In [1]:

```
import numpy as np
import pandas as pd

# hide warnings
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 500)

master_df = pd.read_csv("telecom_churn_data.csv")
master_df_copy = master_df.copy()
```

In [2]:

```
master_df.shape
```

Out[2]:

```
(99999, 226)
```

In [3]:

```
master_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Columns: 226 entries, mobile_number to sep_vbc_3g
dtypes: float64(179), int64(35), object(12)
memory usage: 172.4+ MB
```

In [4]:

```
master_df.head()
```

Out[4]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_
0	7000842753	109	0.0	0.0	0.0	
1	7001865778	109	0.0	0.0	0.0	
2	7001625959	109	0.0	0.0	0.0	
3	7001204172	109	0.0	0.0	0.0	
4	7000142493	109	0.0	0.0	0.0	

In [5]:

```
master_df.columns
```

Out[5]:

```
Index(['mobile_number', 'circle_id', 'loc_og_t2o_mou', 'std_og_t2o_mou',
       'loc_ic_t2o_mou', 'last_date_of_month_6', 'last_date_of_month_7',
       'last_date_of_month_8', 'last_date_of_month_9', 'arpu_6',
       ...
       'sachet_3g_9', 'fb_user_6', 'fb_user_7', 'fb_user_8', 'fb_user_9',
       'aon', 'aug_vbc_3g', 'jul_vbc_3g', 'jun_vbc_3g', 'sep_vbc_3g'],
      dtype='object', length=226)
```

Identify the columns with more than 25% NANs, and remove them

In [6]:

```
xyz = round(100*(master_df.isna().sum()/len(master_df.index)), 0)
xyz_copy = xyz

na_df = pd.DataFrame(xyz)
na_df.columns = ['perc_va_values']

na_df = na_df[na_df['perc_va_values']>25.0]
na_df
```

Out[6]:

	perc_va_values
date_of_last_rech_data_6	75.0
date_of_last_rech_data_7	74.0
date_of_last_rech_data_8	74.0
date_of_last_rech_data_9	74.0
total_rech_data_6	75.0
total_rech_data_7	74.0
total_rech_data_8	74.0
total_rech_data_9	74.0
max_rech_data_6	75.0
max_rech_data_7	74.0
max_rech_data_8	74.0
max_rech_data_9	74.0
count_rech_2g_6	75.0
count_rech_2g_7	74.0
count_rech_2g_8	74.0
count_rech_2g_9	74.0
count_rech_3g_6	75.0
count_rech_3g_7	74.0
count_rech_3g_8	74.0
count_rech_3g_9	74.0
av_rech_amt_data_6	75.0
av_rech_amt_data_7	74.0
av_rech_amt_data_8	74.0
av_rech_amt_data_9	74.0
arpu_3g_6	75.0
arpu_3g_7	74.0
arpu_3g_8	74.0
arpu_3g_9	74.0
arpu_2g_6	75.0
arpu_2g_7	74.0
arpu_2g_8	74.0
arpu_2g_9	74.0
night_pck_user_6	75.0
night_pck_user_7	74.0
night_pck_user_8	74.0
night_pck_user_9	74.0
fb_user_6	75.0

	perc_va_values
fb_user_7	74.0
fb_user_8	74.0
fb_user_9	74.0

In [7]:

```
master_df = master_df.loc[:, (round(100*(master_df.isna().sum()/len(master_df.index)),
0)<25.0) ]
master_df = master_df.dropna()
master_df.shape
```

Out[7]:

```
(84185, 186)
```

Identify the columns with only single value across all the rows and remove them

In [8]:

```
def single_value(df):
    col_list = []
    for each_item in df.columns:
        if len(set(list(df[each_item]))) == 1:
            col_list.append(each_item)
    return col_list

single_val_columns = single_value(master_df)
single_val_columns
```

Out[8]:

```
['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']
```

Remove all the columns with a single value in it

In [9]:

```
master_df = master_df.drop(single_val_columns, axis = 1)
master_df.shape
```

Out[9]:

```
(84185, 170)
```

2. Filter high-value customers

Avergae recharge amount of the good phase

In [10]:

```
master_df['avg_rech_good_phase'] = round((master_df['total_rech_amt_6'] + master_df['to
tal_rech_amt_7'])/ (master_df['total_rech_num_6']+master_df['total_rech_num_6']),2)
avg_rch_70th_percentile = master_df['avg_rech_good_phase'].quantile(.7)
avg_rch_70th_percentile
```

Out[10]:

```
60.75
```

Select only the customers whose recharge spend is more than that of 70th percentile on average

In [11]:

```
master_df = master_df[master_df.avg_rech_good_phase > avg_rch_70th_percentile]
master_df.shape
```

Out[11]:

```
(25240, 171)
```

In [12]:

```
master_df.head()
```

Out[12]:

	mobile_number	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	onnet_mou_7	onnet_
13	7002191713	492.846	205.671	593.260	322.732	501.76	108.39	
19	7001754084	163.430	241.218	326.920	75.229	4.04	7.38	
23	7000887461	74.350	193.897	366.966	811.480	48.96	50.66	
24	7001125315	422.050	359.730	354.793	473.030	124.19	55.19	
25	7000852702	244.436	285.403	172.773	161.284	255.14	327.18	

In [13]:

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25240 entries, 13 to 99997
Data columns (total 171 columns):
mobile_number          int64
arpu_6                 float64
arpu_7                 float64
arpu_8                 float64
arpu_9                 float64
onnet_mou_6            float64
onnet_mou_7            float64
onnet_mou_8            float64
onnet_mou_9            float64
offnet_mou_6           float64
offnet_mou_7           float64
offnet_mou_8           float64
offnet_mou_9           float64
roam_ic_mou_6          float64
roam_ic_mou_7          float64
roam_ic_mou_8          float64
roam_ic_mou_9          float64
roam_og_mou_6          float64
roam_og_mou_7          float64
roam_og_mou_8          float64
roam_og_mou_9          float64
loc_og_t2t_mou_6       float64
loc_og_t2t_mou_7       float64
loc_og_t2t_mou_8       float64
loc_og_t2t_mou_9       float64
loc_og_t2m_mou_6       float64
loc_og_t2m_mou_7       float64
loc_og_t2m_mou_8       float64
loc_og_t2m_mou_9       float64
loc_og_t2f_mou_6       float64
loc_og_t2f_mou_7       float64
loc_og_t2f_mou_8       float64
loc_og_t2f_mou_9       float64
loc_og_t2c_mou_6       float64
loc_og_t2c_mou_7       float64
loc_og_t2c_mou_8       float64
loc_og_t2c_mou_9       float64
loc_og_mou_6           float64
loc_og_mou_7           float64
loc_og_mou_8           float64
loc_og_mou_9           float64
std_og_t2t_mou_6       float64
std_og_t2t_mou_7       float64
std_og_t2t_mou_8       float64
std_og_t2t_mou_9       float64
std_og_t2m_mou_6       float64
std_og_t2m_mou_7       float64
std_og_t2m_mou_8       float64
std_og_t2m_mou_9       float64
std_og_t2f_mou_6       float64
std_og_t2f_mou_7       float64
std_og_t2f_mou_8       float64
std_og_t2f_mou_9       float64
std_og_mou_6           float64
std_og_mou_7           float64
std_og_mou_8           float64
std_og_mou_9           float64
isd_og_mou_6           float64
```

```
isd_og_mou_7          float64
isd_og_mou_8          float64
isd_og_mou_9          float64
spl_og_mou_6          float64
spl_og_mou_7          float64
spl_og_mou_8          float64
spl_og_mou_9          float64
og_others_6           float64
og_others_7           float64
og_others_8           float64
og_others_9           float64
total_og_mou_6        float64
total_og_mou_7        float64
total_og_mou_8        float64
total_og_mou_9        float64
loc_ic_t2t_mou_6      float64
loc_ic_t2t_mou_7      float64
loc_ic_t2t_mou_8      float64
loc_ic_t2t_mou_9      float64
loc_ic_t2m_mou_6      float64
loc_ic_t2m_mou_7      float64
loc_ic_t2m_mou_8      float64
loc_ic_t2m_mou_9      float64
loc_ic_t2f_mou_6      float64
loc_ic_t2f_mou_7      float64
loc_ic_t2f_mou_8      float64
loc_ic_t2f_mou_9      float64
loc_ic_mou_6          float64
loc_ic_mou_7          float64
loc_ic_mou_8          float64
loc_ic_mou_9          float64
std_ic_t2t_mou_6      float64
std_ic_t2t_mou_7      float64
std_ic_t2t_mou_8      float64
std_ic_t2t_mou_9      float64
std_ic_t2m_mou_6      float64
std_ic_t2m_mou_7      float64
std_ic_t2m_mou_8      float64
std_ic_t2m_mou_9      float64
std_ic_t2f_mou_6      float64
std_ic_t2f_mou_7      float64
std_ic_t2f_mou_8      float64
std_ic_t2f_mou_9      float64
std_ic_mou_6          float64
std_ic_mou_7          float64
std_ic_mou_8          float64
std_ic_mou_9          float64
total_ic_mou_6        float64
total_ic_mou_7        float64
total_ic_mou_8        float64
total_ic_mou_9        float64
spl_ic_mou_6          float64
spl_ic_mou_7          float64
spl_ic_mou_8          float64
spl_ic_mou_9          float64
isd_ic_mou_6          float64
isd_ic_mou_7          float64
isd_ic_mou_8          float64
isd_ic_mou_9          float64
ic_others_6           float64
ic_others_7           float64
```

```
ic_others_8           float64
ic_others_9           float64
total_rech_num_6      int64
total_rech_num_7      int64
total_rech_num_8      int64
total_rech_num_9      int64
total_rech_amt_6      int64
total_rech_amt_7      int64
total_rech_amt_8      int64
total_rech_amt_9      int64
max_rech_amt_6        int64
max_rech_amt_7        int64
max_rech_amt_8        int64
max_rech_amt_9        int64
date_of_last_rech_6   object
date_of_last_rech_7   object
date_of_last_rech_8   object
date_of_last_rech_9   object
last_day_rch_amt_6    int64
last_day_rch_amt_7    int64
last_day_rch_amt_8    int64
last_day_rch_amt_9    int64
vol_2g_mb_6           float64
vol_2g_mb_7           float64
vol_2g_mb_8           float64
vol_2g_mb_9           float64
vol_3g_mb_6           float64
vol_3g_mb_7           float64
vol_3g_mb_8           float64
vol_3g_mb_9           float64
monthly_2g_6          int64
monthly_2g_7          int64
monthly_2g_8          int64
monthly_2g_9          int64
sachet_2g_6           int64
sachet_2g_7           int64
sachet_2g_8           int64
sachet_2g_9           int64
monthly_3g_6          int64
monthly_3g_7          int64
monthly_3g_8          int64
monthly_3g_9          int64
sachet_3g_6           int64
sachet_3g_7           int64
sachet_3g_8           int64
sachet_3g_9           int64
aon                   int64
aug_vbc_3g            float64
jul_vbc_3g            float64
jun_vbc_3g            float64
sep_vbc_3g            float64
avg_rech_good_phase   float64
dtypes: float64(133), int64(34), object(4)
memory usage: 33.1+ MB
```

3. Tag churners and remove attributes of the churn phase

In [14]:

```
#list(map(lambda x : True if ((int(x['total_ic_mou_9']) == 0) & (int(x['total_og_mou_9']) == 0) & (int(x['vol_2g_mb_9']) == 0) & (int(x['vol_3g_mb_9']) == 0)) else False, master_df))
master_df['churn_tag'] = list(map(lambda a,b,c,d : 1 if ((a == 0) & (b==0) & (c==0) & (d==0)) else 0, master_df['total_ic_mou_9'], master_df['total_og_mou_9'], master_df['vol_2g_mb_9'], master_df['vol_3g_mb_9'] ))

master_df.shape
```

Out[14]:

```
(25240, 172)
```

Find all the columns with the '_9' suffix which are mainly for churn phase

In [15]:

```
def find_churn_columns(df):
    col_list = []
    for each_item in df.columns:
        if (each_item.find('_9') != -1):
            col_list.append(each_item)
    return col_list

churn_columns = find_churn_columns(master_df)
churn_columns
```

Out[15]:

```
['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',
 'date_of_last_rech_9',
 'last_day_rch_amt_9',
 'vol_2g_mb_9',
 'vol_3g_mb_9',
 'monthly_2g_9',
 'sachet_2g_9',
 'monthly_3g_9',
 'sachet_3g_9']
```

In [16]:

```
master_df = master_df.drop(churn_columns, axis = 1)
#master_df = master_df.drop('sep_vbc_3g', axis = 1)
master_df.shape
```

Out[16]:

```
(25240, 131)
```

Take the backup of mobile_number column

In [17]:

```
mobile_df = master_df['mobile_number']
master_df = master_df.drop('mobile_number', axis = 1)
master_df.shape
```

Out[17]:

```
(25240, 130)
```

4. Derived Columns

Calculate the Average values of all the co

In [18]:

```
## Average arpu
master_df_derived = pd.DataFrame()

master_df_derived['avg_arpu'] = round(((master_df['arpu_6']+master_df['arpu_7']+master_df['arpu_8'])/3),2)
master_df_derived['avg_onnet_mou'] = round(((master_df['onnet_mou_6']+master_df['onnet_mou_7']+master_df['onnet_mou_8'])/3),2)
master_df_derived['avg_offnet_mou'] = round(((master_df['offnet_mou_6']+master_df['offnet_mou_7']+master_df['offnet_mou_8'])/3),2)
master_df_derived['avg_roam_ic_mou'] = round(((master_df['roam_ic_mou_6']+master_df['roam_ic_mou_7']+master_df['roam_ic_mou_8'])/3),2)
master_df_derived['avg_roam_og_mou'] = round(((master_df['roam_og_mou_6']+master_df['roam_og_mou_7']+master_df['roam_og_mou_8'])/3),2)
master_df_derived['avg_loc_og_t2t_mou'] = round(((master_df['loc_og_t2t_mou_6']+master_df['loc_og_t2t_mou_7']+master_df['loc_og_t2t_mou_8'])/3),2)


master_df_derived['avg_loc_og_t2m_mou'] = round(((master_df['loc_og_t2m_mou_6']+master_df['loc_og_t2m_mou_7']+master_df['loc_og_t2m_mou_8'])/3),2)
master_df_derived['avg_loc_og_t2f_mou'] = round(((master_df['loc_og_t2f_mou_6']+master_df['loc_og_t2f_mou_7']+master_df['loc_og_t2f_mou_8'])/3),2)
master_df_derived['avg_loc_og_t2c_mou'] = round(((master_df['loc_og_t2c_mou_6']+master_df['loc_og_t2c_mou_7']+master_df['loc_og_t2c_mou_8'])/3),2)
master_df_derived['avg_loc_og_mou'] = round(((master_df['loc_og_mou_6']+master_df['loc_og_mou_7']+master_df['loc_og_mou_8'])/3),2)
master_df_derived['avg_std_og_t2t_mou'] = round(((master_df['std_og_t2t_mou_6']+master_df['std_og_t2t_mou_7']+master_df['std_og_t2t_mou_8'])/3),2)


master_df_derived['avg_std_og_t2m_mou'] = round(((master_df['std_og_t2m_mou_6']+master_df['std_og_t2m_mou_7']+master_df['std_og_t2m_mou_8'])/3),2)
master_df_derived['avg_std_og_t2f_mou'] = round(((master_df['std_og_t2f_mou_6']+master_df['std_og_t2f_mou_7']+master_df['std_og_t2f_mou_8'])/3),2)
master_df_derived['avg_std_og_mou'] = round(((master_df['std_og_mou_6']+master_df['std_og_mou_7']+master_df['std_og_mou_8'])/3),2)
master_df_derived['avg_isd_og_mou'] = round(((master_df['isd_og_mou_6']+master_df['isd_og_mou_7']+master_df['isd_og_mou_8'])/3),2)
master_df_derived['avg_spl_og_mou'] = round(((master_df['spl_og_mou_6']+master_df['spl_og_mou_7']+master_df['spl_og_mou_8'])/3),2)

master_df_derived['avg_og_others'] = round(((master_df['og_others_6']+master_df['og_others_7']+master_df['og_others_8'])/3),2)
master_df_derived['avg_total_og_mou'] = round(((master_df['total_og_mou_6']+master_df['total_og_mou_7']+master_df['total_og_mou_8'])/3),2)
master_df_derived['avg_loc_ic_t2t_mou'] = round(((master_df['loc_ic_t2t_mou_6']+master_df['loc_ic_t2t_mou_7']+master_df['loc_ic_t2t_mou_8'])/3),2)
master_df_derived['avg_loc_ic_t2m_mou'] = round(((master_df['loc_ic_t2m_mou_6']+master_df['loc_ic_t2m_mou_7']+master_df['loc_ic_t2m_mou_8'])/3),2)
master_df_derived['avg_loc_ic_t2f_mou'] = round(((master_df['loc_ic_t2f_mou_6']+master_df['loc_ic_t2f_mou_7']+master_df['loc_ic_t2f_mou_8'])/3),2)

master_df_derived['avg_loc_ic_mou'] = round(((master_df['loc_ic_mou_6']+master_df['loc_ic_mou_7']+master_df['loc_ic_mou_8'])/3),2)
master_df_derived['avg_std_ic_t2t_mou'] = round(((master_df['std_ic_t2t_mou_6']+master_df['std_ic_t2t_mou_7']+master_df['std_ic_t2t_mou_8'])/3),2)
master_df_derived['avg_std_ic_t2m_mou'] = round(((master_df['std_ic_t2m_mou_6']+master_df['std_ic_t2m_mou_7']+master_df['std_ic_t2m_mou_8'])/3),2)
master_df_derived['avg_std_ic_t2f_mou'] = round(((master_df['std_ic_t2f_mou_6']+master_df['std_ic_t2f_mou_7']+master_df['std_ic_t2f_mou_8'])/3),2)
```

```
master_df_derived['avg_std_ic_mou'] = round(((master_df['std_ic_mou_6']+master_df['std_
ic_mou_7']+master_df['std_ic_mou_8'])/3),2)

master_df_derived['avg_total_ic_mou'] = round(((master_df['total_ic_mou_6']+master_df[
'total_ic_mou_7']+master_df['total_ic_mou_8'])/3),2)
master_df_derived['avg_spl_ic_mou'] = round(((master_df['spl_ic_mou_6']+master_df['spl_
ic_mou_7']+master_df['spl_ic_mou_8'])/3),2)
master_df_derived['avg_isd_ic_mou'] = round(((master_df['isd_ic_mou_6']+master_df['isd_
ic_mou_7']+master_df['isd_ic_mou_8'])/3),2)
master_df_derived['avg_ic_others'] = round(((master_df['ic_others_6']+master_df['ic_oth
ers_7']+master_df['ic_others_8'])/3),2)
master_df_derived['avg_total_rech_num'] = round(((master_df['total_rech_num_6']+master_
df['total_rech_num_7']+master_df['total_rech_num_8'])/3),2)

master_df_derived['avg_total_rech_amt'] = round(((master_df['total_rech_amt_6']+master_
df['total_rech_amt_7']+master_df['total_rech_amt_8'])/3),2)
master_df_derived['avg_max_rech_amt'] = round(((master_df['max_rech_amt_6']+master_df[
'max_rech_amt_7']+master_df['max_rech_amt_8'])/3),2)
master_df_derived['avg_last_day_rch_amt'] = round(((master_df['last_day_rch_amt_6']+mas
ter_df['last_day_rch_amt_7']+master_df['last_day_rch_amt_8'])/3),2)
master_df_derived['avg_vol_2g_mb'] = round(((master_df['vol_2g_mb_6']+master_df['vol_2g
_mb_7']+master_df['vol_2g_mb_8'])/3),2)

master_df_derived['avg_vol_3g_mb'] = round(((master_df['vol_3g_mb_6']+master_df['vol_3g
_mb_7']+master_df['vol_3g_mb_8'])/3),2)
master_df_derived['avg_monthly_2g'] = round(((master_df['monthly_2g_6']+master_df['mont
hly_2g_7']+master_df['monthly_2g_8'])/3),2)
master_df_derived['avg_sachet_2g'] = round(((master_df['sachet_2g_6']+master_df['sachet
_2g_7']+master_df['sachet_2g_8'])/3),2)
master_df_derived['avg_monthly_3g'] = round(((master_df['monthly_3g_6']+master_df['mont
hly_3g_7']+master_df['monthly_3g_8'])/3),2)
master_df_derived['avg_sachet_3g_6'] = round(((master_df['sachet_3g_6']+master_df['sach
et_3g_7']+master_df['sachet_3g_8'])/3),2)
master_df_derived['avg_vbc_3g'] = round(((master_df['aug_vbc_3g']+master_df['jul_vbc_3
g']+master_df['jun_vbc_3g'])/3),2)


## find the avg number of days between recharges
master_df['date_of_last_rech_6'] = pd.to_datetime(master_df['date_of_last_rech_6'])
master_df['date_of_last_rech_7'] = pd.to_datetime(master_df['date_of_last_rech_7'])
master_df['date_of_last_rech_8'] = pd.to_datetime(master_df['date_of_last_rech_8'])

#master_df['avg_days_between_rchg'] = round(((master_df['date_of_last_rech_8'] - master
_df['date_of_last_rech_7']) + (master_df['date_of_last_rech_7'] - master_df['date_of_la
st_rech_6']))/2,2)
master_df['days1'] = (master_df['date_of_last_rech_8'] - master_df['date_of_last_rech_
7']).dt.days
master_df['days2'] = (master_df['date_of_last_rech_7'] - master_df['date_of_last_rech_
6']).dt.days
master_df_derived['avg_days_between_rchg'] = round((master_df['days1']+master_df['days
2'])/2,2)
```

In [19]:

```
## Copy the non-derived columns
master_df_derived['aon'] = master_df['aon']
master_df_derived['avg_rech_good_phase'] = master_df['avg_rech_good_phase']
master_df_derived['churn_tag'] = master_df['churn_tag']
master_df_derived.shape
```

Out[19]:

```
(25240, 45)
```

Remove all the original columns for which derived columns are developed

In [20]:

```
master_df_derived.head(10)
```

Out[20]:

	avg_arpu	avg_onnet_mou	avg_offnet_mou	avg_roam_ic_mou	avg_roam_og_mou	avg_l
13	430.59	381.46	338.35	79.96	14.89	
19	243.86	8.34	17.08	0.00	0.00	
23	211.74	44.40	126.89	0.00	0.00	
24	378.86	106.83	368.06	7.71	10.94	
25	234.20	240.19	147.24	0.00	0.00	
33	1249.69	0.00	0.00	0.00	0.00	
34	198.89	1.19	9.23	0.00	0.00	
36	138.02	0.37	190.62	0.00	0.00	
40	106.30	1.65	32.02	0.00	0.50	
41	379.46	95.24	216.61	0.00	0.00	

Check if there are any non-numerical columns left

In [21]:

```
master_df_derived.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25240 entries, 13 to 99997
Data columns (total 45 columns):
avg_arpu                 25240 non-null float64
avg_onnet_mou            25240 non-null float64
avg_offnet_mou           25240 non-null float64
avg_roam_ic_mou          25240 non-null float64
avg_roam_og_mou          25240 non-null float64
avg_loc_og_t2t_mou       25240 non-null float64
avg_loc_og_t2m_mou       25240 non-null float64
avg_loc_og_t2f_mou       25240 non-null float64
avg_loc_og_t2c_mou       25240 non-null float64
avg_loc_og_mou           25240 non-null float64
avg_std_og_t2t_mou       25240 non-null float64
avg_std_og_t2m_mou       25240 non-null float64
avg_std_og_t2f_mou       25240 non-null float64
avg_std_og_mou           25240 non-null float64
avg_isd_og_mou           25240 non-null float64
avg_spl_og_mou           25240 non-null float64
avg_og_others            25240 non-null float64
avg_total_og_mou         25240 non-null float64
avg_loc_ic_t2t_mou       25240 non-null float64
avg_loc_ic_t2m_mou       25240 non-null float64
avg_loc_ic_t2f_mou       25240 non-null float64
avg_loc_ic_mou           25240 non-null float64
avg_std_ic_t2t_mou       25240 non-null float64
avg_std_ic_t2m_mou       25240 non-null float64
avg_std_ic_t2f_mou       25240 non-null float64
avg_std_ic_mou           25240 non-null float64
avg_total_ic_mou         25240 non-null float64
avg_spl_ic_mou           25240 non-null float64
avg_isd_ic_mou           25240 non-null float64
avg_ic_others            25240 non-null float64
avg_total_rech_num       25240 non-null float64
avg_total_rech_amt       25240 non-null float64
avg_max_rech_amt         25240 non-null float64
avg_last_day_rch_amt     25240 non-null float64
avg_vol_2g_mb            25240 non-null float64
avg_vol_3g_mb            25240 non-null float64
avg_monthly_2g           25240 non-null float64
avg_sachet_2g            25240 non-null float64
avg_monthly_3g           25240 non-null float64
avg_sachet_3g_6          25240 non-null float64
avg_vbc_3g               25240 non-null float64
avg_days_between_rchg    25240 non-null float64
aon                      25240 non-null int64
avg_rech_good_phase      25240 non-null float64
churn_tag                25240 non-null int64
dtypes: float64(43), int64(2)
memory usage: 8.9 MB
```

In [22]:

```
master_df_derived.head()
```

Out[22]:

	avg_arpu	avg_onnet_mou	avg_offnet_mou	avg_roam_ic_mou	avg_roam_og_mou	avg_l
13	430.59	381.46	338.35	79.96	14.89	
19	243.86	8.34	17.08	0.00	0.00	
23	211.74	44.40	126.89	0.00	0.00	
24	378.86	106.83	368.06	7.71	10.94	
25	234.20	240.19	147.24	0.00	0.00	

Check for Outliers

In [23]:

```
## Check the outliers at 25%,50%,75%,90%,95% and 99%
master_df_derived.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[23]:

	avg_arpu	avg_onnet_mou	avg_offnet_mou	avg_roam_ic_mou	avg_roam_og_mou
count	25240.00000	25240.000000	25240.000000	25240.000000	25240.000000
mean	480.36802	223.517150	336.247213	13.575743	21.914723
std	452.73720	385.638711	407.858235	56.813798	86.849717
min	40.01000	0.000000	0.000000	0.000000	0.000000
25%	238.04250	29.107500	97.977500	0.000000	0.000000
50%	380.80000	86.550000	212.680000	0.000000	0.000000
75%	604.91250	241.785000	421.272500	4.730000	8.302500
90%	900.47900	587.954000	763.312000	29.160000	48.840000
95%	1150.00200	929.143000	1059.316000	65.250000	105.351500
99%	1841.83700	1831.228600	1986.094800	237.236200	384.454700
max	32140.18000	7104.600000	10059.140000	2199.730000	3298.940000

In [24]:

```
## Clear all the rows above 99 percentile, which seem to be an outliers
master_df_derived = master_df_derived[master_df_derived['avg_arpu'] < 1842]
master_df_derived = master_df_derived[master_df_derived['avg_onnet_mou'] < 1777]
master_df_derived = master_df_derived[master_df_derived['avg_offnet_mou'] < 1020]
master_df_derived = master_df_derived[master_df_derived['avg_roam_ic_mou'] < 222]
master_df_derived = master_df_derived[master_df_derived['avg_roam_ic_mou'] < 224]
master_df_derived = master_df_derived[master_df_derived['avg_roam_og_mou'] < 224]
master_df_derived = master_df_derived[master_df_derived['avg_loc_og_t2t_mou'] < 684]
master_df_derived = master_df_derived[master_df_derived['avg_loc_og_t2f_mou'] < 89]
master_df_derived = master_df_derived[master_df_derived['avg_loc_og_t2c_mou'] < 16]

master_df_derived = master_df_derived[master_df_derived['avg_std_og_t2f_mou'] < 44]

master_df_derived = master_df_derived[master_df_derived['avg_isd_og_mou'] < 22]

master_df_derived = master_df_derived[master_df_derived['avg_vol_3g_mb'] < 2835]

master_df_derived = master_df_derived[master_df_derived['avg_max_rech_amt'] < 616]

master_df_derived = master_df_derived[master_df_derived['avg_last_day_rch_amt'] < 410]

master_df_derived = master_df_derived[master_df_derived['avg_vol_2g_mb'] < 1037]


master_df_derived.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[24]:

	avg_arpu	avg_onnet_mou	avg_offnet_mou	avg_roam_ic_mou	avg_roam_og_mou
count	21169.000000	21169.000000	21169.000000	21169.000000	21169.000000
mean	390.380895	180.444936	252.403464	7.400101	11.226186
std	235.043207	265.965922	218.574046	21.096984	28.980194
min	40.010000	0.000000	0.000000	0.000000	0.000000
25%	219.840000	26.250000	88.630000	0.000000	0.000000
50%	337.460000	77.300000	189.400000	0.000000	0.000000
75%	507.760000	209.980000	353.950000	3.630000	6.150000
90%	707.260000	493.160000	575.324000	21.112000	35.552000
95%	845.570000	745.900000	730.130000	43.536000	68.650000
99%	1150.012800	1352.236400	935.196000	110.842800	157.666000
max	1822.180000	1772.360000	1019.680000	220.900000	223.050000

In [25]:

```
master_df_derived['churn_tag'] = master_df_derived.churn_tag.astype(int)
```

Model Building

Split the data into train and test

In [26]:

```
from sklearn.model_selection import train_test_split

# Putting feature variable to X
X = master_df_derived.drop('churn_tag', axis = 1)

# Putting response variable to y
y = master_df_derived['churn_tag']

## Normalizing all the columns with continuous values and round them to nearest 1 decim
al value
X = round(((X -X.mean())/X.std()),1)

## Split the data
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7,test_size=0.3,r
andom_state=100)
```

In [27]:

```
X_train.isna().sum()
```

Out[27]:

```
avg_arpu                  0
avg_onnet_mou             0
avg_offnet_mou            0
avg_roam_ic_mou           0
avg_roam_og_mou           0
avg_loc_og_t2t_mou        0
avg_loc_og_t2m_mou        0
avg_loc_og_t2f_mou        0
avg_loc_og_t2c_mou        0
avg_loc_og_mou            0
avg_std_og_t2t_mou        0
avg_std_og_t2m_mou        0
avg_std_og_t2f_mou        0
avg_std_og_mou            0
avg_isd_og_mou            0
avg_spl_og_mou            0
avg_og_others             0
avg_total_og_mou          0
avg_loc_ic_t2t_mou        0
avg_loc_ic_t2m_mou        0
avg_loc_ic_t2f_mou        0
avg_loc_ic_mou            0
avg_std_ic_t2t_mou        0
avg_std_ic_t2m_mou        0
avg_std_ic_t2f_mou        0
avg_std_ic_mou            0
avg_total_ic_mou          0
avg_spl_ic_mou            0
avg_isd_ic_mou            0
avg_ic_others             0
avg_total_rech_num        0
avg_total_rech_amt        0
avg_max_rech_amt          0
avg_last_day_rch_amt      0
avg_vol_2g_mb             0
avg_vol_3g_mb             0
avg_monthly_2g            0
avg_sachet_2g             0
avg_monthly_3g            0
avg_sachet_3g_6           0
avg_vbc_3g                0
avg_days_between_rchg     0
aon                       0
avg_rech_good_phase       0
dtype: int64
```

Check the corelation of the features w.r.t target churn column

In [28]:

```
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (20,10))
sns.heatmap(master_df_derived.corr(),annot = True)
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1812eb8bac8>
```

Checking for the class imbalance

In [29]:

```
print(round((y.sum()/y.count())*100,2), '%')
```

```
2.38 %
```

Only 2.4% of the data is tagged for churn. There is a high class imbalance. As a first step, proceed building the model and see how its performance translates

Let us do PCA and build first model to see the impacts of class imbalance

In [30]:

```
from sklearn.decomposition import PCA
pca = PCA(random_state=42)
pca.fit(X_train)
```

Out[30]:

```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=42,
    svd_solver='auto', tol=0.0, whiten=False)
```

Components of the PCA

In [31]:

```
pca.components_
```

Out[31]:

```
array([[ 3.31530564e-01,  1.79483226e-01,  2.70535412e-01, ...,
        -6.00324744e-02,  4.43826685e-02,  4.53587240e-02],
       [-6.15823582e-02, -3.00551376e-01, -9.21597136e-02, ...,
        -2.04675540e-02,  1.61087626e-01,  3.93098889e-02],
       [-1.77515911e-01,  2.33577402e-02,  1.06169481e-01, ...,
        -5.08735973e-03,  6.72524106e-02, -8.70021185e-02],
       ...,
       [ 3.18519775e-03, -7.44840682e-02, -5.98837376e-02, ...,
        -2.92414749e-04,  9.09577012e-04,  3.62690242e-04],
       [-9.79573279e-04, -2.24990601e-01, -1.89506392e-01, ...,
        -3.64981600e-05, -3.75915647e-06,  2.83322498e-04],
       [-6.21878025e-04,  1.55812371e-01,  1.20535881e-01, ...,
        -5.76767756e-04,  3.69225826e-04,  3.43348509e-04]])
```

In [32]:

```
pca.explained_variance_ratio_
```

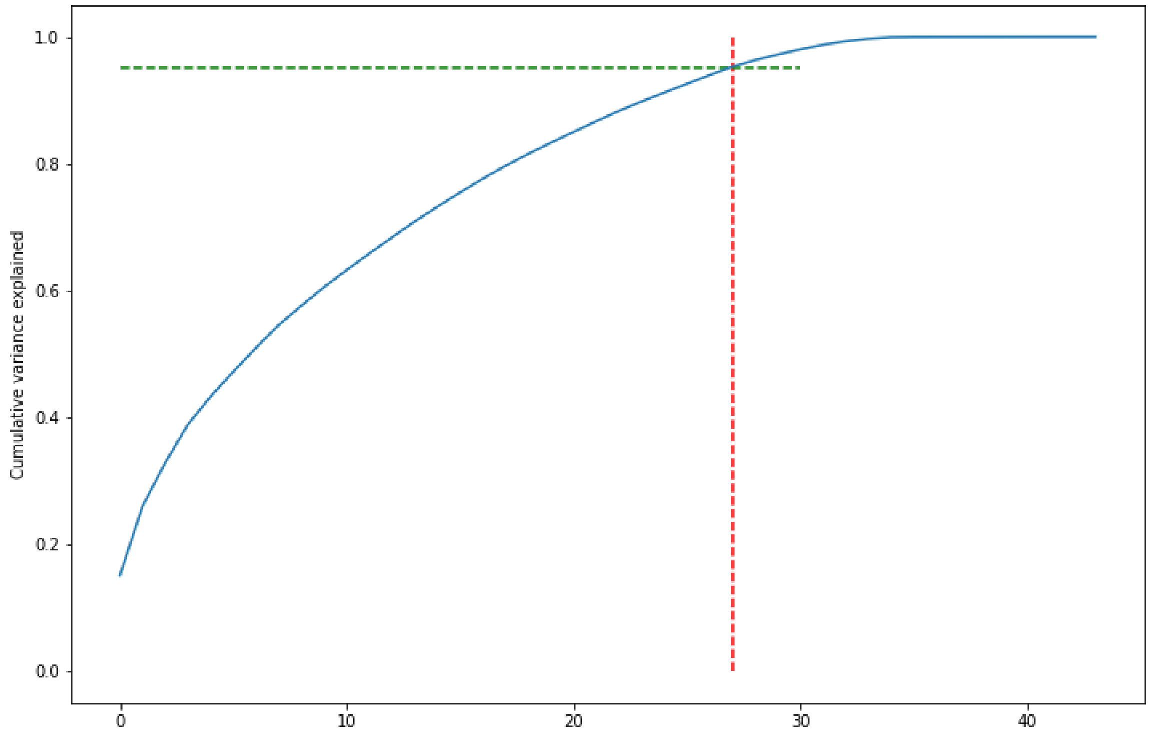
Out[32]:

```
array([1.50652948e-01, 1.09219810e-01, 6.85751171e-02, 5.99454503e-02,
       4.44021221e-02, 3.91160502e-02, 3.76651997e-02, 3.57502169e-02,
       3.05299879e-02, 2.93558350e-02, 2.70843733e-02, 2.61960343e-02,
       2.52625265e-02, 2.48155984e-02, 2.35185987e-02, 2.24380526e-02,
       2.17304772e-02, 2.02588244e-02, 1.88320733e-02, 1.76351471e-02,
       1.70110026e-02, 1.67100997e-02, 1.63895509e-02, 1.47929021e-02,
       1.41997955e-02, 1.38123901e-02, 1.36355614e-02, 1.32303305e-02,
       1.04480908e-02, 8.78388745e-03, 8.13980916e-03, 7.30754888e-03,
       5.57891160e-03, 3.86042714e-03, 2.51567154e-03, 4.23315648e-04,
       4.16116818e-05, 2.60665633e-05, 1.97904845e-05, 1.88418942e-05,
       1.87022868e-05, 1.79977374e-05, 1.71942429e-05, 1.60574199e-05])
```

Creating a scree plot of the variance

In [33]:

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

fig = plt.figure(figsize=[12,8])
plt.vlines(x=27, ymax=1, ymin=0, colors="r", linestyles="--")
plt.hlines(y=0.95, xmax=30, xmin=0, colors="g", linestyles="--")
plt.plot(var_cumu)
plt.ylabel("Cumulative variance explained")
plt.show()
```



Perform PCA with 27 components

In [34]:

```
from sklearn.decomposition import IncrementalPCA
pca_final = IncrementalPCA(n_components=27)
df_train_pca = pca_final.fit_transform(X_train)
df_train_pca.shape
```

Out[34]:

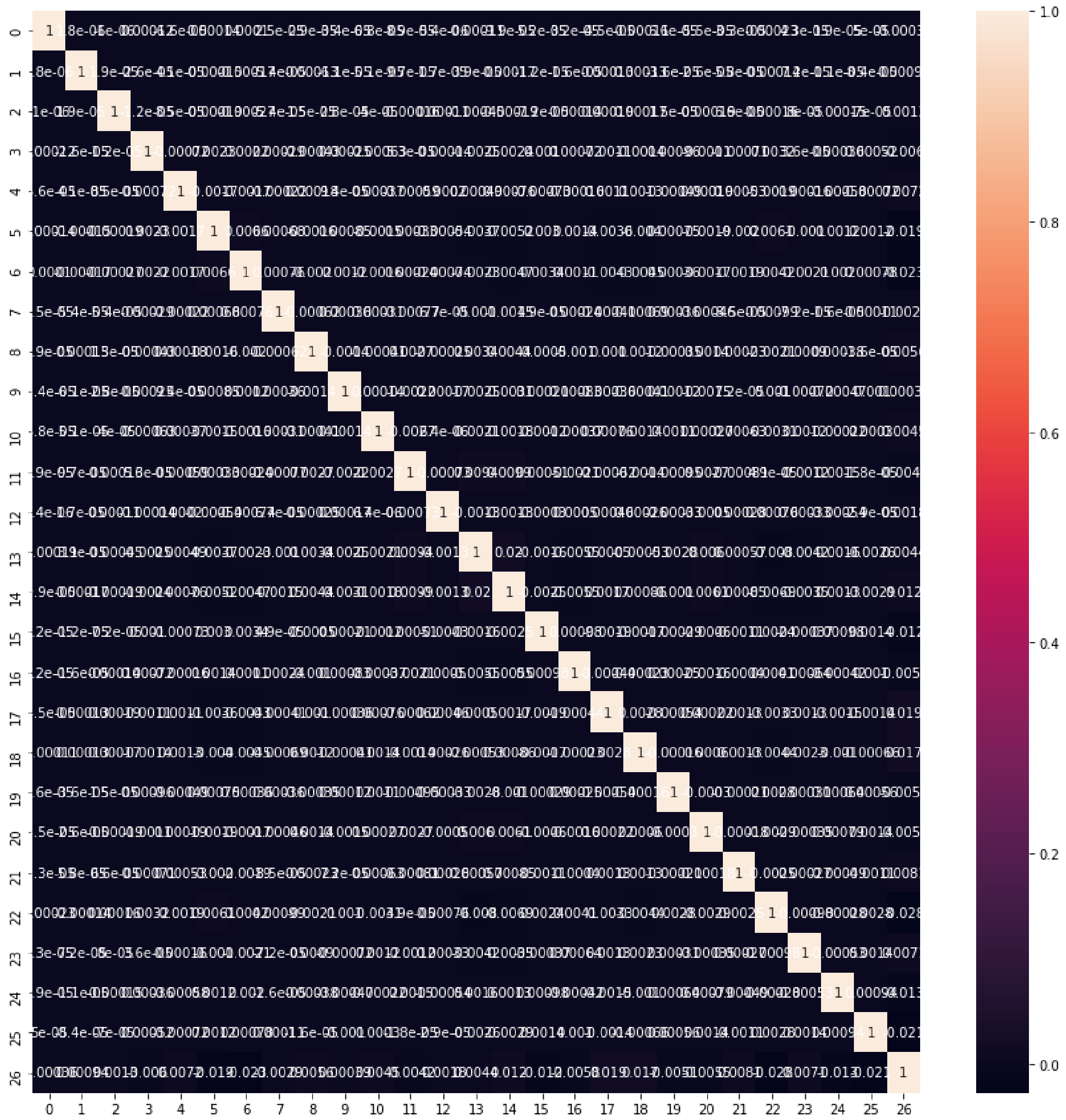
```
(14818, 27)
```

In [35]:

```
corrmat = np.corrcoef(df_train_pca.transpose())

plt.figure(figsize=[15,15])
sns.heatmap(corrmat, annot=True)
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1812ed34be0>
```



Applying the transformation on test data set

In [36]:

```
df_test_pca = pca_final.transform(X_test)
df_test_pca.shape
```

Out[36]:

```
(6351, 27)
```

In [37]:

```
df_train_pca.shape
```

Out[37]:

```
(14818, 27)
```

In [38]:

```
y_train.shape
```

Out[38]:

```
(14818,)
```

Applying the logistic regression on principal components

In [39]:

```
from sklearn.linear_model import LogisticRegression
learner_pca = LogisticRegression()
model_pca = learner_pca.fit(df_train_pca, y_train)
```

Making predictions on the test set

In [40]:

```
pred_probs_test = model_pca.predict_proba(df_test_pca)
pred_probs_test
```

Out[40]:

```
array([[9.92210168e-01, 7.78983152e-03],
       [9.79338691e-01, 2.06613088e-02],
       [9.90951731e-01, 9.04826940e-03],
       ...,
       [9.99829803e-01, 1.70196916e-04],
       [9.74246948e-01, 2.57530517e-02],
       [9.33891914e-01, 6.61080857e-02]])
```

Model Evaluation

AUC Score

In [41]:

```
from sklearn import metrics
"{:2.2}".format(metrics.roc_auc_score(y_test, pred_probs_test[:,1]))
```

Out[41]:

```
'0.88'
```

Accuracy

In [42]:

```
from sklearn.metrics import accuracy_score

df_test_pca = pca_final.fit_transform(X_test)
lr_pred = learner_pca.predict(df_test_pca)
accuracy_score(y_test, lr_pred)
```

Out[42]:

```
0.9707132735002362
```

F1 Score

In [43]:

```
from sklearn.metrics import f1_score
f1_score(y_test, lr_pred)
```

Out[43]:

```
0.0
```

Recall Score

Such a low recall score indicates high number of false negatives - which is not good

In [44]:

```
from sklearn.metrics import recall_score
recall_score(y_test, lr_pred)
```

Out[44]:

```
0.0
```

As you can see above - Though the accuracy of the model is quite high, the F1score/Recall scores are quite low. This is classic example of damage caused by Class Imbalance

Handling Class Imbalance

Method 1 - change the algorithm to Decision Trees

In [45]:

```
from sklearn.ensemble import RandomForestClassifier

df_train_pca1 = pca_final.fit_transform(X_train)
rfc = RandomForestClassifier(n_estimators=10).fit(df_train_pca1, y_train)

# predict on test set
df_test_pca1 = pca_final.transform(X_test)
rfc_pred = rfc.predict(df_test_pca1)
```

In [46]:

```
accuracy_score(y_test, rfc_pred)
```

Out[46]:

```
0.9744922059518186
```

In [47]:

```
f1_score(y_test, rfc_pred)
```

Out[47]:

```
0.024096385542168676
```

In [48]:

```
recall_score(y_test, rfc_pred)
```

Out[48]:

```
0.0125
```

There is an increase in all three metrics - which clearly shows that the Decision Trees (Randowm Forests) outperform the regression algorithms in the case of class imbalance

Method 2 - Resampling : Oversample minority class

In this case - pick more samples in the training data those are marked for chrun

In [49]:

```
from sklearn.utils import resample

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_churn = X[X.churn_tag==0]
churn = X[X.churn_tag==1]

# upsample minority
churn_upsampled = resample(churn,
                          replace=True, # sample with replacement
                          n_samples=len(not_churn), # match number in majority class
                          random_state=27) # reproducible results

# combine majority and upsampled minority
oversampled = pd.concat([not_churn, churn_upsampled])
```

Apply logical regression on the oversampled data

In [50]:

```
# trying logistic regression again with the balanced dataset
y_train = oversampled.churn_tag
X_train = oversampled.drop('churn_tag', axis=1)

df_train_pca2 = pca_final.fit_transform(X_train)
upsampled = learner_pca.fit(df_train_pca2, y_train)

df_test_pca2 = pca_final.transform(X_test)
upsampled_pred = upsampled.predict(df_test_pca2)
```

In [51]:

```
print(upsampled.coef_)
```

```
[[-0.09558195 -1.14498986  0.52586585  0.31314456 -0.04482715  0.11506998
   0.22972895  0.10505888 -0.09390976 -0.1283478   0.18507663 -0.06139957
   0.02643086  0.21038774 -0.2264266  -0.22199149  0.02870295 -0.59232924
  -0.28593426  0.01987363 -0.5268877  -0.28027742 -0.18891298  0.54260031
  -0.35486406 -0.49706145 -0.31150594]]
```

In [52]:

```
accuracy_score(y_test, upsampled_pred)
```

Out[52]:

```
0.75484175720359
```

In [53]:

```
f1_score(y_test, upsampled_pred)
```

Out[53]:

```
0.15149863760217985
```

In [54]:

```
recall_score(y_test, upsampled_pred)
```

Out[54]:

```
0.86875
```

While there is a drop in the accuracy - there is a huge increase in the F1 score and Recall scores.

Method 3 - Resampling : Undersample majority class

In [55]:

```
# downsample majority
not_churn_downsampled = resample(not_churn,
                                replace = False, # sample without replacement
                                n_samples = len(churn), # match minority n
                                random_state = 27) # reproducible results

# combine minority and downsampled majority
downsampled = pd.concat([not_churn_downsampled, churn])
```

In [56]:

```
# trying logistic regression again with the balanced dataset
y_train = downsampled.churn_tag
X_train = downsampled.drop('churn_tag', axis=1)

df_train_pca3 = pca_final.fit_transform(X_train)
resampled_down = learner_pca.fit(df_train_pca3, y_train)

df_test_pca3 = pca_final.transform(X_test)
downsampled_pred = resampled_down.predict(df_test_pca3)
```

In [57]:

```
accuracy_score(y_test, downsampled_pred)
```

Out[57]:

```
0.7474413478192411
```

In [58]:

```
f1_score(y_test, downsampled_pred)
```

Out[58]:

```
0.14771519659936239
```

In [59]:

```
recall_score(y_test, downsampled_pred)
```

Out[59]:

```
0.86875
```

Though the overall performance remained same - there is a bit of improvement in the recall_score which reduces the false negatives by a margin

Based on the above models outcome - it is recommended to use the Model 2 to work around the Class Imbalance problem

Finding the top features contributing to the model and thereby influence the churn rate

Method 1 - Using the co-efficients from PCA

In [60]:

```
pd.set_option('display.max_columns', 500)
disp_df = pd.DataFrame(pca_final.components_,columns=X_train.columns,
                      index = ['PC-1','PC-2','PC-3','PC-4','PC-5','PC-6',
                               'PC-7','PC-8','PC-9','PC-10','PC-11','PC-12',
                               'PC-13','PC-14','PC-15','PC-16','PC-17','PC-18',
                               'PC-19','PC-20','PC-21','PC-22','PC-23','PC-24',
                               'PC-25','PC-26', 'PC-27'])

disp_df.head()
```

Out[60]:

	avg_arpu	avg_onnet_mou	avg_offnet_mou	avg_roam_ic_mou	avg_roam_og_mou	avg_
PC-1	0.311041	0.297957	0.250393	-0.010079	-0.026757	
PC-2	0.061697	-0.252108	0.013271	0.012110	-0.096180	
PC-3	-0.015766	-0.091547	0.051213	0.404569	0.355649	
PC-4	0.160203	0.024651	0.033680	0.508351	0.582570	
PC-5	0.066954	0.116558	-0.300698	0.016925	-0.033801	

Finding the top 2 features those influence the top 5 proncipal components

In [61]:

```
disp_dft = disp_df.T

pc1_df = disp_dft.drop(['PC-2','PC-3','PC-4','PC-5','PC-6','PC-7','PC-8','PC-9','PC-10'
,'PC-11','PC-12','PC-13','PC-14',
                          'PC-15','PC-16','PC-17','PC-18','PC-19','PC-20','PC-21','PC-22'
,'PC-23','PC-24','PC-25','PC-26', 'PC-27'], 1)
pc1_df = pc1_df.sort_values(by = 'PC-1', ascending = False)
pc1_top = list(pc1_df.head(2).index)

pc2_df = disp_dft.drop(['PC-1','PC-3','PC-4','PC-5','PC-6','PC-7','PC-8','PC-9','PC-10'
,'PC-11','PC-12','PC-13','PC-14',
                          'PC-15','PC-16','PC-17','PC-18','PC-19','PC-20','PC-21','PC-22'
,'PC-23','PC-24','PC-25','PC-26', 'PC-27'], 1)
pc2_df = pc2_df.sort_values(by = 'PC-2', ascending = False)
pc2_top = list(pc2_df.head(2).index)

pc3_df = disp_dft.drop(['PC-1','PC-2','PC-4','PC-5','PC-6','PC-7','PC-8','PC-9','PC-10'
,'PC-11','PC-12','PC-13','PC-14',
                          'PC-15','PC-16','PC-17','PC-18','PC-19','PC-20','PC-21','PC-22'
,'PC-23','PC-24','PC-25','PC-26', 'PC-27'], 1)
pc3_df = pc3_df.sort_values(by = 'PC-3', ascending = False)
pc3_top = list(pc3_df.head(2).index)


pc4_df = disp_dft.drop(['PC-1','PC-2','PC-3','PC-5','PC-6','PC-7','PC-8','PC-9','PC-10'
,'PC-11','PC-12', 'PC-13','PC-14',
                          'PC-15','PC-16','PC-17','PC-18','PC-19','PC-20','PC-21','PC-22'
,'PC-23','PC-24','PC-25','PC-26', 'PC-27'], 1)
pc4_df = pc4_df.sort_values(by = 'PC-4', ascending = False)
pc4_top = list(pc4_df.head(2).index)


pc5_df = disp_dft.drop(['PC-1','PC-2','PC-3','PC-4','PC-6','PC-7','PC-8','PC-9','PC-10'
,'PC-11','PC-12', 'PC-13','PC-14',
                          'PC-15','PC-16','PC-17','PC-18','PC-19','PC-20','PC-21','PC-22'
,'PC-23','PC-24','PC-25','PC-26', 'PC-27'], 1)
pc5_df = pc5_df.sort_values(by = 'PC-5', ascending = False)
pc5_top = list(pc5_df.head(2).index)


print(pc1_top)
print(pc2_top)
print(pc3_top)
print(pc4_top)
print(pc5_top)
```

```
['avg_total_og_mou', 'avg_std_og_mou']
['avg_total_ic_mou', 'avg_loc_ic_mou']
['avg_std_ic_t2m_mou', 'avg_std_ic_mou']
['avg_roam_og_mou', 'avg_roam_ic_mou']
['avg_vol_3g_mb', 'avg_vbc_3g']
```

In [62]:

```
import itertools

top_features_set = set(itertools.chain(pc1_top,pc2_top,pc3_top,pc4_top,pc5_top))
top_features_set
```

Out[62]:

```
{'avg_loc_ic_mou',
 'avg_roam_ic_mou',
 'avg_roam_og_mou',
 'avg_std_ic_mou',
 'avg_std_ic_t2m_mou',
 'avg_std_og_mou',
 'avg_total_ic_mou',
 'avg_total_og_mou',
 'avg_vbc_3g',
 'avg_vol_3g_mb'}
```

Though this is a way to find the top features - PCA analysis crumbles the co-efficients for the purpose of deriving combined variance, and so it may not be practical to identify the top contributing features correctly.

Let us try with Lasso - as the co-efficients from the Lasso are linear

Method 2 - Using the Lasso regression

In [63]:

```
# hide warnings
import warnings
warnings.filterwarnings('ignore')

from sklearn import linear_model
```

In [64]:

```
from sklearn.linear_model import LassoCV
lasso_cv = LassoCV(alphas=[0.0001, 0.001, 0.01, 0.05, 0.1,
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ])

model_lasso_cv = lasso_cv.fit(X_train, y_train)
model_lasso_cv.alpha_
```

Out[64]:

```
0.01
```

In [65]:

```
alpha =0.01

lasso = linear_model.Lasso(alpha=alpha)

lasso.fit(X_train, y_train)
```

Out[65]:

```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=Non
e,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [66]:

```
lasso.coef_
```

Out[66]:

```
array([-0.        ,  0.        , -0.        ,  0.03823426,  0.07397634,
       -0.        , -0.02694079, -0.03037053, -0.0203785 , -0.08070232,
        0.        ,  0.03048331, -0.        ,  0.02436209, -0.00195132,
        0.        , -0.        ,  0.        , -0.        , -0.0308791 ,
       -0.02814693, -0.00752667, -0.        , -0.01164545, -0.        ,
       -0.00120627, -0.        , -0.03283979,  0.00663063,  0.        ,
        0.        , -0.        , -0.04810913, -0.0595583 , -0.01694236,
       -0.00447921, -0.02635048, -0.01306067, -0.00560456, -0.        ,
       -0.00661127, -0.06070582, -0.02081517,  0.01818021])
```

Find the top 10 predictors using the Lasso regression

In [67]:

```
arr_lasso = lasso.coef_
val_list = list(arr_lasso)
col_list = list(X_train.columns)

df_lasso = pd.DataFrame(columns = ['Features', 'Lasso_coeff'])
df_lasso['Features'] = col_list
df_lasso['Lasso_coeff'] = val_list
df_lasso = df_lasso.sort_values(by = 'Lasso_coeff', ascending = False)
df_lasso = df_lasso.set_index('Features')

df1_lasso = df_lasso.head()
df2_lasso = df_lasso.tail()
df_combined_lasso = pd.concat([df_lasso.head(),df_lasso.tail()])

df_combined_lasso['Lasso_coeff'] = list(map(lambda x : abs(x), list(df_combined_lasso[
'Lasso_coeff'])))
df_combined_lasso = df_combined_lasso.sort_values(by = 'Lasso_coeff', ascending = False
).head(10)
top10_lasso_features = list(df_combined_lasso.index)
top10_lasso_features
```

Out[67]:

```
['avg_loc_og_mou',
 'avg_roam_og_mou',
 'avg_days_between_rchg',
 'avg_last_day_rch_amt',
 'avg_max_rech_amt',
 'avg_roam_ic_mou',
 'avg_spl_ic_mou',
 'avg_std_og_t2m_mou',
 'avg_std_og_mou',
 'avg_rech_good_phase']
```

Let us use the list from Lasso regression as Lasso practically a better approach to confirm the key features - it also marks co-efficients to zero if the features dont add any values to the model building

Visual representation of top 10 key features

In [68]:

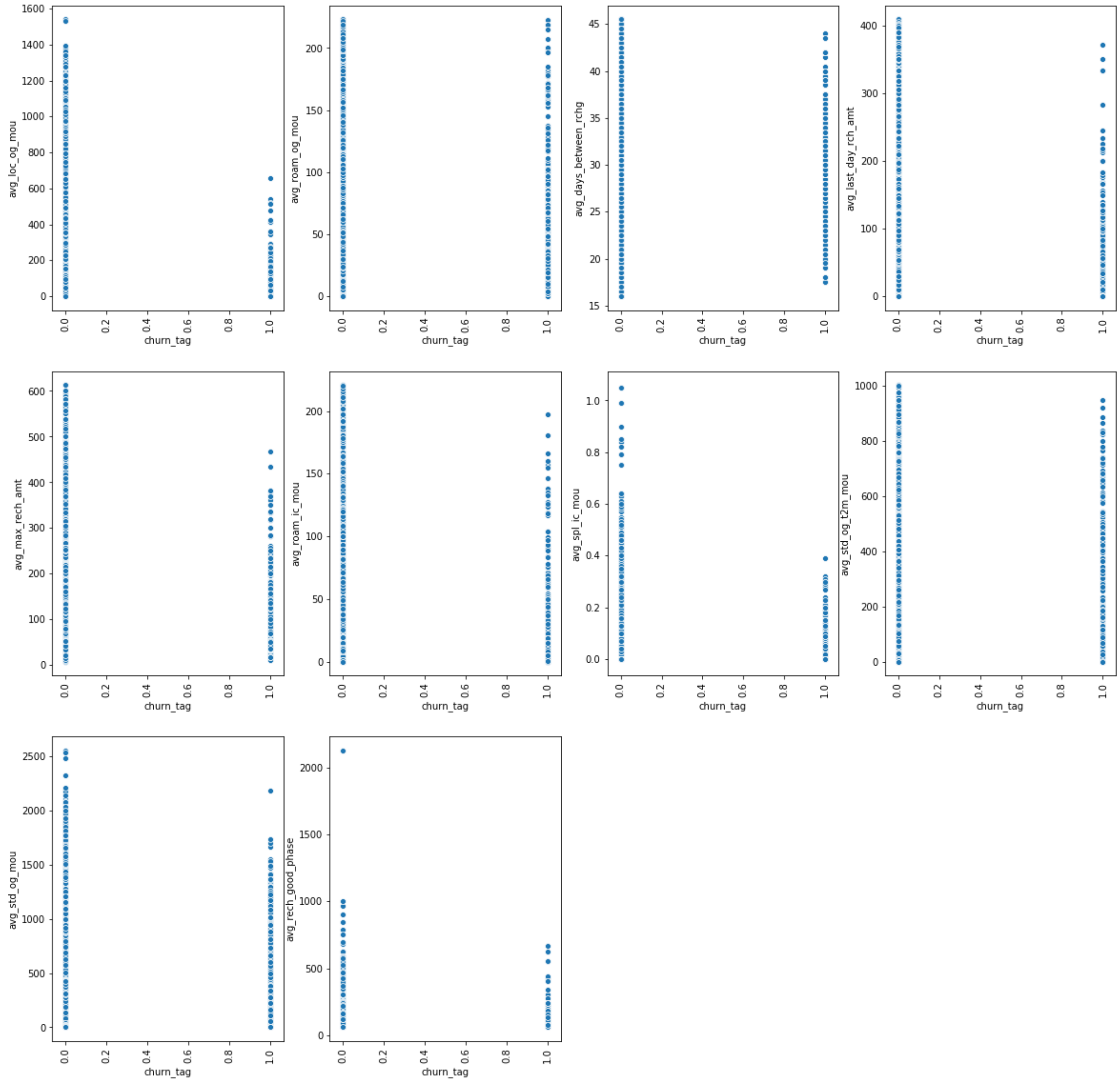
```
df_display = master_df_derived[top10_lasso_features]
df_display['churn_tag'] = master_df_derived['churn_tag']
df_display.head()
```

Out[68]:

	avg_loc_og_mou	avg_roam_og_mou	avg_days_between_rchg	avg_last_day_rch_amt	avg
13	204.26	14.89	35.5	53.33	
19	12.01	0.00	30.0	102.67	
23	149.24	0.00	33.5	59.67	
24	198.51	10.94	34.5	116.67	
25	123.45	0.00	25.0	0.00	

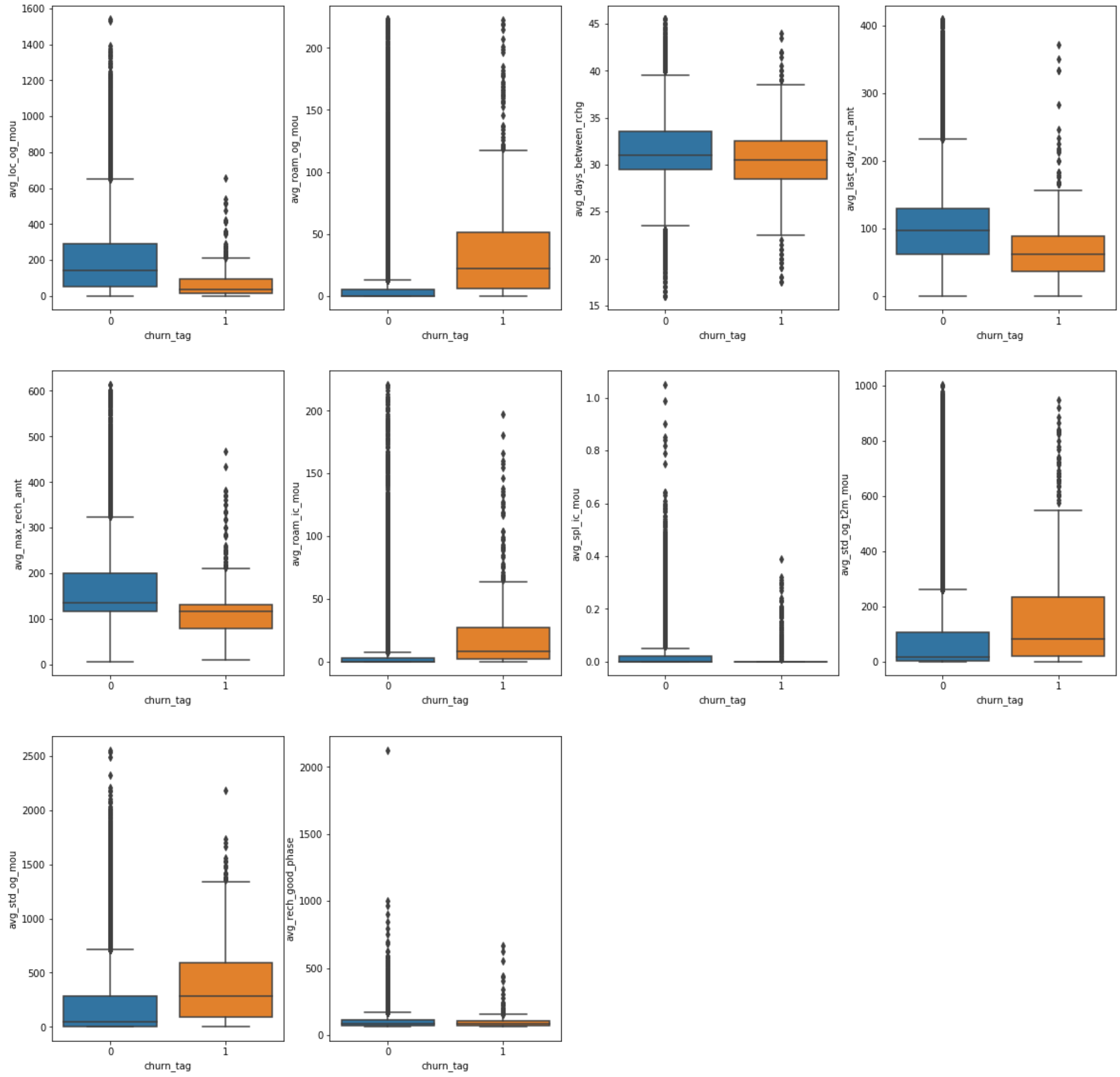
In [69]:

```
import matplotlib.pyplot as plt
import seaborn as sns

fig_dropout = plt.figure(figsize = (20,20))

cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)
count_plot = 1

for each_item in list(df_display.columns):
    if each_item != 'churn_tag':
        plt.subplot(3,4,count_plot)
        snsplot = sns.scatterplot(y = each_item , x = 'churn_tag', data=df_display, pal
ette = cmap)
        plt.xticks(rotation='vertical')
        count_plot = count_plot+1

plt.show()
```


avg_loc_og_mou
churn_tag
avg_roam_og_mou
churn_tag
avg_days_between_rchg
churn_tag
avg_last_day_rch_amt
churn_tag
avg_max_rech_amt
churn_tag
avg_roam_ic_mou
churn_tag
avg_spl_ic_mou
churn_tag
avg_std_og_t2m_mou
churn_tag
avg_std_og_mou
churn_tag
avg_rech_good_phase
churn_tag

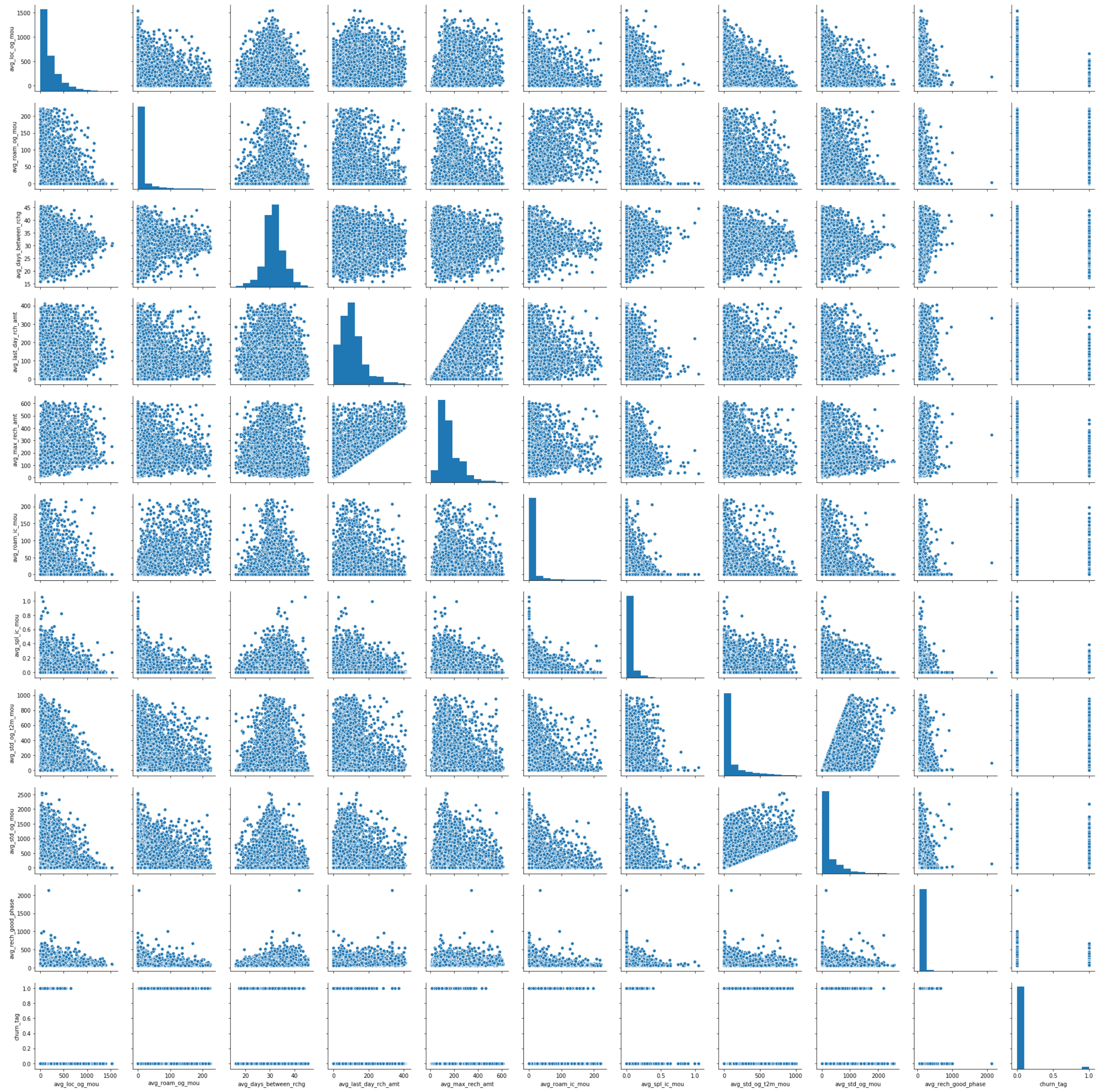
In [70]:

```
dist_count = 1

fig_dropout = plt.figure(figsize = (20,20))

for each_item in list(df_display.columns):
    if each_item != 'churn_tag':
        plt.subplot(3,4,dist_count)
        sns.boxplot(x= 'churn_tag', y=each_item, data=df_display)
        dist_count = dist_count+1

#plt.show()
```



In [71]:

```
sns.pairplot(df_display)
plt.show()
```



Key obervations and Recommended strategies

Key Observations

Based on the analysis of high value customers, the 'potential' churn customers appear to be exhibiting the below behaviour in comparison to the non-churn customers:

1. Spend less usage on the outgoing local calls
2. Spend very high on the calls, both outgoing and incoming, when they are away (on roaming)
3. Recharge lesser amounts on the last day of balance expiry
4. Recharge amounts are lesser for each recharge
5. Have high outgoing minutes of usage in both standard and t2m

Also note that -

1. There is no much difference between number of days between the recharges between churners and non-churners
2. The recharge patterns are similar between churners and non-churners in 'good phase' a. This indicates that there would be a drastic change in pattern in 'action' phase

Key Recommendation

This telecom operator seems to be losing the customers whose usage is very high in outgoing minutes and also the customers who travel a lot (roamers). It is recommended to introduce offers or discounts around outgoing calls and special travel packages for roamers.

The high-value customers represent only 20% of the overall customer base under observation. While it is definitely a value-add to business bottomline by retaining the high-value customers, it is also recommended to see the churn pattern in low-value customers. After all they represent 80% of the customer base and could hit the bottomline drastically if they start churning out in lesser time.

In [ ]:

```

```