In [1]:
```
#Importing Required libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np

#Reading the dataset
df = pd.read_csv(r'New credit card data.csv')

df.drop(df.index[0], inplace = True)
df = df.astype(int)
df.head(3)
```

Out[1]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19	X20	X21	X22	X
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	689	0	0	0	
2	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1000	1000	1000	0	20
3	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	50

3 rows × 24 columns

Exploratory data analysis

X1: LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit

X2: SEX: Gender (1=male, 2=female)

X3: EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

X4: MARRIAGE: Marital status (1=married, 2=single, 3=others)

X5: AGE: Age in years

X6: PAY_1: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)

X7: PAY_2: Repayment status in August, 2005 (scale same as above)

X8: PAY_3: Repayment status in July, 2005 (scale same as above)

X9: PAY_4: Repayment status in June, 2005 (scale same as above)

X10: PAY_5: Repayment status in May, 2005 (scale same as above)

X11: PAY_6: Repayment status in April, 2005 (scale same as above)

X12: BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)

X13: BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

X14: BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)

X15: BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)

X16: BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)

X17: BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)

X18: PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)

X19: PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)

X20: PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)

X21: PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)

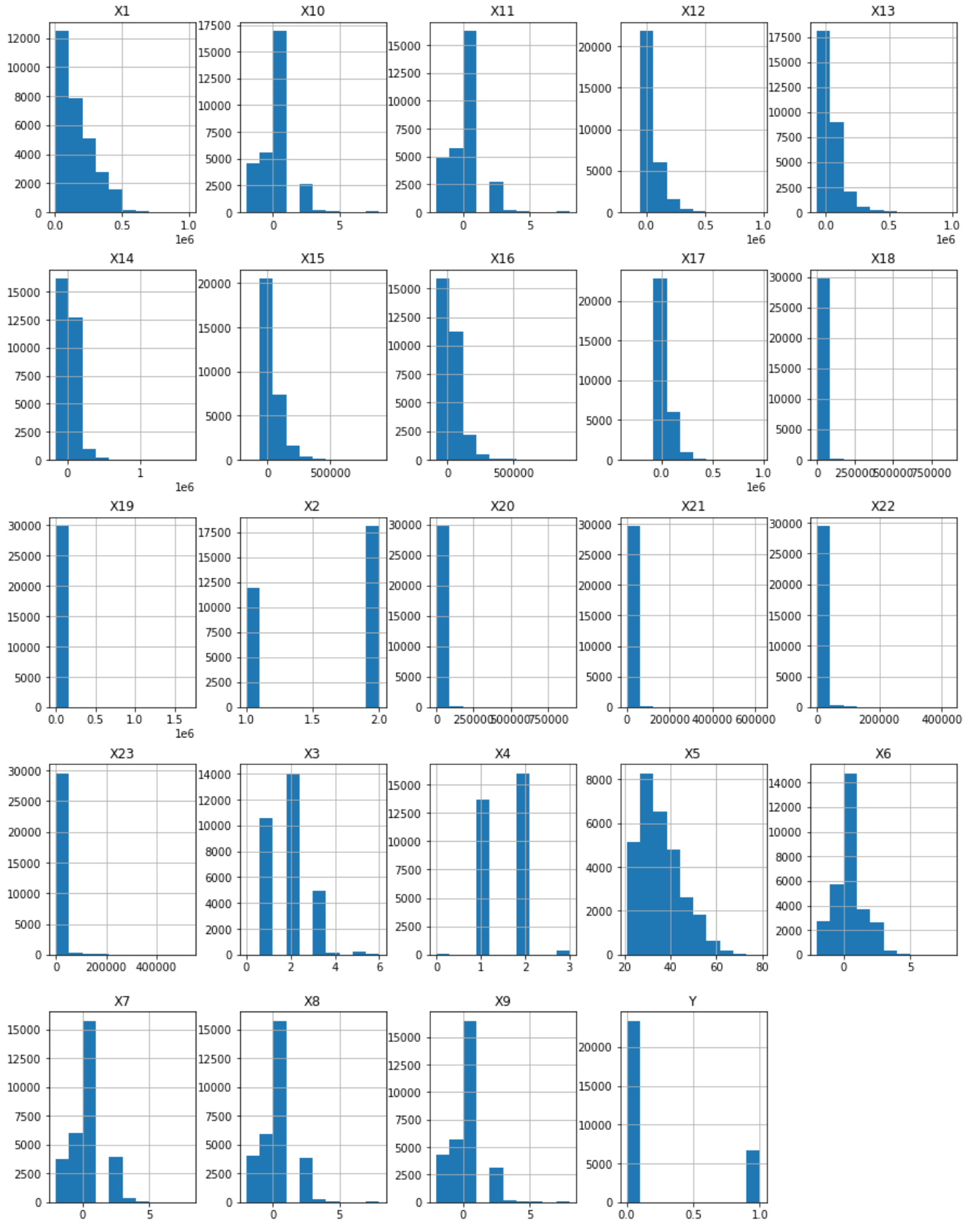
X22: PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)

X23: PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

Y: default.payment.next.month: Default payment (1=yes, 0=no)

In [2]:
```
#Plotting Histogram
fig = plt.figure(figsize = (15,20))
ax = fig.gca()
df.hist(ax = ax)
plt.show()
```

```
<ipython-input-2-195ecd9c3abb>:4: UserWarning: To output multiple subplots, the figure co
ntaining the passed axes is being cleared
  df.hist(ax = ax)
```



In [3]:
```
df.info
```

Out[3]:
```
<bound method DataFrame.info of            X1  X2  X3  X4  X5  X6  X7  X8  X9  X10  ...
X15    X16    X17  \
1        20000   2   2   1  24   2   2  -1  -1   -2  ...      0      0      0
2       120000   2   2   2  26  -1   2   0   0    0  ...   3272   3455   3261
3        90000   2   2   2  34   0   0   0   0    0  ...  14331  14948  15549
4        50000   2   2   1  37   0   0   0   0    0  ...  28314  28959  29547
5        50000   1   2   1  57  -1   0  -1   0    0  ...  20940  19146  19131
...        ...  ..  ..  ..  ..  ..  ..  ..  ..  ...  ...    ...    ...    ...
29996   220000   1   3   1  39   0   0   0   0    0  ...  88004  31237  15980
29997   150000   1   3   2  43  -1  -1  -1  -1    0  ...   8979   5190      0
29998    30000   1   2   2  37   4   3   2  -1    0  ...  20878  20582  19357
29999    80000   1   3   1  41   1  -1   0   0    0  ...  52774  11855  48944
30000    50000   1   2   1  46   0   0   0   0    0  ...  36535  32428  15313

          X18    X19    X20   X21    X22   X23  Y
1           0    689      0     0      0     0  1
2           0   1000   1000  1000      0  2000  1
3        1518   1500   1000  1000   1000  5000  0
4        2000   2019   1200  1100   1069  1000  0
5        2000  36681  10000  9000    689   679  0
...       ...    ...    ...   ...    ...   ... ..
29996    8500  20000   5003  3047   5000  1000  0
29997    1837   3526   8998   129      0     0  0
29998       0      0  22000  4200   2000  3100  1
29999   85900   3409   1178  1926  52964  1804  1
30000    2078   1800   1430  1000   1000  1000  1

[30000 rows x 24 columns]>
```

In [4]:
```
df.describe()
```

Out[4]:

	X1	X2	X3	X4	X5	X6	X7	3
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	
mean	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	
std	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	

8 rows × 24 columns

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
X1     0
X2     0
X3     0
X4     0
X5     0
X6     0
X7     0
X8     0
X9     0
X10    0
X11    0
X12    0
X13    0
X14    0
X15    0
X16    0
X17    0
X18    0
X19    0
X20    0
X21    0
X22    0
X23    0
Y      0
dtype: int64
```

In [6]:
```
# Payment delay description
df[['X6', 'X7', 'X8', 'X9', 'X10', 'X11']].describe()
```

Out[6]:

	X6	X7	X8	X9	X10	X11
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	-0.016700	-0.133767	-0.166200	-0.220667	-0.266200	-0.291100
std	1.123802	1.197186	1.196868	1.169139	1.133187	1.149988
min	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000

In [7]:
```
# Bill Statement description
df[['X12', 'X13', 'X14', 'X15', 'X16', 'X17']].describe()
```

Out[7]:

	X12	X13	X14	X15	X16	X17
count	30000.000000	30000.000000	3.000000e+04	30000.000000	30000.000000	30000.000000
mean	51223.330900	49179.075167	4.701315e+04	43262.948967	40311.400967	38871.760400
std	73635.860576	71173.768783	6.934939e+04	64332.856134	60797.155770	59554.107537
min	-165580.000000	-69777.000000	-1.572640e+05	-170000.000000	-81334.000000	-339603.000000
25%	3558.750000	2984.750000	2.666250e+03	2326.750000	1763.000000	1256.000000
50%	22381.500000	21200.000000	2.008850e+04	19052.000000	18104.500000	17071.000000
75%	67091.000000	64006.250000	6.016475e+04	54506.000000	50190.500000	49198.250000
max	964511.000000	983931.000000	1.664089e+06	891586.000000	927171.000000	961664.000000

In [8]:
```
#Previous Payment Description
df[['X18', 'X19', 'X20', 'X21', 'X22', 'X23']].describe()
```

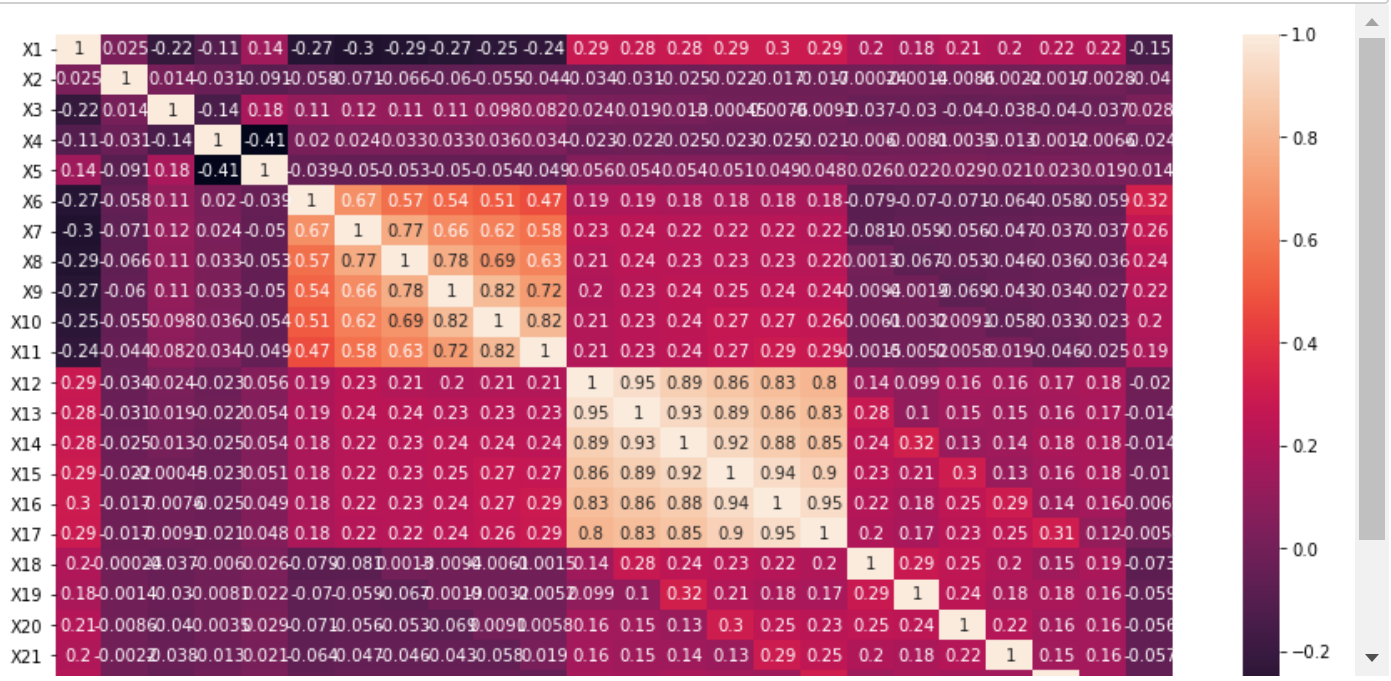
Out[8]:

	X18	X19	X20	X21	X22	X23
count	30000.000000	3.000000e+04	30000.00000	30000.000000	30000.000000	30000.000000
mean	5663.580500	5.921163e+03	5225.68150	4826.076867	4799.387633	5215.502567
std	16563.280354	2.304087e+04	17606.96147	15666.159744	15278.305679	17777.465775
min	0.000000	0.000000e+00	0.00000	0.000000	0.000000	0.000000
25%	1000.000000	8.330000e+02	390.00000	296.000000	252.500000	117.750000
50%	2100.000000	2.009000e+03	1800.00000	1500.000000	1500.000000	1500.000000
75%	5006.000000	5.000000e+03	4505.00000	4013.250000	4031.500000	4000.000000
max	873552.000000	1.684259e+06	896040.00000	621000.000000	426529.000000	528666.000000

In [9]:
```
# Getting a general idea of the default probability
df.Y.sum() / len(df.Y)
```

Out[9]: 0.2212

In [10]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

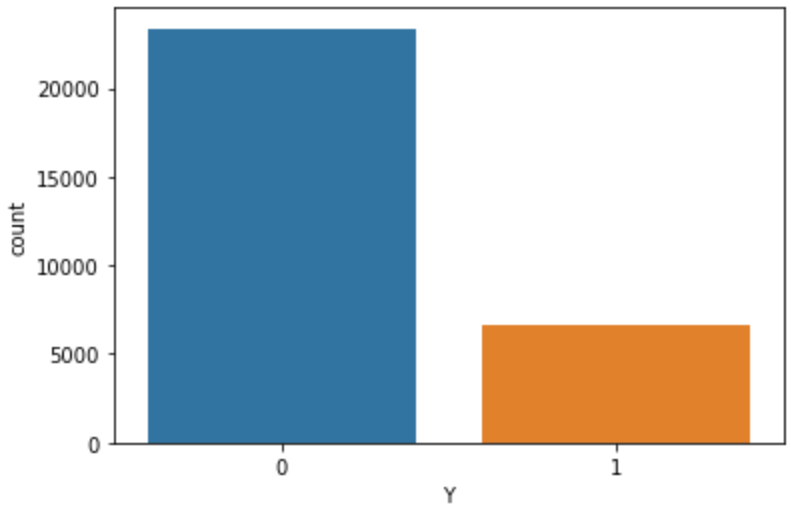


In [11]:
```
#Defining data into X and Y
X = df.drop(['Y'], axis = 1)
y = df['Y']
```

In [12]:
```
# Checking if data is balanced
print('Y = 1: {}%'.format(100*y.mean()))
print('Y = 0: {}%'.format(100 - 100*y.mean()))
```

```
Y = 1: 22.12%
Y = 0: 77.88%
```

In [13]:
```
#Countplot for 'Y'
sns.countplot('Y', data=df)
plt.show()
```



In [14]:
```
y.value_counts()
```

Out[14]:
```
0    23364
1     6636
Name: Y, dtype: int64
```

Balancing the dataset using SMOTE

In [15]:
```
# SMOTE
from imblearn.over_sampling import SMOTE
os = SMOTE(sampling_strategy = 0.9, random_state=1)
X_os, y_os = os.fit_resample(X, y)
```

In [16]:
```
# Resampled data
df_os = pd.DataFrame(X_os, columns = X.columns)
df_os['Y'] = y_os
df_os.head(3)
```

Out[16]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X15	X16	X17	X18	X19	X20	X21	X22	X
0	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	689	0	0	0	
1	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1000	1000	1000	0	20
2	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	50

3 rows × 24 columns

In [17]:
```
print('Y = 1: {}%'.format(100*y_os.mean()))
print('Y = 0: {}%'.format(100 - 100*y_os.mean()))
```

```
Y = 1: 47.36770967087923%
Y = 0: 52.63229032912077%
```

In [18]:
```
X_os=df_os.drop(['Y'], axis = 1)
X_os
```

Out[18]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X14	X15	X16	X17	X18	X19	X
0	20000	2	2	1	24	2	2	-1	-1	-2	...	689	0	0	0	0	689	
1	120000	2	2	2	26	-1	2	0	0	0	...	2682	3272	3455	3261	0	1000	10
2	90000	2	2	2	34	0	0	0	0	0	...	13559	14331	14948	15549	1518	1500	10
3	50000	2	2	1	37	0	0	0	0	0	...	49291	28314	28959	29547	2000	2019	12
4	50000	1	2	1	57	-1	0	-1	0	0	...	35835	20940	19146	19131	2000	36681	100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
44386	273256	2	1	2	27	0	0	0	0	0	...	177319	168629	308684	219119	7996	7546	68
44387	50000	1	2	1	29	0	0	0	0	0	...	24009	11976	8568	5483	2729	2523	8
44388	50000	2	1	2	31	2	2	1	2	2	...	12634	12135	13055	12702	309	1364	
44389	70000	1	3	1	57	2	2	2	2	2	...	71569	68345	72083	70151	3647	2639	
44390	30000	2	2	1	28	1	2	0	0	0	...	19683	19764	20014	20218	1336	1378	4

44391 rows × 23 columns

In [19]:
```
y_os
```

Out[19]:
```
0        1
1        1
2        0
3        0
4        0
        ..
44386    1
44387    1
44388    1
44389    1
44390    1
Name: Y, Length: 44391, dtype: int32
```

In [20]:
```
y_os.value_counts()
```

Out[20]:
```
0    23364
1    21027
Name: Y, dtype: int64
```

In [21]:
```
print(df_os.shape)
```

```
(44391, 24)
```

In [22]:
```
#Standardising the Values
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
df_os=sc.fit_transform(df_os)
df_os
```

Out[22]:
```
array([[-1.07320263,  0.97338902,  0.30230937, ..., -0.30664477,
        -0.28226256,  1.05410759],
       [-0.27857153,  0.97338902,  0.30230937, ..., -0.30664477,
        -0.15680221,  1.05410759],
       [-0.51696086,  0.97338902,  0.30230937, ..., -0.23250596,
         0.0313883 , -0.94866976],
       ...,
       [-0.8348133 ,  0.97338902, -1.04777264, ..., -0.30664477,
        -0.21614496,  1.05410759],
       [-0.67588708, -1.02733849,  1.65239138, ..., -0.28907387,
        -0.13340386,  1.05410759],
       [-0.99373952,  0.97338902,  0.30230937, ..., -0.25304241,
        -0.24506357,  1.05410759]])
```

Detection of Outliers

In [24]:
```
# Isolation forest
from sklearn.ensemble import IsolationForest
ifc = IsolationForest(random_state = 1)
ifc.fit(df_os)
ifc_pred = ifc.predict(df_os)
```

In [25]:
```
ifc_pred[ifc_pred == 1] = 0
ifc_pred[ifc_pred == -1] = 1
```

In [26]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

print('Accuracy  :\t', accuracy_score(y_os, ifc_pred))
print('Recall    :\t', recall_score(y_os, ifc_pred))
print('Precision :\t', precision_score(y_os, ifc_pred))
print('F1-score  :\t', f1_score(y_os, ifc_pred))
```

```
Accuracy  :      0.508684192741772
Recall    :      0.08013506444095686
Precision :      0.4057307970142066
F1-score  :      0.13383637807783955
```

In [27]:
```
# Constructing the confusion matrix.
from sklearn.metrics import confusion_matrix
confusion_matrix(y_os, ifc_pred)
```

Out[27]:
```
array([[20896,  2468],
       [19342,  1685]], dtype=int64)
```

In [28]:
```
#Local Outlier Factor
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None)

clf= LocalOutlierFactor()
lof_pred = clf.fit_predict(X_os)

from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

print('Accuracy  :\t', accuracy_score(y_os, lof_pred))
```

```
Accuracy  :      0.4022662251357257
```

In [29]:
```
# Remove outliers using Isolation Forest
df_os = pd.DataFrame(df_os, columns = df.columns)
df_no_outliers = df_os.copy()
df_no_outliers['ifc'] = ifc_pred
```

In [30]:
```
# Filter outliers
df_no_outliers = df_no_outliers[df_no_outliers['ifc'] == 0]
df_no_outliers.drop(['ifc'], axis = 1, inplace = True)
```

In [31]:
```
df_no_outliers.head(3)
```

Out[31]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	..
0	-1.073203	0.973389	0.302309	-0.894165	-1.291334	1.646442	1.625821	-0.793507	-0.748569	-1.575255	..
1	-0.278572	0.973389	0.302309	1.042924	-1.065086	-1.022273	1.625821	0.026031	0.077959	0.122686	..
2	-0.516961	0.973389	0.302309	1.042924	-0.160093	-0.132701	-0.013684	0.026031	0.077959	0.122686	..

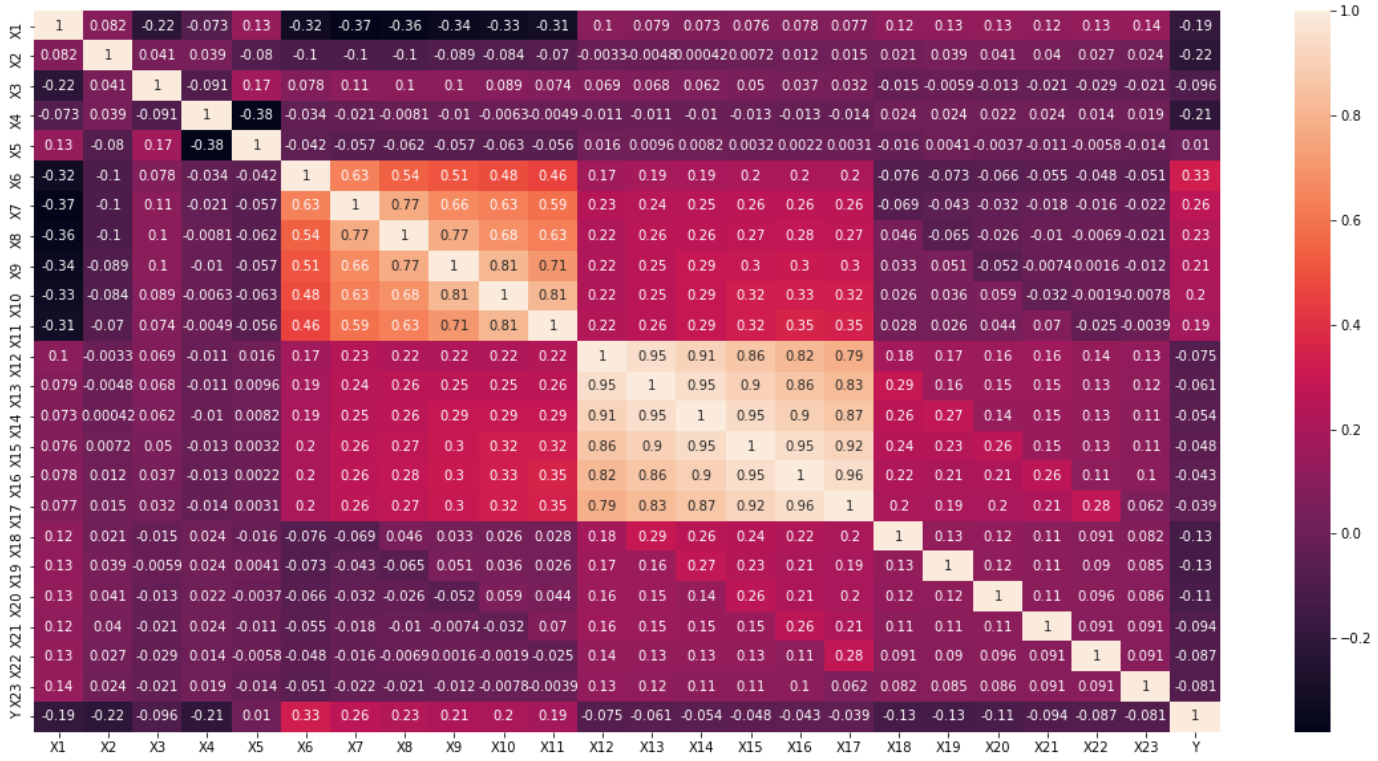
3 rows × 24 columns

In [32]:
```
print(df_no_outliers.shape)
```

```
(40238, 24)
```

In [33]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df_no_outliers.corr(),annot=True)
```

Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dc7b0fabb0>
```



In [34]:
```
#Standardising the Values
from sklearn.preprocessing import RobustScaler
rsc=RobustScaler()
df_no_outliers=rsc.fit_transform(df_no_outliers)
df_no_outliers
```

Out[34]:
```
array([[-0.53333333,  0.        ,  0.        , ..., -0.39333568,
        -0.37175619,  1.        ],
       [ 0.13333333,  0.        ,  0.        , ..., -0.39333568,
         0.31795844,  1.        ],
       [-0.06666667,  0.        ,  0.        , ..., -0.04072638,
         1.35253039,  0.        ],
       ...,
       [-0.33333333,  0.        , -1.        , ..., -0.39333568,
        -0.00827658,  1.        ],
       [-0.2       , -1.        ,  1.        , ..., -0.30976728,
         0.44659022,  1.        ],
       [-0.46666667,  0.        ,  0.        , ..., -0.13839915,
        -0.1672558 ,  1.        ]])
```

In [62]:
```
#Redifining dataframe after Smote
df_no_outliers = pd.DataFrame(df_no_outliers, columns = df.columns)
X_os_no1= df_no_outliers.drop(['Y'], axis = 1)
y_os_no = df_no_outliers['Y']
y_os_no
```

Out[62]:
```
0        1.0
1        1.0
2        0.0
3        0.0
4        0.0
        ...
40233    1.0
40234    1.0
40235    1.0
40236    1.0
40237    1.0
Name: Y, Length: 40238, dtype: float64
```

In [63]:
```
X_os_no1
```

Out[63]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X14	X15	X16	
0	-0.533333	0.0	0.0	0.0	-0.769231	2.0	2.0	-1.0	-1.0	-2.0	...	-0.374931	-0.397036	-0.408193	-0.38
1	0.133333	0.0	0.0	1.0	-0.615385	-1.0	2.0	0.0	0.0	0.0	...	-0.332782	-0.323064	-0.321440	-0.30
2	-0.066667	0.0	0.0	1.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	-0.102750	-0.073045	-0.032856	0.01
3	-0.333333	0.0	0.0	0.0	0.230769	0.0	0.0	0.0	0.0	0.0	...	0.652927	0.243078	0.318954	0.37
4	-0.333333	-1.0	0.0	0.0	1.769231	-1.0	0.0	-1.0	0.0	0.0	...	0.368353	0.076369	0.072554	0.10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
40233	-0.533333	-1.0	0.0	1.0	-0.538462	1.0	-2.0	-2.0	-2.0	-2.0	...	-0.389502	-0.397036	-0.408193	-0.38
40234	-0.333333	-1.0	0.0	0.0	-0.384615	0.0	0.0	0.0	0.0	0.0	...	0.118252	-0.126287	-0.193055	-0.24
40235	-0.333333	0.0	-1.0	1.0	-0.230769	2.0	2.0	1.0	2.0	2.0	...	-0.122312	-0.122692	-0.080388	-0.05
40236	-0.200000	-1.0	1.0	0.0	1.769231	2.0	2.0	2.0	2.0	2.0	...	1.124073	1.148086	1.401778	1.41
40237	-0.466667	0.0	0.0	0.0	-0.461538	1.0	2.0	0.0	0.0	0.0	...	0.026763	0.049782	0.094349	0.13

40238 rows × 23 columns

In [37]:
```
df_no_outliers['Y'].value_counts()
```

Out[37]:
```
0.0    20896
1.0    19342
Name: Y, dtype: int64
```

Removing Insignificant Variables

In [83]:
```
#PCA
from sklearn.model_selection import train_test_split
X_os_no_train, X_os_no_test, y_os_no_train, y_os_no_test = train_test_split(X_os_no1,y_os_n

from sklearn.decomposition import PCA
X_os_no_train_PCA= X_os_no_train.copy()
pca = PCA()
X_os_no_train = pca.fit_transform(X_os_no_train_PCA)
X_os_no_test = pca.transform(X_os_no_test)
```

In [84]:
```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[84]:
```
array([0.21137073, 0.15302731, 0.13433436, 0.11340517, 0.09224196,
       0.08273505, 0.07515502, 0.052341  , 0.01582784, 0.01197054,
       0.01042398, 0.01015898, 0.00680291, 0.00627985, 0.00531618,
       0.0046952 , 0.00442909, 0.00376427, 0.0032072 , 0.00119221,
       0.00060948, 0.00038723, 0.00032446])
```

In [85]:
```
#Logistic Regression after PCA
'''Model Assumptions:
Target variable is binary
Predictive features are interval (continuous) or categorical
Features are independent of one another
Sample size is adequate
'''

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 0)
lr.fit(X_os_no_train, y_os_no_train)
y_predlr = lr.predict(X_os_no_test)
```

In [86]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
cm = confusion_matrix(y_os_no_test, y_predlr)
acc = accuracy_score(y_os_no_test, y_predlr)
pre = precision_score(y_os_no_test, y_predlr)
recall = recall_score(y_os_no_test, y_predlr)
f1 = f1_score(y_os_no_test, y_predlr)

print(cm)
print(acc)
print(pre)
print(recall)
print(f1)
```

```
[[4638 1654]
 [1777 4003]]
0.7157886017229954
0.7076188792646279
0.692560553633218
0.7000087435516307
```

In [87]:
```
#Random Forest after PCA

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 50)
classifier.fit(X_os_no_train, y_os_no_train)

y_predRF= classifier.predict(X_os_no_test)

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_os_no_test, y_predRF)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_os_no_test, y_predRF)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_os_no_test,y_predRF)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[5217 1075]
 [1418 4362]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.83      0.81      6292
         1.0       0.80      0.75      0.78      5780

    accuracy                           0.79     12072
   macro avg       0.79      0.79      0.79     12072
weighted avg       0.79      0.79      0.79     12072

Accuracy: 0.7934890656063618
```

In [88]:
```
#Splitting cleaned dataset into train and test
from sklearn.model_selection import train_test_split
X_os_no1_train,X_os_no1_test,y_os_no_train,y_os_no_test=train_test_split(X_os_no1,y_os_no,
```

In [89]:
```
#Logistic Regression
import statsmodels.api as sm

model = sm.Logit(y_os_no_train, X_os_no1_train).fit()
predictions = model.predict(X_os_no1_train)

print_model = model.summary()
print(print_model)
```

```
Optimization terminated successfully.
         Current function value: 0.553757
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      Y   No. Observations:                28166
Model:                          Logit   Df Residuals:                    28143
Method:                           MLE   Df Model:                           22
Date:                Sat, 17 Oct 2020   Pseudo R-squ.:                  0.2003
Time:                        20:51:40   Log-Likelihood:                -15597.
converged:                       True   LL-Null:                       -19504.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
X1            -0.1701      0.022     -7.740      0.000      -0.213      -0.127
X2            -0.8892      0.024    -36.497      0.000      -0.937      -0.841
X3            -0.5464      0.021    -25.550      0.000      -0.588      -0.504
X4            -0.9875      0.026    -38.413      0.000      -1.038      -0.937
X5            -0.0877      0.022     -3.932      0.000      -0.131      -0.044
X6             0.5522      0.017     31.699      0.000       0.518       0.586
X7             0.1226      0.020      6.005      0.000       0.083       0.163
X8             0.0459      0.023      2.020      0.043       0.001       0.090
X9             0.0514      0.025      2.050      0.040       0.002       0.101
X10            0.0488      0.027      1.830      0.067      -0.003       0.101
X11            0.0047      0.022      0.214      0.831      -0.038       0.048
X12           -0.5442      0.063     -8.702      0.000      -0.667      -0.422
X13            0.2206      0.086      2.563      0.010       0.052       0.389
X14            0.1984      0.078      2.551      0.011       0.046       0.351
X15            0.1195      0.076      1.568      0.117      -0.030       0.269
X16           -0.3481      0.088     -3.943      0.000      -0.521      -0.175
X17            0.1926      0.070      2.737      0.006       0.055       0.330
X18           -0.1144      0.011    -10.730      0.000      -0.135      -0.093
X19           -0.1006      0.011     -9.282      0.000      -0.122      -0.079
X20           -0.0412      0.008     -5.337      0.000      -0.056      -0.026
X21           -0.0218      0.008     -2.586      0.010      -0.038      -0.005
X22           -0.0427      0.007     -5.708      0.000      -0.057      -0.028
X23           -0.0225      0.005     -4.184      0.000      -0.033      -0.012
==============================================================================
```

In [51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report

#Build the model
clf = LogisticRegression()

# Train the classifier
clf.fit(X_os_no_train, y_os_no_train)

#test the model
y_os_no_pred = clf.predict(X_os_no_test)

#classification report
cr = (classification_report(y_os_no_test, y_os_no_pred))
```

In [66]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
confm = confusion_matrix(y_os_no_test, y_os_no_pred)
accuracy= accuracy_score(y_os_no_test, y_os_no_pred)
pre = precision_score(y_os_no_test, y_os_no_pred)
recall = recall_score(y_os_no_test, y_os_no_pred)
f1_score = f1_score(y_os_no_test, y_os_no_pred)

print("Confusion Matrix: \n", confm)
print("Accuracy:", accuracy)
print("Precision: ", pre)
print("Recall: ", recall)
print("F1 Score: ", f1_score)
```

```
Confusion Matrix:
 [[4631 1661]
 [2034 3746]]
Accuracy: 0.6939198144466534
Precision:  0.6928056223414093
Recall:  0.6480968858131488
F1 Score:  0.6697059086439617
```

In [67]:
```
#Dropping Insignificant Variables
df_reg= df_no_outliers.drop(['X10','X11','X15'], axis = 1)
X_reg= df_reg.drop(['Y'], axis = 1)
y_reg = df_reg['Y']
y_reg
```

Out[67]:
```
0        1.0
1        1.0
2        0.0
3        0.0
4        0.0
        ...
40233    1.0
40234    1.0
40235    1.0
40236    1.0
40237    1.0
Name: Y, Length: 40238, dtype: float64
```

In [68]:
```
X_reg
```

Out[68]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X12	X13	X14	X16	
0	-0.533333	0.0	0.0	0.0	-0.769231	2.0	2.0	-1.0	-1.0	-0.303331	-0.324320	-0.374931	-0.408193	-0.
1	0.133333	0.0	0.0	1.0	-0.615385	-1.0	2.0	0.0	0.0	-0.327969	-0.352710	-0.332782	-0.321440	-0.
2	-0.066667	0.0	0.0	1.0	0.000000	0.0	0.0	0.0	0.0	0.203575	-0.099084	-0.102750	-0.032856	0.
3	-0.333333	0.0	0.0	0.0	0.230769	0.0	0.0	0.0	0.0	0.558865	0.606129	0.652927	0.318954	0.
4	-0.333333	-1.0	0.0	0.0	1.769231	-1.0	0.0	-1.0	0.0	-0.209179	-0.271377	0.368353	0.072554	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
40233	-0.533333	-1.0	0.0	1.0	-0.538462	1.0	-2.0	-2.0	-2.0	-0.381650	-0.388273	-0.389502	-0.408193	-0.
40234	-0.333333	-1.0	0.0	0.0	-0.384615	0.0	0.0	0.0	0.0	0.592971	0.522900	0.118252	-0.193055	-0.
40235	-0.333333	0.0	-1.0	1.0	-0.230769	2.0	2.0	1.0	2.0	-0.142748	-0.145863	-0.122312	-0.080388	-0.
40236	-0.200000	-1.0	1.0	0.0	1.769231	2.0	2.0	2.0	2.0	0.999680	1.072333	1.124073	1.401778	1.
40237	-0.466667	0.0	0.0	0.0	-0.461538	1.0	2.0	0.0	0.0	-0.021096	-0.001443	0.026763	0.094349	0.

40238 rows × 20 columns

In [69]:
```
#Standardising the Values
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
df_reg=sc.fit_transform(df_reg)
X_reg=sc.fit_transform(X_reg)
df_reg
```

Out[69]:
```
array([[-1.05195268,  0.96660914,  0.2963766 , ..., -0.36512904,
        -0.33488584,  1.03939564],
       [-0.1721172 ,  0.96660914,  0.2963766 , ..., -0.36512904,
        -0.10846244,  1.03939564],
       [-0.43606785,  0.96660914,  0.2963766 , ..., -0.23598758,
         0.23117265, -0.96209755],
       ...,
       [-0.78800204,  0.96660914, -1.07611753, ..., -0.36512904,
        -0.21556071,  1.03939564],
       [-0.61203494, -1.03454432,  1.66887072, ..., -0.33452251,
        -0.06623448,  1.03939564],
       [-0.96396913,  0.96660914,  0.2963766 , ..., -0.27175976,
        -0.2677513 ,  1.03939564]])
```

In [70]:
```
#Splitting new cleaned dataset into train and test
from sklearn.model_selection import train_test_split
X_reg_train,X_reg_test,y_reg_train,y_reg_test=train_test_split(X_reg,y_reg,test_size=0.30,
```

In [71]:
```
#Logistic Regression after removing insignificant variable
import statsmodels.api as sm

model = sm.Logit(y_reg_train, X_reg_train).fit()
predictions = model.predict(X_reg_train)

print_model = model.summary()
print(print_model)
```

```
Optimization terminated successfully.
         Current function value: 0.554359
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      Y   No. Observations:                28166
Model:                          Logit   Df Residuals:                    28146
Method:                           MLE   Df Model:                           19
Date:                Sat, 17 Oct 2020   Pseudo R-squ.:                  0.1994
Time:                        20:47:18   Log-Likelihood:                -15614.
converged:                       True   LL-Null:                       -19504.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.1394      0.016     -8.451      0.000      -0.172      -0.107
x2            -0.4049      0.014    -29.117      0.000      -0.432      -0.378
x3            -0.3855      0.015    -24.905      0.000      -0.416      -0.355
x4            -0.5545      0.016    -35.745      0.000      -0.585      -0.524
x5            -0.0798      0.016     -5.140      0.000      -0.110      -0.049
x6             0.5819      0.019     30.857      0.000       0.545       0.619
x7             0.1574      0.024      6.521      0.000       0.110       0.205
x8             0.0572      0.026      2.169      0.030       0.006       0.109
x9             0.1090      0.023      4.649      0.000       0.063       0.155
x10           -0.5283      0.059     -8.921      0.000      -0.644      -0.412
x11            0.1861      0.080      2.312      0.021       0.028       0.344
x12            0.2258      0.063      3.560      0.000       0.101       0.350
x13           -0.2310      0.074     -3.102      0.002      -0.377      -0.085
x14            0.1584      0.067      2.371      0.018       0.027       0.289
x15           -0.2425      0.022    -10.852      0.000      -0.286      -0.199
x16           -0.2216      0.023     -9.526      0.000      -0.267      -0.176
x17           -0.0824      0.016     -5.165      0.000      -0.114      -0.051
x18           -0.0641      0.017     -3.831      0.000      -0.097      -0.031
x19           -0.1164      0.020     -5.806      0.000      -0.156      -0.077
x20           -0.0728      0.016     -4.507      0.000      -0.105      -0.041
==============================================================================
```

In [72]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report

#Build the model
clf = LogisticRegression()

# Train the classifier
clf.fit(X_reg_train, y_reg_train)

#test the model
y_reg_pred = clf.predict(X_reg_test)

#classification report
cr = (classification_report(y_reg_test, y_reg_pred))
```

In [73]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
confm = confusion_matrix(y_reg_test, y_reg_pred)
accuracy= accuracy_score(y_reg_test, y_reg_pred)
pre = precision_score(y_reg_test, y_reg_pred)
recall = recall_score(y_reg_test, y_reg_pred)
f1_score = f1_score(y_reg_test, y_reg_pred)

print("Confusion Matrix: \n", confm)
print("Accuracy:", accuracy)
print("Precision: ", pre)
print("Recall: ", recall)
print("F1 Score: ", f1_score)
```

```
Confusion Matrix:
 [[4637 1655]
 [1777 4003]]
Accuracy: 0.7157057654075547
Precision:  0.7074938140685755
Recall:  0.692560553633218
F1 Score:  0.6999475432767968
```

Building Predictive Models

In [74]:
```
#Random Forest

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 50)
classifier.fit(X_reg_train, y_reg_train)
```

Out[74]:
```
RandomForestClassifier(n_estimators=50)
```

In [75]:
```
y_predrf = classifier.predict(X_reg_test)
```

In [76]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result = confusion_matrix(y_reg_test, y_predrf)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_reg_test, y_predrf)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_reg_test, y_predrf)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[5383  909]
 [1247 4533]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.86      0.83      6292
         1.0       0.83      0.78      0.81      5780

    accuracy                           0.82     12072
   macro avg       0.82      0.82      0.82     12072
weighted avg       0.82      0.82      0.82     12072

Accuracy: 0.8214049039098741
```

In [77]:
```
#SVC

from sklearn import svm
clf = svm.SVC()
clf.fit(X_reg_train, y_reg_train)
```

Out[77]:
```
SVC()
```

In [78]:
```
predictions = clf.predict(X_reg_test)
print("Size of training set: ", X_reg_test.shape)
print(predictions.shape)
```

```
Size of training set:  (12072, 20)
(12072,)
```

In [79]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_reg_test,predictions))
```

```
[[5036 1256]
 [1708 4072]]
```

In [80]:
```
print(classification_report(y_reg_test,predictions))
from sklearn.metrics import accuracy_score
accuracy_score(y_reg_test, predictions)
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.80      0.77      6292
         1.0       0.76      0.70      0.73      5780

    accuracy                           0.75     12072
   macro avg       0.76      0.75      0.75     12072
weighted avg       0.76      0.75      0.75     12072
```

Out[80]:
```
0.7544731610337972
```

In [81]:
```
#K Nearest Neighbours

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier()

# fit the model with the training data
model.fit(X_reg_train, y_reg_train)

# Number of Neighbors used to predict the target
print('\nThe number of neighbors used to predict the target : ',model.n_neighbors)

y_predKNN= model.predict(X_reg_test)

result = confusion_matrix(y_reg_test, y_predKNN)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_reg_test, y_predKNN)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_reg_test,y_predKNN)
print("Accuracy:",result2)
```

```
The number of neighbors used to predict the target :  5
Confusion Matrix:
[[4667 1625]
 [1407 4373]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.77      0.74      0.75      6292
         1.0       0.73      0.76      0.74      5780

    accuracy                           0.75     12072
   macro avg       0.75      0.75      0.75     12072
weighted avg       0.75      0.75      0.75     12072

Accuracy: 0.7488402915838304
```