```
#Load Required Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import regex

import warnings
warnings.filterwarnings("ignore")

warnings.filterwarnings("ignore")
pd.set_option("display.max_rows",300)
pd.set_option("display.max_columns",300)
```

```
# Get the dataset
telecom = pd.read_csv("telecom_churn_data.csv")
```

```
#check the data
telecom.head(5)
```

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last
0	7000842753	109	0.0	0.0	0.0	6/30/2014	
1	7001865778	109	0.0	0.0	0.0	6/30/2014	
2	7001625959	109	0.0	0.0	0.0	6/30/2014	
3	7001204172	109	0.0	0.0	0.0	6/30/2014	
4	7000142493	109	0.0	0.0	0.0	6/30/2014	

```
#this represents shape
telecom.shape
```

```
(99999, 226)
```

```
# Finding the dtypes of Columns to get some Insights
telecom.info(verbose=1)
```

```
 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
```

- Dataset contains 99999 no of rows.
- 226 no of columns.
- Number of Float data type - 179
- Number of int datatype - 35
- Number of object datatype- 12

 # Segregate Categorcial, ID and Numeric columns for ease of analysis

```
#Categorcial columns separation : categorical columns are only date here
date_columns = [col for col in telecom.columns if telecom[col].dtype =="object"]
print(f"Total Categorical columns:{len(date_columns)}")
```

```
Total Categorical columns:12
```

```
#ID columns separation
id_columns = ["mobile_number","circle_id"]  # total ID columns are 2
print(f"Total numeric columns:{len(id_columns)}")
```

```
Total numeric columns:2
```

```
#Numeric columns separation
numeric_columns = [ col for col in telecom.columns if col not in date_columns + id_columns]
print(f"Total numeric columns:{len(numeric_columns)}")
```

```
Total numeric columns:212
```

```
#check the date columns
telecom[date_columns].head()
```

	last_date_of_month_6	last_date_of_month_7	last_date_of_month_8	last_date_of_month_9	date_of_last_rech_6	date_of_last_rech_7
0	6/30/2014	7/31/2014	8/31/2014	9/30/2014	6/21/2014	7/16/2014
1	6/30/2014	7/31/2014	8/31/2014	9/30/2014	6/29/2014	7/31/2014
2	6/30/2014	7/31/2014	8/31/2014	9/30/2014	6/17/2014	7/24/2014
3	6/30/2014	7/31/2014	8/31/2014	9/30/2014	6/28/2014	7/31/2014
4	6/30/2014	7/31/2014	8/31/2014	9/30/2014	6/26/2014	7/28/2014

Missing value Treatment and Initial data analysis

```
#check the Null values column wise
(telecom.isnull().sum()/len(telecom)).sort_values(ascending = False)
```

```
sachet_3g_7             0.000000
monthly_2g_8            0.000000
monthly_3g_9            0.000000
monthly_3g_8            0.000000
sachet_3g_9             0.000000
monthly_3g_7            0.000000
monthly_3g_6            0.000000
sachet_2g_9             0.000000
sachet_2g_8             0.000000
sachet_2g_7             0.000000
sachet_2g_6             0.000000
monthly_2g_7            0.000000
monthly_2g_6            0.000000
mobile_number           0.000000
vol_3g_mb_8             0.000000
total_og_mou_9          0.000000
total_rech_num_7        0.000000
total_rech_num_6        0.000000
total_ic_mou_9          0.000000
total_ic_mou_8          0.000000
total_ic_mou_7          0.000000
total_ic_mou_6          0.000000
circle_id               0.000000
total_og_mou_8          0.000000
vol_3g_mb_7             0.000000
total_og_mou_7          0.000000
total_og_mou_6          0.000000
arpu_9                  0.000000
arpu_8                  0.000000
arpu_7                  0.000000
arpu_6                  0.000000
last_date_of_month_6    0.000000
total_rech_num_8        0.000000
total_rech_num_9        0.000000
total_rech_amt_6        0.000000
total_rech_amt_7        0.000000
vol_3g_mb_6             0.000000
vol_2g_mb_9             0.000000
vol_2g_mb_8             0.000000
vol_2g_mb_7             0.000000
vol_2g_mb_6             0.000000
last_day_rch_amt_9      0.000000
last_day_rch_amt_8      0.000000
last_day_rch_amt_7      0.000000
last_day_rch_amt_6      0.000000
max_rech_amt_9          0.000000
max_rech_amt_8          0.000000
max_rech_amt_7          0.000000
max_rech_amt_6          0.000000
total_rech_amt_9        0.000000
total_rech_amt_8        0.000000
sep_vbc_3g              0.000000
dtype: float64
```

- Many false null value columns are available. if customer did not recharge, the value assigned as NaN
- Hence we can not drop these values blindly.
- We can impute these columns as zero.

- When customer did not recharge, the total_rech_data_* and date_of_last_rech_data_* are null
  - Total Recharge data in month 6,7,8,9 would be null
  - Maximum Recharge Data in Month 6,7,8 ,9 would be null
  - Average Amount recharge Data in Month 6,7,8,9, would be null
- Hence this NULL can not be dropped out.
- We will impute with Zero.

```
# when total_rech_data and date_of_last_rech_data is null, Check date_of_last_rech_data,total_rech_data, max_rech_data etc.
telecom[telecom["total_rech_data_6"].isna() & telecom["date_of_last_rech_data_6"].isna()][ \
    ["date_of_last_rech_data_6","total_rech_data_6","max_rech_data_6","max_rech_data_6","av_rech_amt_data_6"]]
```

	date_of_last_rech_data_6	total_rech_data_6	max_rech_data_6	max_rech_data_6	av_rech_amt_data_6
1	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN

```
# Columns which we have to impute as Zero.
zero_impute = ['total_rech_data_6', 'total_rech_data_7', 'total_rech_data_8', 'total_rech_data_9',
        'av_rech_amt_data_6', 'av_rech_amt_data_7', 'av_rech_amt_data_8', 'av_rech_amt_data_9',
        'max_rech_data_6', 'max_rech_data_7', 'max_rech_data_8', 'max_rech_data_9'
       ]
# Put zero in these columns
telecom[zero_impute] = telecom[zero_impute].apply(lambda x: x.fillna(0))
```

Below columns are imputed with zeros.

- Total Recharge data in month 6,7,8,9
- Maximum Recharge Data in Month 6,7,8 ,9
- Average Amount recharge Data in Month 6,7,8,9

```
# We will drop date columns and ID columns as these will not contribute further to our analysis.

telecom.drop(columns=id_columns,inplace=True)
telecom.drop(columns=date_columns,inplace=True)
```

```
# Check the columns associated with month 6 . From this, we can get an overview of columns/features in 7,8,9 months
month_6_cols = [col for col in telecom.columns if "_6" in col]
print(len(month_6_cols))
month_6_cols
```

```
51
['arpu_6',
 'onnet_mou_6',
 'offnet_mou_6',
 'roam_ic_mou_6',
 'roam_og_mou_6',
 'loc_og_t2t_mou_6',
 'loc_og_t2m_mou_6',
 'loc_og_t2f_mou_6',
 'loc_og_t2c_mou_6',
 'loc_og_mou_6',
 'std_og_t2t_mou_6',
 'std_og_t2m_mou_6',
 'std_og_t2f_mou_6',
 'std_og_t2c_mou_6',
 'std_og_mou_6',
 'isd_og_mou_6',
 'spl_og_mou_6',
 'og_others_6',
 'total_og_mou_6',
 'loc_ic_t2t_mou_6',
 'loc_ic_t2m_mou_6',
 'loc_ic_t2f_mou_6',
 'loc_ic_mou_6',
 'std_ic_t2t_mou_6',
 'std_ic_t2m_mou_6',
 'std_ic_t2f_mou_6',
 'std_ic_t2o_mou_6',
 'std_ic_mou_6',
 'total_ic_mou_6',
 'spl_ic_mou_6',
 'isd_ic_mou_6',
 'ic_others_6',
 'total_rech_num_6',
 'total_rech_amt_6',
 'max_rech_amt_6',
 'last_day_rch_amt_6',
 'total_rech_data_6',
 'max_rech_data_6',
 'count_rech_2g_6',
 'count_rech_3g_6',
 'av_rech_amt_data_6',
 'vol_2g_mb_6',
 'vol_3g_mb_6',
 'arpu_3g_6',
 'arpu_2g_6',
 'night_pck_user_6',
 'monthly_2g_6',
 'sachet_2g_6',
```

```
 'monthly_3g_6',
 'sachet_3g_6',
 'fb_user_6']
```

```
# check how the data looks for month 6
telecom[month_6_cols].head(5)
```

	arpu_6	onnet_mou_6	offnet_mou_6	roam_ic_mou_6	roam_og_mou_6	loc_og_t2t_mou_6	loc_og_t2m_mou_6	loc_og_t2f_mou_6	loc_og_t2
0	197.385	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	34.047	24.11	15.74	0.0	0.0	23.88	11.51	0.00	
2	167.690	11.54	143.33	0.0	0.0	7.19	29.34	24.11	
3	221.338	99.91	123.31	0.0	0.0	73.68	107.43	1.91	
4	261.636	50.31	76.96	0.0	0.0	50.31	67.64	0.00	

```
# Check again the null values percentages
(telecom.isnull().sum()/len(telecom)).sort_values(ascending = False).head(50)
```

```
count_rech_2g_6      0.748467
night_pck_user_6     0.748467
fb_user_6            0.748467
arpu_2g_6            0.748467
arpu_3g_6            0.748467
count_rech_3g_6      0.748467
count_rech_2g_7      0.744287
count_rech_3g_7      0.744287
night_pck_user_7     0.744287
arpu_3g_7            0.744287
arpu_2g_7            0.744287
fb_user_7            0.744287
night_pck_user_9     0.740777
arpu_3g_9            0.740777
count_rech_3g_9      0.740777
fb_user_9            0.740777
arpu_2g_9            0.740777
count_rech_2g_9      0.740777
arpu_3g_8            0.736607
arpu_2g_8            0.736607
night_pck_user_8     0.736607
count_rech_2g_8      0.736607
fb_user_8            0.736607
count_rech_3g_8      0.736607
std_og_mou_9         0.077451
std_og_t2c_mou_9     0.077451
loc_ic_t2t_mou_9     0.077451
isd_og_mou_9         0.077451
std_og_t2f_mou_9     0.077451
og_others_9          0.077451
std_ic_t2t_mou_9     0.077451
loc_ic_t2m_mou_9     0.077451
loc_ic_t2f_mou_9     0.077451
loc_ic_mou_9         0.077451
std_ic_t2m_mou_9     0.077451
std_ic_t2o_mou_9     0.077451
std_ic_mou_9         0.077451
std_og_t2t_mou_9     0.077451
spl_ic_mou_9         0.077451
isd_ic_mou_9         0.077451
ic_others_9          0.077451
std_og_t2m_mou_9     0.077451
std_ic_t2f_mou_9     0.077451
spl_og_mou_9         0.077451
onnet_mou_9          0.077451
roam_og_mou_9        0.077451
loc_og_t2c_mou_9     0.077451
loc_og_t2f_mou_9     0.077451
loc_og_t2m_mou_9     0.077451
offnet_mou_9         0.077451
dtype: float64
```

Night pack user columns and FB User columns are categorical column

- night_pck_user_6
- night_pck_user_7
- night_pck_user_8
- night_pck_user_9
- fb_user_6

- fb_user_7
- fb_user_8
- fb_user_9

```
# Check night_pck_user unique values in month 6
telecom["night_pck_user_6"].unique()
```

```
array([ 0., nan,  1.])
```

```
#Check the percetages null values of these columns
categorical_columns = ["night_pck_user_6",
                       "night_pck_user_7",
                       "night_pck_user_8",
                       "night_pck_user_9","fb_user_6",
                       "fb_user_7",
                       "fb_user_8",
                       "fb_user_9"]

telecom[categorical_columns].isna().sum()/len(telecom)
```

```
night_pck_user_6    0.748467
night_pck_user_7    0.744287
night_pck_user_8    0.736607
night_pck_user_9    0.740777
fb_user_6           0.748467
fb_user_7           0.744287
fb_user_8           0.736607
fb_user_9           0.740777
dtype: float64
```

In the above columns,We can impute the NaN as -1, as a part to mark as missing value

```
#Fill NaN value as -1 to mark missing value
telecom[categorical_columns] = telecom[categorical_columns].fillna(-1)
```

```
# Check if the null value is filled with -1
telecom[categorical_columns].isna().sum()
```

```
night_pck_user_6    0
night_pck_user_7    0
night_pck_user_8    0
night_pck_user_9    0
fb_user_6           0
fb_user_7           0
fb_user_8           0
fb_user_9           0
dtype: int64
```

Hence there are no null values in night_pck_user and fb_user columns in month 6,7,8,9

```
#Check the null value pecentage
(telecom.isna().sum()/len(telecom)).sort_values(ascending=False)
```

```
last_day_rch_amt_7    0.000000
max_rech_amt_9        0.000000
vol_3g_mb_6           0.000000
max_rech_amt_8        0.000000
max_rech_amt_7        0.000000
max_rech_amt_6        0.000000
total_rech_amt_9      0.000000
total_rech_amt_8      0.000000
total_rech_amt_7      0.000000
total_rech_amt_6      0.000000
max_rech_data_6       0.000000
max_rech_data_7       0.000000
max_rech_data_8       0.000000
max_rech_data_9       0.000000
vol_2g_mb_9           0.000000
vol_2g_mb_8           0.000000
vol_2g_mb_7           0.000000
vol_2g_mb_6           0.000000
av_rech_amt_data_9    0.000000
av_rech_amt_data_8    0.000000
av_rech_amt_data_7    0.000000
av_rech_amt_data_6    0.000000
total_og_mou_9        0.000000
total_ic_mou_6        0.000000
total_ic_mou_7        0.000000
total_ic_mou_8        0.000000
total_ic_mou_9        0.000000
total_rech_num_6      0.000000
total_rech_num_7      0.000000
sep_vbc_3g            0.000000
dtype: float64
```

```
# Many columns have more than 70% null values
#Function to drop columns where there are more than 40% null values
def columns_tobe_dropped(cols):
    '''cols: list of columns in dataframe
      '''
    for col in cols:
        if (telecom[col].isna().sum()/len(telecom)) > .40:   # Check the condition if null values GT .40
            telecom.drop(columns=[col],inplace=True)
```

```
# drop colums
columns_tobe_dropped(telecom.columns)
```

```
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Columns: 196 entries, loc_og_t2o_mou to sep_vbc_3g
dtypes: float64(163), int64(33)
memory usage: 149.5 MB
```

- We have removed 30 columns from the dataframe

```
# check the null value row wise.
telecom.isna().sum(axis=1).sort_values(ascending = False).head(30)
```

```
51296    119
47936    119
48177    119
48376    119
48474    119
48582    119
48651    119
48707    119
48740    119
48839    119
49153    119
49211    119
49582    119
49594    119
49651    119
49772    119
49857    119
49903    119
49909    119
49981    119
50006    119
48138    119
47786    119
45426    119
47739    119
45836    119
46033    119
46295    119
```

```
46515    119
46694    119
dtype: int64
```

- We have many rows having multiple null values. We are not dropping these and will fill these gradually

```
# check the null value again
(telecom.isna().sum()/len(telecom)).sort_values(ascending = False)
```

```
av_rech_amt_data_9    0.000000
monthly_3g_6          0.000000
jun_vbc_3g            0.000000
jul_vbc_3g            0.000000
aug_vbc_3g            0.000000
aon                   0.000000
fb_user_9             0.000000
fb_user_8             0.000000
fb_user_7             0.000000
fb_user_6             0.000000
sachet_3g_9           0.000000
sachet_3g_8           0.000000
sachet_3g_7           0.000000
sachet_3g_6           0.000000
monthly_3g_9          0.000000
monthly_3g_8          0.000000
monthly_3g_7          0.000000
vol_2g_mb_6           0.000000
av_rech_amt_data_7    0.000000
av_rech_amt_data_8    0.000000
arpu_7                0.000000
total_rech_amt_7      0.000000
total_rech_amt_6      0.000000
total_rech_num_9      0.000000
total_rech_num_8      0.000000
total_rech_num_7      0.000000
total_rech_num_6      0.000000
arpu_6                0.000000
arpu_8                0.000000
total_ic_mou_9        0.000000
arpu_9                0.000000
total_og_mou_6        0.000000
total_og_mou_7        0.000000
total_og_mou_8        0.000000
total_og_mou_9        0.000000
total_ic_mou_6        0.000000
total_ic_mou_7        0.000000
total_rech_amt_8      0.000000
total_rech_amt_9      0.000000
max_rech_amt_6        0.000000
max_rech_amt_7        0.000000
av_rech_amt_data_6    0.000000
max_rech_data_9       0.000000
max_rech_data_8       0.000000
max_rech_data_7       0.000000
max_rech_data_6       0.000000
total_rech_data_9     0.000000
total_rech_data_8     0.000000
total_rech_data_7     0.000000
total_rech_data_6     0.000000
last_day_rch_amt_9    0.000000
last_day_rch_amt_8    0.000000
last_day_rch_amt_7    0.000000
last_day_rch_amt_6    0.000000
max_rech_amt_9        0.000000
max_rech_amt_8        0.000000
sep_vbc_3g            0.000000
dtype: float64
```

```
#check columns which have only 1 value.

# Create a DataFrame of no. of unique values and filter where only one value is available.
zero_variance_columns = pd.DataFrame(telecom.nunique()).reset_index().rename(columns = {'index': 'feature', 0: 'nunique'})
print(zero_variance_columns[zero_variance_columns['nunique'] == 1])
```

```
              feature  nunique
0      loc_og_t2o_mou        1
1      std_og_t2o_mou        1
2      loc_ic_t2o_mou        1
55   std_og_t2c_mou_6        1
56   std_og_t2c_mou_7        1
57   std_og_t2c_mou_8        1
58   std_og_t2c_mou_9        1
107  std_ic_t2o_mou_6        1
108  std_ic_t2o_mou_7        1
109  std_ic_t2o_mou_8        1
110  std_ic_t2o_mou_9        1
```

- The above columns have just one Unique value.
- Hence they have zero variance and can be dropped.

```
# create columns list whihc have zero variance i:e 1 unique value.
columns_tobe_dropped = list(zero_variance_columns[zero_variance_columns['nunique'] == 1]["feature"])
columns_tobe_dropped
```

```
['loc_og_t2o_mou',
 'std_og_t2o_mou',
 'loc_ic_t2o_mou',
 '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']
```

```
# drop columns whish are having 1 unique values
telecom.drop(columns=columns_tobe_dropped,inplace=True)
```

```
#check the shape
telecom.shape
```

```
(99999, 185)
```

```
# Check the null values again
(telecom.isna().sum()/len(telecom)).sort_values(ascending=False).reset_index()
```

	index	0
0	loc_ic_mou_9	0.077451
1	loc_og_mou_9	0.077451
2	std_ic_t2t_mou_9	0.077451
3	loc_og_t2m_mou_9	0.077451
4	loc_ic_t2t_mou_9	0.077451
5	spl_ic_mou_9	0.077451
6	std_ic_t2f_mou_9	0.077451
7	loc_og_t2f_mou_9	0.077451
8	loc_og_t2c_mou_9	0.077451
9	std_ic_t2m_mou_9	0.077451
10	loc_ic_t2m_mou_9	0.077451
11	std_og_t2t_mou_9	0.077451
12	og_others_9	0.077451
13	std_ic_mou_9	0.077451
14	std_og_t2m_mou_9	0.077451
15	std_og_t2f_mou_9	0.077451
16	std_og_mou_9	0.077451
17	spl_og_mou_9	0.077451
18	loc_og_t2t_mou_9	0.077451
19	isd_og_mou_9	0.077451
20	roam_og_mou_9	0.077451
21	roam_ic_mou_9	0.077451
22	loc_ic_t2f_mou_9	0.077451
23	onnet_mou_9	0.077451
24	offnet_mou_9	0.077451
25	isd_ic_mou_9	0.077451
26	ic_others_9	0.077451
27	std_og_mou_8	0.053781
28	std_ic_t2f_mou_8	0.053781
29	ic_others_8	0.053781
30	loc_ic_mou_8	0.053781
31	loc_ic_t2m_mou_8	0.053781
32	std_og_t2f_mou_8	0.053781
33	std_ic_mou_8	0.053781
34	og_others_8	0.053781
35	std_og_t2m_mou_8	0.053781
36	onnet_mou_8	0.053781
37	spl_og_mou_8	0.053781
38	isd_og_mou_8	0.053781
39	std_og_t2t_mou_8	0.053781
40	loc_ic_t2f_mou_8	0.053781
41	std_ic_t2t_mou_8	0.053781
42	loc_ic_t2t_mou_8	0.053781
43	loc_og_t2t_mou_8	0.053781
44	roam_og_mou_8	0.053781
45	isd_ic_mou_8	0.053781
46	loc_og_t2m_mou_8	0.053781
47	roam_ic_mou_8	0.053781
48	offnet_mou_8	0.053781

- Still we have null values in 107 columns.
- Majority of the null values are in Minitue of Usage columns
- As these values are not available, so we are imputing those values as 0 instead of iteratively imputing.

52 std_ic_t2m_mou_8 0.053781

```
# Fill hr NaN as zero.
telecom = telecom.fillna(0)
```

54 loc_ic_mou_6 0.039370

```
telecom.isna().sum()
```

```
max_rech_amt_6        0
max_rech_amt_7        0
max_rech_amt_8        0
max_rech_amt_9        0
last_day_rch_amt_6    0
last_day_rch_amt_7    0
last_day_rch_amt_8    0
last_day_rch_amt_9    0
total_rech_data_6     0
total_rech_data_7     0
total_rech_data_8     0
total_rech_data_9     0
max_rech_data_6       0
max_rech_data_7       0
max_rech_data_8       0
max_rech_data_9       0
av_rech_amt_data_6    0
av_rech_amt_data_7    0
av_rech_amt_data_8    0
av_rech_amt_data_9    0
vol_2g_mb_6           0
vol_2g_mb_7           0
vol_2g_mb_8           0
vol_2g_mb_9           0
vol_3g_mb_6           0
vol_3g_mb_7           0
vol_3g_mb_8           0
vol_3g_mb_9           0
night_pck_user_6      0
night_pck_user_7      0
night_pck_user_8      0
night_pck_user_9      0
monthly_2g_6          0
monthly_2g_7          0
monthly_2g_8          0
monthly_2g_9          0
sachet_2g_6           0
sachet_2g_7           0
sachet_2g_8           0
sachet_2g_9           0
monthly_3g_6          0
monthly_3g_7          0
monthly_3g_8          0
monthly_3g_9          0
sachet_3g_6           0
sachet_3g_7           0
sachet_3g_8           0
sachet_3g_9           0
fb_user_6             0
fb_user_7             0
fb_user_8             0
fb_user_9             0
aon                   0
aug_vbc_3g            0
jul_vbc_3g            0
jun_vbc_3g            0
sep_vbc_3g            0
dtype: int64
```

```
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Columns: 185 entries, arpu_6 to sep_vbc_3g
dtypes: float64(152), int64(33)
memory usage: 141.1 MB
```

- Now there is no null values in the data.
- And still we have 99999 rows of data.
- No. of columns reduced from 226 to 185.

Filter High Value Customer

- We need to predict churn only for the high-value customers.
- Those who have recharged with an amount more than or equal to X, where X is the 70th percentile of the average recharge amount in the first two months (the good phase).

101 std_og_t2m_mou_7 0.038590

Create derive columns to filter high value customer

```
#Calculate total Data recharge amount--> Total Data Recharge * Average Amount of Data recharge
telecom["total_data_recharge_amnt_6"] = telecom.total_rech_data_6 * telecom.av_rech_amt_data_6
telecom["total_data_recharge_amnt_7"] = telecom.total_rech_data_7 * telecom.av_rech_amt_data_7
```

```
#Calculate Total Amount recharge --> total talktime recharge + total data recharge
telecom["total_recharge_amnt_6"] = telecom.total_rech_amt_6 + telecom.total_data_recharge_amnt_6
telecom["total_recharge_amnt_7"] = telecom.total_rech_amt_7 + telecom.total_data_recharge_amnt_7
```

```
#Calculate Average amount of recharge of 6th and 7th month
telecom['average_amnt_6_7'] = (telecom["total_recharge_amnt_6"] + telecom["total_recharge_amnt_7"])/2
```

```
# Check the 70th percentile of "average_amnt_6_7"
telecom['average_amnt_6_7'].quantile(.70)
```

```
478.0
```

- 70th percentile of average amount recharge in 6th and 7th month comes as 478.0.
- Now we need to filter the data based on this value.

```
#filter based on 70th percentile
telecom_highvalue = telecom[telecom["average_amnt_6_7"]>= telecom["average_amnt_6_7"].quantile(.70)]
```

119 vol_3g_mb_7 0.000000

```
#Delete the derived columns created in above step
telecom_highvalue.drop(columns=["total_data_recharge_amnt_6","total_data_recharge_amnt_7","total_recharge_amnt_6",\
                                "total_recharge_amnt_7","average_amnt_6_7"],inplace=True)
```

```
telecom_highvalue.shape
```

```
(30001, 185)
```

Finally we have 30001 rows of high value customer data with 185 columns

```
# check the data
telecom_highvalue.head()
```

	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_mou_8	onnet_mou_9	offnet_mou_6	offnet_mou_7	offnet_mou_
0	197.385	214.816	213.803	21.100	0.00	0.00	0.00	0.00	0.00	0.00	0.0
7	1069.180	1349.850	3171.480	500.000	57.84	54.68	52.29	0.00	453.43	567.16	325.9
8	378.721	492.223	137.362	166.787	413.69	351.03	35.08	33.46	94.66	80.63	136.4
21	514.453	597.753	637.760	578.596	102.41	132.11	85.14	161.63	757.93	896.68	983.3
23	74.350	193.897	366.966	811.480	48.96	50.66	33.58	15.74	85.41	89.36	205.8

Tag churners and remove attributes of the churn phase

- Now we need to tag the churned customers (churn=1, else 0) based on the fourth month as follows:
- Those who have not made any calls (either incoming or outgoing) and have not used mobile internet even once in the churn phase.
- Based on these below attributes we need to decide churners
  - total_ic_mou_9
  - total_og_mou_9
  - vol_2g_mb_9
  - vol_3g_mb_9

```
#Calculate total call in minus by adding Incoming and Outgoing calls
telecom_highvalue['total_calls_9'] = telecom_highvalue.total_ic_mou_9 + telecom_highvalue.total_og_mou_9
```

```
# Calculate total 2G and 3G consumption of data
telecom_highvalue["total_data_consumptions"] = telecom_highvalue.vol_2g_mb_9 + telecom_highvalue.vol_3g_mb_9
```

150 total rech num 9 0.000000

- Now we need to create Churn variable.
- Customer who have not used any calls or have not consumed any data on month of 9 are tagged as Churn customer.
- Churn customer is marked as 1
- non-churn custoner is marked as 0

154 total_ic_mou_9 0.000000

```
#Tag 1 as churner  where total_calls_9=0 and total_data_consumptions=0
# else 0 as non-churner
telecom_highvalue["churn"]=telecom_highvalue.apply(lambda row:1 if (row.total_calls_9==0 and row.total_data_consumptions==0) else 0,axis=
```

157 av rech amt data 6 0.000000

```
#check the percentages of churn and non churn data
telecom_highvalue["churn"].value_counts(normalize=True)
```

```
0    0.918636
1    0.081364
Name: churn, dtype: float64
```

162 total_og_mou_6 0.000000

- The data is imbalance.
- Churn percentage is close 8 and non-churn percentage is close to 92.

```
#Drop the derived columns
telecom_highvalue.drop(columns=["total_calls_9","total_data_consumptions"],inplace=True)
```

167 total rech amt 9 0.000000

Delete columns belong to the 9th month : Churn Month

- After tagging churners, remove all the attributes corresponding to the churn phase (all attributes having ' _9', etc. in their names.
- These columns contain data for users, where these users are already churned.
- Hence those will not contribute anything to churn prediction.

```
# drop all 9th month columns
telecom_highvalue = telecom_highvalue.filter(regex='[^9]$',axis=1)
```

```
# check the baisc info about high value customer
telecom_highvalue.info(verbose=1)
```

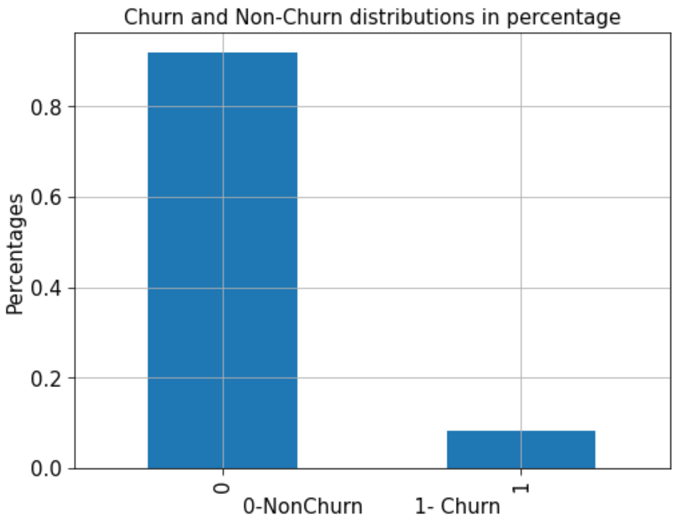
```
 125  sachet_2g_8         int64
 126  monthly_3g_6        int64
 127  monthly_3g_7        int64
 128  monthly_3g_8        int64
 129  sachet_3g_6         int64
 130  sachet_3g_7         int64
 131  sachet_3g_8         int64
 132  fb_user_6           float64
 133  fb_user_7           float64
 134  fb_user_8           float64
 135  aon                 int64
 136  aug_vbc_3g          float64
 137  jul_vbc_3g          float64
 138  jun_vbc_3g          float64
 139  sep_vbc_3g          float64
 140  churn               int64
dtypes: float64(115), int64(26)
memory usage: 32.5 MB
```

Exploratory Data Analysis

```
# Check the percenatges of churn and non-churn customers
telecom_highvalue["churn"].value_counts(normalize=True)
```

```
0    0.918636
1    0.081364
Name: churn, dtype: float64
```

```
# plot to Check percetanges of churn and non churn data
plt.figure(figsize=(8,6))
telecom_highvalue["churn"].value_counts(normalize=True).plot.bar()
plt.tick_params(size=5,labelsize = 15)
plt.title("Churn and Non-Churn distributions in percentage",fontsize=15)
plt.ylabel("Percentages",fontsize=15)
plt.xlabel("0-NonChurn        1- Churn",fontsize=15)
plt.grid(0.3)
plt.show()
```



We have almost 92% customers belong non-churn and 8% customers belong to Churn type

```
# check basic statistics
telecom_highvalue.describe()
```

	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8
count	30001.000000	30001.000000	30001.000000	30001.000000	30001.000000	30001.000000	30001.000000	30001.000000	30001.000000

```
#check columns associated with month 6, From month 6 we can figure out how the columns and data are in other months
cols_6 = [col for col in telecom_highvalue.columns if "_6" in col]
cols_6
```

```
['arpu_6',
 'onnet_mou_6',
 'offnet_mou_6',
 'roam_ic_mou_6',
 'roam_og_mou_6',
 'loc_og_t2t_mou_6',
 'loc_og_t2m_mou_6',
 'loc_og_t2f_mou_6',
 'loc_og_t2c_mou_6',
 'loc_og_mou_6',
 'std_og_t2t_mou_6',
 'std_og_t2m_mou_6',
 'std_og_t2f_mou_6',
 'std_og_mou_6',
 'isd_og_mou_6',
 'spl_og_mou_6',
 'og_others_6',
 'total_og_mou_6',
 'loc_ic_t2t_mou_6',
 'loc_ic_t2m_mou_6',
 'loc_ic_t2f_mou_6',
 'loc_ic_mou_6',
 'std_ic_t2t_mou_6',
 'std_ic_t2m_mou_6',
 'std_ic_t2f_mou_6',
 'std_ic_mou_6',
 'total_ic_mou_6',
 'spl_ic_mou_6',
 'isd_ic_mou_6',
 'ic_others_6',
 'total_rech_num_6',
 'total_rech_amt_6',
 'max_rech_amt_6',
 'last_day_rch_amt_6',
 'total_rech_data_6',
 'max_rech_data_6',
 'av_rech_amt_data_6',
 'vol_2g_mb_6',
 'vol_3g_mb_6',
 'night_pck_user_6',
 'monthly_2g_6',
 'sachet_2g_6',
 'monthly_3g_6',
 'sachet_3g_6',
 'fb_user_6']
```

Derive new faetures by comparing month 8 features vs month 6 and month 7 features

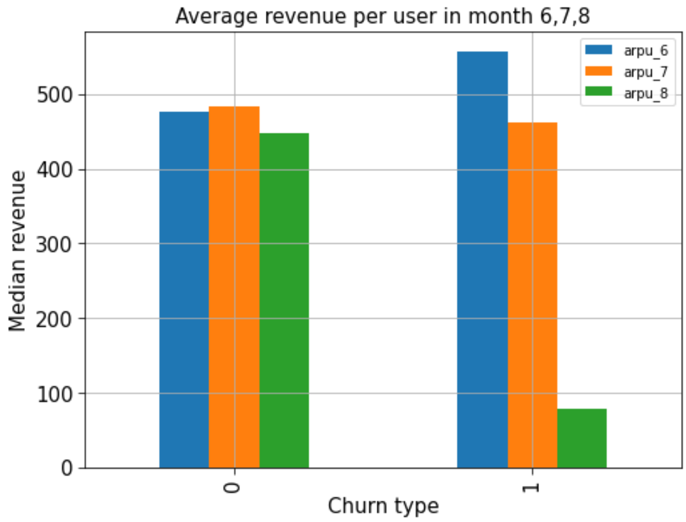
```
#compare average revenue and calculate the difference
telecom_highvalue['arpu_diff'] = telecom_highvalue.arpu_8 - ((telecom_highvalue.arpu_6 + telecom_highvalue.arpu_7)/2)
```

```
# Check various columns related to Minutes of Usage and calculate difference
telecom_highvalue['onnet_mou_diff'] = telecom_highvalue.onnet_mou_8 - ((telecom_highvalue.onnet_mou_6 + telecom_highvalue.onnet_mou_7)/2)
telecom_highvalue['offnet_mou_diff'] = telecom_highvalue.offnet_mou_8 - ((telecom_highvalue.offnet_mou_6 + telecom_highvalue.offnet_mou_7
telecom_highvalue['roam_ic_mou_diff'] = telecom_highvalue.roam_ic_mou_8 - ((telecom_highvalue.roam_ic_mou_6 + telecom_highvalue.roam_ic_m
telecom_highvalue['roam_og_mou_diff'] = telecom_highvalue.roam_og_mou_8 - ((telecom_highvalue.roam_og_mou_6 + telecom_highvalue.roam_og_m
telecom_highvalue['loc_og_mou_diff'] = telecom_highvalue.loc_og_mou_8 - ((telecom_highvalue.loc_og_mou_6 + telecom_highvalue.loc_og_mou_7
telecom_highvalue['std_og_mou_diff'] = telecom_highvalue.std_og_mou_8 - ((telecom_highvalue.std_og_mou_6 + telecom_highvalue.std_og_mou_7
telecom_highvalue['isd_og_mou_diff'] = telecom_highvalue.isd_og_mou_8 - ((telecom_highvalue.isd_og_mou_6 + telecom_highvalue.isd_og_mou_7
telecom_highvalue['spl_og_mou_diff'] = telecom_highvalue.spl_og_mou_8 - ((telecom_highvalue.spl_og_mou_6 + telecom_highvalue.spl_og_mou_7
telecom_highvalue['total_og_mou_diff'] = telecom_highvalue.total_og_mou_8 - ((telecom_highvalue.total_og_mou_6 + telecom_highvalue.total_
telecom_highvalue['loc_ic_mou_diff'] = telecom_highvalue.loc_ic_mou_8 - ((telecom_highvalue.loc_ic_mou_6 + telecom_highvalue.loc_ic_mou_7
telecom_highvalue['std_ic_mou_diff'] = telecom_highvalue.std_ic_mou_8 - ((telecom_highvalue.std_ic_mou_6 + telecom_highvalue.std_ic_mou_7
telecom_highvalue['isd_ic_mou_diff'] = telecom_highvalue.isd_ic_mou_8 - ((telecom_highvalue.isd_ic_mou_6 + telecom_highvalue.isd_ic_mou_7
telecom_highvalue['spl_ic_mou_diff'] = telecom_highvalue.spl_ic_mou_8 - ((telecom_highvalue.spl_ic_mou_6 + telecom_highvalue.spl_ic_mou_7
telecom_highvalue['total_ic_mou_diff'] = telecom_highvalue.total_ic_mou_8 - ((telecom_highvalue.total_ic_mou_6 + telecom_highvalue.total_
```

```
# Check total Recharge number
telecom_highvalue['total_rech_num_diff'] = telecom_highvalue.total_rech_num_8 - ((telecom_highvalue.total_rech_num_6 + telecom_highvalue.
#check total recharge amount
telecom_highvalue['total_rech_amt_diff'] = telecom_highvalue.total_rech_amt_8 - ((telecom_highvalue.total_rech_amt_6 + telecom_highvalue.
#Check maximum recharge amount
telecom_highvalue['max_rech_amt_diff'] = telecom_highvalue.max_rech_amt_8 - ((telecom_highvalue.max_rech_amt_6 + telecom_highvalue.max_re
#check total recharge data
```

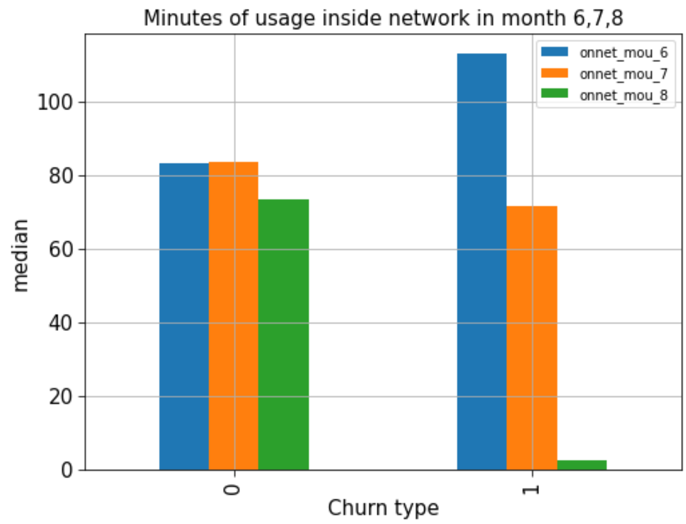
```
telecom_highvalue['total_rech_data_diff'] = telecom_highvalue.total_rech_data_8 - ((telecom_highvalue.total_rech_data_6 + telecom_highval
#check maximum recharge data
telecom_highvalue['max_rech_data_diff'] = telecom_highvalue.max_rech_data_8 - ((telecom_highvalue.max_rech_data_6 + telecom_highvalue.max
#Check average recharge amount in Data
telecom_highvalue['av_rech_amt_data_diff'] = telecom_highvalue.av_rech_amt_data_8 - ((telecom_highvalue.av_rech_amt_data_6 + telecom_high
#check 2G data consumption difference in MB
telecom_highvalue['vol_2g_mb_diff'] = telecom_highvalue.vol_2g_mb_8 - ((telecom_highvalue.vol_2g_mb_6 + telecom_highvalue.vol_2g_mb_7)/2)
#Check 3G data consumption in MB
telecom_highvalue['vol_3g_mb_diff'] = telecom_highvalue.vol_3g_mb_8 - ((telecom_highvalue.vol_3g_mb_6 + telecom_highvalue.vol_3g_mb_7)/2)
```

```
# Plot to visualize average revenue per user(ARPU)
telecom_highvalue.groupby("churn")["arpu_6","arpu_7","arpu_8"].median().plot.bar(figsize=[8,6])
plt.title("Average revenue per user in month 6,7,8",fontsize=15)
plt.tick_params(size=5,labelsize = 15)
plt.ylabel("Median revenue",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



*** Average revenue per user more in month 6 means, if they are unsatisfied, those useres are more likely to churn**

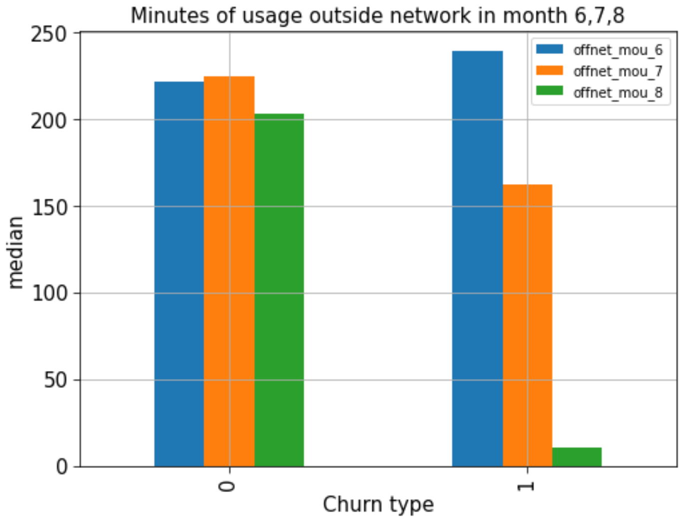
```
## Plot to visualize onnet_mou
telecom_highvalue.groupby("churn")["onnet_mou_6","onnet_mou_7","onnet_mou_8" ].median().plot.bar(figsize=[8,6])
plt.tick_params(size=5,labelsize = 15)
plt.title("Minutes of usage inside network in month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



- Users whose minutes of usage are more in month 6, they are more likely to churn.

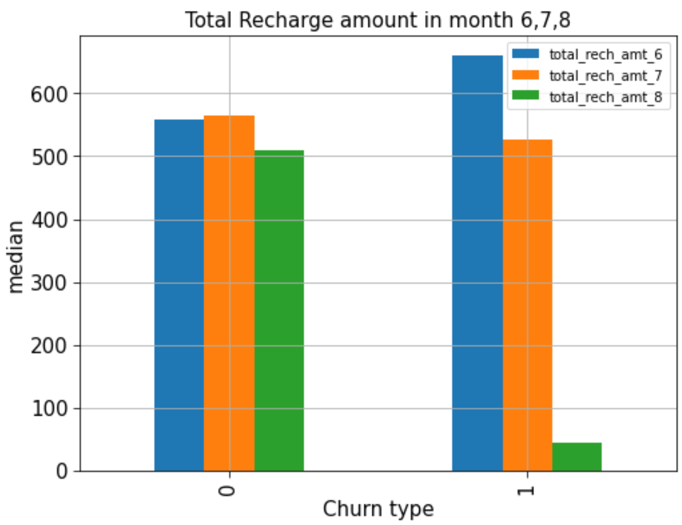
```
# Plot to visualize  offnet_mou
telecom_highvalue.groupby("churn")["offnet_mou_6","offnet_mou_7","offnet_mou_8" ].median().plot.bar(figsize=[8,6])
plt.tick_params(size=5,labelsize = 15)
```

```
plt.tick_params(size=5,labelsize = 15)
plt.title("Minutes of usage outside network in month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



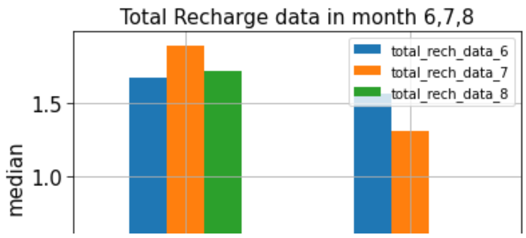
->The users who have big difference of minutes of call duration to other network between month 6 and month 7,are likely to churn

```
# Plot to visualize total_rech_amt
telecom_highvalue.groupby("churn")["total_rech_amt_6","total_rech_amt_7","total_rech_amt_8" ].median().plot.bar(figsize=[8,6])
plt.tick_params(size=5,labelsize = 15)
plt.title("Total Recharge amount in month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



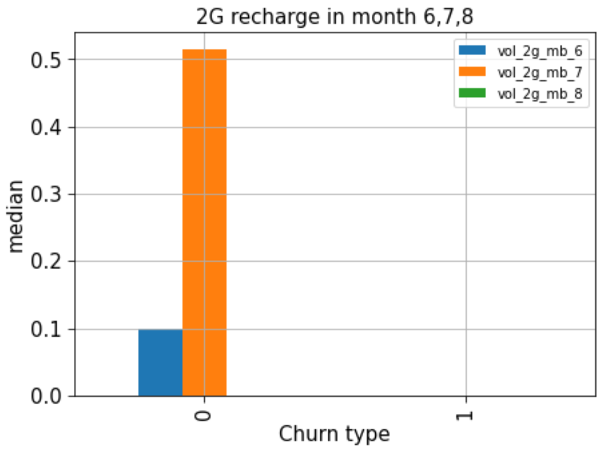
-> when the difference of total recharge amount is more, those users are more likely to churn.

```
# Plot to visualize total_rech_data_
telecom_highvalue.groupby("churn")["total_rech_data_6","total_rech_data_7","total_rech_data_8" ].mean().plot.bar()
plt.tick_params(size=5,labelsize = 15)
plt.title("Total Recharge data in month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



Users who have not recharge in month 6, 7, 8 may or may not churn, we do not have much evidence from data

```
## Plot to visualize vol_2g_mb_6
telecom_highvalue.groupby("churn")["vol_2g_mb_6","vol_2g_mb_7","vol_2g_mb_8" ].median().plot.bar(figsize=[7,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("2G recharge in month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



2g recharge who have not done may or may not churn, There is no concrete evidence from data

```
#Check the percenatges of churn in each category of Night Pack Users in month 8
pd.crosstab(telecom_highvalue.churn, telecom_highvalue.night_pck_user_8, normalize='columns')*100
```

night_pck_user_8	-1.0	0.0	1.0
churn			
0	85.89123	97.117602	97.360704
1	14.10877	2.882398	2.639296

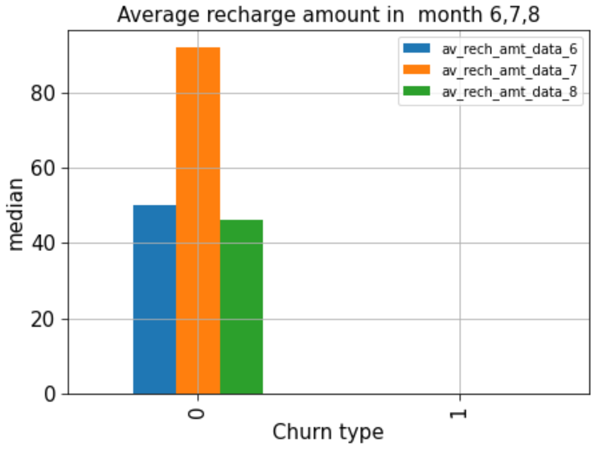
```
#Check the percenatges of churn in each category of Facebook Users in month 6
(pd.crosstab(telecom_highvalue.churn, telecom_highvalue.fb_user_8, normalize='columns')*100)
```

fb_user_8	-1.0	0.0	1.0
churn			
0	85.89123	93.231707	97.568644
1	14.10877	6.768293	2.431356

- Night pack users(which we do not know whether using or not) in month 8 , high churn rate: close to 14%
- Among Facebook users in month 8, close to 2% churns
- Customers who are not using facebook, close to 7% churns in month 8

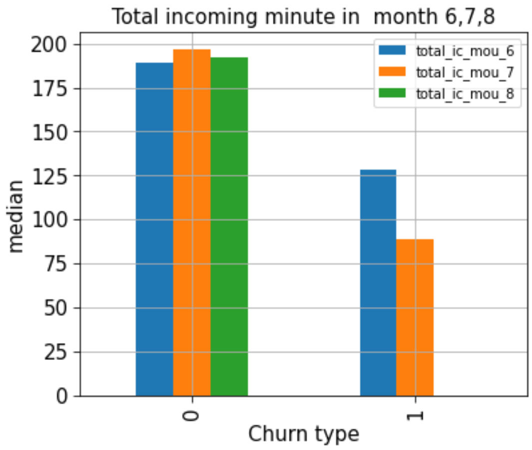
```
# plot to visualize av_rech_amt_data
telecom_highvalue.groupby("churn")["av_rech_amt_data_6","av_rech_amt_data_7","av_rech_amt_data_8" ].median().plot.\
bar(figsize=[7,5])

plt.tick_params(size=5,labelsize = 15)
plt.title("Average recharge amount in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
```

```
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



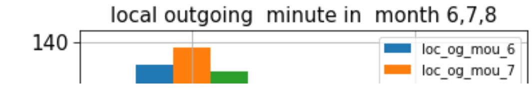
Average recharge amount in month 6,7,8 is none, from dataset, they are more likely to churn

```
#Plot to visualize total_ic_mou
telecom_highvalue.groupby("churn")["total_ic_mou_6","total_ic_mou_7","total_ic_mou_8"].median().plot.bar(figsize=[6,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("Total incoming minute in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



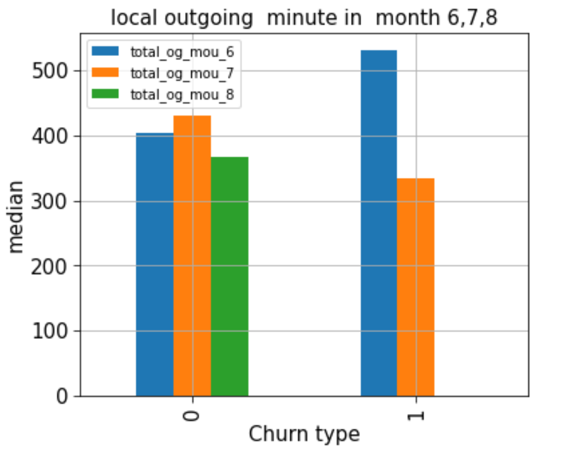
Users who have more difference in Total incoming minutes in month 6,7,8 are more likely to churn

```
#plot to visualize loc_og_mou
telecom_highvalue.groupby("churn")["loc_og_mou_6","loc_og_mou_7","loc_og_mou_8"].median().plot.bar(figsize=[6,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("local outgoing  minute in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



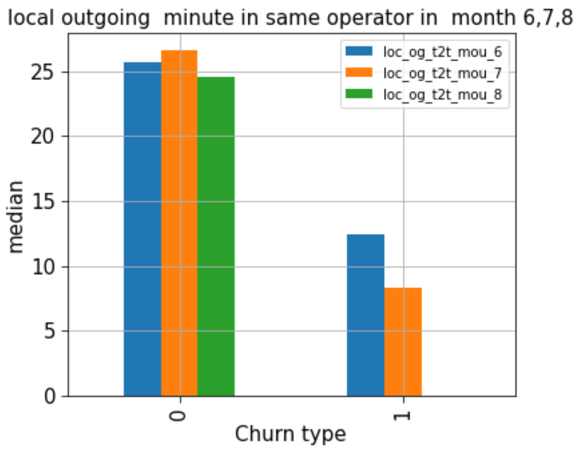
- local outgoing minute are less, users are more likely to churn

```
# total_og_mou_6
telecom_highvalue.groupby("churn")["total_og_mou_6","total_og_mou_7","total_og_mou_8"].median().plot.bar(figsize=[6,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("local outgoing  minute in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



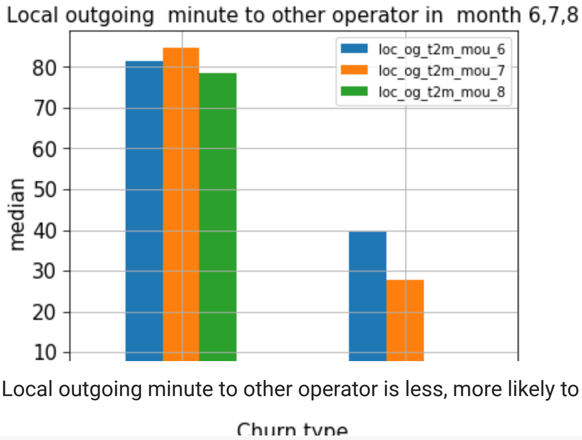
- Total outgoing minute usage difference is more between month 6 and 7, users are mor likely to chrun

```
# loc_og_t2t_mou_6
telecom_highvalue.groupby("churn")["loc_og_t2t_mou_6","loc_og_t2t_mou_7","loc_og_t2t_mou_8"].median().plot.bar(figsize=[6,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("local outgoing  minute in same operator in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



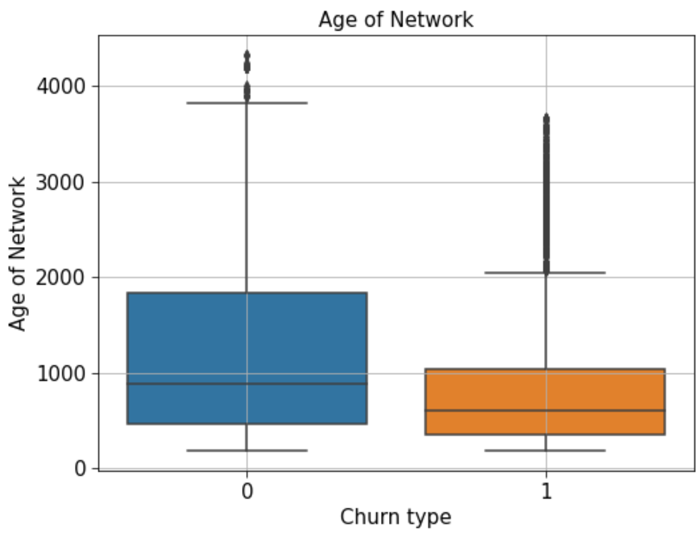
- Local outgoing minute in same operator in month 6,7,8 are less, users are more likely to churn.

```
telecom_highvalue.groupby("churn")["loc_og_t2m_mou_6","loc_og_t2m_mou_7","loc_og_t2m_mou_8"].median().plot.bar(figsize=[6,5])
plt.tick_params(size=5,labelsize = 15)
plt.title("Local outgoing  minute to other operator in  month 6,7,8",fontsize=15)
plt.ylabel("median",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.grid(0.3)
plt.show()
```



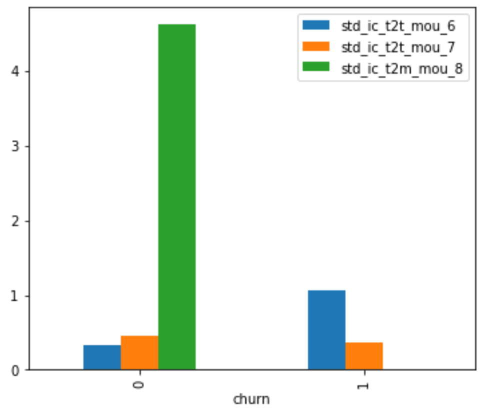
- Local outgoing minute to other operator is less, more likely to churn

```
#Network
plt.figure(figsize=[8,6])
sns.boxplot(data=telecom_highvalue,x="churn",y="aon")
plt.tick_params(size=5,labelsize = 15)
plt.title("Age of Network",fontsize=15)
plt.xlabel("Churn type",fontsize=15)
plt.ylabel("Age of Network",fontsize=15)
plt.grid(0.3)
plt.show()
```



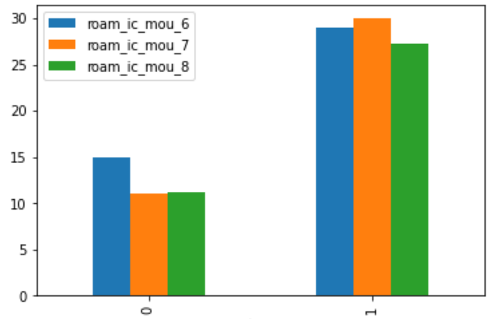
- Median Age of network less, more likely to churn

```
telecom_highvalue.groupby("churn")["std_ic_t2t_mou_6","std_ic_t2t_mou_7","std_ic_t2m_mou_8"].median().plot.bar(figsize=[6,5])
plt.show()
```



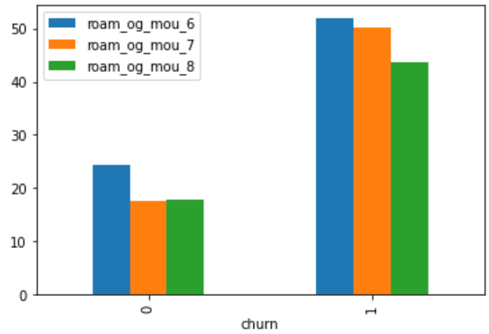
- Users who are using more STD calls are more likely to churn.

```
telecom_highvalue.groupby("churn")["roam_ic_mou_6","roam_ic_mou_7","roam_ic_mou_8"].mean().plot.bar()
plt.show()
```



- Roaming in incoming minutes more, they are likely to churn more.

```
telecom_highvalue.groupby("churn")["roam_og_mou_6","roam_og_mou_7","roam_og_mou_8"].mean().plot.bar()
plt.show()
```



- roaming in outgoing minutes more, Users are more likely to churn.

```
telecom_highvalue.head()
```

	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6	roa
0	197.385	214.816	213.803	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
7	1069.180	1349.850	3171.480	57.84	54.68	52.29	453.43	567.16	325.91	16.23	
8	378.721	492.223	137.362	413.69	351.03	35.08	94.66	80.63	136.48	0.00	
21	514.453	597.753	637.760	102.41	132.11	85.14	757.93	896.68	983.39	0.00	
23	74.350	193.897	366.966	48.96	50.66	33.58	85.41	89.36	205.89	0.00	

Model Building

```
#Load required library
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
```

Train test split of data

```
#Peform the train test split
train,test = train_test_split(telecom_highvalue,test_size=0.2,random_state=48)
```

```
# check the training and testing data shape
print(f"train data shape:{train.shape}")
print(f"Test data shape:{test.shape}")
```

```
train data shape:(24000, 164)
Test data shape:(6001, 164)
```

```
#Convert categorical data to numeric columns by aggregation.
categorical_columns = ["night_pck_user_6","night_pck_user_7",
                       "night_pck_user_8","fb_user_6",
                       "fb_user_8","fb_user_7"]
```

```
train[categorical_columns].head()
```

	night_pck_user_6	night_pck_user_7	night_pck_user_8	fb_user_6	fb_user_8	fb_user_7
33114	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
4101	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
40361	0.0	0.0	0.0	1.0	1.0	1.0
11213	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
14484	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0

```
#Calculate categorical features mean and replace those with categorical value
print(train.groupby('night_pck_user_6')["churn"].mean())
print(train.groupby('night_pck_user_7')["churn"].mean())
print(train.groupby('night_pck_user_8')["churn"].mean())
print(train.groupby('fb_user_6')["churn"].mean())
print(train.groupby('fb_user_7')["churn"].mean())
print(train.groupby('fb_user_8')["churn"].mean())
```

```
night_pck_user_6
-1.0    0.099621
 0.0    0.066717
 1.0    0.098462
Name: churn, dtype: float64
night_pck_user_7
-1.0    0.116741
 0.0    0.054784
 1.0    0.058020
Name: churn, dtype: float64
night_pck_user_8
-1.0    0.141980
 0.0    0.028647
 1.0    0.019084
Name: churn, dtype: float64
fb_user_6
-1.0    0.099621
 0.0    0.083333
 1.0    0.066233
Name: churn, dtype: float64
fb_user_7
-1.0    0.116741
 0.0    0.065279
 1.0    0.053977
Name: churn, dtype: float64
fb_user_8
-1.0    0.141980
 0.0    0.067373
 1.0    0.023955
Name: churn, dtype: float64
```

Need to perform based on complete data

```
#Map each categorical value with mean value
mapping = {'night_pck_user_6' : {-1: 0.099621, 0: 0.066717, 1: 0.098462},
           'night_pck_user_7' : {-1: 0.116741, 0: 0.054784, 1: 0.058020},
           'night_pck_user_8' : {-1: 0.141980, 0: 0.028647, 1: 0.019084},
           'fb_user_6'        : {-1: 0.099621, 0: 0.083333, 1: 0.066233},
           'fb_user_7'        : {-1: 0.116741, 0: 0.065279, 1: 0.053977},
           'fb_user_8'        : {-1: 0.141980, 0: 0.067373, 1: 0.023955}}

#convert categorical to Numeric features by aggregation and replace in train data
train.replace(mapping, inplace = True)
#replace the same in test data
test.replace(mapping, inplace = True)
```

```
# segregate  X_train and y_train
y_train = train.pop("churn")
X_train = train
```

```
# Segregate X_test and y_test
y_test = test.pop("churn")
X_test = test
```

Perform Oversampling with SMOTE

```
* As we have imbalance data set, we will oversample only the training set data
```

```
# If imblearn is not install in your system, install using
# !pip install imblearn
```

```
# Perform oversampling with traing data and pass both X_train and y_train to SMOTE
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=48)
X_train_resample,y_train_resample = smote.fit_resample(X_train,y_train)
```

```
# Check the shape after Oversampling
print(f"Shape of train data after oversampling: {X_train_resample.shape}")
print(f"Value count of training target variable:\n{y_train_resample.value_counts()}")
```

```
Shape of train data after oversampling: (44082, 163)
Value count of training target variable:
1    22041
0    22041
Name: churn, dtype: int64
```

Now the non-churn and churn data is balanced.

Scaling

- We need to perform the scaling to feed the scaled data to PCA
- We are using minmax scaling

```
# Import library and perform scaling
from sklearn.preprocessing import MinMaxScaler,StandardScaler
scale = MinMaxScaler()
temp_x_train = scale.fit_transform(X_train_resample)

#Form the dataframe after scaling
X_train_scale = pd.DataFrame(temp_x_train,columns=X_train.columns)
# Check the shape of scaled data
X_train_scale.shape
```

```
(44082, 163)
```

```
# check the scaled train data head
X_train_scale.head()
```

	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6	roam
0	0.088949	0.079537	0.035792	0.012317	0.033941	0.010586	0.113348	0.222906	0.043536	0.0	
1	0.091309	0.072997	0.044006	0.068476	0.113913	0.105521	0.029152	0.036250	0.016693	0.0	
2	0.078872	0.071509	0.059257	0.000655	0.005340	0.010668	0.008990	0.069703	0.051038	0.0	
3	0.092193	0.064149	0.038217	0.001225	0.001190	0.004448	0.083064	0.034989	0.040130	0.0	
4	0.091403	0.067002	0.036085	0.018706	0.009548	0.013353	0.046217	0.029105	0.022605	0.0	

```
# Perform the scaling on test set
temp_x_test = scale.transform(X_test)
# form the test set dataframe after scaling
X_test_scale = pd.DataFrame(temp_x_test,columns=X_test.columns)
```

```
# check the scaled test data head
X_test_scale.head()
```

	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet_mou_7	onnet_mou_8	offnet_mou_6	offnet_mou_7	offnet_mou_8	roam_ic_mou_6	roam_
0	0.090964	0.075823	0.042129	0.008711	0.010712	0.026989	0.148318	0.286030	0.107864	0.000000	
1	0.091021	0.069884	0.037593	0.000381	0.001873	0.001295	0.039266	0.039306	0.027770	0.027926	
2	0.092684	0.072383	0.052856	0.021061	0.011957	0.025669	0.041658	0.054902	0.035280	0.000000	
4	0.084756	0.055982	0.027394	0.000000	0.000000	0.000000	0.009412	0.002223	0.000000	0.000000	

- Use X_train_scale and X_test_scale in PCA

PCA

- We have almost 140 features to train the model
- To remove collinearity and faster training we can perform dimensionality reduction technique PCA.

```
# Load the library
from sklearn.decomposition import PCA
pc_class = PCA(random_state=60)
X_train_pca = pc_class.fit(X_train_scale)
```

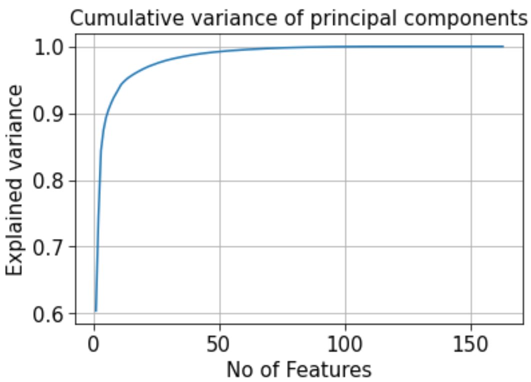
```
# Check the explained_variance_ratio_ whihc tells us individual principal component variance.
X_train_pca.explained_variance_ratio_
```

```
array([6.03828617e-01, 1.40973413e-01, 9.91155206e-02, 3.07631658e-02,
       1.86234669e-02, 1.18416165e-02, 9.48517523e-03, 8.28397598e-03,
       6.72748174e-03, 6.66714826e-03, 6.48765005e-03, 4.03245946e-03,
       3.54239498e-03, 3.02625956e-03, 2.62601134e-03, 2.36736548e-03,
       2.22188138e-03, 2.11335787e-03, 2.04027551e-03, 1.85954948e-03,
       1.80479200e-03, 1.69169327e-03, 1.44245222e-03, 1.39627736e-03,
       1.38652402e-03, 1.32326350e-03, 1.20520271e-03, 1.16539268e-03,
       1.09180753e-03, 9.87446077e-04, 9.32626988e-04, 8.50296303e-04,
       8.22568575e-04, 8.09298469e-04, 7.93824971e-04, 7.32863836e-04,
       7.15982444e-04, 6.64938130e-04, 6.26470521e-04, 6.16430322e-04,
       5.83678714e-04, 5.55821255e-04, 5.11635457e-04, 4.75816882e-04,
       4.69430576e-04, 4.39348363e-04, 4.29142579e-04, 4.02705885e-04,
       3.62016817e-04, 3.53569223e-04, 3.44779423e-04, 3.28864327e-04,
       3.16730775e-04, 3.00533635e-04, 2.84082281e-04, 2.80172818e-04,
       2.67941661e-04, 2.49376878e-04, 2.45090226e-04, 2.39718600e-04,
       2.36020524e-04, 2.23137116e-04, 2.13475123e-04, 2.06288140e-04,
       2.04234389e-04, 1.86475052e-04, 1.85448944e-04, 1.83841747e-04,
       1.78237485e-04, 1.73120864e-04, 1.66886531e-04, 1.56023037e-04,
       1.47194714e-04, 1.36069987e-04, 1.29581166e-04, 1.27037603e-04,
       1.24456177e-04, 1.16628357e-04, 1.08673978e-04, 1.07240266e-04,
       9.29987514e-05, 9.10470624e-05, 8.78679297e-05, 8.51975928e-05,
       8.34410713e-05, 8.18590604e-05, 7.90754286e-05, 7.83113520e-05,
       7.42033077e-05, 7.26272764e-05, 6.61710852e-05, 6.28257541e-05,
       5.57391381e-05, 5.26393136e-05, 4.65259006e-05, 4.29468127e-05,
       4.19308301e-05, 3.50162659e-05, 3.21562865e-05, 3.14906387e-05,
       2.68407413e-05, 2.63149202e-05, 2.58889358e-05, 2.52070867e-05,
       2.11631837e-05, 2.04077442e-05, 1.88961189e-05, 1.62875797e-05,
       1.60788263e-05, 1.54661107e-05, 1.38307963e-05, 1.11231003e-05,
       6.42800252e-06, 5.64039104e-06, 5.48031307e-06, 3.45802509e-06,
       3.25406489e-06, 2.69060807e-06, 2.55494293e-06, 6.75562441e-07,
       3.96168311e-07, 2.63648152e-07, 1.16641112e-07, 5.81244119e-10,
       1.07269167e-11, 8.63968223e-13, 4.16026605e-13, 3.34367701e-13,
       2.68365974e-13, 2.36409214e-13, 2.07744509e-13, 1.97390141e-13,
       1.83146481e-13, 1.70096986e-13, 1.39491133e-13, 1.31906181e-13,
       1.25002555e-13, 1.12894502e-13, 6.81036656e-14, 6.47805070e-14,
       5.07939199e-14, 3.26151205e-14, 1.89692338e-14, 1.09014154e-32,
       4.62998599e-33, 4.62998599e-33, 4.62998599e-33, 4.62998599e-33,
       4.62998599e-33, 4.62998599e-33, 4.62998599e-33, 4.62998599e-33,
       4.62998599e-33, 4.62998599e-33, 4.62998599e-33, 4.62998599e-33,
       4.62998599e-33, 4.62998599e-33, 4.62998599e-33, 4.62998599e-33,
       4.62998599e-33, 4.62998599e-33, 2.02869985e-34])
```

```
# perform the cumulaltive sum of explained variance
var_cumu = np.cumsum(X_train_pca.explained_variance_ratio_)
#Convert explained variance to DataFrame
var_cumu_df = pd.DataFrame({"variance":var_cumu})
var_cumu_df.head(30)
```

	variance
0	0.603829
1	0.744802
2	0.843918
3	0.874681
4	0.893304
5	0.905146
6	0.914631
7	0.922915
8	0.929642
9	0.936310
10	0.942797
11	0.946830
12	0.950372
13	0.953398
14	0.956024
15	0.958392
16	0.960614
17	0.962727
18	0.964767
19	0.966627
20	0.968432
21	0.970123
22	0.971566
23	0.972962

```
# Plot the cumulative explained variance : SCREE Plot
plt.figure(figsize=[6,4])
plt.plot(range(1,len(var_cumu)+1), var_cumu)
plt.title("Cumulative variance of principal components",size=15)
plt.ylabel("Explained variance",size=15)
plt.xlabel("No of Features",size=15)
plt.tick_params(size=5,labelsize = 15) # Tick size in both X and Y axes
plt.grid(0.3)
```




```
# By providing variance value we can also get the suitable principal components.
pca_demo = PCA(0.96,random_state=40)
X_train_pca1 = pca_demo.fit_transform(X_train_scale)
print(f"suitable principal components for 96% of variance:{X_train_pca1.shape[1]}")
```

```
suitable principal components for 96% of variance:17
```

- Now we got suitable no of principal components as 17
- Hence we will do PCA again with 17 components for train and test set

```
# Instantiate PCA with 17 components
pca_object = PCA(n_components=17,random_state=48)
# get the PCs for train data
X_train_pca_final = pca_object.fit_transform(X_train_scale)
# get the PCs for test data
X_test_pca_final = pca_object.fit_transform(X_test_scale)

#check the shape of train and test data after PCA
print(X_train_pca_final.shape)
print(X_test_pca_final.shape)
```

```
(44082, 17)
(6001, 17)
```

```
# Check the correlations after PCA
np.corrcoef(X_train_pca_final.transpose())
```

```
        1.28394432e-16,  1.00000000e+00,  3.73792894e-16,
        2.42026664e-16, -1.15354415e-16,  2.19458969e-17,
       -2.03797615e-16,  1.54501708e-16,  1.49057820e-16,
        5.10971505e-17, -1.61476714e-16],
      [ 2.12951429e-18, -1.47958057e-17,  5.94443920e-18,
        1.13720206e-17, -7.24655461e-17, -4.87335083e-16,
       -4.65680359e-16,  3.73792894e-16,  1.00000000e+00,
        1.22127340e-16,  6.93985346e-17, -1.71709564e-16,
       -8.93668402e-17,  2.19790636e-16,  6.59312321e-17,
       -6.36872479e-17, -7.64873973e-17],
      [ 5.33254023e-17, -1.28466628e-17,  1.06697258e-17,
        1.48362638e-17, -7.38077261e-17,  5.83505151e-17,
       -7.92040504e-16,  2.42026664e-16,  1.22127340e-16,
        1.00000000e+00, -7.86177480e-18,  3.67099761e-16,
        8.16244609e-17, -3.90866621e-16,  1.46384159e-16,
       -2.09519886e-17, -8.22982232e-17],
      [-1.32563592e-17, -1.00245009e-17, -2.68516391e-18,
       -3.84580661e-18, -2.86645420e-18,  8.19190266e-17,
        5.56651887e-17, -1.15354415e-16,  6.93985346e-17,
       -7.86177480e-18,  1.00000000e+00,  3.47497823e-18,
       -5.23566070e-16, -5.73728721e-17,  6.34197647e-17,
        5.77071981e-17, -1.07615895e-17],
      [ 1.68146059e-17,  8.37377276e-18,  5.25346224e-18,
       -4.03938551e-18,  5.75678211e-18, -4.61811448e-18,
        1.21161415e-17,  2.19458969e-17, -1.71709564e-16,
        3.67099761e-16,  3.47497823e-18,  1.00000000e+00,
       -6.04234356e-16,  1.07457129e-16, -6.98908697e-16,
       -1.05295576e-16,  1.59342375e-16],
      [ 2.68313945e-17, -2.01731848e-17,  1.29347874e-19,
       -2.64098553e-18,  1.49200359e-18, -4.05402795e-17,
       -2.11432516e-16, -2.03797615e-16, -8.93668402e-17,
        8.16244609e-17, -5.23566070e-16, -6.04234356e-16,
        1.00000000e+00,  1.37690075e-16,  2.14707684e-16,
       -2.58959180e-16,  1.99170746e-16],
      [ 4.96119593e-18, -1.55678408e-17,  9.31815374e-18,
       -2.09333344e-18, -4.95041358e-18,  8.61051444e-17,
        1.29609585e-16,  1.54501708e-16,  2.19790636e-16,
       -3.90866621e-16, -5.73728721e-17,  1.07457129e-16,
        1.37690075e-16,  1.00000000e+00, -8.77582700e-16,
       -3.06165029e-16, -1.21689443e-17],
      [ 6.41140986e-18, -5.71075601e-18, -4.50707702e-19,
        1.32811377e-17,  2.24127305e-17, -1.30394813e-17,
        7.84119664e-17,  1.49057820e-16,  6.59312321e-17,
        1.46384159e-16,  6.34197647e-17, -6.98908697e-16,
        2.14707684e-16, -8.77582700e-16,  1.00000000e+00,
        8.57339218e-17,  7.44203859e-16],
      [-7.52391503e-18,  1.54830851e-19,  1.03537315e-18,
       -2.11177388e-17,  6.57236325e-18,  1.29715434e-17,
        1.03050961e-16,  5.10971505e-17, -6.36872479e-17,
       -2.09519886e-17,  5.77071981e-17, -1.05295576e-16,
       -2.58959180e-16, -3.06165029e-16,  8.57339218e-17,
        1.00000000e+00,  2.19681559e-16],
      [ 1.43135347e-17,  1.50859987e-17,  6.81162470e-18,
       -1.34070672e-17,  2.28441023e-17, -7.09799601e-18,
       -1.54563039e-16, -1.61476714e-16, -7.64873973e-17,
       -8.22982232e-17, -1.07615895e-17,  1.59342375e-16,
        1.99170746e-16, -1.21689443e-17,  7.44203859e-16,
        2.19681559e-16,  1.00000000e+00]])
```

- The correlation values are almost close to 0(power raised to -17,-18,-19) except the diagonal.

Model Building:

- We will explore below models.
  - Logistic regression
  - Decision tree

- Randomforest
- Gradientboosting
- XGboost

```
#Function definition to check the performance of model on test data
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV
# Check the performance on test set
#Precision
#recall
#f1_score
#ROC_AUC
def calculate_peformance_testdata(model_name,y_test,y_pred,pred_prob):

    '''y_test:Test Labels,
       y_pred: Prediction Labels ,
       pred_prob:Predicted Probability  '''

    print(f"{model_name}:")
    precision = metrics.precision_score(y_test,y_pred)
    print(f"precision: {precision}")
    recall = metrics.recall_score(y_test,y_pred)
    print(f"recall: {recall}")
    f1_score = metrics.f1_score(y_test,y_pred)
    print(f"f1_score: {f1_score}")
    roc_auc = metrics.roc_auc_score(y_test,pred_prob)
    print(f"roc_auc: {roc_auc}")
#     return a DataFrame with all the score
    return pd.DataFrame({"Model":[model_name],"precision":[precision],"recall":[recall],"f1_score":[f1_score],
                     "roc_auc":[roc_auc]})
```

```
# Create a DataFrame which stores all test score for each model
score_df = pd.DataFrame({"Model":[None],"precision":[None],"recall":[None],"f1_score":[None],"roc_auc":[None]})
```

Logistic regression

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

#Instantiate logistic regression
lr_obj = LogisticRegression(random_state=40)
#pass PCA data as input
lr_obj.fit(X_train_pca_final, y_train_resample)
cv_score = cross_val_score(lr_obj, X_train_pca_final, y_train_resample, cv=5, scoring='f1_micro')
print(f"Cross validation score: {cv_score}")
```

```
Cross validation score: [0.82862652 0.84450493 0.84029038 0.83824864 0.83938294]
```

```
#Prediction on  pca testdata
y_pred_lr = lr_obj.predict(X_test_pca_final)
#check predict probability on pca data
pred_prob = lr_obj.predict_proba(X_test_pca_final)
```

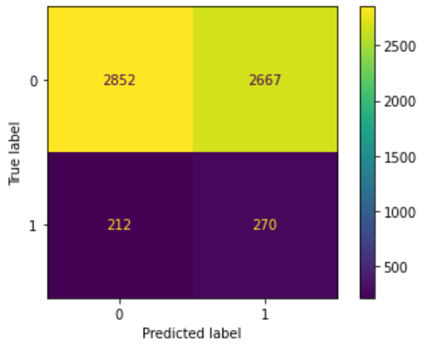
```
#check various scores on test data
df1 = calculate_peformance_testdata("LogisticRegression",y_test,y_pred_lr,pred_prob[:,1])
```

```
LogisticRegression:
precision: 0.09193054136874361
recall: 0.5601659751037344
f1_score: 0.15794091839719218
roc_auc: 0.575711292336771
```

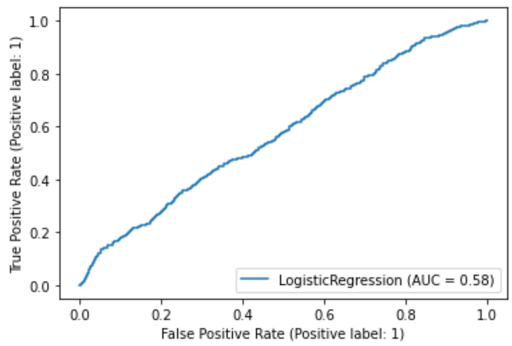
```
#Add the score to dataframe for comparision with other model performance
score_df= score_df.dropna()
score_df = score_df.append(df1)
score_df
```

	Model	precision	recall	f1_score	roc_auc
0	LogisticRegression	0.091931	0.560166	0.157941	0.575711

```
#Plot confusion matrix for Logistic Regression
metrics.plot_confusion_matrix(lr_obj, X_test_pca_final, y_test)
plt.show()
```




```
#Plot ROC_AUC Curve for Logistic Regression
metrics.plot_roc_curve(lr_obj, X_test_pca_final, y_test)
plt.show()
```



Decision Tree

- X_train_resample, y_train_resample

```
from sklearn.tree import DecisionTreeClassifier
#Instantiate Decision tree with defautl parameter
dt_obj=  DecisionTreeClassifier(random_state=40)

# here we have used data generated by SMOTE.
dt_obj.fit(X_train_scale, y_train_resample)
cv_score = cross_val_score(dt_obj, X_train_scale, y_train_resample, cv=5, scoring='f1_micro')
print(cv_score)
```

```
[0.88125213 0.92491777 0.92343466 0.93160163 0.93216878]
```

```
#check the default paramters
dt_obj.get_params()
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 40,
 'splitter': 'best'}
```

```
#Perform hyperparamter tuning with randomizedsearchcv
param_grid = dict({"max_leaf_nodes":[4,5,6],"min_samples_leaf":[3,4,5],'min_samples_split':[3,4,5]})
dt_clf = DecisionTreeClassifier(random_state=40)
dt_clf_rcv = RandomizedSearchCV(dt_clf,param_grid,cv=5,scoring="f1_micro")# n_jobs=-1
dt_clf_rcv.fit(X_train_scale, y_train_resample)
```

```
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=40),
                   param_distributions={'max_leaf_nodes': [4, 5, 6],
                                        'min_samples_leaf': [3, 4, 5],
```

```
                                   'min_samples_split': [3, 4, 5]},
                   scoring='f1_micro')
```

```
#check the beat score and best estimator paramters
print(dt_clf_rcv.best_score_)
print(dt_clf_rcv.best_estimator_)
```

```
0.8546802785906701
DecisionTreeClassifier(max_leaf_nodes=6, min_samples_leaf=4,
                       min_samples_split=3, random_state=40)
```

```
#Train the decision tree with best paramters obtained from above step

dt_clf = DecisionTreeClassifier(max_leaf_nodes=6,min_samples_leaf=4,min_samples_split=5,random_state=40)
dt_clf.fit(X_train_scale,y_train_resample)
```

```
DecisionTreeClassifier(max_leaf_nodes=6, min_samples_leaf=4,
                       min_samples_split=5, random_state=40)
```

```
#perform the prediction
y_pred_dt = dt_clf.predict(X_test_scale)
#Perform the prediction probability
pred_prob = dt_clf.predict_proba(X_test_scale)
```

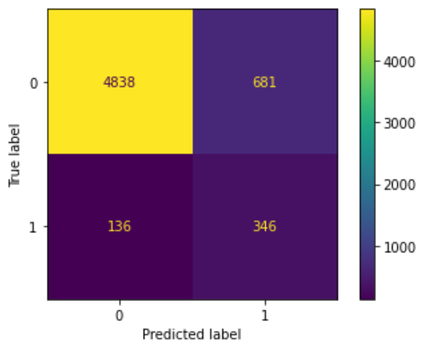
```
##check the scores.
df2 = calculate_peformance_testdata("DecisionTree",y_test,y_pred_dt,pred_prob[:,1])
```

```
DecisionTree:
precision: 0.33690360272638753
recall: 0.7178423236514523
f1_score: 0.45858184227965537
roc_auc: 0.8510624180969703
```

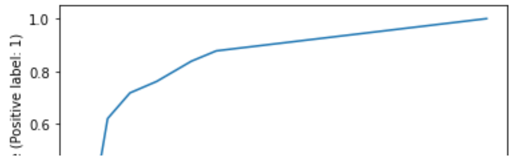
```
#Add the score to Dataframe  for comparision
score_df = score_df.append(df2)
score_df.dropna(inplace=True)
score_df.drop_duplicates(inplace=True)
score_df
```

	Model	precision	recall	f1_score	roc_auc
0	LogisticRegression	0.091931	0.560166	0.157941	0.575711
0	DecisionTree	0.336904	0.717842	0.458582	0.851062

```
#visualize the confusion matrix
metrics.plot_confusion_matrix(dt_clf, X_test_scale, y_test)
plt.show()
```




```
#plot the ROC_AUC curve
metrics.plot_roc_curve(dt_clf, X_test_scale, y_test)
plt.show()
```



Randomforest

```
#Instantiate RandomForest, train with default parameters
rf_class = RandomForestClassifier(n_jobs=-1) #class_weight={0:1,1:2}
rf_class.fit(X_train_scale,y_train_resample)
y_pred_rf = rf_class.predict(X_test_scale)
pred_prob = rf_class.predict_proba(X_test_scale)
```

```
#check the default parameters
rf_class.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': -1,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
#Use best paramters to train the model
rf_class = RandomForestClassifier(min_samples_leaf=3,n_estimators=120,n_jobs=-1,random_state=40)
rf_class.fit(X_train_scale,y_train_resample)
y_pred_rf = rf_class.predict(X_test_scale)
pred_prob = rf_class.predict_proba(X_test_scale)
```

```
#check the scores
df3 = calculate_peformance_testdata("RandomForest",y_test,y_pred_rf,pred_prob[:,1])
```

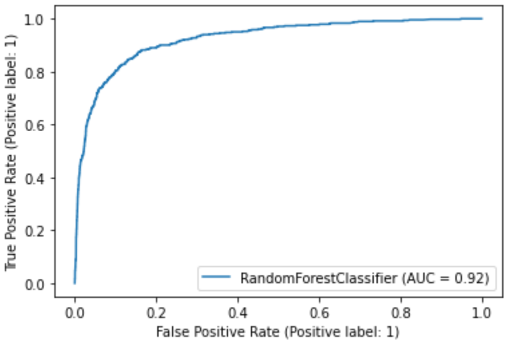
```
RandomForest:
precision: 0.5862708719851577
recall: 0.6556016597510373
f1_score: 0.6190009794319296
roc_auc: 0.9244024227132372
```

```
#Add score to the dataframe for comparision
score_df = score_df.append(df3)
score_df
```

	Model	precision	recall	f1_score	roc_auc
0	LogisticRegression	0.091931	0.560166	0.157941	0.575711
0	DecisionTree	0.336904	0.717842	0.458582	0.851062
0	RandomForest	0.586271	0.655602	0.619001	0.924402

```
#visualize confusion matrix
metrics.plot_confusion_matrix(rf_class, X_test_scale, y_test)
plt.show()
```


```
#plot roc auc cureve
metrics.plot_roc_curve(rf_class, X_test_scale, y_test)
plt.show()
```



GradientBoosting

```
#Train gradient boosting with default parameters
from sklearn.ensemble import GradientBoostingClassifier
gb_class = GradientBoostingClassifier(random_state=42,min_samples_leaf=4,min_samples_split=5)
# n_estimators=110,min_samples_leaf=2,min_samples_split=3,learning_rate=0.2
gb_class.fit(X_train_scale,y_train_resample)

#get the predicated label
y_pred_gb = gb_class.predict(X_test_scale)
#get the predicted probability
pred_prob = gb_class.predict_proba(X_test_scale)
```

```
#check the training default parameters
gb_class.get_params()
```

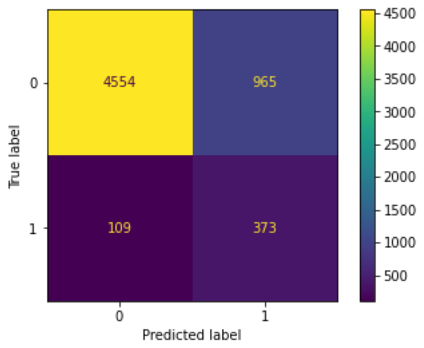
```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 4,
 'min_samples_split': 5,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 42,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

```
#Check the test scores
df4 = calculate_peformance_testdata("GradientBoosting",y_test,y_pred_gb,pred_prob[:,1])
```

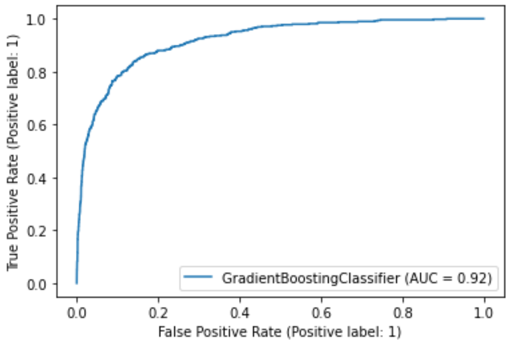
```
GradientBoosting:
precision: 0.48326055312954874
recall: 0.6887966804979253
f1_score: 0.5680068434559453
roc_auc: 0.9197948016621568
```

```
#Add the scores to dataframe
score_df=score_df.append(df4)
score_df
```

```
#Plot the confusion matrix
metrics.plot_confusion_matrix(gb_class, X_test, y_test)
plt.show()
```




```
#plot the roc curve
metrics.plot_roc_curve(gb_class, X_test_scale, y_test)
plt.show()
```



Xgboost

```
# !pip install xgboost
import xgboost as xgb
```

```
# Model training with default paamters

xgb_class = xgb.XGBClassifier(max_depth=10)
xgb_class.fit(X_train_scale,y_train_resample)

#Model prediction
y_pred_xgb = xgb_class.predict(X_test_scale)
#Model predict probability
pred_prob = xgb_class.predict_proba(X_test_scale)
```

```
#check the model default paramters
xgb_class.get_params()
```

```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'gamma': 0,
 'learning_rate': 0.1,
 'max_delta_step': 0,
 'max_depth': 10,
 'min_child_weight': 1,
 'missing': None,
 'n_estimators': 100,
 'n_jobs': 1,
 'nthread': None,
 'objective': 'binary:logistic',
 'random_state': 0,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'seed': None,
 'silent': None,
```

```
       'subsample': 1,
       'verbosity': 1}
```

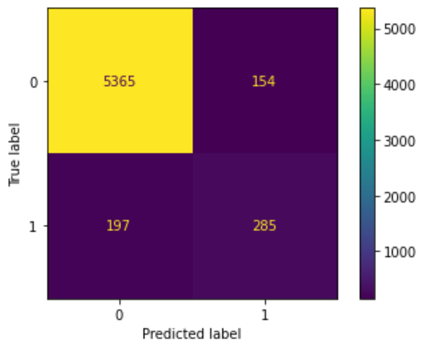
```
#chekc the scores
df5 = calculate_peformance_testdata("XGBoost",y_test,y_pred_xgb,pred_prob[:,1])
```

```
XGBoost:
precision: 0.6492027334851936
recall: 0.5912863070539419
f1_score: 0.6188925081433225
roc_auc: 0.9314074577525094
```

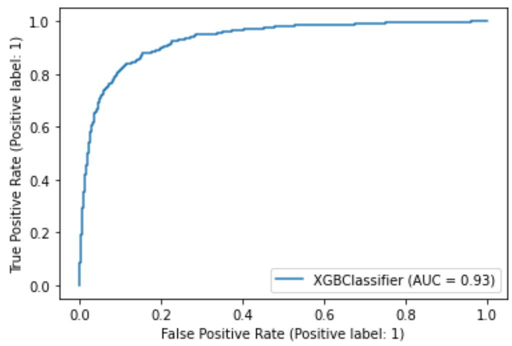
```
#add the score to dataframe
score_df= score_df.append(df5)
score_df.drop_duplicates()
```

	Model	precision	recall	f1_score	roc_auc
0	LogisticRegression	0.091931	0.560166	0.157941	0.575711
0	DecisionTree	0.336904	0.717842	0.458582	0.851062
0	RandomForest	0.586271	0.655602	0.619001	0.924402
0	GradientBoosting	0.483261	0.688797	0.568007	0.919795
0	XGBoost	0.649203	0.591286	0.618893	0.931407

```
#Plot confusion matrix
metrics.plot_confusion_matrix(xgb_class, X_test_scale, y_test)
plt.show()
```




```
#plot roc curve
metrics.plot_roc_curve(xgb_class, X_test_scale, y_test)
plt.show()
```




```
#check how various model is performing on test set on Churn=1.
score_df
```


	Model	precision	recall	f1_score	roc_auc
0	LogisticRegression	0.091931	0.560166	0.157941	0.575711
0	DecisionTree	0.336904	0.717842	0.458582	0.851062
0	RandomForest	0.586271	0.655602	0.619001	0.924402
0	GradientBoosting	0.483261	0.688797	0.568007	0.919795
0	XGBoost	0.649203	0.591286	0.618893	0.931407

- The randomforest worked well on this data in churn with precision close to 58%, recall close to 65% and f1_score close to 61%.
- In Logistic regression we have used PCA.
- In this scenario, Without PCA model works well.

Fearure Importance and Model Interpretation

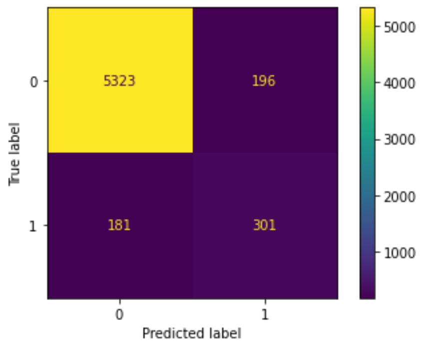
```
# Randomforest model training
gb_object = RandomForestClassifier(random_state=40)
gb_object.fit(X_train_resample,y_train_resample)
y_pred = gb_object.predict(X_test)
```

```
#check the performance on test data
calculate_peformance_testdata("RandomForest",y_test,y_pred,pred_prob[:,1])
```

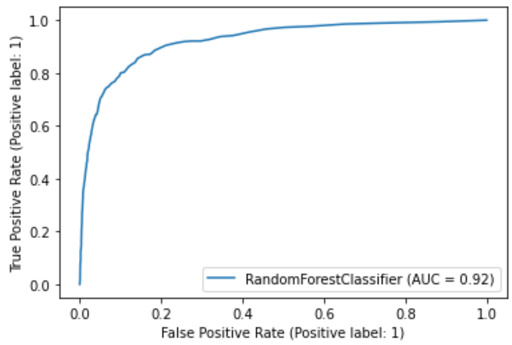
```
RandomForest:
precision: 0.6056338028169014
recall: 0.6244813278008299
f1_score: 0.6149131767109295
roc_auc: 0.9314074577525094
```

	Model	precision	recall	f1_score	roc_auc
0	RandomForest	0.605634	0.624481	0.614913	0.931407

```
#plot confusion matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(gb_object, X_test, y_test)
plt.show()
```




```
#plot ROC curve
metrics.plot_roc_curve(gb_object, X_test, y_test)
plt.show()
```




```
#check the classification report
print(metrics.classification_report(y_test,y_pred))
```

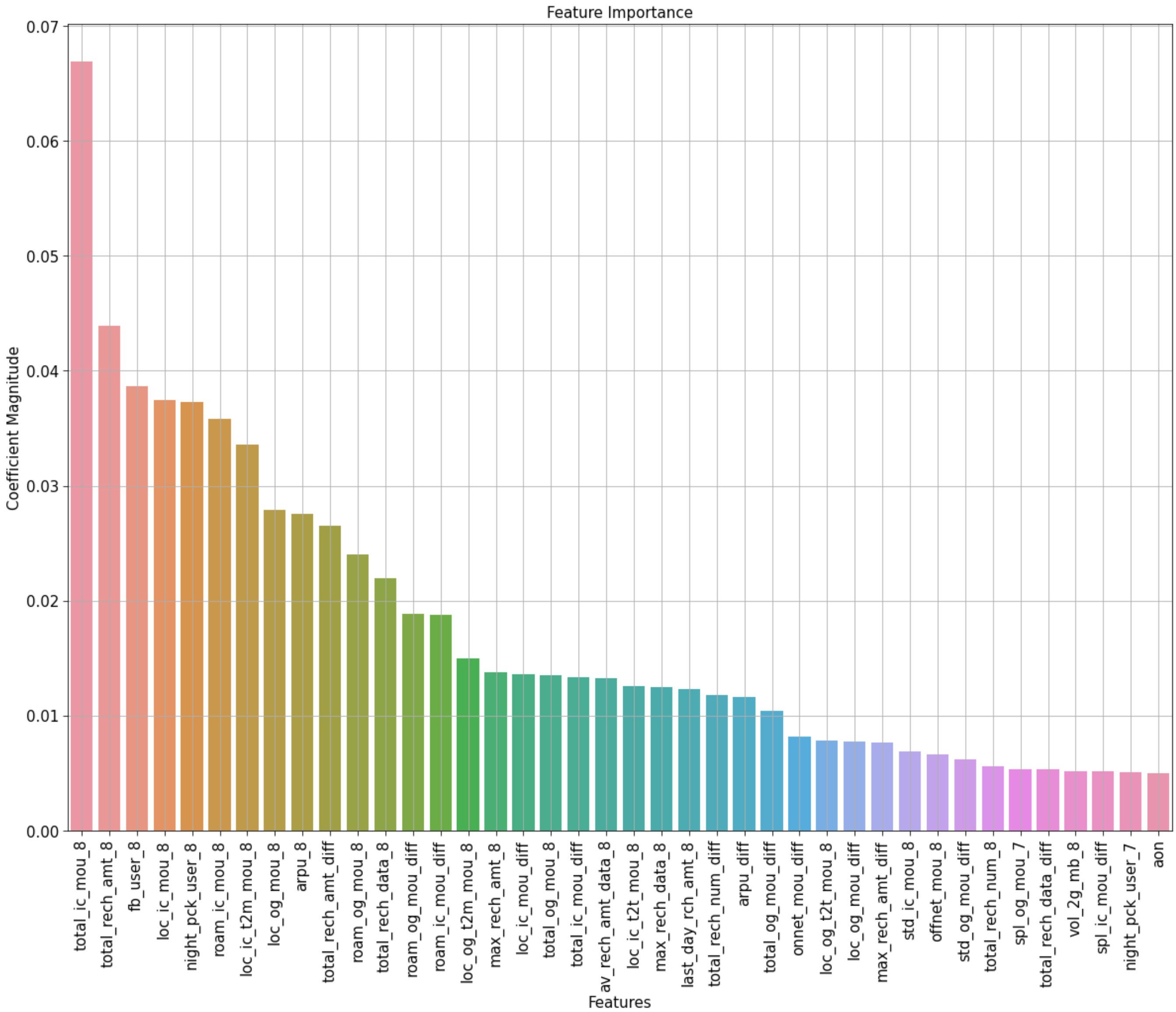
```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      5519
           1       0.61      0.62      0.61       482

    accuracy                           0.94      6001
   macro avg       0.79      0.79      0.79      6001
weighted avg       0.94      0.94      0.94      6001
```

```
#Create a Feature importance dataframe
Feature_importance = pd.DataFrame({"columns":X_train.columns,"feature_importance":gb_object.feature_importances_})
```

```
#check 40 important features
fi = Feature_importance.sort_values(by="feature_importance",ascending=False).head(40)
fi
```

	columns	feature_importance
80	total_ic_mou_8	0.066874
95	total_rech_amt_8	0.043899
134	fb_user_8	0.038620
65	loc_ic_mou_8	0.037469
119	night_pck_user_8	0.037256
11	roam_ic_mou_8	0.035839
59	loc_ic_t2m_mou_8	0.033583
29	loc_og_mou_8	0.027910
2	arpu_8	0.027545
156	total_rech_amt_diff	0.026513
14	roam_og_mou_8	0.024042
104	total_rech_data_8	0.021991
144	roam_og_mou_diff	0.018811
143	roam_ic_mou_diff	0.018789
20	loc_og_t2m_mou_8	0.015012
98	max_rech_amt_8	0.013761
150	loc_ic_mou_diff	0.013625
53	total_og_mou_8	0.013502
154	total_ic_mou_diff	0.013340
110	av_rech_amt_data_8	0.013241
56	loc_ic_t2t_mou_8	0.012584
107	max_rech_data_8	0.012510
101	last_day_rch_amt_8	0.012273
155	total_rech_num_diff	0.011773
140	arpu_diff	0.011626
149	total_og_mou_diff	0.010407
141	onnet_mou_diff	0.008210
17	loc_og_t2t_mou_8	0.007867
145	loc_og_mou_diff	0.007731
157	max_rech_amt_diff	0.007625
77	std_ic_mou_8	0.006873
8	offnet_mou_8	0.006631
146	std_og_mou_diff	0.006177
92	total_rech_num_8	0.005596
46	spl_og_mou_7	0.005352
158	total_rech_data_diff	0.005312
113	vol_2g_mb_8	0.005133
153	spl_ic_mou_diff	0.005132
118	night_pck_user_7	0.005101
135	aon	0.004965

```
#Plot to show the feature importance
plt.figure(figsize=[20,15])
sns.barplot(x = "columns",y="feature_importance",data=fi)
plt.title("Feature Importance",size=15)
```



Conclusion:

- The most important features are as shown in above graph.
- Average revenue per user more, those are likely to churn if they are not happy with the network.
- Local calls minutes of usage has also has impact on churn .
- Large difference between recharge amount between 6th and 7th month, also impact churn.
- Users who are using more Roaminng in Outgoing and Incoming calls, are likely to churn.Compnay can focus on them too.