TELECOM CHURN CASE STUDY

ANN MARY PHILIP

PROBLEM STATEMENT

In the telecom industry, customers are able to choose from multiple service providers and actively switch from one operator to another. In this highly competitive market, the telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition. To reduce customer churn, telecom companies need to predict which customers are at high risk of churn.
In this project, customer-level data of a leading telecom firm is analysed, predictive models are built to identify customers at high risk of churn and the main indicators of churn are identified.
The dataset contains customer-level information for a span of four consecutive months - June, July, August and September. The months are encoded as 6, 7, 8 and 9, respectively. The business objective is to predict the churn in the last (i.e. the ninth) month using the data (features) from the first three months.

ANALYZING THE DATA:

In [6]:
```
#Import libraries
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings('ignore')

#Reading the data from csv
rawdata_read= pd.read_csv('telecom_churn_data.csv', sep=',', encoding='ISO-8859-1')
rawdata_read.head()
```

Out[6]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_of_month_7	last_date_of_month_8	last_date_of
0	7000842753	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
1	7001865778	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
2	7001625959	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
3	7001204172	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
4	7000142493	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	

5 rows × 226 columns

In [9]:
```
#Lets get the distribution of numeric data all the columns.
rawdata_read.describe()
```

Out[9]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	...
count	9.999900e+04	99999.0	98981.0	98981.0	98981.0	99999.000000	99999.000000	99999.000000	99999.000000	96062.000000	...
mean	7.001207e+09	109.0	0.0	0.0	0.0	282.987358	278.536648	279.154731	261.645069	132.395875	...
std	6.956694e+05	0.0	0.0	0.0	0.0	328.439770	338.156291	344.474791	341.998630	297.207406	...
min	7.000000e+09	109.0	0.0	0.0	0.0	-2258.709000	-2014.045000	-945.808000	-1899.505000	0.000000	...
25%	7.000606e+09	109.0	0.0	0.0	0.0	93.411500	86.980500	84.126000	62.685000	7.380000	...
50%	7.001205e+09	109.0	0.0	0.0	0.0	197.704000	191.640000	192.080000	176.849000	34.310000	...
75%	7.001812e+09	109.0	0.0	0.0	0.0	371.060000	365.344500	369.370500	353.466500	118.740000	...
max	7.002411e+09	109.0	0.0	0.0	0.0	27731.088000	35145.834000	33543.624000	38805.617000	7376.710000	...

8 rows × 214 columns

FINDING OUT THE NULL % IN THE DATA SET:

```
In [10]: #Get the null % of all the columns
         print('\n Null % \n',round(100*(rawdata_read.isnull().sum()/len(rawdata_read.index)), 2))
```

```
 Null %
 mobile_number     0.00
circle_id         0.00
loc_og_t2o_mou    1.02
std_og_t2o_mou    1.02
loc_ic_t2o_mou    1.02
                  ...
aon               0.00
vbc_3g_8          0.00
vbc_3g_7          0.00
vbc_3g_6          0.00
vbc_3g_9          0.00
Length: 226, dtype: float64
```

Lets look at the features that we need to filter high value customers. There are no null values in the total_rech_amt columns. But there are null values in av_rech_amt_data and total_rech_data columns. Lets look at them

In [12]:
```
rawdata_read[['av_rech_amt_data_6','av_rech_amt_data_7']].head(10)
```
Out[12]:

	av_rech_amt_data_6	av_rech_amt_data_7
0	252.0	252.0
1	NaN	154.0
2	NaN	NaN
3	NaN	NaN
4	56.0	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	177.0
9	NaN	154.0

n [13]:
```
rawdata_read[['total_rech_data_6','total_rech_data_7']].head(10)
```
ut[13]:

	total_rech_data_6	total_rech_data_7
0	1.0	1.0
1	NaN	1.0
2	NaN	NaN
3	NaN	NaN
4	1.0	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	2.0
9	NaN	1.0

av_rech_amt_data indicates average recharge amount of data while total_rech_data represents recharge done or not done.
They have the same percentage of null values, lets impute them with 0's.

HANDLING THE MISSING VALUES IN DATA SET

```
n [14]: #Fill missing values to filter high value customers
        rawdata_read[['av_rech_amt_data_6','av_rech_amt_data_7','av_rech_amt_data_8','av_rech_amt_data_9','total_rech_data_6','total_rech
```

Let's filter the high value customers from the original set who have recharged with an amount more than or equal to the 70th percentile of the average recharge amount in the first two Months.

```
In [15]: #Get the top 30% customers based on the sum of recharges in month 6 and 7
         high_value_cust=rawdata_read[rawdata_read[['total_rech_amt_6', 'total_rech_amt_7','av_rech_amt_data_6','av_rech_amt_data_7']].mea

In [16]: #Unnderstand the no of datapoints, column and column types
         high_value_cust.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 29949 entries, 0 to 99998
         Columns: 226 entries, mobile_number to vbc_3g_9
         dtypes: float64(179), int64(35), object(12)
         memory usage: 51.9+ MB
```

In [17]:
```
#Getting the string columns in the data frame
str_cols = high_value_cust.select_dtypes(['object'])
#Stripping the leading and trailing whitespaces from the data set
high_value_cust[str_cols.columns]=str_cols.apply(lambda x: x.str.strip())
```

In [18]:
```
#Converting all the string columns to upper case in the Data set
high_value_cust[str_cols.columns]=str_cols.apply(lambda x: x.str.upper())
```

In [19]:
```
#Getting the no of unique mobile numbers, this is an id column
len(high_value_cust['mobile_number'].unique())
```

Out[19]:
```
29949
```

In [20]:
```
#setting the option to display a maximum of 250 rows
pd.set_option("display.max_rows",250)
```

In [21]:
```
#Get null% of all columns
round(100*(high_value_cust.isnull().sum()/len(high_value_cust.index)), 2)
```

Out[21]:
```
mobile_number            0.00
circle_id                0.00
loc_og_t2o_mou           0.74
std_og_t2o_mou           0.74
loc_ic_t2o_mou           0.74
last_date_of_month_6     0.00
last_date_of_month_7     0.09
last_date_of_month_8     0.52
last_date_of_month_9     1.19
arpu_6                   0.00
arpu_7                   0.00
arpu_8                   0.00
arpu_9                   0.00
onnet_mou_6              1.65
onnet_mou_7              1.62
onnet_mou_8              3.63
onnet_mou_9              6.06
offnet_mou_6             1.65
offnet_mou_7             1.62
offnet_mou_8             3.63
offnet_mou_9             6.06
roam_ic_mou_6            1.65
```

```
dtype: float64

In [22]: #Identify churn based on the criteria that the usage of 'vol_3g_mb_9', 'vol_2g_mb_9','total_ic_mou_9','total_og_mou_9' should be
         high_value_cust['churn']=high_value_cust[['vol_3g_mb_9', 'vol_2g_mb_9','total_ic_mou_9','total_og_mou_9']].apply(lambda x: 1 if (

In [23]: #Get only the data where the customers churned
         churned_total = high_value_cust[high_value_cust['churn']==1]

In [24]: #No of churned datapoints
         len(churned_total)

Out[24]: 2451

In [25]: #No of non churned customers
         len(high_value_cust[high_value_cust['churn']==0])

Out[25]: 27498
```

Hence only 2451 data points are present for churned customers, so dropping null rows would significantly affect the minority class

CALCULATING THE NULL PERCENTAGE:

```
In [26]: #Get the null % of columns where the customers churned
         print('\n Null % \n',round(100*(churned_total.isnull().sum()/len(churned_total.index)), 2))
```

```
 Null %
 mobile_number                 0.00
circle_id                     0.00
loc_og_t2o_mou                3.75
std_og_t2o_mou                3.75
loc_ic_t2o_mou                3.75
last_date_of_month_6          0.00
last_date_of_month_7          1.14
last_date_of_month_8          6.32
last_date_of_month_9         14.48
arpu_6                        0.00
arpu_7                        0.00
arpu_8                        0.00
arpu_9                        0.00
onnet_mou_6                   6.16
onnet_mou_7                   8.04
onnet_mou_8                  30.52
onnet_mou_9                  64.46
offnet_mou_6                  6.16
offnet_mou_7                  8.04
offnet_mou_8                 30.52
offnet_mou_9                 64.46
roam_ic_mou_6                 6.16
roam_ic_mou_7                 8.04
roam_ic_mou_8                30.52
roam_ic_mou_9                64.46
roam_og_mou_6                 6.16
roam_og_mou_7                 8.04
roam_og_mou_8                30.52
roam_og_mou_9                64.46
loc_og_t2t_mou_6              6.16
loc_og_t2t_mou_7              8.04
loc_og_t2t_mou_8             30.52
```

The data columns are having high percentage of null values for churned customers wheareas call columns are having low percentage of null values. It could mean most customers are using the mobile service for calls rather than data. Since majority of the columns are having null values we cannot simply drop them. We cant drop the rows as this would further diminish minority class. Before imputingthe values lets look at all the columns having null values

```
In [28]: #Get the data of all the rows with null values in more than 40 columns
         high_value_cust[high_value_cust.isnull().sum(axis=1)>=40]
```

Out[28]:

	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_d
0	7000842753	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
7	7000701601	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
67	7000800341	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
77	7001328263	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
111	7001300706	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
...	...	...	...	...	...	...	...	...	
99726	7000224828	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
99790	7000008246	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
99827	7000231239	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
99961	7000992757	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	
99998	7001905007	109	0.0	0.0	0.0	6/30/2014	7/31/2014	8/31/2014	

1914 rows × 227 columns

```
In [29]: #Lets see which columns are null when 'date_of_last_rech_data_6' are null
         high_value_cust[high_value_cust.isnull().sum(axis=1)>=40].columns[high_value_cust[pd.isnull(high_value_cust['date_of_last_rech_da

Out[29]: ['date_of_last_rech_data_6',
          'max_rech_data_6',
          'count_rech_2g_6',
          'count_rech_3g_6',
          'arpu_3g_6',
          'arpu_2g_6',
          'night_pck_user_6',
          'fb_user_6']

In [30]: #Lets see which columns are null when 'date_of_last_rech_data_7' are null
         high_value_cust[high_value_cust.isnull().sum(axis=1)>=40].columns[high_value_cust[pd.isnull(high_value_cust['date_of_last_rech_da

Out[30]: ['date_of_last_rech_data_7',
          'max_rech_data_7',
          'count_rech_2g_7',
          'count_rech_3g_7',
          'arpu_3g_7',
          'arpu_2g_7',
          'night_pck_user_7',
          'fb_user_7']
```

In [35]:

```
#Convert the date columns from string to date type
dateColumns = ['last_date_of_month_6','last_date_of_month_7','last_date_of_month_8','date_of_last_rech_6','date_of_last_rech_7','
high_value_cust[dateColumns]=high_value_cust[dateColumns].apply(lambda x: pd.to_datetime(x, format='%m/%d/%Y', errors='coerce'))
high_value_cust[dateColumns]
```

Out[35]:

	last_date_of_month_6	last_date_of_month_7	last_date_of_month_8	date_of_last_rech_6	date_of_last_rech_7	date_of_last_rech_8	date_of_last_rech_da
0	2014-06-30	2014-07-31	2014-08-31	2014-06-21	2014-07-16	2014-08-08	2014-0
7	2014-06-30	2014-07-31	2014-08-31	2014-06-27	2014-07-25	2014-08-26	
8	2014-06-30	2014-07-31	2014-08-31	2014-06-25	2014-07-31	2014-08-30	
16	2014-06-30	2014-07-31	2014-08-31	2014-06-30	2014-07-31	2014-08-14	
21	2014-06-30	2014-07-31	2014-08-31	2014-06-30	2014-07-31	2014-08-31	
...	...	...	...	...	...	...	
99984	2014-06-30	2014-07-31	2014-08-31	2014-06-21	2014-07-31	2014-08-27	2014-0
99986	2014-06-30	2014-07-31	2014-08-31	2014-06-20	2014-07-28	2014-08-18	2014-0
99988	2014-06-30	2014-07-31	2014-08-31	2014-06-30	2014-07-28	2014-08-29	
99997	2014-06-30	2014-07-31	2014-08-31	2014-06-17	2014-07-19	2014-08-20	2014-0
99998	2014-06-30	2014-07-31	2014-08-31	2014-06-16	NaT	NaT	2014-0

29949 rows × 9 columns

The missing values of the last recharge date for call or data can be assumed there is no recharge done for that particular month. Let's create new columns which indicate whether a recharge has been done or not for that particular month for both data and calls

In [36]:
```
#Transform the existing date of last recharge columns with 1 indicating a recharge and 0 indicating no recharge
dateColumnsToTransform = ['date_of_last_rech_6','date_of_last_rech_7','date_of_last_rech_8','date_of_last_rech_data_6','date_of_l

high_value_cust[['rech_amt_6','rech_amt_7','rech_amt_8','rech_data_6','rech_data_7','rech_data_8' ]] = high_value_cust[dateColumn
```

In [37]:
```
#Convert the features to category
high_value_cust[['rech_amt_6','rech_amt_7','rech_amt_8','rech_data_6','rech_data_7','rech_data_8' ]] = high_value_cust[['rech_amt
```

In [38]:
```
#Lets look at the columns we created
high_value_cust[['rech_amt_6','rech_amt_7','rech_amt_8','rech_data_6','rech_data_7','rech_data_8' ]]
```

Out[38]:

	rech_amt_6	rech_amt_7	rech_amt_8	rech_data_6	rech_data_7	rech_data_8
0	1	1	1	1	1	1
7	1	1	1	-1	-1	-1
8	1	1	1	-1	1	1
16	1	1	1	-1	-1	-1
21	1	1	1	-1	-1	-1
...	...	...	...	...	...	...
99984	1	1	1	1	1	1
99986	1	1	1	1	1	1
99988	1	1	1	-1	1	1
99997	1	1	1	1	1	1
99998	1	-1	-1	1	-1	-1

29949 rows × 6 columns

In [41]:
```
#Lets get the info of all columns
high_value_cust.describe()
```

Out[41]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	arpu_9	onnet_mou_6	...
count	2.994900e+04	29949.0	29726.0	29726.0	29726.0	29949.000000	29949.000000	29949.000000	29949.000000	29455.000000	...
mean	7.001216e+09	109.0	0.0	0.0	0.0	577.006877	578.921924	525.105455	481.919263	281.138675	...
std	6.867758e+05	0.0	0.0	0.0	0.0	449.261901	469.335744	496.373165	498.689560	463.649942	...
min	7.000000e+09	109.0	0.0	0.0	0.0	-2258.709000	-2014.045000	-945.808000	-1899.505000	0.000000	...
25%	7.000631e+09	109.0	0.0	0.0	0.0	345.808000	347.071000	266.252000	219.637000	29.030000	...
50%	7.001221e+09	109.0	0.0	0.0	0.0	490.933000	489.043000	443.470000	404.792000	106.610000	...
75%	7.001806e+09	109.0	0.0	0.0	0.0	702.776000	700.512000	666.980000	632.975000	325.685000	...
max	7.002411e+09	109.0	0.0	0.0	0.0	27731.088000	35145.834000	33543.624000	38805.617000	7376.710000	...

8 rows × 215 columns

CHECKING FOR ALL THE COLUMNS WITH NULL VALUES

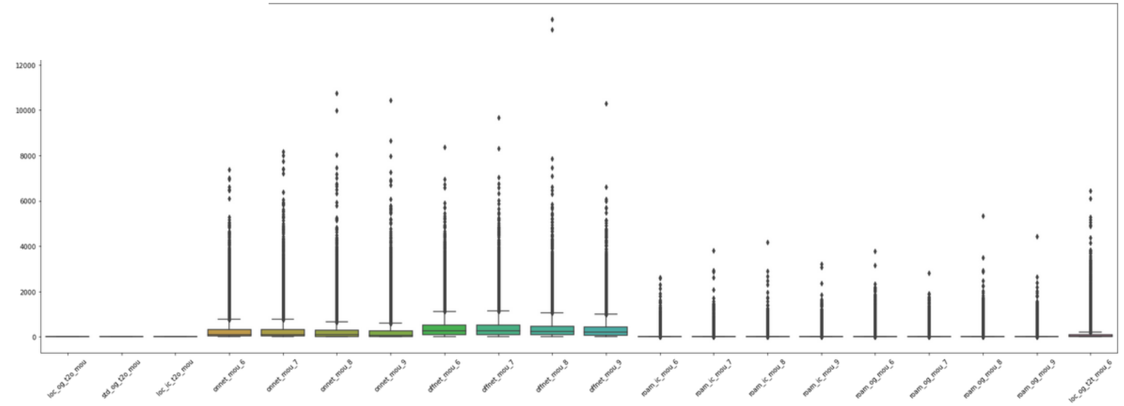
In [42]:
```
#Get all the null columns
null_cols=high_value_cust.columns[high_value_cust.isna().any()].tolist()
```

In [43]:
```
#Date columns to list
datecols=['date_of_last_rech_6',
'date_of_last_rech_7',
'date_of_last_rech_8',
'date_of_last_rech_9',
'date_of_last_rech_data_6',
'date_of_last_rech_data_7',
'date_of_last_rech_data_8',
'date_of_last_rech_data_9',
'last_date_of_month_9']
```

In [44]:
```
#Get numerical columns to explore
cols_explore  = [x for x in null_cols if x not in datecols]
```

In [45]:
```


plt.figure(figsize = (30,10))
ax=sns.boxplot(data=high_value_cust[cols_explore[:20]])
ax.set_xticklabels(ax.get_xticklabels(),rotation=45)
```

Out[45]:
```
[Text(0, 0, 'loc_og_t2o_mou'),
 Text(1, 0, 'std_og_t2o_mou'),
 Text(2, 0, 'loc_ic_t2o_mou'),
 Text(3, 0, 'onnet_mou_6'),
 Text(4, 0, 'onnet_mou_7'),
 Text(5, 0, 'onnet_mou_8'),
 Text(6, 0, 'onnet_mou_9'),
 Text(7, 0, 'offnet_mou_6'),
 Text(8, 0, 'offnet_mou_7'),
 Text(9, 0, 'offnet_mou_8'),
```



```
ut[56]: ['last_date_of_month_9', 'date_of_last_rech_9', 'date_of_last_rech_data_9']
```

```
n [57]: #Get all data columns to impute
columnstoImpute = ['total_rech_data_6'
,'total_rech_data_7'
,'total_rech_data_8'
,'total_rech_data_9'
,'max_rech_data_6'
,'max_rech_data_7'
,'max_rech_data_8'
,'max_rech_data_9'
,'count_rech_2g_6'
,'count_rech_2g_7'
,'count_rech_2g_8'
,'count_rech_2g_9'
,'count_rech_3g_6'
,'count_rech_3g_7'
,'count_rech_3g_8'
,'count_rech_3g_9'
,'av_rech_amt_data_6'
,'av_rech_amt_data_7'
,'av_rech_amt_data_8'
,'av_rech_amt_data_9'
,'arpu_3g_6'
,'arpu_3g_7'
,'arpu_3g_8'
,'arpu_3g_9'
,'arpu_2g_6'
,'arpu_2g_7'
,'arpu_2g_8'
,'arpu_2g_9']
```

```
In [58]: #Impute the null values with 0 for all data columns
high_value_cust[columnstoImpute] = high_value_cust[columnstoImpute].fillna(0, axis=1)
```

```
In [59]: #Recheck the null percentage
print('\n Null % \n',round(100*(high_value_cust.isnull().sum()/len(high_value_cust.index)), 2))
```

```
 Null %
 mobile_number          0.00
circle_id               0.00
loc_og_t2o_mou          0.74
std_og_t2o_mou          0.74
loc_ic_t2o_mou          0.74
last_date_of_month_6    0.00
last_date_of_month_7    0.00
last_date_of_month_8    0.00
last_date_of_month_9    1.19
arpu_6                  0.00
arpu_7                  0.00
arpu_8                  0.00
arpu_9                  0.00
onnet_mou_6             1.65
onnet_mou_7             1.62
onnet_mou_8             3.63
onnet_mou_9             6.06
```

The columns night_pack_user and fb_user are categorical variables where 1 indicates usage of the service and null indicates no usage. Let's impute the missing values with -1 and convert the type to category

IMPUTING THE MISSING VALUES

In [60]:
```
#Impute the missing values and convert the features to category
high_value_cust[['night_pck_user_6','night_pck_user_7','night_pck_user_8','fb_user_6','fb_user_7','fb_user_8' ]]=high_value_cust[
high_value_cust[['night_pck_user_6','night_pck_user_7','night_pck_user_8','fb_user_6','fb_user_7','fb_user_8' ]] = high_value_cus
```

In [61]:
```
high_value_cust[['night_pck_user_6','night_pck_user_7','night_pck_user_8','fb_user_6','fb_user_7','fb_user_8' ]].head()
```

Out[61]:

	night_pck_user_6	night_pck_user_7	night_pck_user_8	fb_user_6	fb_user_7	fb_user_8
0	1	1	1	1	1	1
7	-1	-1	-1	-1	-1	-1
8	-1	1	1	-1	1	1
16	-1	-1	-1	-1	-1	-1
21	-1	-1	-1	-1	-1	-1

In [62]:
```
#Get all the data when loc_og_t20_mou are nulls
high_value_cust[pd.isnull(high_value_cust['loc_og_t2o_mou'])]
```

Out[62]:

	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_d
687	7001662284	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
2185	7000237332	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
3154	7000606599	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
3506	7002324263	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
3677	7001016201	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
4024	7000102255	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
4190	7000293582	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
5175	7002074759	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
5220	7000635396	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
5539	7001126462	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
6567	7001848388	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
6946	7002368732	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
6984	7001738538	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
8265	7000144821	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
8277	7000664549	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
8363	7000604685	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
8975	7000822092	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
9518	7002391910	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
10879	7001588698	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
11364	7000746562	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	
11626	7000145218	109	NaN	NaN	NaN	2014-06-30	2014-07-31	2014-08-31	

'loc_og_t2o_mou','std_og_t2o_mou','loc_ic_t2o_mou' are having same null% and are from same customers. So lets impute them with 0.

In [63]:
```
#Impute 'loc_og_t2o_mou','std_og_t2o_mou','loc_ic_t2o_mou' columns with 0's
high_value_cust[['loc_og_t2o_mou','std_og_t2o_mou','loc_ic_t2o_mou']] = high_value_cust[['loc_og_t2o_mou','std_og_t2o_mou','loc_i
```

In [64]:
```
#Recheck the null percentage
print('\n Null % \n',round(100*(high_value_cust.isnull().sum()/len(high_value_cust.index)), 2))
```

```
 Null %
 mobile_number              0.00
circle_id                   0.00
loc_og_t2o_mou              0.00
std_og_t2o_mou              0.00
loc_ic_t2o_mou              0.00
last_date_of_month_6        0.00
last_date_of_month_7        0.00
last_date_of_month_8        0.00
last_date_of_month_9        1.19
arpu_6                      0.00
arpu_7                      0.00
arpu_8                      0.00
arpu_9                      0.00
onnet_mou_6                 1.65
onnet_mou_7                 1.62
onnet_mou_8                 3.63
onnet_mou_9                 6.06
```

ANALYZING THE METRICS OF THE DATA SET

In [77]:
```
#Get the mterics of all numeric columns
high_value_cust.describe()
```

Out[77]:

	mobile_number	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet_mou_7
count	2.994900e+04	29949.0	29949.0	29949.0	29949.0	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000
mean	7.001216e+09	109.0	0.0	0.0	0.0	577.006877	578.921924	525.105455	276.501375	284.502051
std	6.867758e+05	0.0	0.0	0.0	0.0	449.261901	469.335744	496.373165	461.202260	483.088197
min	7.000000e+09	109.0	0.0	0.0	0.0	-2258.709000	-2014.045000	-945.808000	0.000000	0.000000
25%	7.000631e+09	109.0	0.0	0.0	0.0	345.808000	347.071000	266.252000	26.410000	25.540000
50%	7.001221e+09	109.0	0.0	0.0	0.0	490.933000	489.043000	443.470000	102.690000	100.530000
75%	7.001806e+09	109.0	0.0	0.0	0.0	702.776000	700.512000	666.980000	319.590000	322.760000
max	7.002411e+09	109.0	0.0	0.0	0.0	27731.088000	35145.834000	33543.624000	7376.710000	8157.780000

The columns 'mobile_number','circle_id', '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_ic_t2o_mou_6','std_ic_t2o_mou_7','std_ic_t2o_mou_8' have no new information to provide to the learning algorithm so lets drop them

```
In [78]: #Drop the columns
high_value_cust = high_value_cust.drop(['mobile_number','circle_id', '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_ic_t2o_mou_6',
                        'std_ic_t2o_mou_7','std_ic_t2o_mou_8'], axis=1)
```

```
In [79]: #Get the info of the dataframe after droping columns
high_value_cust.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29949 entries, 0 to 99998
Columns: 167 entries, last_date_of_month_6 to rech_data_8
dtypes: category(12), datetime64[ns](9), float64(120), int64(26)
memory usage: 36.0 MB
```

```
In [80]: #Get the metrics of remaining columns
high_value_cust.describe()
```

Out[80]:

	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
count	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	29949.000000	2
mean	577.006877	578.921924	525.105455	276.501375	284.502051	248.826141	393.905095	398.808925	352.778474	16.762548	
std	449.261901	469.335744	496.373165	461.202260	483.088197	462.953151	478.888295	494.947243	480.726120	77.987846	
min	-2258.709000	-2014.045000	-945.808000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	345.808000	347.071000	266.252000	26.410000	25.540000	16.930000	98.510000	96.210000	66.890000	0.000000	
50%	490.933000	489.043000	443.470000	102.690000	100.530000	79.680000	251.340000	247.640000	209.510000	0.000000	
75%	702.776000	700.512000	666.980000	319.590000	322.760000	265.710000	507.610000	512.380000	459.710000	0.000000	
max	27731.088000	35145.834000	33543.624000	7376.710000	8157.780000	10752.560000	8362.360000	9667.130000	14007.340000	2613.310000	

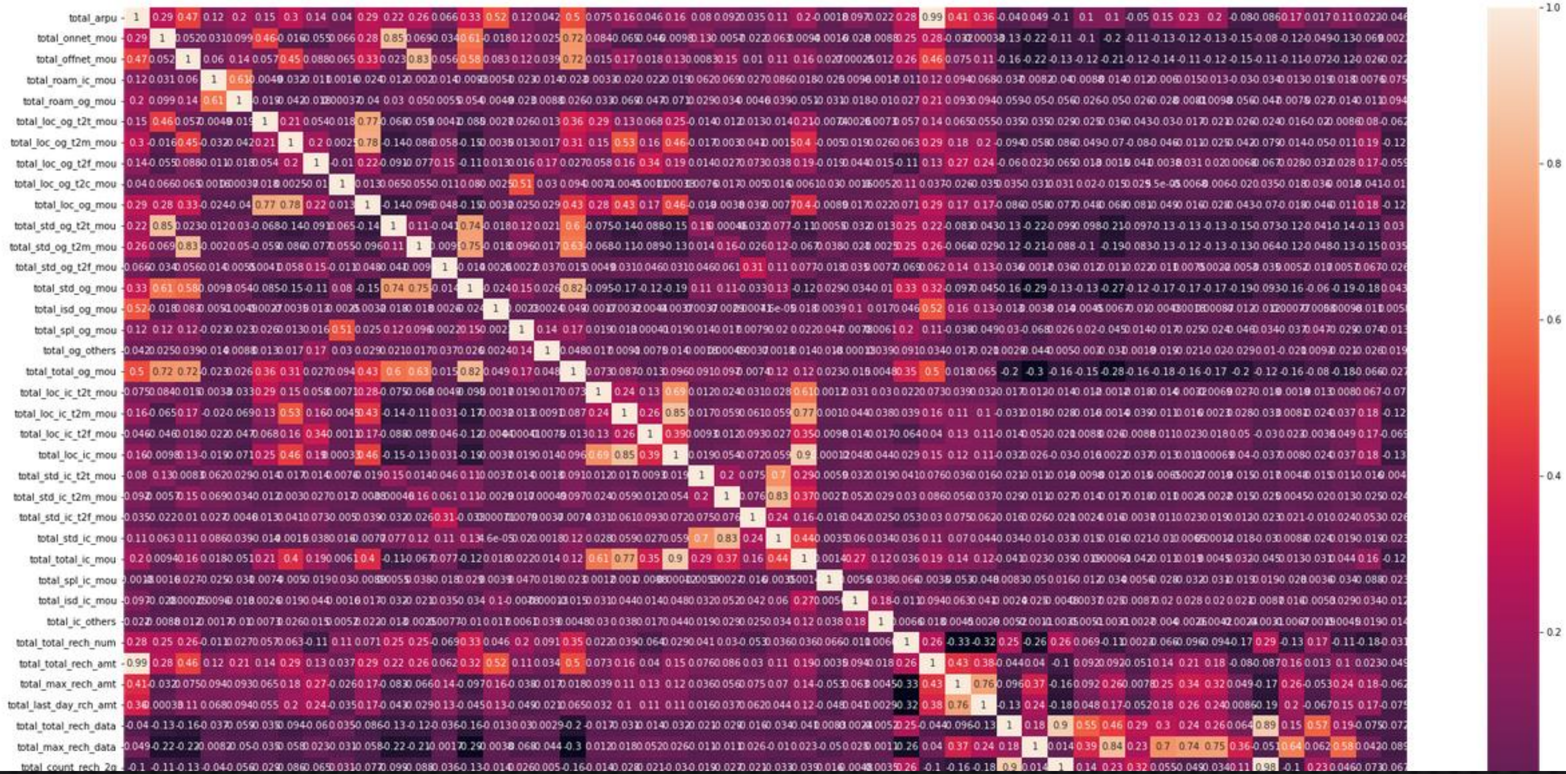
The columns roam_ic_mou, roam_og_mou, loc_og_t2f_mou, loc_og_t2c_mou, loc_og_t2c_mou, std_og_t2f_mou, isd_og_mou, spl_og_mou, og_others, std_ic_t2f_mou, spl_ic_mou, isd_ic_mou, ic_others, total_rech_data, max_rech_data, count_rech_2g, count_rech_3g, av_rech_amt_data, vol_2g_mb, vol_3g_mb, arpu_3g, arpu_2g, night_pck_user, monthly_2g, monthly_sachet_2g, monthly_3g,montlhy_sachet_3g, fbb_user, vbc_3g all are having values 0 for more than 50% of datapoints. So lets create a new dataframe to have totals of all three monehts for respective features

```
In [81]: #Create a new dataframe
         total_data = pd.DataFrame()

In [82]: #Get columns excluding date columns
         cols = high_value_cust.select_dtypes(exclude=['datetime64[ns]','object','category']).columns.tolist()

In [83]: #Remove aon and churn columns
         cols.remove('aon')
         cols.remove('churn')

In [84]: #Create an empty list
         sublist=[]

         #Get the columns names removing last two characters
         for col in cols:
             sublist.append(col[:-2])

In [85]: #Import ordered dict package
         from collections import OrderedDict

         #Get unique column names after removing last two characters into a list
         collist=[]
         collist=list(OrderedDict.fromkeys(sublist))
         collist

Out[85]: ['arpu',
          'onnet_mou',
          'offnet_mou',
          'roam_ic_mou',
          'roam_og_mou',
```

CORRELATION MATRIX OF THE DATA SET:

```
#%matplotlib inline
%matplotlib inline
# Let's see the correlation matrix
plt.figure(figsize = (30,20))        # Size of the figure
sns.heatmap(total_data.corr(),annot = True)
```

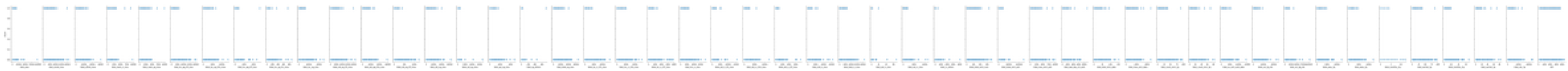
Out[89]: <AxesSubplot:>



OBSERVING THE OUTLIERS:

In [90]:
```
#Pair plot of all the columns with respect to churn
sns.pairplot(total_data, x_vars=total_data.drop('churn', axis=1).columns, y_vars='churn', size=5, aspect=0.5,kind='scatter')
```

Out[90]: <seaborn.axisgrid.PairGrid at 0x23928a88430>



There are outliers present in some of the columns, lets observe them and remove the outliers

In [92]:
```
#Outliers in total_arpu
total_data[total_data['total_arpu']>40000]
```

Out[92]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
38610	96420.546	17.84	10725.83	0.0	0.00	17.84	11.20	0.0
51314	46152.276	0.18	17232.18	1.0	139.98	0.18	0.00	0.6
67717	40614.779	37.53	4913.58	0.0	0.00	37.53	43.21	3.3

In [93]:
```
#Outliers in total_offnet_mou
total_data[total_data['total_offnet_mou']>30000]
```

Out[93]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
33035	12124.74	2351.28	30177.43	0.0	0.0	96.07	352.46	4.9

In [94]:
```
#Outliers in total_roam_ic
total_data[total_data['total_roam_ic_mou']>9500]
```

Out[94]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
28436	1238.586	366.65	887.01	9716.2	1099.49	83.73	61.58	0.

In [95]:
```
#Outliers in total_roam_og_mou
total_data[total_data['total_roam_og_mou']>7500]
```

Out[95]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
74987	8101.6	8394.67	1502.15	27.66	9896.82	0.0	0.0	0.

In [96]:
```
#Outliers in total_loc_og_t2f_mou
total_data[total_data['total_loc_og_t2f_mou']>2000]
```

Out[96]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
67497	4478.437	29.61	2675.15	0.00	0.0	29.61	312.41	2357.0
95046	7318.820	395.73	5203.92	92.46	1625.7	40.10	4.85	2673.8

In [97]:
```
#Outliers in total_std_og_t2m_mou
total_data[total_data['total_std_og_t2m_mou']>20000]
```

Out[97]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
33035	12124.74	2351.28	30177.43	0.0	0.0	96.07	352.46	4.9

```
In [98]: #Outliers in total_std_og_mou
total_data[total_data['total_std_og_mou']>30000]
```

Out[98]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
33035	12124.74	2351.28	30177.43	0.0	0.0	96.07	352.46	4.9

```
In [99]: #Outliers in total_isd_og_mou
total_data[total_data['total_isd_og_mou']>15000]
```

Out[99]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
51314	46152.276	0.18	17232.18	1.0	139.98	0.18	0.0	0.

```
In [100]: #Outliers in total_og_others
total_data[total_data['total_og_others']>600]
```

Out[100]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
95046	7318.82	395.73	5203.92	92.46	1625.7	40.1	4.85	2673.8

```
In [101]: #Outliers in total_total_og_mou
total_data[total_data['total_total_og_mou']>30000]
```

Out[101]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
33035	12124.74	2351.28	30177.43	0.0	0.0	96.07	352.46	4.9

```
In [102]: #Outliers in total_loc_ic_t2t_mou
total_data[total_data['total_loc_ic_t2t_mou']>12000]
```

Out[102]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
34952	2785.079	1438.76	1142.54	0.0	0.0	1201.29	536.78	36.

```
In [103]: #Outliers in total_std_ic_t2m_mou
total_data[total_data['total_std_ic_t2m_mou']>10000]
```

Out[103]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
12898	2019.799	421.58	3518.78	24.13	16.85	24.83	233.33	0.

```
In [104]: #Outliers in total_isd_ic_mou
total_data[total_data['total_isd_ic_mou']>7500]
```

Out[104]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
71514	1758.512	278.68	612.53	0.0	0.0	222.19	210.52	21.6
83599	3386.160	111.60	578.20	0.0	0.0	111.27	392.88	138.3

```
In [105]: #Outliers in total_arpu_2g
total_data[total_data['total_arpu_2g']>10000]
```

Out[105]:

	total_arpu	total_onnet_mou	total_offnet_mou	total_roam_ic_mou	total_roam_og_mou	total_loc_og_t2t_mou	total_loc_og_t2m_mou	total_loc_og_t2f_mo
27531	12204.97	0.0	11.92	0.0	0.0	0.0	0.53	0.

```
In [108]: #Get the info of the total_data dataframe after dropping outliers
total_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29935 entries, 0 to 99998
Data columns (total 50 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   total_arpu              29935 non-null  float64
 1   total_onnet_mou         29935 non-null  float64
 2   total_offnet_mou        29935 non-null  float64
 3   total_roam_ic_mou       29935 non-null  float64
 4   total_roam_og_mou       29935 non-null  float64
 5   total_loc_og_t2t_mou    29935 non-null  float64
 6   total_loc_og_t2m_mou    29935 non-null  float64
 7   total_loc_og_t2f_mou    29935 non-null  float64
 8   total_loc_og_t2c_mou    29935 non-null  float64
 9   total_loc_og_mou        29935 non-null  float64
 10  total_std_og_t2t_mou    29935 non-null  float64
 11  total_std_og_t2m_mou    29935 non-null  float64
 12  total_std_og_t2f_mou    29935 non-null  float64
 13  total_std_og_mou        29935 non-null  float64
 14  total_isd_og_mou        29935 non-null  float64
 15  total_spl_og_mou        29935 non-null  float64
 16  total_og_others         29935 non-null  float64
 17  total_total_og_mou      29935 non-null  float64
 18  total_loc_ic_t2t_mou    29935 non-null  float64
 19  total_loc_ic_t2m_mou    29935 non-null  float64
 20  total_loc_ic_t2f_mou    29935 non-null  float64
 21  total_loc_ic_mou        29935 non-null  float64
 22  total_std_ic_t2t_mou    29935 non-null  float64
 23  total_std_ic_t2m_mou    29935 non-null  float64
 24  total_std_ic_t2f_mou    29935 non-null  float64
 25  total_std_ic_mou        29935 non-null  float64
 26  total_total_ic_mou      29935 non-null  float64
 27  total_spl_ic_mou        29935 non-null  float64
```

Lets create another derived features where we indicate 1 if average first two months value(good months) is greater than last month(action month)

In [116]:
```python
#Create new feature as difference of incoming and outgoing usages. Since we have 0 values we cannot use ratio
total_data['in_out_difference']=total_data['total_total_ic_mou']-total_data['total_total_og_mou']
```

In [117]:
```python
#Create new feature as difference of onnet and offnet usages. Since we have 0 values we cannot use ratio
total_data['onnet_offnet_difference'] = total_data['total_onnet_mou']-total_data['total_offnet_mou']
```

In [118]:
```python
#Get ratio of data arpu to amount arpu
total_data['arpu_data_to_total_ratio'] = (total_data['total_arpu_2g']+total_data['total_arpu_3g'])/total_data['total_arpu']
```

In [119]:
```python
#Get the ratio of average data recharge to amount recharge
total_data['data_to_amt_ratio']=total_data['total_av_rech_amt_data']/total_data['total_total_rech_amt']
```

In [120]:
```python
#Drop aon and churn columns from total_data
total_data=total_data.drop(['aon','churn'], axis=1)
```

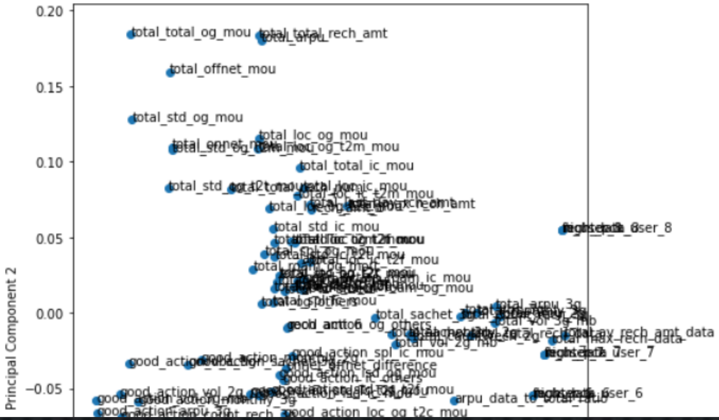
In [121]:
```python
#Get info of total_data dataframe
total_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29935 entries, 0 to 99998
Data columns (total 52 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   total_arpu                 29935 non-null  float64
```

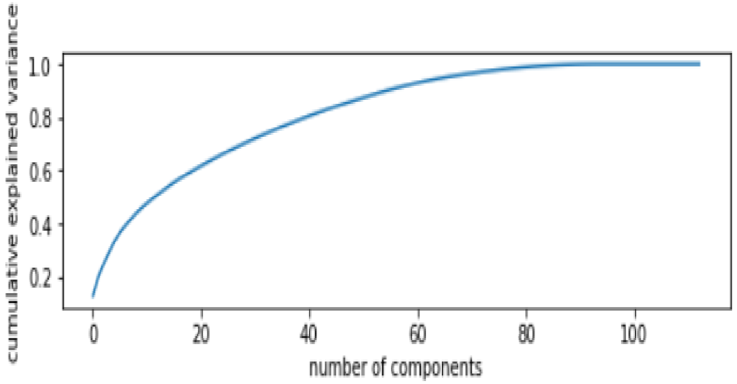
PCA ANALYSIS:

In [154]:
```
#Improting the PCA module
from sklearn.decomposition import PCA
pca = PCA(svd_solver='randomized', random_state=100)
```

In [155]:
```
# Lets see the distributions of columns in PCA in 2d space
pca.fit(X_train)
fig = plt.figure(figsize = (8,8))
plt.scatter(pca.components_[0],pca.components_[1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
for i, txt in enumerate(X_train.columns):
    plt.annotate(txt, (pca.components_[0][i],pca.components_[1][i]))
plt.tight_layout()
plt.show()
```



In [156]:
```
#Making the screeplot - plotting the cumulative variance against the number of components
%matplotlib inline
fig = plt.figure(figsize = (8,2))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



From the graph it is clear that around 30 components convey more than 95% of variance data

```
In [157]: #Import incremental PCA
          from sklearn.decomposition import IncrementalPCA
          pca_final = IncrementalPCA(n_components=30)

In [158]: #Fit the train data
          df_train_pca = pca_final.fit_transform(X_train)
          df_train_pca.shape

Out[158]: (14667, 30)

In [159]: #Fit the validation data
          df_val_pca = pca_final.fit_transform(X_val)
          df_val_pca.shape

Out[159]: (6287, 30)

In [160]: #Fit the validation data
          df_test_pca = pca_final.fit_transform(X_test)
          df_test_pca.shape

Out[160]: (8981, 30)

In [161]: # Importing random forest classifier from sklearn library
          from sklearn.ensemble import RandomForestClassifier

          # Running the random forest with default parameter and balanced subsample to tackle class imbalance.
          rfc = RandomForestClassifier(class_weight='balanced_subsample')
```

```
In [162]: # fit
          rfc.fit(df_train_pca,y_train)

Out[162]:         RandomForestClassifier
          RandomForestClassifier(class_weight='balanced_subsample')

In [163]: # Let's check the report of our default model on training data
          print(classification_report(y_train,rfc.predict(df_train_pca)))

                        precision    recall  f1-score   support

                     0       1.00      1.00      1.00     13487
                     1       1.00      1.00      1.00      1180

              accuracy                           1.00     14667
             macro avg       1.00      1.00      1.00     14667
          weighted avg       1.00      1.00      1.00     14667

In [164]: # Printing confusion matrix
          print(confusion_matrix(y_train,rfc.predict(df_train_pca)))

          [[13487     0]
           [    1  1179]]

In [165]: # Let's check the report of our default model on validation data
          print(classification_report(y_val,rfc.predict(df_val_pca)))

                        precision    recall  f1-score   support

                     0       0.92      1.00      0.96      5768
                     1       0.67      0.07      0.13       519

              accuracy                           0.92      6287
```

```
In [166]: # Printing confusion matrix
          print(confusion_matrix(y_val,rfc.predict(df_val_pca)))

          [[5749   19]
           [ 481   38]]
```

The F1 score for training data is almost 1 whereas for validation data it is much less. It is clearly overfitting. We will use GridSearch on each parameter to check the optimal range. Since it is the classification we use auc_score for our scoring metric.

We will use greedy based algorithm to identify best hyperparameters using grid search

GRIDSEARCHCV TO FIND OPTIMAL MAX_DEPTH

```
# GridSearchCV to find optimal max_depth
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5
scoring = {'AUC': 'roc_auc'}
# parameters to build the model on
parameters = {'max_depth': range(2, 20, 1)}

# instantiate the model
rf = RandomForestClassifier(class_weight='balanced_subsample')


# fit tree on training data
grid_search = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring='roc_auc',verbose=1)
grid_search.fit(df_train_pca, y_train)
train_results = grid_search.cv_results_
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
[155] #Add the results to dataframe
pd.DataFrame(train_results).head(10)
```

```python
[155] #Add the results to dataframe
      pd.DataFrame(train_results).head(10)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1_test_score	split2_test_score	split3_test_scor
0	1.586294	0.024479	0.031552	0.000723	2	{'max_depth': 2}	0.892883	0.880500	0.880467	0.87479
1	2.053650	0.012873	0.034979	0.001314	3	{'max_depth': 3}	0.900509	0.887753	0.886379	0.89051
2	2.632062	0.220793	0.040322	0.005622	4	{'max_depth': 4}	0.905114	0.893452	0.892213	0.89385
3	3.015187	0.219152	0.039508	0.001031	5	{'max_depth': 5}	0.910034	0.899912	0.898000	0.89586
4	3.287752	0.037942	0.042384	0.001283	6	{'max_depth': 6}	0.909034	0.900392	0.898156	0.90102
5	3.790820	0.278104	0.083064	0.074183	7	{'max_depth': 7}	0.908136	0.904346	0.898889	0.90698
6	3.918222	0.022403	0.046821	0.000500	8	{'max_depth': 8}	0.911198	0.903332	0.902119	0.90354
7	4.180690	0.032951	0.053582	0.002267	9	{'max_depth': 9}	0.908585	0.900014	0.902346	0.90369
8	4.355918	0.023397	0.052143	0.001388	10	{'max_depth': 10}	0.907620	0.904782	0.906343	0.90464
9	4.473270	0.013283	0.052875	0.000759	11	{'max_depth': 11}	0.905766	0.902518	0.900279	0.90592

```python
[156] #Fit the range of feature on validation data to plot
      auc_list = []
      feature_list=[]
      for feature in range(2, 20, 1):
          rf = RandomForestClassifier(max_depth=feature,class_weight='balanced_subsample')
          rf.fit(df_train_pca,y_train)
          fpr, tpr, thresholds = roc_curve(y_val, rf.predict(df_val_pca))
          auc_list.append(auc(fpr, tpr))
          feature_list.append(feature)
```

```python
[161] # plotting accuracies with max_depth
      plt.figure()
      plt.plot(train_results["param_max_depth"],
               train_results["mean_test_score"],
               label="train auc score")
      plt.plot(train_results["param_max_depth"],
               train_results["mean_test_score"],
               label="test auc score")
      plt.plot(feature_list,
               auc_list,
               label="valid auc score")
      plt.xlabel("max_depth")
      plt.ylabel("AUC Score")
```

```
Text(0, 0.5, 'AUC Score')
```


```python
[162] #Fit the range of features
      parameters = {'n_estimators': range(200, 1800, 200)}

      # instantiate the model ()
      rf = RandomForestClassifier(class_weight='balanced_subsample', max_depth=6)

      # fit tree on training data
      grid_search = GridSearchCV(rf, parameters,
                          cv=n_folds,
                          scoring='roc_auc',n_jobs=-1,verbose=10)
      grid_search.fit(df_train_pca, y_train)
      train_results = grid_search.cv_results_
```

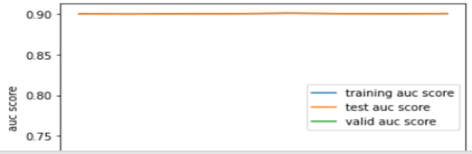
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```python
[163] pd.DataFrame(train_results).head(20)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	split3_test_
0	9.331249	0.117475	0.117807	0.009017	200	{'n_estimators': 200}	0.908811	0.901559	0.898955	0.9
1	18.591524	0.298544	0.216841	0.005816	400	{'n_estimators': 400}	0.909077	0.902187	0.900084	0.9
2	28.121088	0.248164	0.327308	0.008908	600	{'n_estimators': 600}	0.910382	0.902605	0.899859	0.9
3	38.198847	1.558456	0.433357	0.008767	800	{'n_estimators': 800}	0.910279	0.902396	0.900580	0.9
4	46.419666	0.097878	0.534289	0.005150	1000	{'n_estimators':	0.910321	0.902009	0.901230	0.9

```python
[164] #Fit the range of feature on validation data to plot
      auc_list = []
      feature_list=[]
      for feature in range(200, 1800, 200):
          rf = RandomForestClassifier(n_estimators=feature, class_weight='balanced_subsample', max_depth=6)
          rf.fit(df_train_pca,y_train)
          fpr, tpr, thresholds = roc_curve(y_val, rf.predict(df_val_pca))
          auc_list.append(auc(fpr, tpr))
          feature_list.append(feature)
```

```python
[166] # plotting accuracies with n_estimators
      plt.figure()
      plt.plot(train_results["param_n_estimators"],
               train_results["mean_test_score"],
               label="training auc score")
      plt.plot(train_results["param_n_estimators"],
               train_results["mean_test_score"],
               label="test auc score")
      plt.plot(feature_list,
               auc_list,
               label="valid auc score")
      plt.xlabel("n_estimators")
      plt.ylabel("auc score")
      plt.legend()
      plt.show()
```



```
[167] # parameters to build the model on max_features
      parameters = {'max_features': range(2, 30, 2)}

      # instantiate the model
      rf = RandomForestClassifier(class_weight='balanced_subsample')


      # fit tree on training data
      grid_search = GridSearchCV(rf, parameters,
                          cv=n_folds,
                         scoring='roc_auc',n_jobs=-1, verbose=10)
      grid_search.fit(df_train_pca, y_train)
      train_results = grid_search.cv_results_

      Fitting 5 folds for each of 14 candidates, totalling 70 fits
```

```
[168] pd.DataFrame(train_results).head(20)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	params	split0_test_score	split1_test_score	split2_test_score	split3_te
0	3.612083	0.036814	0.087696	0.008430	2	{'max_features': 2}	0.896796	0.889623	0.893787	
1	5.959802	0.053548	0.080570	0.007730	4	{'max_features': 4}	0.900072	0.897108	0.903498	
2	8.451101	0.056427	0.076382	0.003048	6	{'max_features': 6}	0.905209	0.892395	0.901476	
3	10.946402	0.145556	0.079285	0.006597	8	{'max_features': 8}	0.901637	0.893259	0.893421	
4	14.082851	0.631121	0.093824	0.039933	10	{'max_features': 10}	0.896448	0.893153	0.894730	
5	16.891600	0.768545	0.073577	0.001898	12	{'max_features': 12}	0.896145	0.895663	0.903723	

```
# parameters to build the model on min_samples_split
parameters = {'min_samples_split': range(200, 500, 50)}

# instantiate the model
rf = RandomForestClassifier(class_weight='balanced_subsample')


# fit tree on training data
grid_search = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring='roc_auc',n_jobs=-1)
grid_search.fit(df_train_pca, y_train)
train_results = grid_search.cv_results_
```

```
[178] pd.DataFrame(train_results).head(10)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_test_score	split1_test_score	split2_test_score	sp
0	6.030660	0.094864	0.063834	0.001238	200	{'min_samples_split': 200}	0.906716	0.900605	0.902104	
1	5.844471	0.059071	0.063167	0.002786	250	{'min_samples_split': 250}	0.905473	0.903754	0.900145	
2	5.681923	0.041294	0.061418	0.002228	300	{'min_samples_split': 300}	0.905059	0.903518	0.898977	
3	5.506897	0.094370	0.059792	0.000653	350	{'min_samples_split': 350}	0.905382	0.904007	0.901904	
4	5.362813	0.050728	0.058349	0.000481	400	{'min_samples_split': 400}	0.903820	0.903827	0.897932	
5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	{'min_samples_split':	[illegible]	[illegible]	[illegible]	

```
[182] # model with the best hyperparameters
      from sklearn.ensemble import RandomForestClassifier
      rfc = RandomForestClassifier(bootstrap=True,
                                   max_depth=6,
                                   min_samples_leaf=100,
                                   min_samples_split=370,
                                   max_features=28,
                                   n_estimators=600,
                                   class_weight='balanced_subsample')
```

```
[183] # fit
      rfc.fit(df_train_pca,y_train)

      RandomForestClassifier(class_weight='balanced_subsample', max_depth=6,
                             max_features=28, min_samples_leaf=100,
                             min_samples_split=370, n_estimators=600)
```

```
[184] # Let's check the report of our optimal model on training data
      print(classification_report(y_train,rfc.predict(df_train_pca)))

                    precision    recall  f1-score   support

                 0       0.98      0.86      0.92     12500
                 1       0.34      0.83      0.49      1118

          accuracy                           0.86     13618
         macro avg       0.66      0.84      0.70     13618
      weighted avg       0.93      0.86      0.88     13618
```

```
[185] # Printing confusion matrix
      print(confusion_matrix(y_train,rfc.predict(df_train_pca)))

      [[10742  1758]
       [  193   925]]
```

```
[187] # Printing confusion matrix
      print(confusion_matrix(y_val,rfc.predict(df_val_pca)))

      [[4573  801]
       [ 179  284]]
```

```
[188] # Let's check the report of our default model on validation data
      print(classification_report(y_test,rfc.predict(df_test_pca)))

                    precision    recall  f1-score   support

                 0       0.96      0.87      0.91      7655
                 1       0.30      0.63      0.40       684

          accuracy                           0.85      8339
         macro avg       0.63      0.75      0.66      8339
      weighted avg       0.91      0.85      0.87      8339
```

```
[189] # Printing confusion matrix
      print(confusion_matrix(y_test,rfc.predict(df_test_pca)))

      [[6632 1023]
       [ 254  430]]
```

```
[190] from sklearn.ensemble import RandomForestClassifier
      rfc_base = RandomForestClassifier(bootstrap=True,
                                        max_depth=1,
                                        class_weight='balanced_subsample')
```

```
[191] #import Adaboost classifier
      from sklearn.ensemble import AdaBoostClassifier
      # parameter grid
```

RESULTS THROUGH PCA

WE GET AROUND 0.85 F1 SCORE ON THE TEST DATA WHICH IS REASONABLE AND 0.88 OF PRECISCION AND 0.82 OF RECALL