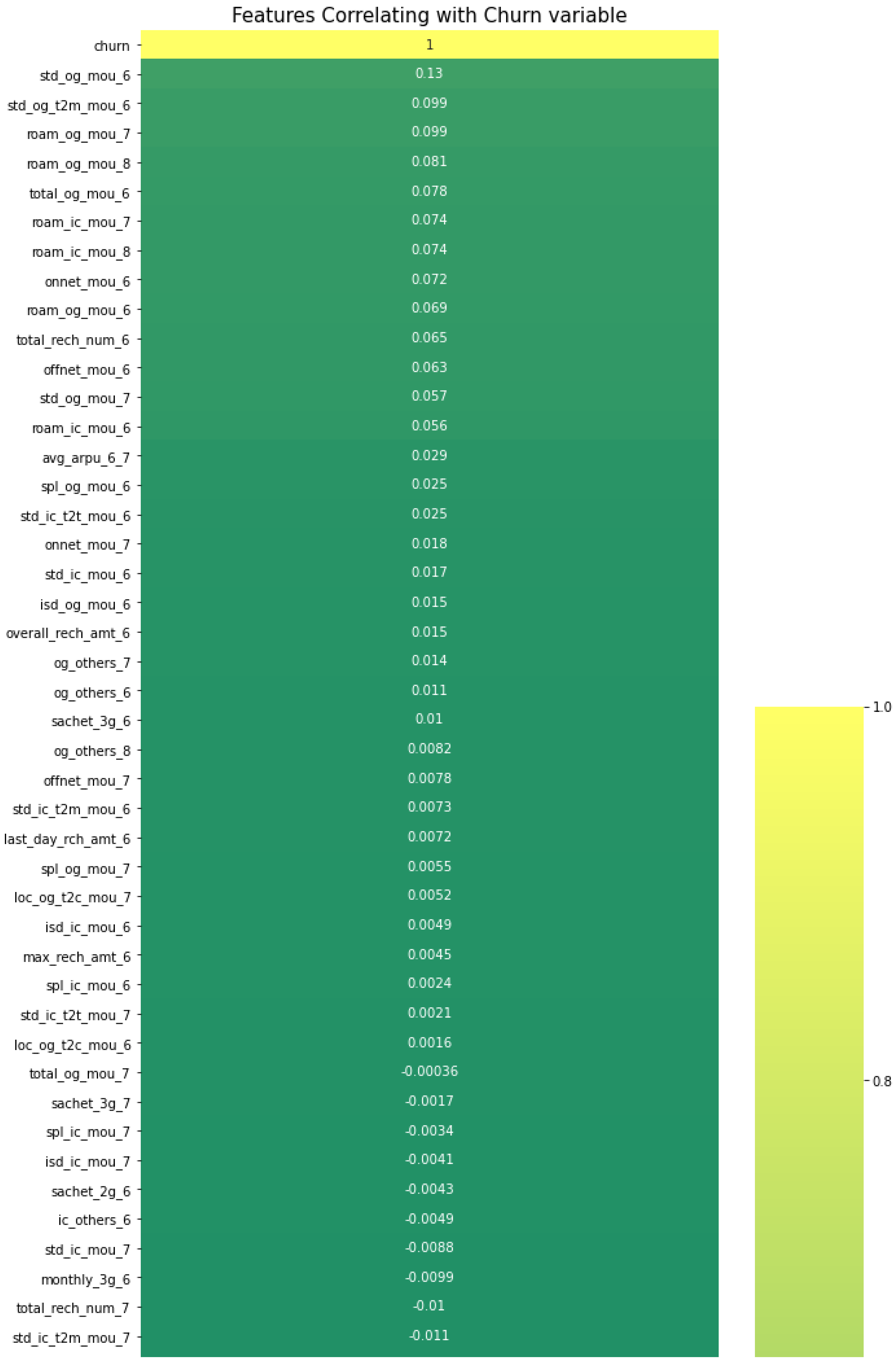
```
(30001, 121)
```

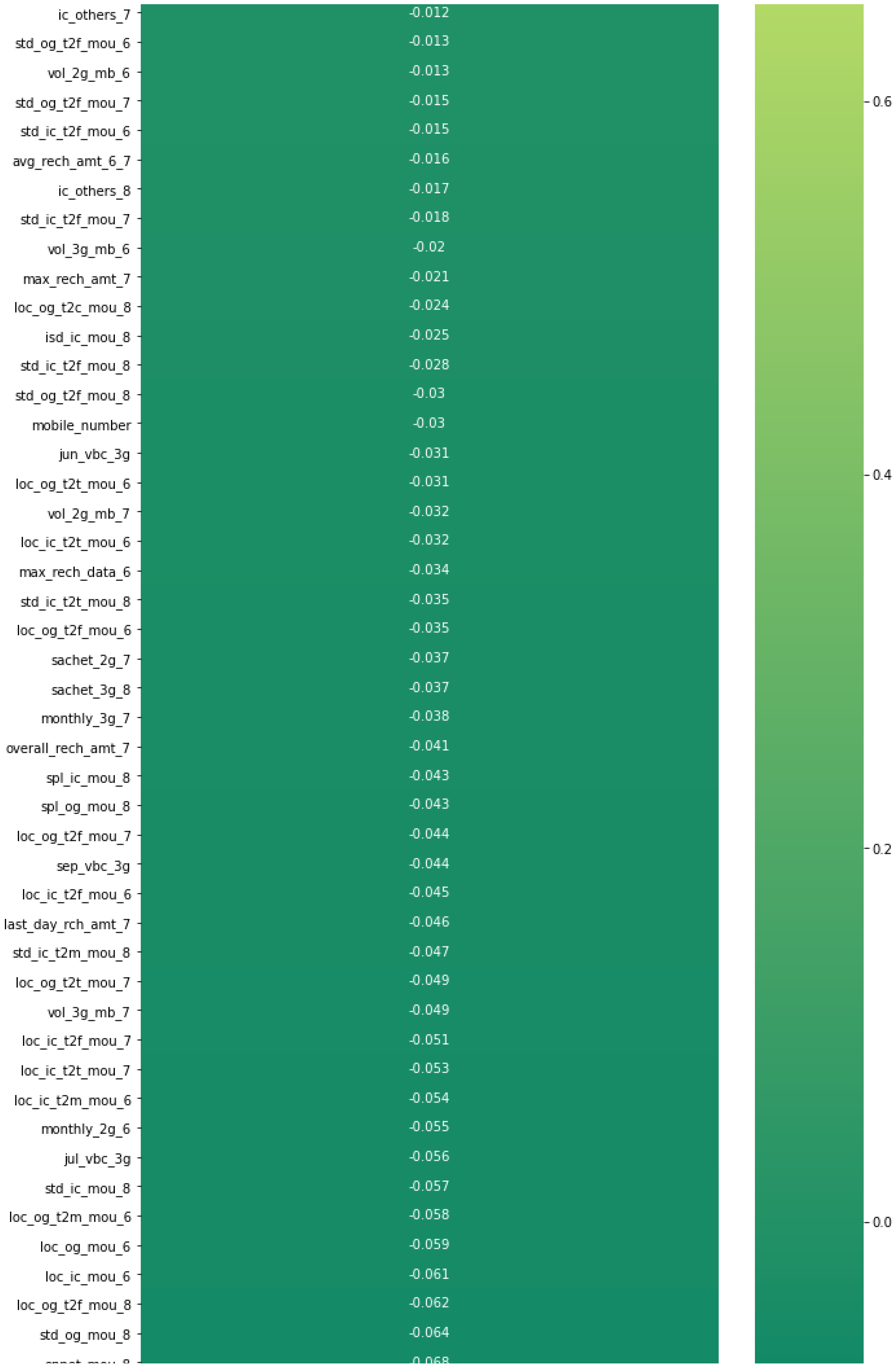
```
# Visualizing the column created
sns.distplot(telecom_data['avg_arpu_6_7'])
plt.show()
```

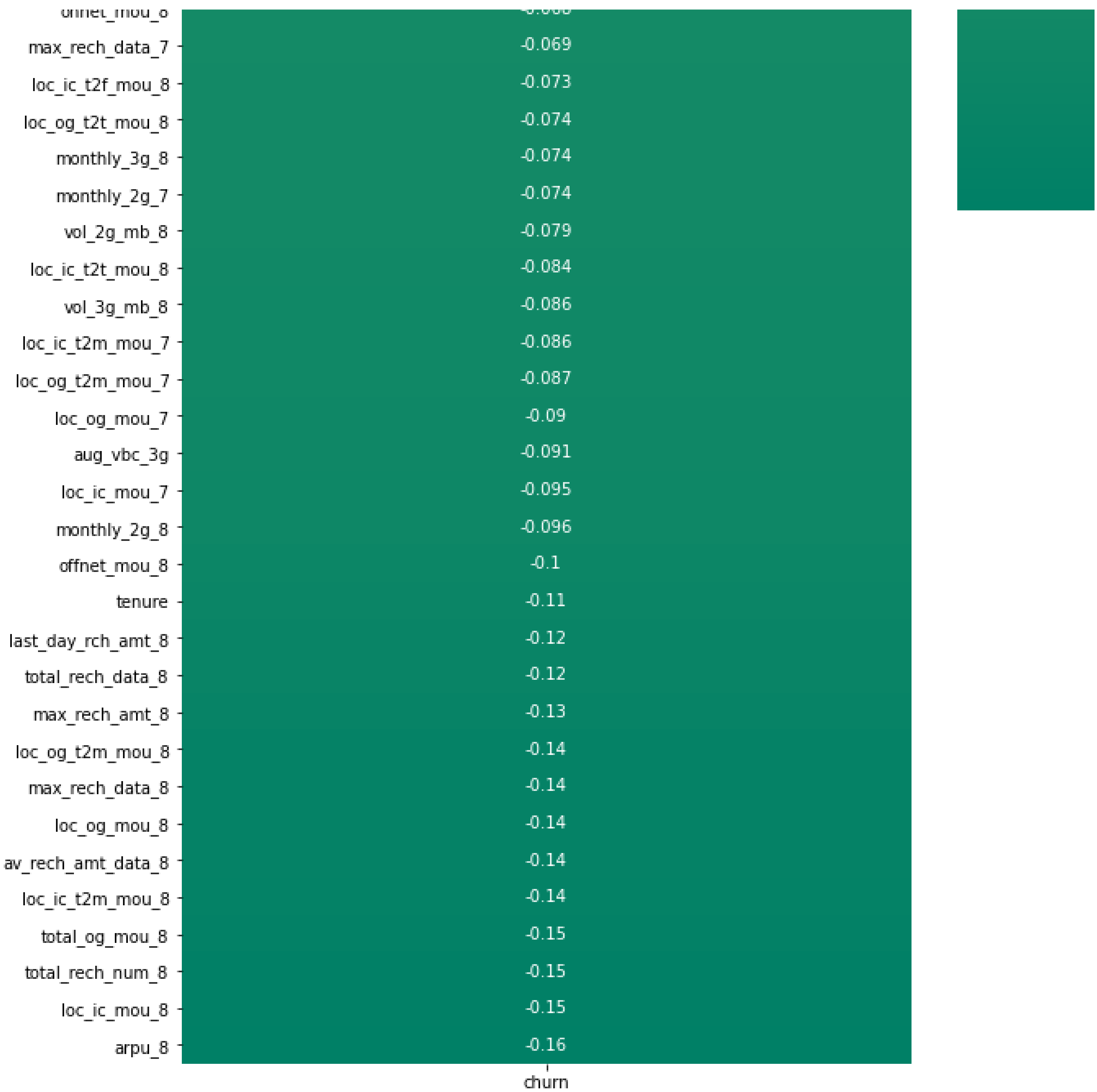



```
# Checking Correlation between target variable(SalePrice) with the other variable in th
plt.figure(figsize=(10,50))
heatmap_churn = sns.heatmap(telecom_data.corr()[['churn']].sort_values(ascending=False,
                                cmap='summer')
heatmap_churn.set_title("Features Correlating with Churn variable", fontsize=15)
```

```
Text(0.5, 1.0, 'Features Correlating with Churn variable')
```

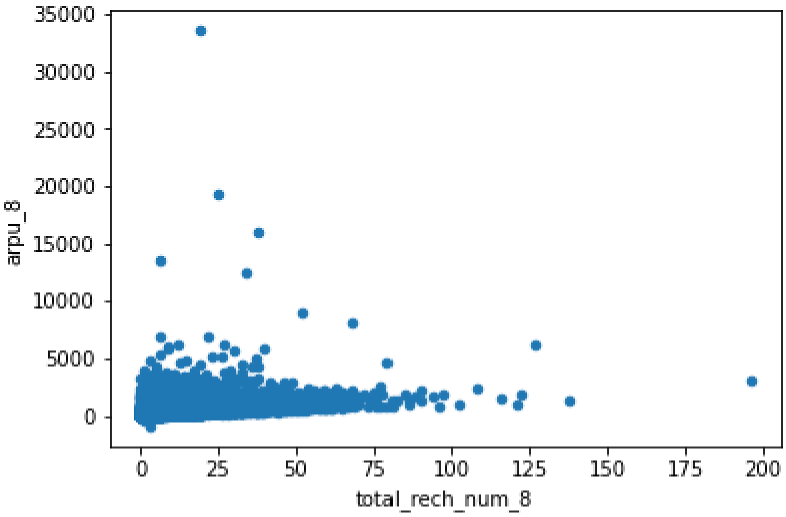

Features Correlating with Churn variable
churn 1
std_og_mou_6 0.13
std_og_t2m_mou_6 0.099
roam_og_mou_7 0.099
roam_og_mou_8 0.081
total_og_mou_6 0.078
roam_ic_mou_7 0.074
roam_ic_mou_8 0.074
onnet_mou_6 0.072
roam_og_mou_6 0.069
total_rech_num_6 0.065
offnet_mou_6 0.063
std_og_mou_7 0.057
roam_ic_mou_6 0.056
avg_arpu_6_7 0.029
spl_og_mou_6 0.025
std_ic_t2t_mou_6 0.025
onnet_mou_7 0.018
std_ic_mou_6 0.017
isd_og_mou_6 0.015
overall_rech_amt_6 0.015
og_others_7 0.014
og_others_6 0.011
sachet_3g_6 0.01
og_others_8 0.0082
offnet_mou_7 0.0078
std_ic_t2m_mou_6 0.0073
last_day_rch_amt_6 0.0072
spl_og_mou_7 0.0055
loc_og_t2c_mou_7 0.0052
isd_ic_mou_6 0.0049
max_rech_amt_6 0.0045
spl_ic_mou_6 0.0024
std_ic_t2t_mou_7 0.0021
loc_og_t2c_mou_6 0.0016
total_og_mou_7 -0.00036
sachet_3g_7 -0.0017
spl_ic_mou_7 -0.0034
isd_ic_mou_7 -0.0041
sachet_2g_6 -0.0043
ic_others_6 -0.0049
std_ic_mou_7 -0.0088
monthly_3g_6 -0.0099
total_rech_num_7 -0.01
std_ic_t2m_mou_7 -0.011
1.0
0.8


ic_others_7 -0.012
std_og_t2f_mou_6 -0.013
vol_2g_mb_6 -0.013
std_og_t2f_mou_7 -0.015
std_ic_t2f_mou_6 -0.015
avg_rech_amt_6_7 -0.016
ic_others_8 -0.017
std_ic_t2f_mou_7 -0.018
vol_3g_mb_6 -0.02
max_rech_amt_7 -0.021
loc_og_t2c_mou_8 -0.024
isd_ic_mou_8 -0.025
std_ic_t2f_mou_8 -0.028
std_og_t2f_mou_8 -0.03
mobile_number -0.03
jun_vbc_3g -0.031
loc_og_t2t_mou_6 -0.031
vol_2g_mb_7 -0.032
loc_ic_t2t_mou_6 -0.032
max_rech_data_6 -0.034
std_ic_t2t_mou_8 -0.035
loc_og_t2f_mou_6 -0.035
sachet_2g_7 -0.037
sachet_3g_8 -0.037
monthly_3g_7 -0.038
overall_rech_amt_7 -0.041
spl_ic_mou_8 -0.043
spl_og_mou_8 -0.043
loc_og_t2f_mou_7 -0.044
sep_vbc_3g -0.044
loc_ic_t2f_mou_6 -0.045
last_day_rch_amt_7 -0.046
std_ic_t2m_mou_8 -0.047
loc_og_t2t_mou_7 -0.049
vol_3g_mb_7 -0.049
loc_ic_t2f_mou_7 -0.051
loc_ic_t2t_mou_7 -0.053
loc_ic_t2m_mou_6 -0.054
monthly_2g_6 -0.055
jul_vbc_3g -0.056
std_ic_mou_8 -0.057
loc_og_t2m_mou_6 -0.058
loc_og_mou_6 -0.059
loc_ic_mou_6 -0.061
loc_og_t2f_mou_8 -0.062
std_og_mou_8 -0.064
0.6
0.4
0.2
0.0

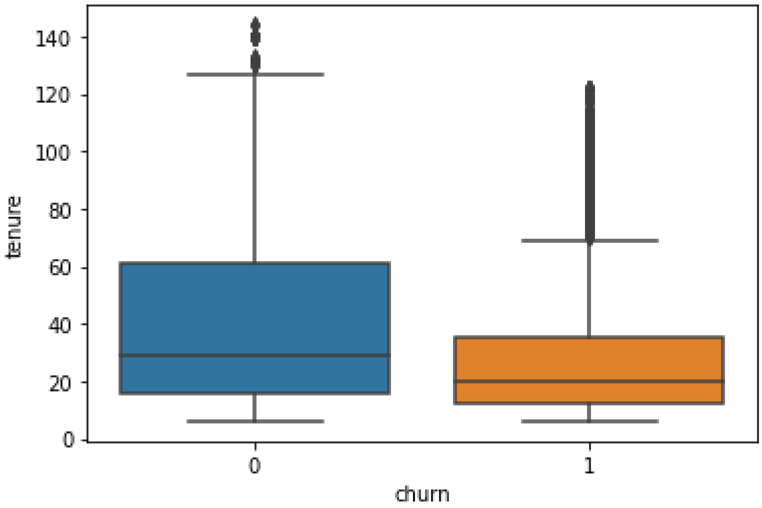


- Avg Outgoing Calls & calls on romaning for 6 & 7th months are positively correlated with churn.
- Avg Revenue, No. Of Recharge for 8th month has negative correlation with churn.

```
# lets now draw a scatter plot between total recharge and avg revenue for the 8th month
telecom_data[['total_rech_num_8', 'arpu_8']].plot.scatter(x = 'total_rech_num_8',
                                                          y='arpu_8')
plt.show()
```

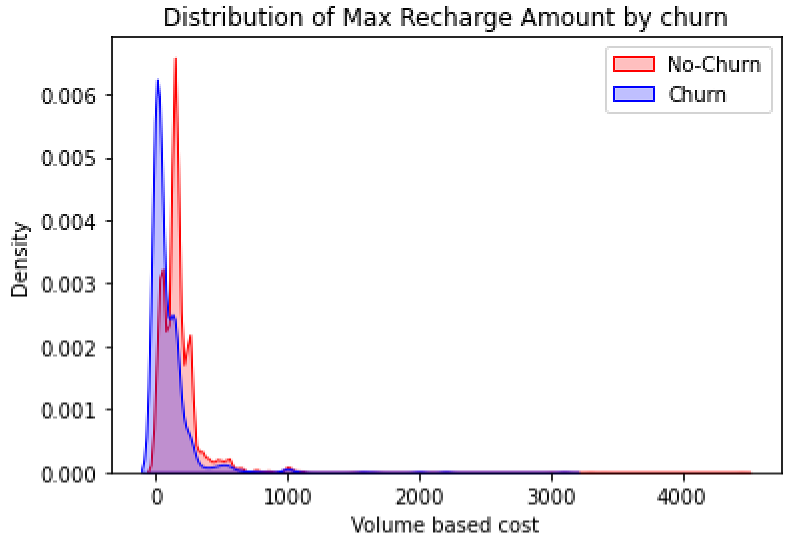



```
sns.boxplot(x = telecom_data.churn, y = telecom_data.tenure)

plt.show()
```

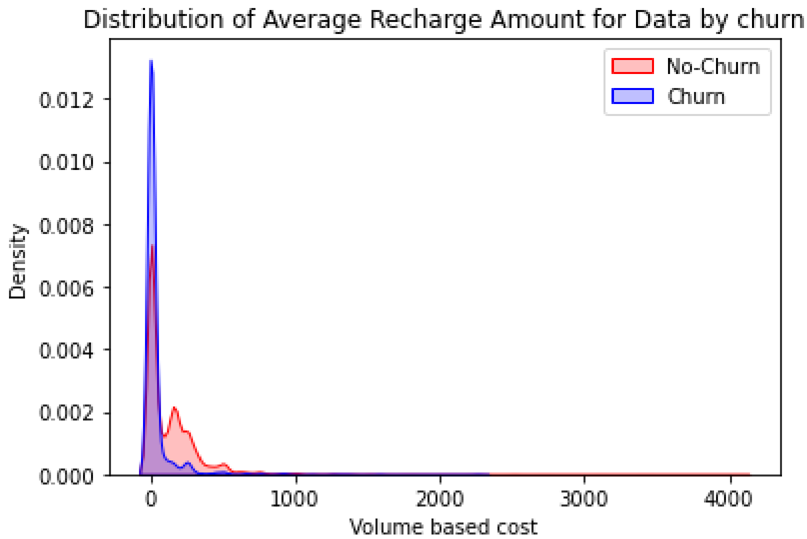


From the above plot , its clear tenured customers do no churn and they keep availing telecom services

```
# Plot between churn vs max recharе amount
ax = sns.kdeplot(telecom_data.max_rech_amt_8[(telecom_data["churn"] == 0)],
                color="Red", shade = True)
ax = sns.kdeplot(telecom_data.max_rech_amt_8[(telecom_data["churn"] == 1)],
                ax =ax, color="Blue", shade= True)
ax.legend(["No-Churn","Churn"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Volume based cost')
ax.set_title('Distribution of Max Recharge Amount by churn')
plt.show()
```




```
# churn vs max recharge amount
ax = sns.kdeplot(telecom_data.av_rech_amt_data_8[(telecom_data["churn"] == 0)],
                color="Red", shade = True)
ax = sns.kdeplot(telecom_data.av_rech_amt_data_8[(telecom_data["churn"] == 1)],
                ax =ax, color="Blue", shade= True)
ax.legend(["No-Churn","Churn"],loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Volume based cost')
ax.set_title('Distribution of Average Recharge Amount for Data by churn')
plt.show()
```

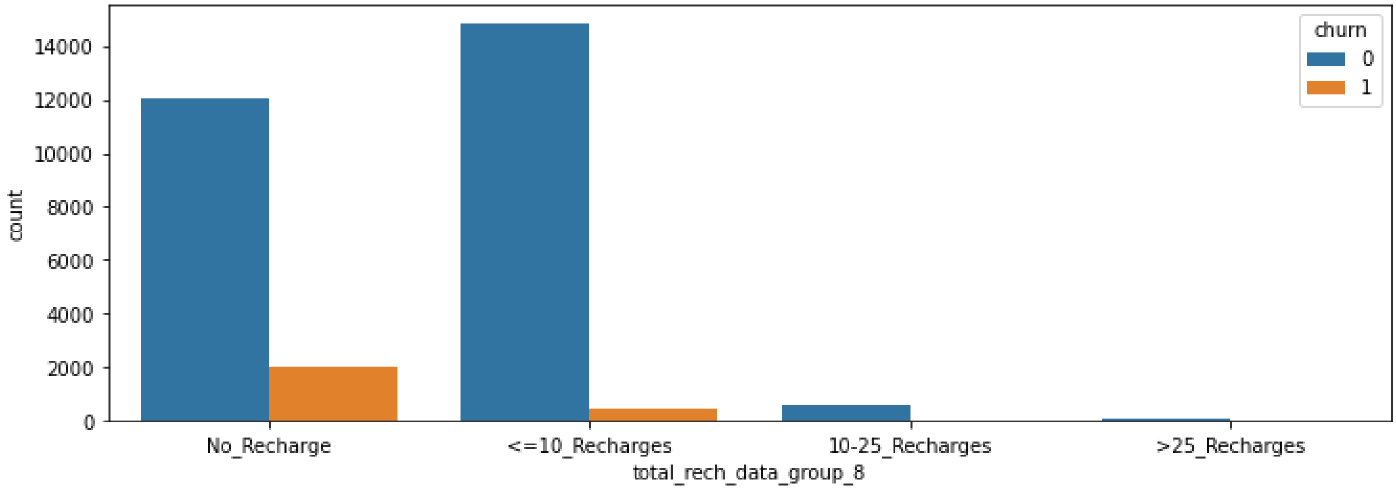



```
# Creating categories for month 8 column totalrecharge and their count
telecom_data['total_rech_data_group_8']=pd.cut(telecom_data['total_rech_data_8'],[-1,0,
telecom_data['total_rech_num_group_8']=pd.cut(telecom_data['total_rech_num_8'],[-1,0,10
```

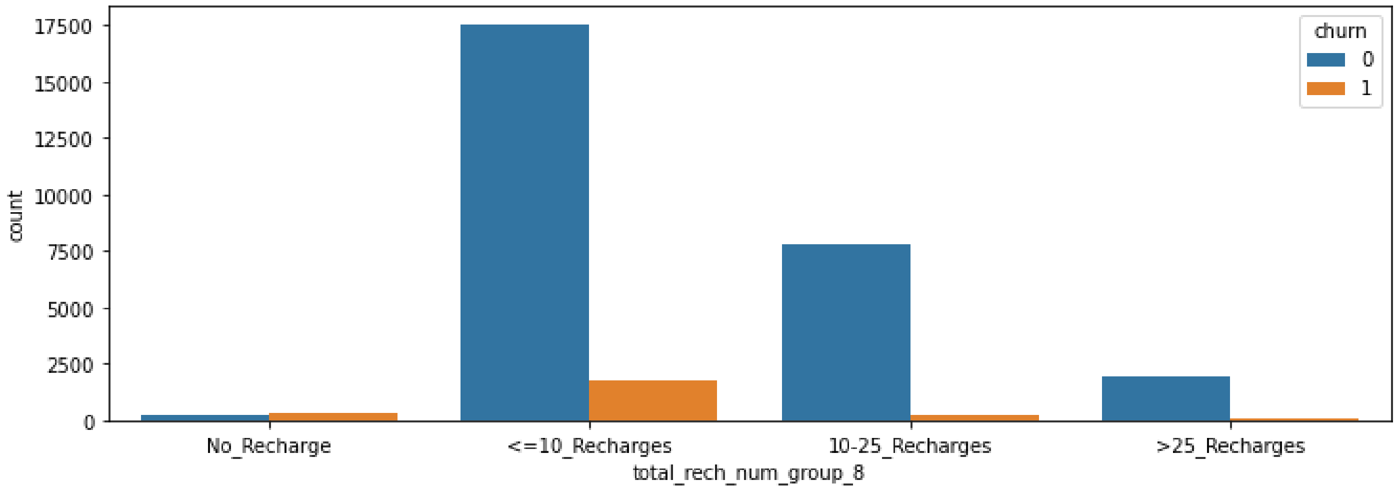
```
# Plotting the results

plt.figure(figsize=[12,4])
sns.countplot(data=telecom_data,x="total_rech_data_group_8",hue="churn")
print("\t\t\t\t\tDistribution of total_rech_data_8 variable\n",telecom_data['total_rech
plt.show()
```

```
plt.figure(figsize=[12,4])
sns.countplot(data=telecom_data,x="total_rech_num_group_8",hue="churn")
print("\t\t\t\t\tDistribution of total_rech_num_8 variable\n",telecom_data['total_rech_
plt.show()
```

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



As the number of recharge rate increases, the churn rate decreases clearly.

```
# Creating a dummy variable for some of the categorical variables and dropping the firs
dummy = pd.get_dummies(telecom_data[['total_rech_data_group_8','total_rech_num_group_8'
dummy.head()
```

	total_rech_data_group_8_<=10_Recharges	total_rech_data_group_8_10-25_Recharges	total_rech_data_group_8_>25_Recharges	total_
0	1	0	0	
7	0	0	0	
8	1	0	0	
21	0	0	0	
23	1	0	0	

```
# Adding the results to the master dataframe
telecom_data = pd.concat([telecom_data, dummy], axis=1)
telecom_data.head()
```

	mobile_number	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8
0	7.000843e+09	213.803	53.27	24.613333	0.00	84.23	23.993333	0.00
7	7.000702e+09	3171.480	57.84	54.680000	52.29	453.43	567.160000	325.91
8	7.001525e+09	137.362	413.69	351.030000	35.08	94.66	80.630000	136.48
21	7.002124e+09	637.760	102.41	132.110000	85.14	757.93	896.680000	983.39
23	7.000887e+09	366.966	48.96	50.660000	33.58	85.41	89.360000	205.89

```
# Creating a copy of the filtered dataframe

df=telecom_data[:].copy()

# Dropping unwanted columns
df.drop(['tenure_range','mobile_number','total_rech_data_group_8','total_rech_num_group
```

```
# Cheking the dataset
df.head()
```

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6
0	213.803	53.27	24.613333	0.00	84.23	23.993333	0.00	0.00
7	3171.480	57.84	54.680000	52.29	453.43	567.160000	325.91	16.23
8	137.362	413.69	351.030000	35.08	94.66	80.630000	136.48	0.00
21	637.760	102.41	132.110000	85.14	757.93	896.680000	983.39	0.00
23	366.966	48.96	50.660000	33.58	85.41	89.360000	205.89	0.00

```
# lets create X dataset for model building.
X = df.drop(['churn'],axis=1)
```

```
X.head()
```

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6
0	213.803	53.27	24.613333	0.00	84.23	23.993333	0.00	0.00
7	3171.480	57.84	54.680000	52.29	453.43	567.160000	325.91	16.23
8	137.362	413.69	351.030000	35.08	94.66	80.630000	136.48	0.00
21	637.760	102.41	132.110000	85.14	757.93	896.680000	983.39	0.00
23	366.966	48.96	50.660000	33.58	85.41	89.360000	205.89	0.00

```
# lets create y dataset for model building.
y=df['churn']
y.head()
```

```
0     1
7     1
8     0
21    0
23    0
Name: churn, dtype: int32
```

```
# split the dateset into train and test datasets
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, train_size=0.7
print("Dimension of X_train:", X_train.shape)
print("Dimension of X_test:", X_test.shape)
```

```
Dimension of X_train: (21000, 126)
Dimension of X_test: (9001, 126)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21000 entries, 15709 to 99093
Data columns (total 126 columns):
 #    Column                                   Dtype
---   ------                                   -----
 0    arpu_8                                   float64
 1    onnet_mou_6                              float64
 2    onnet_mou_7                              float64
 3    onnet_mou_8                              float64
 4    offnet_mou_6                             float64
 5    offnet_mou_7                             float64
 6    offnet_mou_8                             float64
 7    roam_ic_mou_6                            float64
 8    roam_ic_mou_7                            float64
 9    roam_ic_mou_8                            float64
```

10	roam_og_mou_6	float64
11	roam_og_mou_7	float64
12	roam_og_mou_8	float64
13	loc_og_t2t_mou_6	float64
14	loc_og_t2t_mou_7	float64
15	loc_og_t2t_mou_8	float64
16	loc_og_t2m_mou_6	float64
17	loc_og_t2m_mou_7	float64
18	loc_og_t2m_mou_8	float64
19	loc_og_t2f_mou_6	float64
20	loc_og_t2f_mou_7	float64
21	loc_og_t2f_mou_8	float64
22	loc_og_t2c_mou_6	float64
23	loc_og_t2c_mou_7	float64
24	loc_og_t2c_mou_8	float64
25	loc_og_mou_6	float64
26	loc_og_mou_7	float64
27	loc_og_mou_8	float64
28	std_og_t2m_mou_6	float64
29	std_og_t2f_mou_6	float64
30	std_og_t2f_mou_7	float64
31	std_og_t2f_mou_8	float64
32	std_og_mou_6	float64
33	std_og_mou_7	float64
34	std_og_mou_8	float64
35	isd_og_mou_6	float64
36	spl_og_mou_6	float64
37	spl_og_mou_7	float64
38	spl_og_mou_8	float64
39	og_others_6	float64
40	og_others_7	float64
41	og_others_8	float64
42	total_og_mou_6	float64
43	total_og_mou_7	float64
44	total_og_mou_8	float64
45	loc_ic_t2t_mou_6	float64
46	loc_ic_t2t_mou_7	float64
47	loc_ic_t2t_mou_8	float64
48	loc_ic_t2m_mou_6	float64
49	loc_ic_t2m_mou_7	float64
50	loc_ic_t2m_mou_8	float64
51	loc_ic_t2f_mou_6	float64
52	loc_ic_t2f_mou_7	float64

53	loc_ic_t2f_mou_8	float64
54	loc_ic_mou_6	float64
55	loc_ic_mou_7	float64
56	loc_ic_mou_8	float64
57	std_ic_t2t_mou_6	float64
58	std_ic_t2t_mou_7	float64
59	std_ic_t2t_mou_8	float64
60	std_ic_t2m_mou_6	float64
61	std_ic_t2m_mou_7	float64
62	std_ic_t2m_mou_8	float64
63	std_ic_t2f_mou_6	float64
64	std_ic_t2f_mou_7	float64
65	std_ic_t2f_mou_8	float64
66	std_ic_mou_6	float64
67	std_ic_mou_7	float64
68	std_ic_mou_8	float64
69	spl_ic_mou_6	float64
70	spl_ic_mou_7	float64
71	spl_ic_mou_8	float64
72	isd_ic_mou_6	float64
73	isd_ic_mou_7	float64
74	isd_ic_mou_8	float64
75	ic_others_6	float64
76	ic_others_7	float64
77	ic_others_8	float64
78	total_rech_num_6	float64
79	total_rech_num_7	float64
80	total_rech_num_8	float64
81	max_rech_amt_6	float64
82	max_rech_amt_7	float64
83	max_rech_amt_8	float64
84	last_day_rch_amt_6	float64
85	last_day_rch_amt_7	float64
86	last_day_rch_amt_8	float64
87	total_rech_data_8	float64
88	max_rech_data_6	float64
89	max_rech_data_7	float64
90	max_rech_data_8	float64
91	av_rech_amt_data_8	float64
92	vol_2g_mb_6	float64
93	vol_2g_mb_7	float64
94	vol_2g_mb_8	float64
95	vol_3g_mb_6	float64

```
 96   vol_3g_mb_7                               float64
 97   vol_3g_mb_8                               float64
 98   monthly_2g_6                              float64
 99   monthly_2g_7                              float64
 100  monthly_2g_8                              float64
 101  sachet_2g_6                               float64
 102  sachet_2g_7                               float64
 103  monthly_3g_6                              float64
 104  monthly_3g_7                              float64
 105  monthly_3g_8                              float64
 106  sachet_3g_6                               float64
 107  sachet_3g_7                               float64
 108  sachet_3g_8                               float64
 109  aug_vbc_3g                                float64
 110  jul_vbc_3g                                float64
 111  jun_vbc_3g                                float64
 112  overall_rech_amt_6                        float64
 113  overall_rech_amt_7                        float64
 114  avg_rech_amt_6_7                          float64
 115  avg_arpu_6_7                              float64
 116  total_rech_data_group_8_<=10_Recharges    uint8
 117  total_rech_data_group_8_10-25_Recharges   uint8
 118  total_rech_data_group_8_>25_Recharges     uint8
 119  total_rech_num_group_8_<=10_Recharges     uint8
 120  total_rech_num_group_8_10-25_Recharges    uint8
 121  total_rech_num_group_8_>25_Recharges      uint8
 122  tenure_range_6-12 Months                  uint8
 123  tenure_range_1-2 Yrs                      uint8
 124  tenure_range_2-5 Yrs                      uint8
 125  tenure_range_5 Yrs and above              uint8
dtypes: float64(116), uint8(10)
memory usage: 18.9 MB
```

```
num_col = X_train.select_dtypes(include = ['int64','float64']).columns.tolist()
```

```
# apply scaling on the dataset
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train[num_col] = scaler.fit_transform(X_train[num_col])
```

```
X_train.head()
```

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_
15709	0.038904	0.000235	0.000531	0.000238	0.004211	0.003651	0.004095	0
28202	0.032921	0.000493	0.000000	0.000000	0.001631	0.000000	0.000000	0
14943	0.033826	0.000876	0.000275	0.000714	0.003861	0.007485	0.003679	0
92007	0.081645	0.163879	0.105394	0.050406	0.142667	0.177782	0.052962	0
56403	0.042893	0.079633	0.051881	0.004868	0.058346	0.046732	0.010097	0

Data Imbalance Handling

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

```
# Importing necessary libraries for Model creation
import statsmodels.api as sm
```

```
# Logistic regression model
logm1 = sm.GLM(y_train_sm,(sm.add_constant(X_train_sm)), family = sm.families.Binomial(
logm1.fit().summary()
```

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
arpu_8	4.7856	1.723	2.777	0.005	1.409	8.163

onnet_mou_6	-51.0285	28.723	-1.777	0.076	-107.325	5.268
onnet_mou_7	58.2210	16.330	3.565	0.000	26.215	90.227
onnet_mou_8	181.4643	24.155	7.513	0.000	134.122	228.807
offnet_mou_6	-47.8874	32.384	-1.479	0.139	-111.359	15.584
offnet_mou_7	71.5233	19.308	3.704	0.000	33.680	109.367
offnet_mou_8	232.4309	31.430	7.395	0.000	170.830	294.032
roam_ic_mou_6	3.0054	0.846	3.555	0.000	1.348	4.663
roam_ic_mou_7	5.1800	1.572	3.295	0.001	2.099	8.261
roam_ic_mou_8	-1.5652	0.801	-1.953	0.051	-3.136	0.006
roam_og_mou_6	21.8398	14.676	1.488	0.137	-6.924	50.604
roam_og_mou_7	-12.2597	3.827	-3.203	0.001	-19.760	-4.759
roam_og_mou_8	-55.8880	7.814	-7.152	0.000	-71.203	-40.573
loc_og_t2t_mou_6	-6948.9854	1.98e+04	-0.352	0.725	-4.57e+04	3.18e+04
loc_og_t2t_mou_7	-1.823e+04	2.28e+04	-0.800	0.424	-6.29e+04	2.64e+04
loc_og_t2t_mou_8	1.93e+05	3.41e+04	5.662	0.000	1.26e+05	2.6e+05
loc_og_t2m_mou_6	-5118.4731	1.45e+04	-0.352	0.725	-3.36e+04	2.34e+04
loc_og_t2m_mou_7	-1.099e+04	1.37e+04	-0.800	0.424	-3.79e+04	1.59e+04
loc_og_t2m_mou_8	6.367e+04	1.12e+04	5.662	0.000	4.16e+04	8.57e+04
loc_og_t2f_mou_6	-729.5957	2078.273	-0.351	0.726	-4802.935	3343.744
loc_og_t2f_mou_7	-2621.4647	3259.099	-0.804	0.421	-9009.182	3766.253
loc_og_t2f_mou_8	1.667e+04	2943.371	5.664	0.000	1.09e+04	2.24e+04
loc_og_t2c_mou_6	-5.0019	1.068	-4.685	0.000	-7.094	-2.909
loc_og_t2c_mou_7	0.2131	1.864	0.114	0.909	-3.441	3.867
loc_og_t2c_mou_8	0.3722	1.221	0.305	0.761	-2.021	2.765
loc_og_mou_6	1.161e+04	3.27e+04	0.355	0.723	-5.25e+04	7.57e+04
loc_og_mou_7	1.888e+04	2.36e+04	0.799	0.424	-2.75e+04	6.52e+04
loc_og_mou_8	-1.981e+05	3.5e+04	-5.660	0.000	-2.67e+05	-1.29e+05
std_og_t2m_mou_6	-10.9573	4.002	-2.738	0.006	-18.802	-3.113
std_og_t2f_mou_6	-1.8195	1.671	-1.089	0.276	-5.094	1.455
std_og_t2f_mou_7	1.0576	1.824	0.580	0.562	-2.517	4.633
std_og_t2f_mou_8	-9.8438	2.711	-3.631	0.000	-15.158	-4.530
std_og_mou_6	83.7209	33.002	2.537	0.011	19.038	148.403
std_og_mou_7	-54.8592	21.210	-2.586	0.010	-96.430	-13.288
std_og_mou_8	110.9924	30.090	3.689	0.000	52.017	169.968
isd_og_mou_6	13.2105	5.562	2.375	0.018	2.310	24.111
spl_og_mou_6	5.2820	2.044	2.584	0.010	1.275	9.289
spl_og_mou_7	-6.8701	2.397	-2.866	0.004	-11.568	-2.173
spl_og_mou_8	9.1897	2.263	4.061	0.000	4.755	13.625
og_others_6	-3.0564	0.706	-4.331	0.000	-4.440	-1.673
og_others_7	1.3761	7.492	0.184	0.854	-13.307	16.059

og_others_8	-165.7948	51.500	-3.219	0.001	-266.732	-64.858
total_og_mou_6	-29.7757	4.604	-6.467	0.000	-38.800	-20.751
total_og_mou_7	-20.4135	7.917	-2.578	0.010	-35.931	-4.896
total_og_mou_8	-361.4826	12.927	-27.962	0.000	-386.820	-336.145
loc_ic_t2t_mou_6	-1.999e+04	1.82e+04	-1.097	0.273	-5.57e+04	1.57e+04
loc_ic_t2t_mou_7	1.781e+05	1.68e+04	10.599	0.000	1.45e+05	2.11e+05
loc_ic_t2t_mou_8	1.301e+05	1.2e+04	10.828	0.000	1.07e+05	1.54e+05
loc_ic_t2m_mou_6	-1.477e+04	1.35e+04	-1.097	0.273	-4.12e+04	1.16e+04
loc_ic_t2m_mou_7	1.302e+05	1.23e+04	10.600	0.000	1.06e+05	1.54e+05
loc_ic_t2m_mou_8	1.434e+05	1.32e+04	10.827	0.000	1.17e+05	1.69e+05
loc_ic_t2f_mou_6	-5288.4284	4817.200	-1.098	0.272	-1.47e+04	4153.110
loc_ic_t2f_mou_7	6.187e+04	5837.260	10.600	0.000	5.04e+04	7.33e+04
loc_ic_t2f_mou_8	5.162e+04	4767.356	10.828	0.000	4.23e+04	6.1e+04
loc_ic_mou_6	2.347e+04	2.14e+04	1.097	0.273	-1.85e+04	6.54e+04
loc_ic_mou_7	-2.018e+05	1.9e+04	-10.599	0.000	-2.39e+05	-1.64e+05
loc_ic_mou_8	-1.751e+05	1.62e+04	-10.829	0.000	-2.07e+05	-1.43e+05
std_ic_t2t_mou_6	-8.05e+04	1.97e+04	-4.078	0.000	-1.19e+05	-4.18e+04
std_ic_t2t_mou_7	2.023e+04	2.12e+04	0.956	0.339	-2.12e+04	6.17e+04
std_ic_t2t_mou_8	7456.8463	1.75e+04	0.426	0.670	-2.69e+04	4.18e+04
std_ic_t2m_mou_6	-6.826e+04	1.67e+04	-4.077	0.000	-1.01e+05	-3.55e+04
std_ic_t2m_mou_7	1.209e+04	1.27e+04	0.955	0.340	-1.27e+04	3.69e+04
std_ic_t2m_mou_8	9795.6016	2.3e+04	0.427	0.670	-3.52e+04	5.48e+04
std_ic_t2f_mou_6	-1.992e+04	4885.426	-4.078	0.000	-2.95e+04	-1.03e+04
std_ic_t2f_mou_7	3548.4493	3716.763	0.955	0.340	-3736.271	1.08e+04
std_ic_t2f_mou_8	2417.9893	5671.352	0.426	0.670	-8697.657	1.35e+04
std_ic_mou_6	8.05e+04	1.97e+04	4.078	0.000	4.18e+04	1.19e+05
std_ic_mou_7	-2.35e+04	2.46e+04	-0.955	0.340	-7.17e+04	2.47e+04
std_ic_mou_8	-1.035e+04	2.42e+04	-0.427	0.669	-5.78e+04	3.71e+04
spl_ic_mou_6	7.2468	1.960	3.697	0.000	3.405	11.089
spl_ic_mou_7	-6.6548	3.352	-1.985	0.047	-13.225	-0.085
spl_ic_mou_8	-23.7450	1.633	-14.545	0.000	-26.945	-20.545
isd_ic_mou_6	4.2685	2.547	1.676	0.094	-0.724	9.261
isd_ic_mou_7	3.1616	1.963	1.611	0.107	-0.685	7.008
isd_ic_mou_8	-1.7850	1.334	-1.338	0.181	-4.400	0.830
ic_others_6	-14.2950	5.498	-2.600	0.009	-25.070	-3.520
ic_others_7	-1.2308	4.474	-0.275	0.783	-10.000	7.538
ic_others_8	6.3301	3.652	1.733	0.083	-0.828	13.488
total_rech_num_6	-1.1745	0.938	-1.252	0.211	-3.013	0.664
total_rech_num_7	4.1068	0.547	7.501	0.000	3.034	5.180
total_rech_num_8	-8.8626	1.196	-7.407	0.000	-11.208	-6.518

max_rech_amt_6	-1.7025	0.711	-2.393	0.017	-3.097	-0.308
max_rech_amt_7	0.5615	0.583	0.962	0.336	-0.582	1.705
max_rech_amt_8	8.2360	0.867	9.499	0.000	6.537	9.935
last_day_rch_amt_6	0.5915	0.723	0.818	0.413	-0.825	2.008
last_day_rch_amt_7	-1.2024	0.674	-1.785	0.074	-2.523	0.118
last_day_rch_amt_8	-18.5883	0.917	-20.280	0.000	-20.385	-16.792
total_rech_data_8	-3.9021	1.120	-3.484	0.000	-6.097	-1.707
max_rech_data_6	0.2650	0.470	0.564	0.573	-0.656	1.186
max_rech_data_7	1.2588	0.492	2.560	0.010	0.295	2.223
max_rech_data_8	-0.4231	0.885	-0.478	0.633	-2.159	1.312
av_rech_amt_data_8	-12.5302	1.826	-6.863	0.000	-16.109	-8.952
vol_2g_mb_6	2.8277	0.792	3.572	0.000	1.276	4.379
vol_2g_mb_7	3.3457	0.708	4.724	0.000	1.958	4.734
vol_2g_mb_8	-13.3725	1.193	-11.210	0.000	-15.710	-11.035
vol_3g_mb_6	-4.4761	2.297	-1.948	0.051	-8.979	0.027
vol_3g_mb_7	0.2100	1.500	0.140	0.889	-2.730	3.150
vol_3g_mb_8	-4.5393	2.010	-2.258	0.024	-8.479	-0.599
monthly_2g_6	-1.1288	0.259	-4.357	0.000	-1.637	-0.621
monthly_2g_7	-1.9149	0.281	-6.803	0.000	-2.467	-1.363
monthly_2g_8	-1.8365	0.478	-3.846	0.000	-2.772	-0.901
sachet_2g_6	1.4624	0.526	2.780	0.005	0.431	2.494
sachet_2g_7	-0.1821	0.675	-0.270	0.787	-1.506	1.142
monthly_3g_6	1.5719	0.868	1.810	0.070	-0.130	3.274
monthly_3g_7	-2.3536	0.896	-2.627	0.009	-4.109	-0.598
monthly_3g_8	3.0809	1.373	2.244	0.025	0.391	5.771
sachet_3g_6	2.9297	0.842	3.479	0.001	1.279	4.580
sachet_3g_7	1.8321	1.101	1.664	0.096	-0.326	3.990
sachet_3g_8	2.4807	1.646	1.507	0.132	-0.745	5.707
aug_vbc_3g	-5.8841	0.807	-7.290	0.000	-7.466	-4.302
jul_vbc_3g	2.6293	0.788	3.337	0.001	1.085	4.174
jun_vbc_3g	1.8643	0.859	2.170	0.030	0.181	3.548
overall_rech_amt_6	-1.3541	1.833	-0.739	0.460	-4.948	2.239
overall_rech_amt_7	1.1217	1.800	0.623	0.533	-2.406	4.649
avg_rech_amt_6_7	-0.7551	1.361	-0.555	0.579	-3.422	1.912
avg_arpu_6_7	6.2959	1.936	3.252	0.001	2.502	10.090
total_rech_data_group_8_<=10_Recharges	-0.2095	0.059	-3.571	0.000	-0.325	-0.095
total_rech_data_group_8_10-25_Recharges	-3.9029	0.808	-4.832	0.000	-5.486	-2.320
total_rech_data_group_8_>25_Recharges	-0.7705	1.194	-0.645	0.519	-3.112	1.571
total_rech_num_group_8_<=10_Recharges	-0.5896	0.098	-5.988	0.000	-0.783	-0.397
total_rech_num_group_8_10-25_Recharges	-0.4755	0.120	-3.967	0.000	-0.710	-0.241

total_rech_num_group_8_>25_Recharges	-0.5457	0.200	-2.728	0.006	-0.938	-0.154
tenure_range_6-12 Months	0.3585	0.052	6.921	0.000	0.257	0.460
tenure_range_1-2 Yrs	0.1820	0.048	3.753	0.000	0.087	0.277
tenure_range_2-5 Yrs	0.0659	0.047	1.399	0.162	-0.026	0.158
tenure_range_5 Yrs and above	-0.6732	0.346	-1.945	0.052	-1.352	0.005

Logistic Regression using Feature Selection (RFE method)

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

from sklearn.feature_selection import RFE

# running RFE with 20 variables as output
rfe = RFE(logreg, 20)
rfe = rfe.fit(X_train_sm, y_train_sm)
```

```
rfe.support_
```

```
array([ True, False, False, False, False, False, False, False,  True,
       False, False, False,  True, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False,  True, False, False,  True, False, False, False,
        True, False,  True, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False,  True,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False,  True,  True, False, False,
       False,  True, False, False,  True, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False])
```

```
rfe_columns=X_train_sm.columns[rfe.support_]
print("The selected columns by RFE for modelling are: \n\n",rfe_columns)
```

```
The selected columns by RFE for modelling are:

 Index(['arpu_8', 'roam_ic_mou_7', 'roam_og_mou_8', 'loc_og_t2m_mou_8',
       'std_og_mou_7', 'total_og_mou_8', 'loc_ic_t2t_mou_8',
       'loc_ic_t2m_mou_8', 'loc_ic_mou_6', 'loc_ic_mou_8', 'std_ic_mou_8',
       'spl_ic_mou_8', 'total_rech_num_8', 'last_day_rch_amt_8',
       'total_rech_data_8', 'av_rech_amt_data_8', 'vol_2g_mb_8',
       'monthly_2g_8', 'aug_vbc_3g', 'avg_arpu_6_7'],
      dtype='object')
```

```
list(zip(X_train_sm.columns, rfe.support_, rfe.ranking_))
```

```
[('arpu_8', True, 1),
 ('onnet_mou_6', False, 22),
 ('onnet_mou_7', False, 37),
 ('onnet_mou_8', False, 42),
 ('offnet_mou_6', False, 35),
 ('offnet_mou_7', False, 21),
 ('offnet_mou_8', False, 26),
 ('roam_ic_mou_6', False, 13),
 ('roam_ic_mou_7', True, 1),
 ('roam_ic_mou_8', False, 60),
 ('roam_og_mou_6', False, 69),
 ('roam_og_mou_7', False, 33),
 ('roam_og_mou_8', True, 1),
 ('loc_og_t2t_mou_6', False, 65),
 ('loc_og_t2t_mou_7', False, 99),
 ('loc_og_t2t_mou_8', False, 19),
 ('loc_og_t2m_mou_6', False, 67),
 ('loc_og_t2m_mou_7', False, 74),
 ('loc_og_t2m_mou_8', True, 1),
 ('loc_og_t2f_mou_6', False, 107),
 ('loc_og_t2f_mou_7', False, 5),
 ('loc_og_t2f_mou_8', False, 25),
 ('loc_og_t2c_mou_6', False, 7),
 ('loc_og_t2c_mou_7', False, 66),
 ('loc_og_t2c_mou_8', False, 104),
 ('loc_og_mou_6', False, 48),
 ('loc_og_mou_7', False, 105),
 ('loc_og_mou_8', False, 2),
 ('std_og_t2m_mou_6', False, 93),
 ('std_og_t2f_mou_6', False, 79),
 ('std_og_t2f_mou_7', False, 27),
 ('std_og_t2f_mou_8', False, 4),
 ('std_og_mou_6', False, 46),
 ('std_og_mou_7', True, 1),
 ('std_og_mou_8', False, 64),
 ('isd_og_mou_6', False, 14),
 ('spl_og_mou_6', False, 87),
 ('spl_og_mou_7', False, 51),
 ('spl_og_mou_8', False, 36),
 ('og_others_6', False, 23),
 ('og_others_7', False, 82),
 ('og_others_8', False, 98),
 ('total_og_mou_6', False, 47),
 ('total_og_mou_7', False, 90),
 ('total_og_mou_8', True, 1),
 ('loc_ic_t2t_mou_6', False, 45),
 ('loc_ic_t2t_mou_7', False, 77),
```

```
('loc_ic_t2t_mou_8', True, 1),
('loc_ic_t2m_mou_6', False, 6),
('loc_ic_t2m_mou_7', False, 28),
('loc_ic_t2m_mou_8', True, 1),
('loc_ic_t2f_mou_6', False, 52),
('loc_ic_t2f_mou_7', False, 83),
('loc_ic_t2f_mou_8', False, 11),
('loc_ic_mou_6', True, 1),
('loc_ic_mou_7', False, 57),
('loc_ic_mou_8', True, 1),
('std_ic_t2t_mou_6', False, 59),
('std_ic_t2t_mou_7', False, 32),
('std_ic_t2t_mou_8', False, 12),
('std_ic_t2m_mou_6', False, 38),
('std_ic_t2m_mou_7', False, 39),
('std_ic_t2m_mou_8', False, 8),
('std_ic_t2f_mou_6', False, 95),
('std_ic_t2f_mou_7', False, 50),
('std_ic_t2f_mou_8', False, 34),
('std_ic_mou_6', False, 9),
('std_ic_mou_7', False, 73),
('std_ic_mou_8', True, 1),
('spl_ic_mou_6', False, 102),
('spl_ic_mou_7', False, 92),
('spl_ic_mou_8', True, 1),
('isd_ic_mou_6', False, 54),
('isd_ic_mou_7', False, 40),
('isd_ic_mou_8', False, 55),
('ic_others_6', False, 53),
('ic_others_7', False, 70),
('ic_others_8', False, 78),
('total_rech_num_6', False, 103),
('total_rech_num_7', False, 3),
('total_rech_num_8', True, 1),
('max_rech_amt_6', False, 81),
('max_rech_amt_7', False, 16),
('max_rech_amt_8', False, 72),
('last_day_rch_amt_6', False, 89),
('last_day_rch_amt_7', False, 15),
('last_day_rch_amt_8', True, 1),
('total_rech_data_8', True, 1),
('max_rech_data_6', False, 41),
('max_rech_data_7', False, 61),
('max_rech_data_8', False, 100),
('av_rech_amt_data_8', True, 1),
('vol_2g_mb_6', False, 43),
('vol_2g_mb_7', False, 17),
('vol_2g_mb_8', True, 1),
('vol_3g_mb_6', False, 97),
```

```
 ('vol_3g_mb_7', False, 62),
 ('vol_3g_mb_8', False, 71),
 ('monthly_2g_6', False, 44),
 ('monthly_2g_7', False, 18),
 ('monthly_2g_8', True, 1),
 ('sachet_2g_6', False, 63),
 ('sachet_2g_7', False, 106),
 ('monthly_3g_6', False, 84),
 ('monthly_3g_7', False, 49),
 ('monthly_3g_8', False, 75),
 ('sachet_3g_6', False, 10),
 ('sachet_3g_7', False, 24),
 ('sachet_3g_8', False, 76),
 ('aug_vbc_3g', True, 1),
 ('jul_vbc_3g', False, 58),
 ('jun_vbc_3g', False, 88),
 ('overall_rech_amt_6', False, 85),
 ('overall_rech_amt_7', False, 86),
 ('avg_rech_amt_6_7', False, 101),
 ('avg_arpu_6_7', True, 1),
 ('total_rech_data_group_8_<=10_Recharges', False, 68),
 ('total_rech_data_group_8_10-25_Recharges', False, 20),
 ('total_rech_data_group_8_>25_Recharges', False, 80),
 ('total_rech_num_group_8_<=10_Recharges', False, 31),
 ('total_rech_num_group_8_10-25_Recharges', False, 30),
 ('total_rech_num_group_8_>25_Recharges', False, 29),
 ('tenure_range_6-12 Months', False, 91),
 ('tenure_range_1-2 Yrs', False, 94),
 ('tenure_range_2-5 Yrs', False, 96),
 ('tenure_range_5 Yrs and above', False, 56)]
```

Assessing the model with StatsModels

```
X_train_SM = sm.add_constant(X_train_sm[rfe_columns])
logm2 = sm.GLM(y_train_sm,X_train_SM, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	churn	**No. Observations:**	38576
Model:	GLM	**Df Residuals:**	38555
Model Family:	Binomial	**Df Model:**	20
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-15852.
Date:	Mon, 01 Mar 2021	**Deviance:**	31703.
Time:	15:31:57	**Pearson chi2:**	8.44e+10
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5718	0.071	8.101	0.000	0.433	0.710
arpu_8	-7.4189	1.295	-5.730	0.000	-9.957	-4.881
roam_ic_mou_7	8.3147	0.929	8.948	0.000	6.493	10.136
roam_og_mou_8	4.3254	0.602	7.190	0.000	3.146	5.505
loc_og_t2m_mou_8	-3.8292	0.668	-5.731	0.000	-5.139	-2.520
std_og_mou_7	7.7463	0.513	15.114	0.000	6.742	8.751
total_og_mou_8	-20.2090	0.956	-21.143	0.000	-22.082	-18.336
loc_ic_t2t_mou_8	1.0280	4.078	0.252	0.801	-6.966	9.022
loc_ic_t2m_mou_8	-1.2611	4.448	-0.284	0.777	-9.978	7.456
loc_ic_mou_6	9.1611	0.723	12.666	0.000	7.743	10.579
loc_ic_mou_8	-31.0473	5.010	-6.197	0.000	-40.866	-21.228
std_ic_mou_8	-11.9357	1.359	-8.784	0.000	-14.599	-9.272
spl_ic_mou_8	-19.8516	1.375	-14.436	0.000	-22.547	-17.156
total_rech_num_8	-7.0996	0.533	-13.327	0.000	-8.144	-6.055
last_day_rch_amt_8	-18.3227	0.810	-22.610	0.000	-19.911	-16.734
total_rech_data_8	-8.9207	0.612	-14.580	0.000	-10.120	-7.721
av_rech_amt_data_8	-5.2450	0.644	-8.148	0.000	-6.507	-3.983
vol_2g_mb_8	-10.4892	0.934	-11.229	0.000	-12.320	-8.658
monthly_2g_8	-5.7717	0.360	-16.041	0.000	-6.477	-5.066
aug_vbc_3g	-6.8396	0.588	-11.633	0.000	-7.992	-5.687
avg_arpu_6_7	18.2676	1.089	16.781	0.000	16.134	20.401

```
# From the p-value of the individual columns,
    # we can drop the column 'loc_ic_t2t_mou_8' as it has high p-value of 0.80
rfe_columns_1=rfe_columns.drop('loc_ic_t2t_mou_8',1)
print("\nThe new set of edited featured are:\n",rfe_columns_1)
```

```
The new set of columns are:
 Index(['arpu_8', 'roam_ic_mou_7', 'roam_og_mou_8', 'loc_og_t2m_mou_8',
       'std_og_mou_7', 'total_og_mou_8', 'loc_ic_t2m_mou_8', 'loc_ic_mou_6',
       'loc_ic_mou_8', 'std_ic_mou_8', 'spl_ic_mou_8', 'total_rech_num_8',
       'last_day_rch_amt_8', 'total_rech_data_8', 'av_rech_amt_data_8',
       'vol_2g_mb_8', 'monthly_2g_8', 'aug_vbc_3g', 'avg_arpu_6_7'],
      dtype='object')
```

```
# Training the model with the edited feature list
X_train_SM = sm.add_constant(X_train_sm[rfe_columns_1])
logm2 = sm.GLM(y_train_sm,X_train_SM, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	churn	**No. Observations:**	38576
Model:	GLM	**Df Residuals:**	38556
Model Family:	Binomial	**Df Model:**	19
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-15852.
Date:	Mon, 01 Mar 2021	**Deviance:**	31703.
Time:	15:38:29	**Pearson chi2:**	8.49e+10
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.5714	0.071	8.096	0.000	0.433	0.710
arpu_8	-7.4196	1.295	-5.730	0.000	-9.957	-4.882
roam_ic_mou_7	8.3174	0.929	8.950	0.000	6.496	10.139
roam_og_mou_8	4.3268	0.602	7.192	0.000	3.148	5.506
loc_og_t2m_mou_8	-3.8309	0.668	-5.734	0.000	-5.140	-2.521
std_og_mou_7	7.7428	0.512	15.114	0.000	6.739	8.747
total_og_mou_8	-20.2018	0.955	-21.146	0.000	-22.074	-18.329
loc_ic_t2m_mou_8	-2.2429	2.136	-1.050	0.294	-6.429	1.943
loc_ic_mou_6	9.1640	0.723	12.669	0.000	7.746	10.582
loc_ic_mou_8	-29.8843	1.933	-15.463	0.000	-33.672	-26.096
std_ic_mou_8	-11.9422	1.359	-8.789	0.000	-14.605	-9.279
spl_ic_mou_8	-19.8488	1.375	-14.435	0.000	-22.544	-17.154
total_rech_num_8	-7.0935	0.532	-13.330	0.000	-8.136	-6.051
last_day_rch_amt_8	-18.3273	0.810	-22.622	0.000	-19.915	-16.739
total_rech_data_8	-8.9222	0.612	-14.583	0.000	-10.121	-7.723
av_rech_amt_data_8	-5.2484	0.644	-8.155	0.000	-6.510	-3.987
vol_2g_mb_8	-10.4886	0.934	-11.228	0.000	-12.320	-8.658
monthly_2g_8	-5.7723	0.360	-16.043	0.000	-6.478	-5.067
aug_vbc_3g	-6.8433	0.588	-11.642	0.000	-7.995	-5.691
avg_arpu_6_7	18.2734	1.088	16.790	0.000	16.140	20.406

```
# From the p-value of the individual columns,
    # we can drop the column 'loc_ic_t2m_mou_8' as it has high p-value of 0.80
rfe_columns_2=rfe_columns_1.drop('loc_ic_t2m_mou_8',1)
print("\nThe new set of edited featured are:\n",rfe_columns_2)
```

```
The new set of edited featured are:
 Index(['arpu_8', 'roam_ic_mou_7', 'roam_og_mou_8', 'loc_og_t2m_mou_8',
       'std_og_mou_7', 'total_og_mou_8', 'loc_ic_mou_6', 'loc_ic_mou_8',
       'std_ic_mou_8', 'spl_ic_mou_8', 'total_rech_num_8',
```

```
       'last_day_rch_amt_8', 'total_rech_data_8', 'av_rech_amt_data_8',
       'vol_2g_mb_8', 'monthly_2g_8', 'aug_vbc_3g', 'avg_arpu_6_7'],
      dtype='object')
```

```
# Training the model with the edited feature list
X_train_SM = sm.add_constant(X_train_sm[rfe_columns_2])
logm2 = sm.GLM(y_train_sm,X_train_SM, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	churn	**No. Observations:**	38576
Model:	GLM	**Df Residuals:**	38557
Model Family:	Binomial	**Df Model:**	18
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-15852.
Date:	Mon, 01 Mar 2021	**Deviance:**	31704.
Time:	15:41:29	**Pearson chi2:**	8.51e+10
No. Iterations:	7		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5682	0.071	8.055	0.000	0.430	0.706
arpu_8	-7.3871	1.294	-5.709	0.000	-9.923	-4.851
roam_ic_mou_7	8.2919	0.930	8.919	0.000	6.470	10.114
roam_og_mou_8	4.3369	0.602	7.208	0.000	3.158	5.516
loc_og_t2m_mou_8	-3.9987	0.650	-6.156	0.000	-5.272	-2.725
std_og_mou_7	7.7052	0.511	15.082	0.000	6.704	8.707
total_og_mou_8	-20.1259	0.952	-21.139	0.000	-21.992	-18.260
loc_ic_mou_6	9.1605	0.724	12.652	0.000	7.741	10.580
loc_ic_mou_8	-31.5914	1.068	-29.592	0.000	-33.684	-29.499
std_ic_mou_8	-11.9423	1.359	-8.790	0.000	-14.605	-9.280
spl_ic_mou_8	-19.8518	1.375	-14.440	0.000	-22.546	-17.157
total_rech_num_8	-7.1243	0.531	-13.408	0.000	-8.166	-6.083
last_day_rch_amt_8	-18.3312	0.810	-22.622	0.000	-19.919	-16.743
total_rech_data_8	-8.9197	0.612	-14.580	0.000	-10.119	-7.721
av_rech_amt_data_8	-5.2486	0.644	-8.155	0.000	-6.510	-3.987
vol_2g_mb_8	-10.5014	0.934	-11.242	0.000	-12.332	-8.671
monthly_2g_8	-5.7637	0.360	-16.025	0.000	-6.469	-5.059
aug_vbc_3g	-6.8479	0.588	-11.651	0.000	-8.000	-5.696
avg_arpu_6_7	18.3112	1.088	16.823	0.000	16.178	20.445

```
# Getting the predicted values on the train set
y_train_sm_pred = res.predict(X_train_SM)
y_train_sm_pred = y_train_sm_pred.values.reshape(-1)
y_train_sm_pred[:10]
```

```
array([1.38574250e-01, 4.01121753e-01, 3.24275768e-01, 4.14619020e-01,
       5.08729618e-01, 4.31066021e-01, 2.12010834e-05, 2.27844968e-01,
       5.14992869e-02, 7.08374581e-01])
```

Creating a dataframe with the actual churn flag and the predicted probabilities

```
y_train_sm_pred_final = pd.DataFrame({'Converted':y_train_sm.values, 'Converted_prob':y
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

Creating new column 'churn_pred' with 1 if Churn_Prob > 0.5 else 0

```
y_train_sm_pred_final['churn_pred'] = y_train_sm_pred_final.Converted_prob.map(lambda x

# 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

```
from sklearn import metrics

# Confusion matrix
confusion = metrics.confusion_matrix(y_train_sm_pred_final.Converted, y_train_sm_pred_f
print(confusion)
```

```
[[15661  3627]
 [ 2775 16513]]
```

```
# Predicted     not_churn    churn
# Actual
```

```
# not_churn         15661       3627
# churn             2775        16513
```

```
# Checking the overall accuracy.
print("The overall accuracy of the model is:",metrics.accuracy_score(y_train_sm_pred_fi
```

```
The overall accuracy of the model is: 0.8340418913313977
```

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# Create a dataframe that will contain the names of all the feature variables and their
vif = pd.DataFrame()
vif['Features'] = X_train_sm[rfe_columns_2].columns
vif['VIF'] = [variance_inflation_factor(X_train_sm[rfe_columns].values, i) for i in ran
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
9	spl_ic_mou_8	83.90
7	loc_ic_mou_8	42.86
0	arpu_8	18.96
6	loc_ic_mou_6	18.68
5	total_og_mou_8	5.46
12	total_rech_data_8	3.58
4	std_og_mou_7	3.27
8	std_ic_mou_8	2.88
15	monthly_2g_8	2.76
3	loc_og_t2m_mou_8	2.54
14	vol_2g_mb_8	2.06
13	av_rech_amt_data_8	1.76
2	roam_og_mou_8	1.56
16	aug_vbc_3g	1.35
17	avg_arpu_6_7	1.33
1	roam_ic_mou_7	1.30
10	total_rech_num_8	1.15
11	last_day_rch_amt_8	1.05

Metrics beyond simply accuracy

```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
```

```
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

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

# Let us calculate specificity
print("Specificity = ",TN / float(TN+FP))

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

# positive predictive value
print ("Precision = ",TP / float(TP+FP))

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

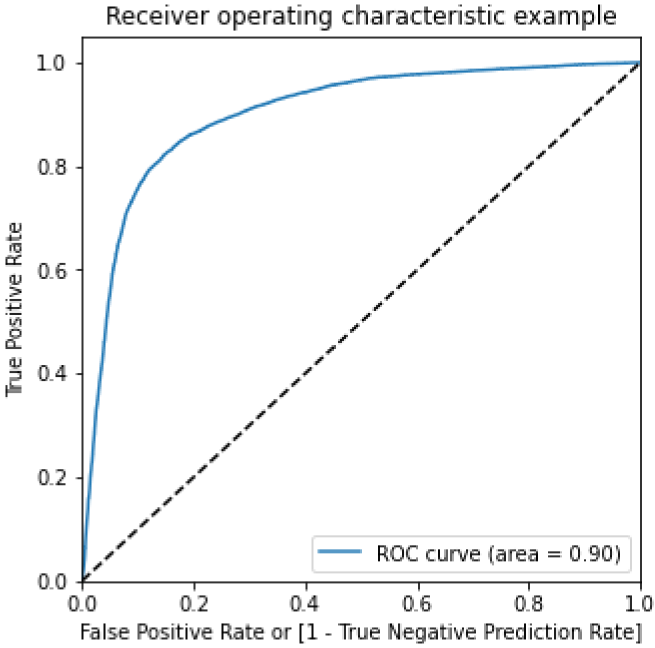
```
Sensitivity =  0.8561281625881377
Specificity =  0.8119556200746578
False Positive Rate =  0.18804437992534218
Precision =  0.8199106256206554
True Negative Prediction Rate =  0.8494792796702104
```

Plotting the ROC Curve

```
# Defining a function to plot the roc curve
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Prediction Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

```
# Defining the variables to plot the curve
fpr, tpr, thresholds = metrics.roc_curve( y_train_sm_pred_final.Converted, y_train_sm_p
```

```
# Plotting the curve for the obtained metrics
draw_roc(y_train_sm_pred_final.Converted, y_train_sm_pred_final.Converted_prob)
```



Finding Optimal Cutoff Point

```
# Let's create columns with different probability cutoffs
numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_sm_pred_final[i]= y_train_sm_pred_final.Converted_prob.map(lambda x: 1 if x
y_train_sm_pred_final.head()
```

	Converted	Converted_prob	churn_pred	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.138574	0	1	1	0	0	0	0	0	0	0	0
1	0	0.401122	0	1	1	1	1	1	0	0	0	0	0
2	0	0.324276	0	1	1	1	1	0	0	0	0	0	0
3	0	0.414619	0	1	1	1	1	1	0	0	0	0	0
4	0	0.508730	1	1	1	1	1	1	1	0	0	0	0

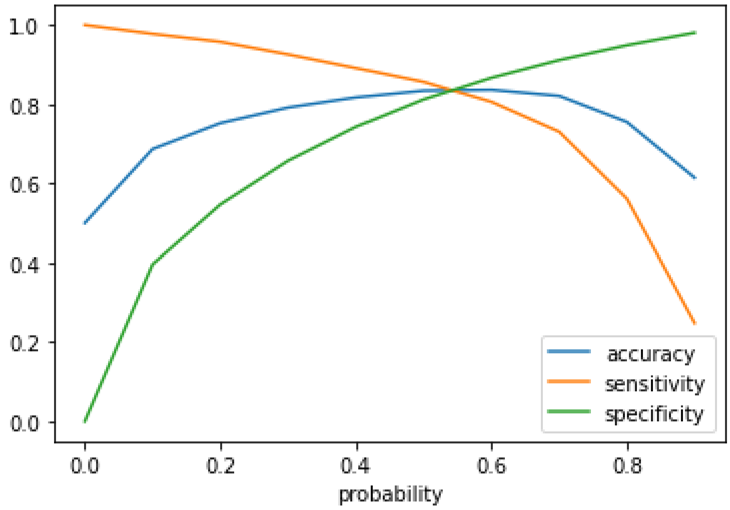
```
# Now let's calculate accuracy sensitivity and specificity for various probability cuto
cutoff_df = pd.DataFrame( columns = ['probability','accuracy','sensitivity','specificit
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_sm_pred_final.Converted, y_train_sm_pred_fir
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    specificity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensitivity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
```

```
    cutoff_df.loc[i] =[ i ,accuracy,sensitivity,specificity]
print(cutoff_df)
```

```
     probability  accuracy  sensitivity  specificity
0.0  0.0          0.500000  1.000000     0.000000
0.1  0.1          0.686696  0.977603     0.395790
0.2  0.2          0.751996  0.957538     0.546454
0.3  0.3          0.791321  0.925653     0.656989
0.4  0.4          0.816881  0.891176     0.742586
0.5  0.5          0.834042  0.856128     0.811956
0.6  0.6          0.836116  0.805682     0.866549
0.7  0.7          0.820795  0.730350     0.911240
0.8  0.8          0.755003  0.561230     0.948776
0.9  0.9          0.614294  0.248185     0.980402
```

```
# plotting accuracy sensitivity and specificity for various probabilities calculated ab
cutoff_df.plot.line(x='probability', y=['accuracy','sensitivity','specificity'])
plt.show()
```

<Figure size 1080x1080 with 0 Axes>



Initially we selected the optimm 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 tweek a little more within this range.

```
# Let's create columns with refined probability cutoffs
numbers = [0.50,0.51,0.52,0.53,0.54,0.55,0.56,0.57,0.58,0.59]
for i in numbers:
    y_train_sm_pred_final[i]= y_train_sm_pred_final.Converted_prob.map(lambda x: 1 if x
y_train_sm_pred_final.head()
```

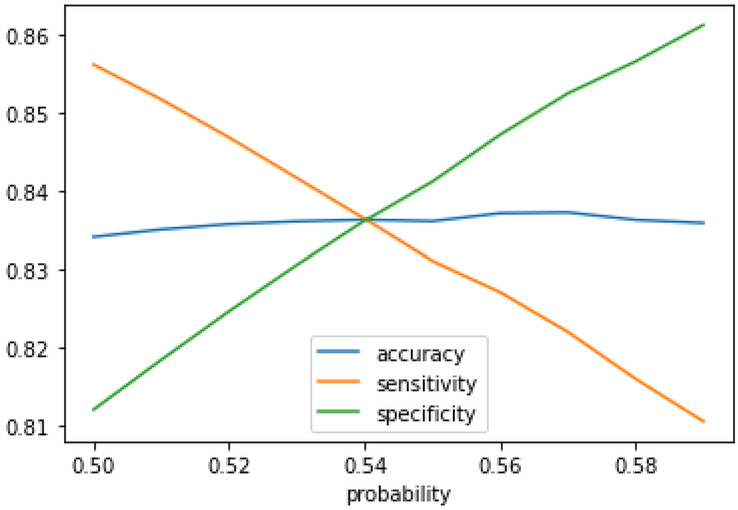
	Converted	Converted_prob	churn_pred	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.51	0.52	0.53	0.54
0	0	0.138574	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.401122	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
2	0	0.324276	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0

	Converted	Converted_prob	churn_pred	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.51	0.52	0.53	0.54
3	0	0.414619	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4	0	0.508730	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

```
# Now let's calculate accuracy sensitivity and specificity for various probability cutc
cutoff_df = pd.DataFrame( columns = ['probability','accuracy','sensitivity','specificit
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.50,0.51,0.52,0.53,0.54,0.55,0.56,0.57,0.58,0.59]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_sm_pred_final.Converted, y_train_sm_pred_fir
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    specificity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensitivity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensitivity,specificity]
print(cutoff_df)
```

```
      probability  accuracy  sensitivity  specificity
0.50  0.50         0.834042  0.856128     0.811956
0.51  0.51         0.835001  0.851669     0.818333
0.52  0.52         0.835675  0.846796     0.824554
0.53  0.53         0.836038  0.841611     0.830465
0.54  0.54         0.836245  0.836375     0.836116
0.55  0.55         0.836064  0.830983     0.841145
0.56  0.56         0.837075  0.826991     0.847159
0.57  0.57         0.837179  0.821910     0.852447
0.58  0.58         0.836219  0.815896     0.856543
0.59  0.59         0.835831  0.810452     0.861209
```

```
# plotting accuracy sensitivity and specificity for various probabilities calculated ab
cutoff_df.plot.line(x='probability', y=['accuracy','sensitivity','specificity'])
plt.show()
```



From the above graph we can conclude, the optimal cutoff point in the probability to define the predicted churn variabe converges at `0.54`

```
#### From the curve above, 0.2 is the optimum point to take it as a cutoff probability.

y_train_sm_pred_final['final_churn_pred'] = y_train_sm_pred_final.Converted_prob.map( l

y_train_sm_pred_final.head()
```

	Converted	Converted_prob	churn_pred	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.51	0.52	0.53	0.54
0	0	0.138574	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.401122	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
2	0	0.324276	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	0	0.414619	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4	0	0.508730	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

```
# Calculating the ovearall accuracy again
print("The overall accuracy of the model now is:",metrics.accuracy_score(y_train_sm_pre
```

```
The overall accuracy of the model now is: 0.8362453338863542
```

```
confusion2 = metrics.confusion_matrix(y_train_sm_pred_final.Converted, y_train_sm_pred_
print(confusion2)
```

```
[[16127  3161]
 [ 3156 16132]]
```

```
TP2 = confusion2[1,1] # true positive
TN2 = confusion2[0,0] # true negatives
FP2 = confusion2[0,1] # false positives
FN2 = confusion2[1,0] # false negatives

# Let's see the sensitivity of our logistic regression model
print("Sensitivity = ",TP2 / float(TP2+FN2))
```

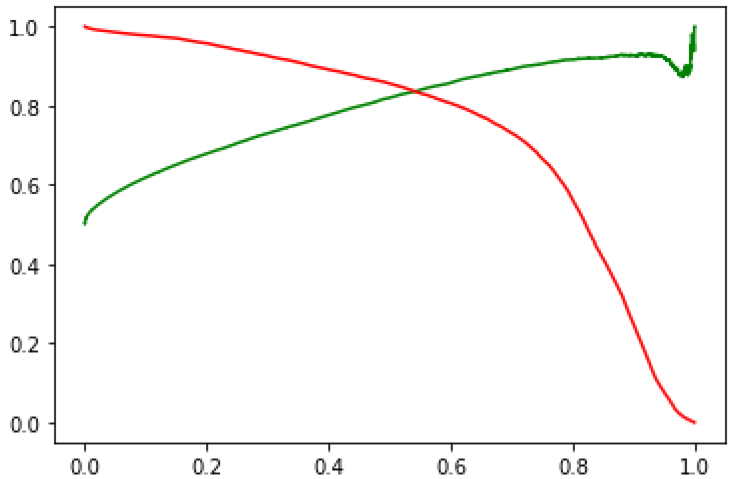
```
# Let us calculate specificity
print("Specificity = ",TN2 / float(TN2+FP2))

# Calculate false postive rate - predicting churn when customer does not have churned
print("False Positive Rate = ",FP2/ float(TN2+FP2))

# positive predictive value
print ("Precision = ",TP2 / float(TP2+FP2))

# Negative predictive value
print ("True Negative Prediction Rate = ",TN2 / float(TN2 + FN2))
```

```
Sensitivity =  0.8363749481542928
Specificity =  0.8361157196184156
False Positive Rate =  0.1638842803815844
Precision =  0.8361581920903954
True Negative Prediction Rate =  0.8363325208733081
```

Precision and recall tradeoff

```
from sklearn.metrics import precision_recall_curve
```

```
p, r, thresholds = precision_recall_curve(y_train_sm_pred_final.Converted, y_train_sm_p

# Plotting the curve
plt.plot(thresholds, p[:-1], "g-")
plt.plot(thresholds, r[:-1], "r-")
plt.show()
```



Making predictions on the test set

Transforming and feature selection for test data

```
# Scaling the test data
X_test[num_col] = scaler.transform(X_test[num_col])
X_test.head()
```

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_
35865	0.026143	0.021027	0.000000	0.000070	0.003412	0.000575	0.000000	0
41952	0.048190	0.005702	0.005250	0.002058	0.011146	0.023873	0.007510	0
98938	0.061230	0.003275	0.037889	0.008157	0.010851	0.025458	0.018789	0
29459	0.042998	0.020180	0.000963	0.000297	0.001588	0.003828	0.000573	0
70682	0.098384	0.005699	0.011111	0.039505	0.084425	0.242612	0.135335	0

```
# Feature selection
X_test=X_test[rfe_columns_2]
X_test.head()
```

	arpu_8	roam_ic_mou_7	roam_og_mou_8	loc_og_t2m_mou_8	std_og_mou_7	total_og_mou_8	loc_ic_mou_6
35865	0.026143	0.000000	0.000000	0.000000	0.000000	0.000053	0.003321
41952	0.048190	0.000000	0.000000	0.005379	0.018971	0.009067	0.023235
98938	0.061230	0.000000	0.000000	0.073716	0.000374	0.024987	0.057580
29459	0.042998	0.000000	0.000000	0.000000	0.000000	0.000800	0.001622
70682	0.098384	0.000721	0.031491	0.041749	0.172443	0.157573	0.021147

```
# Adding constant to the test model.
X_test_SM = sm.add_constant(X_test)
```

Predicting the target variable

```
y_test_pred = res.predict(X_test_SM)
print("\n The first ten probability value of the prediction are:\n",y_test_pred[:10])
```

```
35865    0.772260
41952    0.516558
98938    0.000325
29459    0.128443
70682    0.007754
58317    0.237200
4860     0.007990
16890    0.702931
61329    0.652452
94332    0.491091
dtype: float64
```

```
y_pred = pd.DataFrame(y_test_pred)
y_pred.head()
```

	0
35865	0.772260
41952	0.516558
98938	0.000325

	0
29459	0.128443
70682	0.007754

```
y_pred=y_pred.rename(columns = {0:"Conv_prob"})
```

```
y_test_df = pd.DataFrame(y_test)
y_test_df.head()
```

	churn
35865	0
41952	0
98938	0
29459	0
70682	0

```
y_pred_final = pd.concat([y_test_df,y_pred],axis=1)
y_pred_final.head()
```

	churn	Conv_prob
35865	0	0.772260
41952	0	0.516558
98938	0	0.000325
29459	0	0.128443
70682	0	0.007754

```
y_pred_final['test_churn_pred'] = y_pred_final.Conv_prob.map(lambda x: 1 if x>0.54 else
y_pred_final.head()
```

	churn	Conv_prob	test_churn_pred
35865	0	0.772260	1
41952	0	0.516558	0
98938	0	0.000325	0
29459	0	0.128443	0
70682	0	0.007754	0

```
# Checking the overall accuracy of the predicted set.
metrics.accuracy_score(y_pred_final.churn, y_pred_final.test_churn_pred)
```

```
0.8270192200866571
```

Metrics Evaluation

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

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

```
# Calculating model validation parameters
TP3 = confusion2_test[1,1] # true positive
TN3 = confusion2_test[0,0] # true negatives
FP3 = confusion2_test[0,1] # false positives
FN3 = confusion2_test[1,0] # false negatives
```

```
# 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
```

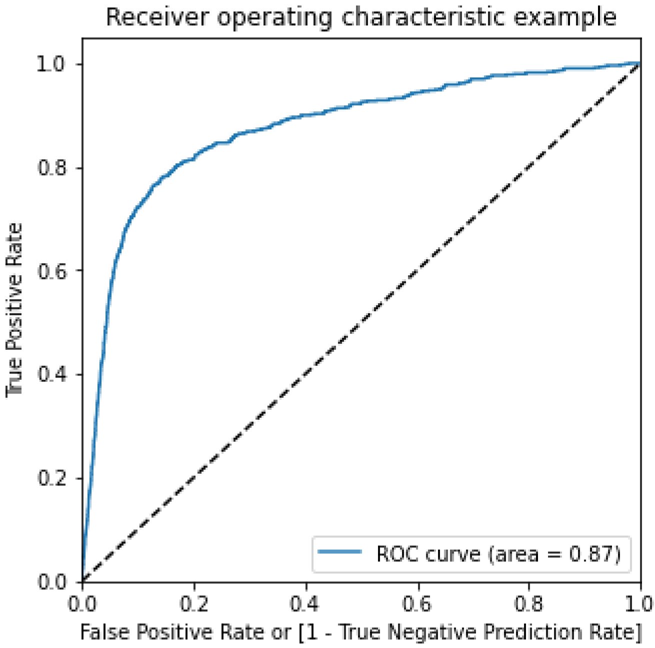
Explaining the results

```
print("The accuracy of the predicted model is: ",round(metrics.accuracy_score(y_pred_fi
print("The sensitivity of the predicted model is: ",round(TP3 / float(TP3+FN3),2)*100,"

print("\nAs the model created is based on a sentivity model, i.e. the True positive rat
```

```
The accuracy of the predicted model is:  83.0 %
The sensitivity of the predicted model is:  80.0 %


As the model created is based on a sentivity model, i.e. the True positive rate is
```

given more importance as the actual and prediction of churn by a customer

```
# ROC curve for the test dataset

# Defining the variables to plot the curve
fpr, tpr, thresholds = metrics.roc_curve(y_pred_final.churn,y_pred_final.Conv_prob, drc
# Plotting the curve for the obtained metrics
draw_roc(y_pred_final.churn,y_pred_final.Conv_prob)
```



The AUC score for train dataset is 0.90 and the test dataset is 0.87.
This model can be considered as a good model.

Logistic Regression using PCA

```
# split the dateset into train and test datasets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, train_size=0.7
print("Dimension of X_train:", X_train.shape)
print("Dimension of X_test:", X_test.shape)

# apply scaling on the dataset

scaler = MinMaxScaler()
X_train[num_col] = scaler.fit_transform(X_train[num_col])
X_test[num_col] = scaler.transform(X_test[num_col])

# Applying SMOTE technique for data imbalance correction

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)

X_train_sm.head()
```

```
Dimension of X_train: (21000, 126)
Dimension of X_test: (9001, 126)
Dimension of X_train_sm Shape: (38576, 126)
Dimension of y_train_sm Shape: (38576,)
```

	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6	r
0	0.038904	0.000235	0.000531	0.000238	0.004211	0.003651	0.004095	0.0	
1	0.032921	0.000493	0.000000	0.000000	0.001631	0.000000	0.000000	0.0	
2	0.033826	0.000876	0.000275	0.000714	0.003861	0.007485	0.003679	0.0	
3	0.081645	0.163879	0.105394	0.050406	0.142667	0.177782	0.052962	0.0	
4	0.042893	0.079633	0.051881	0.004868	0.058346	0.046732	0.010097	0.0	

```
# importing PCA
from sklearn.decomposition import PCA
pca = PCA(random_state=42)

# applying PCA on train data
pca.fit(X_train_sm)
```

```
PCA(random_state=42)
```

```
X_train_sm_pca=pca.fit_transform(X_train_sm)
print("Dimension of X_train_sm_pca: ",X_train_sm_pca.shape)

X_test_pca=pca.transform(X_test)
print("Dimension of X_test_pca: ",X_test_pca.shape)
```

```
Dimension of X_train_sm_pca:  (38576, 126)
Dimension of X_test_pca:  (9001, 126)
```

```
#Viewing the PCA components
pca.components_
```

```
array([[ 1.77080250e-02,  5.62945551e-03,  1.28071557e-02, ...,
        -8.33377373e-02,  2.03169293e-01, -2.25884463e-04],
       [ 1.17884332e-03,  1.36226801e-04,  2.66567649e-03, ...,
         6.62002105e-01, -7.17541378e-01,  1.93966990e-04],
       [ 8.31908962e-03, -2.32698646e-02, -1.53378013e-02, ...,
         7.54642802e-02,  5.50287343e-02,  1.26734621e-03],
       ...,
       [-3.94307290e-07,  1.32661563e-06, -2.21287988e-06, ...,
        -3.76725866e-08, -1.42403279e-08,  2.74517957e-08],
       [ 2.29473384e-07, -1.88640723e-06,  1.53383133e-06, ...,
        -3.64244933e-08, -2.71775061e-08, -3.24942343e-08],
```

```
       [-0.00000000e+00, -1.20429354e-16, -2.26455538e-17, ...,
         3.32681843e-18, -2.16312073e-18, -2.01305223e-17]])
```

Performing Logistic Regression

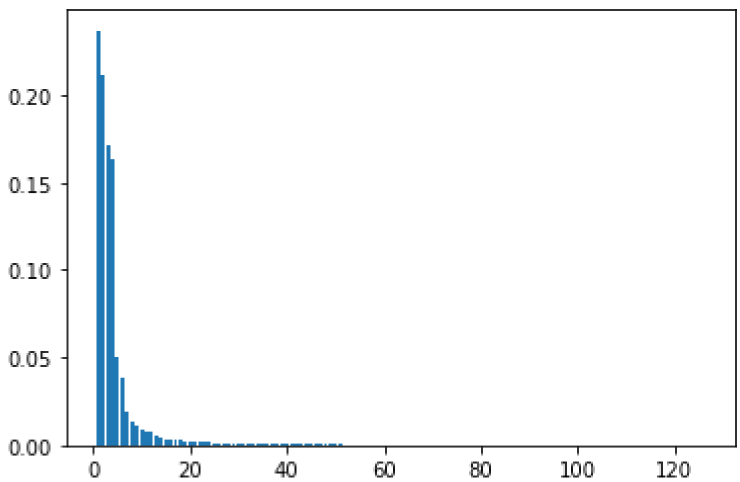
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg_pca = LogisticRegression()
logreg_pca.fit(X_train_sm_pca, y_train_sm)

# 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)
```

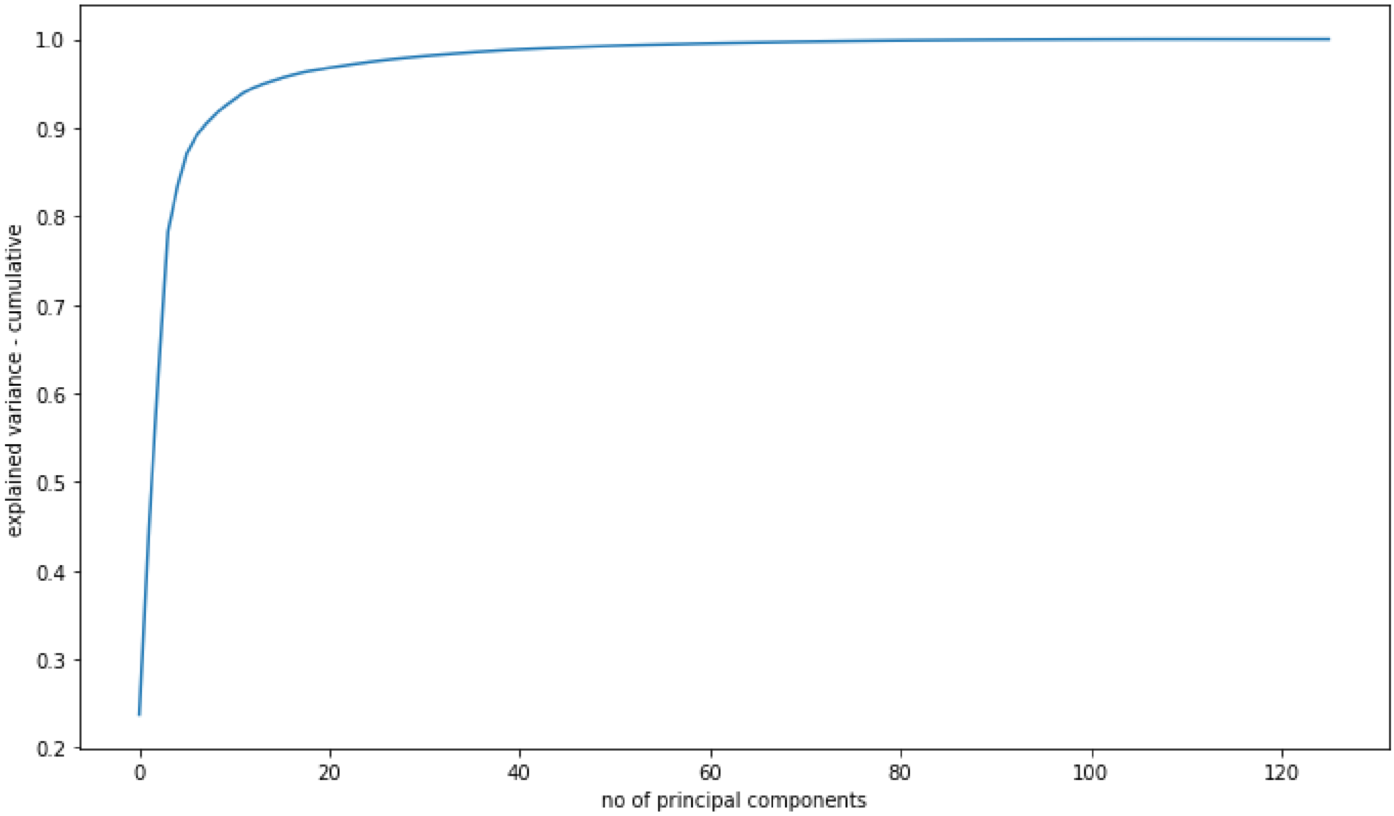
```
from sklearn.metrics import confusion_matrix, accuracy_score

# 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_pr
```

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

Accuracy of the logistic regression model with PCA:  0.818131318742362
```

```
plt.bar(range(1,len(pca.explained_variance_ratio_)+1),pca.explained_variance_ratio_)
plt.show()
```



```
var_cumu = np.cumsum(pca.explained_variance_ratio_)

# Making a scree plot
fig = plt.figure(figsize=[12,7])
plt.plot(var_cumu)
plt.xlabel('no of principal components')
plt.ylabel('explained variance - cumulative')
plt.show()
```




```
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
```

```
array([23.7, 44.8, 62. , 78.3, 83.3, 87.1, 89. , 90.4, 91.5, 92.4, 93.2,
       93.9, 94.5, 94.9, 95.3, 95.6, 95.9, 96.2, 96.4, 96.6, 96.8, 97. ,
       97.2, 97.4, 97.5, 97.6, 97.7, 97.8, 97.9, 98. , 98.1, 98.2, 98.3,
       98.4, 98.5, 98.6, 98.7, 98.8, 98.9, 99. , 99.1, 99.2, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3, 99.3,
       99.3, 99.3, 99.3, 99.3, 99.3])
```

**90% of the data can be explained with 8 PCA components*

Fitting the dataset with the 8 explainable components

```
pca_8 = PCA(n_components=15)

train_pca_8 = pca_8.fit_transform(X_train_sm)
print("Dimension for Train dataset using PCA: ", train_pca_8.shape)

test_pca_8 = pca_8.transform(X_test)
print("Dimension for Test dataset using PCA: ", test_pca_8.shape)
```

```
Dimension for Train dataset using PCA:  (38576, 15)
Dimension for Test dataset using PCA:  (9001, 15)
```

```
logreg_pca_8 = LogisticRegression()
logreg_pca_8.fit(train_pca_8, y_train_sm)

# making the predictions
y_pred_8 = logreg_pca_8.predict(test_pca_8)

# converting the prediction into a dataframe
y_pred_df_8 = pd.DataFrame(y_pred_8)
print("Dimension of y_pred_df_8: ", y_pred_df_8.shape)
```

```
Dimension of y_pred_df_8:  (9001, 1)
```

```
# Checking the Confusion matrix
print("Confusion Matirx for y_test & y_pred\n",confusion_matrix(y_test,y_pred_8),"\n")

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

```
Confusion Matirx for y_test & y_pred
 [[6250 2022]
 [ 185  544]]

Accuracy of the logistic regression model with PCA:  0.7548050216642596
```

```
# df_pca = pd.DataFrame(newdata, columns=["PC1", "PC2"])
# df.head()
```