Problem Statement

In the telecommunication industry, customers tend to change operators if not provided with attractive schemes and offers. It is very important for any telecom operator to prevent the present customers from churning to other operators. As a data scientist, your task in this case study would be to build an ML model which can predict if the customer will churn or not in a particular month based on the past data.

In [1]:
```
#Data Structures
import pandas as pd
import numpy as np
import re
import os
import missingno as msno

#Sklearn
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, precision_score, recall_score
#Plotting
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
pd.set_option('display.max_columns', 200)
#Others
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

Import the data

In [2]:
```
data = pd.read_csv("train.csv")
unseen = pd.read_csv("test.csv")
sample = pd.read_csv("sample.csv")
data_dict = pd.read_csv("data_dictionary.csv")
```

Analyse the data

In [3]:
```
data.head()
```

Out[3]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_o
0	0	109	0.0	0.0	0.0	6/30/2014	
1	1	109	0.0	0.0	0.0	6/30/2014	
2	2	109	0.0	0.0	0.0	6/30/2014	
3	3	109	0.0	0.0	0.0	6/30/2014	
4	4	109	0.0	0.0	0.0	6/30/2014	

In [4]:
```
data.shape
```

Out[4]:
```
(69999, 172)
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 172 entries, id to churn_probability
dtypes: float64(135), int64(28), object(9)
memory usage: 91.9+ MB
```

In [6]:
```
data.describe()
```

Out[6]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_
t	69999.000000	69999.0	69297.0	69297.0	69297.0	69999.000000	69999.00000
[illegible]	34999.000000	109.0	0.0	0.0	0.0	283.134365	278.1859
[illegible]	20207.115084	0.0	0.0	0.0	0.0	334.213918	344.3669
[illegible]	0.000000	109.0	0.0	0.0	0.0	-2258.709000	-1289.7150
[illegible]	17499.500000	109.0	0.0	0.0	0.0	93.581000	86.7140
[illegible]	34999.000000	109.0	0.0	0.0	0.0	197.484000	191.5880
[illegible]	52498.500000	109.0	0.0	0.0	0.0	370.791000	365.3695
[illegible]	69998.000000	109.0	0.0	0.0	0.0	27731.088000	35145.8340

In [7]:
```
data_dict
```

Out[7]:

	Acronyms	Description
0	CIRCLE_ID	Telecom circle area to which the customer belo...
1	LOC	Local calls within same telecom circle
2	STD	STD calls outside the calling circle
3	IC	Incoming calls
4	OG	Outgoing calls
5	T2T	Operator T to T ie within same operator mobile...
6	T2M	Operator T to other operator mobile
7	T2O	Operator T to other operator fixed line
8	T2F	Operator T to fixed lines of T
9	T2C	Operator T to its own call center
10	ARPU	Average revenue per user
11	MOU	Minutes of usage voice calls
12	AON	Age on network number of days the customer is...
13	ONNET	All kind of calls within the same operator net...
14	OFFNET	All kind of calls outside the operator T network
15	ROAM	Indicates that customer is in roaming zone dur...
16	SPL	Special calls
17	ISD	ISD calls
18	RECH	Recharge
19	NUM	Number
20	AMT	Amount in local currency
21	MAX	Maximum
22	DATA	Mobile internet
23	3G	G network
24	AV	Average
25	VOL	Mobile internet usage volume in MB
26	2G	G network
27	PCK	Prepaid service schemes called PACKS
28	NIGHT	Scheme to use during specific night hours only
29	MONTHLY	Service schemes with validity equivalent to a ...
30	SACHET	Service schemes with validity smaller than a m...
31	*.6	KPI for the month of June
32	*.7	KPI for the month of July
33	*.8	KPI for the month of August
34	FB_USER	Service scheme to avail services of Facebook a...
35	VBC	Volume based cost when no specific scheme is ...

In [8]:
```
data
```

Out[8]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last
0	0	109	0.0	0.0	0.0	6/30/2014	
1	1	109	0.0	0.0	0.0	6/30/2014	
2	2	109	0.0	0.0	0.0	6/30/2014	
3	3	109	0.0	0.0	0.0	6/30/2014	
4	4	109	0.0	0.0	0.0	6/30/2014	
...	...	...	...	...	...	...	
69994	69994	109	0.0	0.0	0.0	6/30/2014	
69995	69995	109	0.0	0.0	0.0	6/30/2014	
69996	69996	109	0.0	0.0	0.0	6/30/2014	
69997	69997	109	0.0	0.0	0.0	6/30/2014	
69998	69998	109	0.0	0.0	0.0	6/30/2014	

69999 rows × 172 columns

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 172 entries, id to churn_probability
dtypes: float64(135), int64(28), object(9)
memory usage: 91.9+ MB
```

In [10]:
```
# Checking the percentage of missing values
missing_data = data.apply(lambda x: round(x.isnull().mean()* 100, 2)).sort_values(
missing_data_above_thresold = missing_data[missing_data>50]
missing_columns_above_thresold = missing_data_above_thresold.index.to_list()
missing_data
```

Out[10]:
```
arpu_3g_6                  74.9
count_rech_2g_6            74.9
night_pck_user_6           74.9
arpu_2g_6                  74.9
date_of_last_rech_data_6   74.9
                           ...
last_day_rch_amt_8          0.0
vol_2g_mb_6                 0.0
vol_2g_mb_7                 0.0
vol_2g_mb_8                 0.0
churn_probability           0.0
Length: 172, dtype: float64
```

In [11]:
```
missing_columns_above_thresold
```

Out[11]:
```
['arpu_3g_6',
 'count_rech_2g_6',
 'night_pck_user_6',
 'arpu_2g_6',
 'date_of_last_rech_data_6',
 'total_rech_data_6',
 'av_rech_amt_data_6',
 'max_rech_data_6',
 'count_rech_3g_6',
 'fb_user_6',
 'night_pck_user_7',
 'date_of_last_rech_data_7',
 'total_rech_data_7',
 'max_rech_data_7',
 'fb_user_7',
 'count_rech_2g_7',
 'count_rech_3g_7',
 'arpu_3g_7',
 'av_rech_amt_data_7',
 'arpu_2g_7',
 'count_rech_2g_8',
 'av_rech_amt_data_8',
 'night_pck_user_8',
 'max_rech_data_8',
 'total_rech_data_8',
 'arpu_2g_8',
 'arpu_3g_8',
 'date_of_last_rech_data_8',
 'fb_user_8',
 'count_rech_3g_8']
```

In [12]:
```
# Drop the columns from the dataframe
data.drop(missing_columns_above_thresold, axis=1, inplace=True)
data.head()
```

Out[12]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_o
0	0	109	0.0	0.0	0.0	6/30/2014	
1	1	109	0.0	0.0	0.0	6/30/2014	
2	2	109	0.0	0.0	0.0	6/30/2014	
3	3	109	0.0	0.0	0.0	6/30/2014	
4	4	109	0.0	0.0	0.0	6/30/2014	

In [13]:
```
data.shape
```

Out[13]:
```
(69999, 142)
```

In [14]:
```
data.describe()
```

Out[14]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	
count	69999.000000	69999.0	69297.0	69297.0	69297.0	69999.000000	6999
mean	34999.000000	109.0	0.0	0.0	0.0	283.134365	27
std	20207.115084	0.0	0.0	0.0	0.0	334.213918	34
min	0.000000	109.0	0.0	0.0	0.0	-2258.709000	-128
25%	17499.500000	109.0	0.0	0.0	0.0	93.581000	8
50%	34999.000000	109.0	0.0	0.0	0.0	197.484000	19
75%	52498.500000	109.0	0.0	0.0	0.0	370.791000	36
max	69998.000000	109.0	0.0	0.0	0.0	27731.088000	3514

In [15]:
```
data
```

Out[15]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last
0	0	109	0.0	0.0	0.0	6/30/2014	
1	1	109	0.0	0.0	0.0	6/30/2014	
2	2	109	0.0	0.0	0.0	6/30/2014	
3	3	109	0.0	0.0	0.0	6/30/2014	
4	4	109	0.0	0.0	0.0	6/30/2014	
...	...	...	...	...	...	...	
69994	69994	109	0.0	0.0	0.0	6/30/2014	
69995	69995	109	0.0	0.0	0.0	6/30/2014	
69996	69996	109	0.0	0.0	0.0	6/30/2014	
69997	69997	109	0.0	0.0	0.0	6/30/2014	
69998	69998	109	0.0	0.0	0.0	6/30/2014	

69999 rows × 142 columns

In [16]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69999 entries, 0 to 69998
Columns: 142 entries, id to churn_probability
dtypes: float64(108), int64(28), object(6)
memory usage: 75.8+ MB
```

In [17]:
```
# Checking the percentage of missing values
missing_data = data.apply(lambda x: round(x.isnull().mean()* 100, 2)).sort_values(
missing_data_above_thresold = missing_data[missing_data>0]
missing_columns_above_thresold = missing_data_above_thresold.index.to_list()
missing_columns_above_thresold
```

Out[17]:
```
['roam_og_mou_8',
 'og_others_8',
 'spl_og_mou_8',
 'loc_ic_t2t_mou_8',
 'loc_og_t2m_mou_8',
 'loc_og_t2c_mou_8',
 'loc_ic_t2m_mou_8',
 'loc_og_t2t_mou_8',
 'loc_ic_t2f_mou_8',
 'std_og_t2f_mou_8',
 'loc_ic_mou_8',
 'isd_og_mou_8',
 'roam_ic_mou_8',
 'std_ic_t2t_mou_8',
 'loc_og_mou_8',
 'offnet_mou_8',
 'std_ic_t2m_mou_8',
 'onnet_mou_8',
 'std_ic_t2f_mou_8',
 'std_og_mou_8',
 'std_ic_t2o_mou_8',
 'std_og_t2t_mou_8',
 'std_ic_mou_8',
 'spl_ic_mou_8',
 'std_og_t2c_mou_8',
 'isd_ic_mou_8',
 'std_og_t2m_mou_8',
 'ic_others_8',
 'loc_og_t2f_mou_8',
 'isd_og_mou_6',
 'spl_og_mou_6',
 'std_og_mou_6',
 'loc_ic_t2f_mou_6',
 'loc_ic_t2t_mou_6',
 'loc_ic_t2m_mou_6',
 'loc_ic_mou_6',
 'std_ic_t2t_mou_6',
 'std_ic_t2m_mou_6',
 'std_ic_t2f_mou_6',
 'std_ic_t2o_mou_6',
 'std_ic_mou_6',
 'spl_ic_mou_6',
 'isd_ic_mou_6',
 'ic_others_6',
 'std_og_t2c_mou_6',
 'og_others_6',
 'offnet_mou_6',
 'onnet_mou_6',
 'loc_og_t2m_mou_6',
 'loc_og_t2f_mou_6',
 'roam_og_mou_6',
 'roam_ic_mou_6',
 'loc_og_t2c_mou_6',
 'std_og_t2f_mou_6',
 'loc_og_mou_6',
 'loc_og_t2t_mou_6',
 'std_og_t2t_mou_6',
 'std_og_t2m_mou_6',
 'std_ic_t2o_mou_7',
 'onnet_mou_7',
 'std_ic_t2f_mou_7',
 'loc_og_t2t_mou_7',
 'std_ic_mou_7',
 'std_ic_t2m_mou_7',
```

```
 'spl_ic_mou_7',
 'roam_ic_mou_7',
 'std_ic_t2t_mou_7',
 'isd_ic_mou_7',
 'roam_og_mou_7',
 'loc_ic_mou_7',
 'ic_others_7',
 'std_og_t2f_mou_7',
 'offnet_mou_7',
 'loc_ic_t2f_mou_7',
 'og_others_7',
 'std_og_t2m_mou_7',
 'isd_og_mou_7',
 'loc_og_mou_7',
 'spl_og_mou_7',
 'loc_og_t2c_mou_7',
 'std_og_t2c_mou_7',
 'std_og_mou_7',
 'std_og_t2t_mou_7',
 'loc_og_t2f_mou_7',
 'loc_ic_t2t_mou_7',
 'loc_og_t2m_mou_7',
 'loc_ic_t2m_mou_7',
 'date_of_last_rech_8',
 'date_of_last_rech_7',
 'date_of_last_rech_6',
 'last_date_of_month_8',
 'loc_ic_t2o_mou',
 'std_og_t2o_mou',
 'loc_og_t2o_mou',
 'last_date_of_month_7']
```

In [18]:

```
data.isna().sum().unique() # so no null value is present in the data
```

Out[18]:

```
array([   0,  702,  399,  733, 2768, 2687, 3703, 1101, 1234, 2461],
      dtype=int64)
```

Treat the data

In [19]:

```
# Remove the date column
# List the date columns
date_columns = [k for k in data.columns.to_list() if 'date' in k]
date_columns
```

Out[19]:

```
['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']
```

In [20]:
```
data = data.drop(date_columns, axis=1)
data
```

Out[20]:

	id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8
0	0	109	0.0	0.0	0.0	31.277	87.009	7.527
1	1	109	0.0	0.0	0.0	0.000	122.787	42.953
2	2	109	0.0	0.0	0.0	60.806	103.176	0.000
3	3	109	0.0	0.0	0.0	156.362	205.260	111.095
4	4	109	0.0	0.0	0.0	240.708	128.191	101.565
...	...	...	...	...	...	...	...	...
69994	69994	109	0.0	0.0	0.0	15.760	410.924	329.136
69995	69995	109	0.0	0.0	0.0	160.083	289.129	265.772
69996	69996	109	0.0	0.0	0.0	372.088	258.374	279.782
69997	69997	109	0.0	0.0	0.0	238.575	245.414	145.062
69998	69998	109	0.0	0.0	0.0	168.269	42.815	167.961

69999 rows × 136 columns

In [21]:
```
# remove circle id as it is no longer needed
data = data.drop(['circle_id'], axis=1)
data
```

Out[21]:

	id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_m
0	0	0.0	0.0	0.0	31.277	87.009	7.527	4
1	1	0.0	0.0	0.0	0.000	122.787	42.953	
2	2	0.0	0.0	0.0	60.806	103.176	0.000	
3	3	0.0	0.0	0.0	156.362	205.260	111.095	
4	4	0.0	0.0	0.0	240.708	128.191	101.565	2
...	...	...	...	...	...	...	...	
69994	69994	0.0	0.0	0.0	15.760	410.924	329.136	
69995	69995	0.0	0.0	0.0	160.083	289.129	265.772	11
69996	69996	0.0	0.0	0.0	372.088	258.374	279.782	7
69997	69997	0.0	0.0	0.0	238.575	245.414	145.062	1
69998	69998	0.0	0.0	0.0	168.269	42.815	167.961	

69999 rows × 135 columns

Get the high value customer

Creating column `avg_rech_amt_6_7` by summing up total recharge amount of month 6 and 7. Then taking the average of the sum.

In [22]:
```
data['avg_rech_amt'] = (data['total_rech_amt_6'] + data['total_rech_amt_7'] + data
data['total_rech_amt'] = (data['total_rech_amt_6'] + data['total_rech_amt_7'] + da
data
```

Out[22]:

	id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_m
0	0	0.0	0.0	0.0	31.277	87.009	7.527	4
1	1	0.0	0.0	0.0	0.000	122.787	42.953	
2	2	0.0	0.0	0.0	60.806	103.176	0.000	
3	3	0.0	0.0	0.0	156.362	205.260	111.095	
4	4	0.0	0.0	0.0	240.708	128.191	101.565	2
...	...	...	...	...	...	...	...	
69994	69994	0.0	0.0	0.0	15.760	410.924	329.136	
69995	69995	0.0	0.0	0.0	160.083	289.129	265.772	11
69996	69996	0.0	0.0	0.0	372.088	258.374	279.782	7
69997	69997	0.0	0.0	0.0	238.575	245.414	145.062	1
69998	69998	0.0	0.0	0.0	168.269	42.815	167.961	

69999 rows × 137 columns

In [23]:
```
#X = data['avg_rech_amt'].quantile(0.7)
#X
```

Treat missing values in rows

In [24]:
```
# Checking the missing values in columns again
missing_row_value= (data.apply(lambda x: round(x.isnull().mean()* 100, 2)).to_fram
missing_row_value
```

Out[24]:

	null
std_ic_mou_8	5.29
std_ic_t2m_mou_8	5.29
std_ic_t2o_mou_8	5.29
std_og_mou_8	5.29
loc_og_t2f_mou_8	5.29
...	...
max_rech_amt_7	0.00
max_rech_amt_8	0.00
last_day_rch_amt_6	0.00
last_day_rch_amt_7	0.00
total_rech_amt	0.00

137 rows × 1 columns

In [25]:
```
data = data.dropna(axis = 0)
data
```

Out[25]:

	id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_m
0	0	0.0	0.0	0.0	31.277	87.009	7.527	4
1	1	0.0	0.0	0.0	0.000	122.787	42.953	
2	2	0.0	0.0	0.0	60.806	103.176	0.000	
3	3	0.0	0.0	0.0	156.362	205.260	111.095	
4	4	0.0	0.0	0.0	240.708	128.191	101.565	2
...	...	...	...	...	...	...	...	
69994	69994	0.0	0.0	0.0	15.760	410.924	329.136	
69995	69995	0.0	0.0	0.0	160.083	289.129	265.772	11
69996	69996	0.0	0.0	0.0	372.088	258.374	279.782	7
69997	69997	0.0	0.0	0.0	238.575	245.414	145.062	1
69998	69998	0.0	0.0	0.0	168.269	42.815	167.961	

63842 rows × 137 columns

In [26]:
```
data.shape
```

Out[26]:
```
(63842, 137)
```

In [27]:
```
data.isna().sum()
```

Out[27]:
```
id                   0
loc_og_t2o_mou       0
std_og_t2o_mou       0
loc_ic_t2o_mou       0
arpu_6               0
                    ..
jul_vbc_3g           0
jun_vbc_3g           0
churn_probability    0
avg_rech_amt         0
total_rech_amt       0
Length: 137, dtype: int64
```

In [28]:
```
data = data.set_index("id")
```

In [29]:
```
data['churn_probability'].unique()
```

Out[29]:
```
array([0, 1], dtype=int64)
```

In [30]:
```
X = data.drop(['churn_probability'], axis=1)
y = data['churn_probability']
```

In [31]:
```
X
```

Out[31]:

id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_mou_6	o
0	0.0	0.0	0.0	31.277	87.009	7.527	48.58	
1	0.0	0.0	0.0	0.000	122.787	42.953	0.00	
2	0.0	0.0	0.0	60.806	103.176	0.000	0.53	
3	0.0	0.0	0.0	156.362	205.260	111.095	7.26	
4	0.0	0.0	0.0	240.708	128.191	101.565	21.28	
...	...	...	...	...	...	...	...	
69994	0.0	0.0	0.0	15.760	410.924	329.136	0.00	
69995	0.0	0.0	0.0	160.083	289.129	265.772	116.54	
69996	0.0	0.0	0.0	372.088	258.374	279.782	77.13	
69997	0.0	0.0	0.0	238.575	245.414	145.062	14.01	
69998	0.0	0.0	0.0	168.269	42.815	167.961	0.00	

63842 rows × 135 columns

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_siz
```

In [33]:
```
X_train
```

Out[33]:

id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_mou_6
19904	0.0	0.0	0.0	164.410	83.874	127.699	3.13
15687	0.0	0.0	0.0	700.242	795.870	638.072	34.34
22241	0.0	0.0	0.0	172.334	251.812	198.079	142.39
36639	0.0	0.0	0.0	1238.934	691.492	490.938	51.24
58710	0.0	0.0	0.0	864.527	618.235	335.378	722.04
...	...	...	...	...	...	...	...
13325	0.0	0.0	0.0	98.340	149.570	129.124	1.83
62695	0.0	0.0	0.0	224.289	226.306	267.614	3.56
15561	0.0	0.0	0.0	690.116	833.704	660.159	109.81
61499	0.0	0.0	0.0	116.597	177.829	70.943	24.73
42108	0.0	0.0	0.0	93.612	81.601	59.609	9.86

51073 rows × 135 columns

In [34]:
```
pca = PCA(random_state=40)
pca.fit(X_train)
```

Out[34]:
```
PCA
PCA(random_state=40)
```

In [35]:
```
# Principal components
pca.components_
```

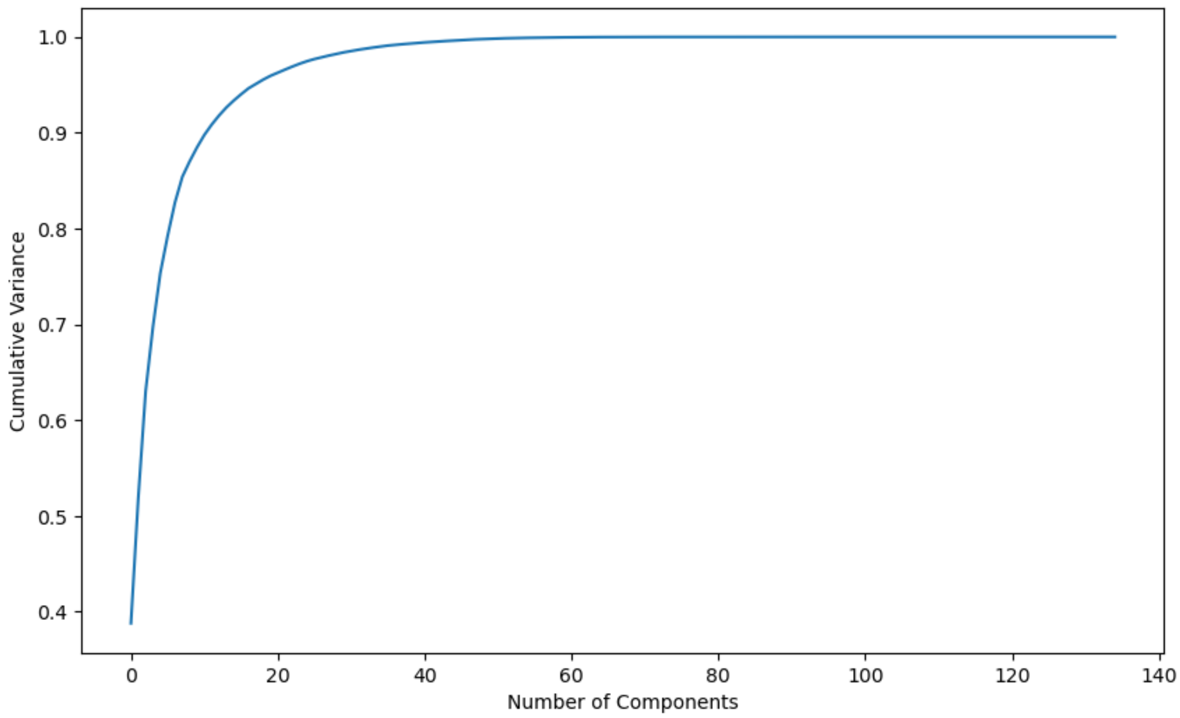
Out[35]:
```
array([[-1.59336414e-19,  1.11022302e-16, -0.00000000e+00, ...,
         2.64738839e-02,  2.13288060e-01,  6.39864181e-01],
       [ 1.81814425e-19, -1.66533454e-16, -1.94289029e-16, ...,
         6.66789193e-02,  4.22317215e-02,  1.26695165e-01],
       [-2.75315783e-19,  5.55111512e-17,  2.22044605e-16, ...,
         1.37991405e-01,  4.19391784e-02,  1.25817535e-01],
       ...,
       [-0.00000000e+00, -3.14203968e-02,  6.88677286e-02, ...,
        -2.11419424e-17,  5.82976775e-06, -7.52610159e-06],
       [ 9.99999916e-01,  2.66387916e-04, -2.58487589e-05, ...,
         4.00646584e-19,  1.70263053e-04,  8.50399352e-05],
       [ 0.00000000e+00, -2.24765643e-01,  8.64221030e-01, ...,
        -5.55111512e-17, -1.20694206e-01,  2.38254730e-01]])
```

In [36]:
```
# Cumuliative varinace of the PCs
variance_cumu = np.cumsum(pca.explained_variance_ratio_)
print(variance_cumu)
```

```
[0.38793958 0.5189239  0.6300407  0.69755679 0.75360874 0.79239937
 0.82727822 0.85398003 0.87013555 0.88472758 0.897722   0.9084958
 0.91797097 0.92636422 0.93356462 0.94010514 0.94624644 0.95086144
 0.95535904 0.95936204 0.9626725  0.96591325 0.96907886 0.97206352
 0.97467583 0.97688534 0.97874304 0.98057436 0.98228089 0.98387903
 0.98530451 0.98659723 0.98780447 0.98892259 0.98998621 0.99093338
 0.99168701 0.99238483 0.99302672 0.9936474  0.99422719 0.99477213
 0.99528123 0.99574287 0.99618939 0.99663043 0.99705851 0.99744064
 0.997752   0.99803366 0.99828855 0.99852705 0.99872953 0.9988894
 0.99902682 0.99916104 0.99928219 0.99938113 0.99946752 0.99954737
 0.99962433 0.99967389 0.99972297 0.99976573 0.99980691 0.99984207
 0.99986408 0.99988542 0.99989958 0.99991178 0.99992325 0.9999328
 0.99994093 0.99994831 0.99995514 0.9999615  0.99996722 0.99997263
 0.99997754 0.999982   0.99998574 0.99998885 0.99999138 0.99999346
 0.99999512 0.99999656 0.99999744 0.99999827 0.99999888 0.99999933
 0.99999958 0.99999969 0.99999978 0.99999985 0.9999999  0.99999992
 0.99999994 0.99999996 0.99999997 0.99999998 0.99999998 0.99999999
 0.99999999 0.99999999 1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.        ]
```

In [37]:
```
# Plotting scree plot
fig = plt.figure(figsize = (10,6))
plt.plot(variance_cumu)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Variance')
```

Out[37]:
```
Text(0, 0.5, 'Cumulative Variance')
```



In [38]:
```
# Importing incremental PCA
from sklearn.decomposition import IncrementalPCA
pca_final = IncrementalPCA(n_components=70)
X_train_pca = pca_final.fit_transform(X_train)
```

In [122]:
```
X_test_pca = pca_final.transform(X_test)
```

In [40]:
```
from sklearn.model_selection import KFold

folds = KFold(n_splits=10, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 0.05, 0.5, 5]}

# Specifing score as recall as we are more focused on acheiving the higher sensiti
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'recall',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_pca, y_train)
```

```
Fitting 10 folds for each of 8 candidates, totalling 80 fits
```

Out[40]:

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

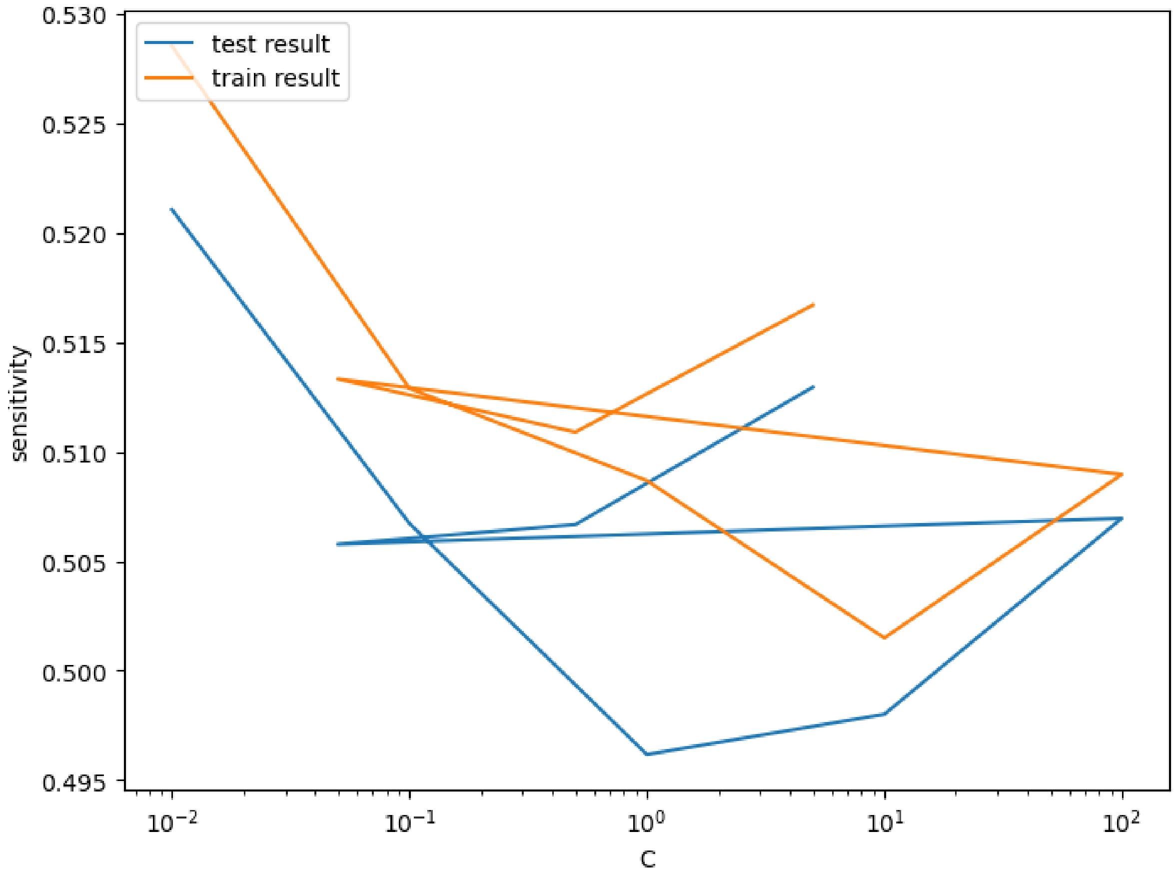
In [41]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

Out[41]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score
0	0.917654	0.041110	0.002856	0.000694	0.01	{'C': 0.01}	0.513889
1	0.907222	0.030095	0.003118	0.000581	0.1	{'C': 0.1}	0.489583
2	0.894384	0.017044	0.003003	0.000641	1	{'C': 1}	0.406250
3	0.908478	0.026851	0.002969	0.000423	10	{'C': 10}	0.493056
4	0.904125	0.023168	0.002979	0.000552	100	{'C': 100}	0.531250
5	0.907335	0.031398	0.002768	0.000271	0.05	{'C': 0.05}	0.520833
6	0.905126	0.027950	0.003294	0.000898	0.5	{'C': 0.5}	0.500000
7	0.894872	0.021792	0.002902	0.000741	5	{'C': 5}	0.527778

In [42]:
```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('sensitivity')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```



In [43]:
```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test sensitivity is {0} at C = {1}".format(best_score, best_C)
```

```
 The highest test sensitivity is 0.5210403311985068 at C = 0.01
```

In [44]:
```
# Instantiate the model with best C
logistic_pca = LogisticRegression(C=best_C)
```

In [118]:
```
# Fit the model on the train set
log_pca_model = logistic_pca.fit(X_train_pca, y_train)
```

Out[118]:
```
array([[-8.55711759e+02,  8.95251078e+01, -6.87048666e+01, ...,
        -3.36729841e-01, -1.60871219e-01,  4.93780246e-01],
       [-1.42011774e+03, -4.34229538e+02, -1.99169014e+02, ...,
         7.37864386e-01, -4.00113216e-01, -1.22816508e-01],
       [-1.11354791e+03,  7.84916539e+02,  6.25202376e+02, ...,
        -1.32456505e+00,  2.80178453e-02,  3.17310310e-01],
       ...,
       [-2.02354105e+02,  6.68341701e+02,  4.37594263e+02, ...,
         5.97052068e-01, -5.86736323e-01,  5.48848380e+00],
       [-1.19040721e+03, -6.29503085e+02, -3.73722473e+02, ...,
        -9.11507725e-01, -3.61361372e-01, -8.59702766e-01],
       [-1.43090387e+03,  2.09546547e+02,  2.66446473e+02, ...,
         1.53052327e+00,  3.73235050e+00, -3.52131310e+00]])
```

In [46]:
```
# Predictions on the train set
y_train_pred = log_pca_model.predict(X_train_pca)
```

In [47]:
```
# Confusion matrix
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion = metrics.confusion_matrix(y_train, y_train_pred)
confusion
```

Out[47]:
```
array([[38634,  9403],
       [ 1676,  1360]], dtype=int64)
```

In [48]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [49]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.7830752060775752
Sensitivity:- 0.4479578392621871
Specificity:- 0.8042550533963403
```

In [50]:
```
# Prediction on the test set
y_test_pred = log_pca_model.predict(X_test_pca)
```

In [51]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[9695 2315]
 [ 446  313]]
```

In [52]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [53]:
```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))
```

```
Accuracy:- 0.783773200720495
Sensitivity:- 0.41238471673254284
Specificity:- 0.8072439633638634
```

In [100]:
```
# Test file
unseen = pd.read_csv("test.csv")
unseen = unseen.set_index("id")
unseen.head()
```

Out[100]:

id	circle_id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	last_date_of_month_6	last_date_c
69999	109	0.0	0.0	0.0	6/30/2014	
70000	109	0.0	0.0	0.0	6/30/2014	
70001	109	0.0	0.0	0.0	6/30/2014	
70002	109	0.0	0.0	0.0	6/30/2014	
70003	109	0.0	0.0	0.0	6/30/2014	

In [101]:
```
unseen['avg_rech_amt'] = (unseen['total_rech_amt_6'] + unseen['total_rech_amt_7']
unseen['total_rech_amt'] = (unseen['total_rech_amt_6'] + unseen['total_rech_amt_7'
```

In [111]:
```
unseen = unseen[X_test.columns]
unseen
```

Out[111]:

id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_mou_6
69999	0.0	0.0	0.0	91.882	65.330	64.445	31.78
70000	0.0	0.0	0.0	414.168	515.568	360.868	75.51
70001	0.0	0.0	0.0	329.844	434.884	746.239	7.54
70002	0.0	0.0	0.0	43.550	171.390	24.400	5.31
70003	0.0	0.0	0.0	306.854	406.289	413.329	450.93
...	...	...	...	...	...	...	...
99994	0.0	0.0	0.0	718.870	396.259	406.150	324.46
99995	0.0	0.0	0.0	218.327	324.070	374.981	263.79
99996	0.0	0.0	0.0	139.473	38.230	180.194	11.08
99997	0.0	0.0	0.0	1122.912	781.121	257.439	122.74
99998	0.0	0.0	0.0	318.980	307.890	605.320	28.09

27343 rows × 135 columns

In [112]:
```
# Checking the percentage of missing values
missing_data = unseen.apply(lambda x: round(x.isnull().mean()* 100, 2)).sort_value
missing_data_above_thresold = missing_data[missing_data>50]
missing_columns_above_thresold = missing_data_above_thresold.index.to_list()
missing_data
```

Out[112]:
```
loc_og_t2o_mou      0.0
std_ic_mou_7        0.0
total_rech_num_6    0.0
ic_others_8         0.0
ic_others_7         0.0
                   ...
std_og_t2f_mou_6    0.0
std_og_t2m_mou_8    0.0
std_og_t2m_mou_7    0.0
std_og_t2m_mou_6    0.0
total_rech_amt      0.0
Length: 135, dtype: float64
```

In [113]:
```
unseen.shape
```

Out[113]:
```
(27343, 135)
```

In [114]:
```
unseen = unseen.dropna(axis = 0)
unseen
```

Out[114]:

id	loc_og_t2o_mou	std_og_t2o_mou	loc_ic_t2o_mou	arpu_6	arpu_7	arpu_8	onnet_mou_6	onnet
99	0.0	0.0	0.0	91.882	65.330	64.445	31.78	
00	0.0	0.0	0.0	414.168	515.568	360.868	75.51	
01	0.0	0.0	0.0	329.844	434.884	746.239	7.54	
02	0.0	0.0	0.0	43.550	171.390	24.400	5.31	
03	0.0	0.0	0.0	306.854	406.289	413.329	450.93	
...	...	...	...	...	...	...	...	
94	0.0	0.0	0.0	718.870	396.259	406.150	324.46	
95	0.0	0.0	0.0	218.327	324.070	374.981	263.79	
96	0.0	0.0	0.0	139.473	38.230	180.194	11.08	
97	0.0	0.0	0.0	1122.912	781.121	257.439	122.74	
98	0.0	0.0	0.0	318.980	307.890	605.320	28.09	

43 rows × 135 columns

In [ ]: