Exploratory Data Analysis

In [18]:
```
#Main Libraries
import pandas as pd
import numpy as np
import time

#Data visualisation libraries
import seaborn as sns
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.style.use('ggplot')

#ML Libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recal
from sklearn.externals import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

###Reading the csv file and renaming last Column to Default which is easier to re
df1 = pd.read_csv('Desktop\card.csv')
df1.head()
```

Checking for the dimensions of the dataframe

In [2]:
```
df1.shape
```

Out[2]:
```
(30000, 25)
```

There are 30000 records in the dataset and 25 characteristics, of which the last column, 'Default' is the target variable

Checking for the datatypes and nullity of the variables

In [4]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
ID                            30000 non-null int64
LIMIT_BAL                     30000 non-null int64
SEX                           30000 non-null int64
EDUCATION                     30000 non-null int64
MARRIAGE                      30000 non-null int64
AGE                           30000 non-null int64
PAY_1                         30000 non-null int64
PAY_2                         30000 non-null int64
PAY_3                         30000 non-null int64
PAY_4                         30000 non-null int64
PAY_5                         30000 non-null int64
PAY_6                         30000 non-null int64
BILL_AMT_SEP                  30000 non-null int64
BILL_AMT_AUG                  30000 non-null int64
BILL_AMT_JUL                  30000 non-null int64
BILL_AMT_JUN                  30000 non-null int64
BILL_AMT_MAY                  30000 non-null int64
BILL_AMT_APR                  30000 non-null int64
PAY_AMT_SEP                   30000 non-null int64
PAY_AMT_AUG                   30000 non-null int64
PAY_AMT_JUL                   30000 non-null int64
PAY_AMT_JUN                   30000 non-null int64
PAY_AMT_MAY                   30000 non-null int64
PAY_AMT_APR                   30000 non-null int64
default payment next month    30000 non-null int64
dtypes: int64(25)
memory usage: 5.7 MB
```

The data only has integer values and has no missing values

Generating summary statistics for the variables

In [17]:
```
df_withoutID = df1.iloc[:, 1:24]
df_withoutID.describe()
```

Out[17]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	300
mean	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	
std	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	

8 rows × 23 columns

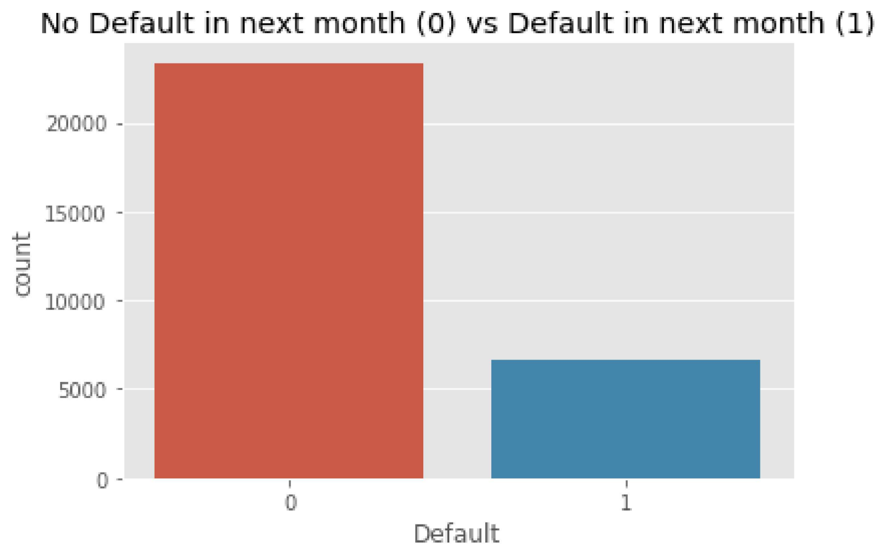
We an see that all the variables barring sex, education, marriage and default are continuous and numerical in nature. Sex, Education and Marriage are categorical variables that have been converted to a numerical variable for easy model fitting.

From the summary statistics, we can see that the bank deals with clients in their 30s-40s that borrow larger amounts (Mean for limit_balance > median; ~50% of Age records are from the 30s-40s range, and mean<median for pay status across all months)

Generating a barplot for the target variable to understand the count of the target variable in the dataset

In [5]:
```
sns.countplot('Default', data=df1)
plt.title('No Default in next month (0) vs Default in next month (1)')
```

Out[5]:
```
Text(0.5,1,'No Default in next month (0) vs Default in next month (1)')
```

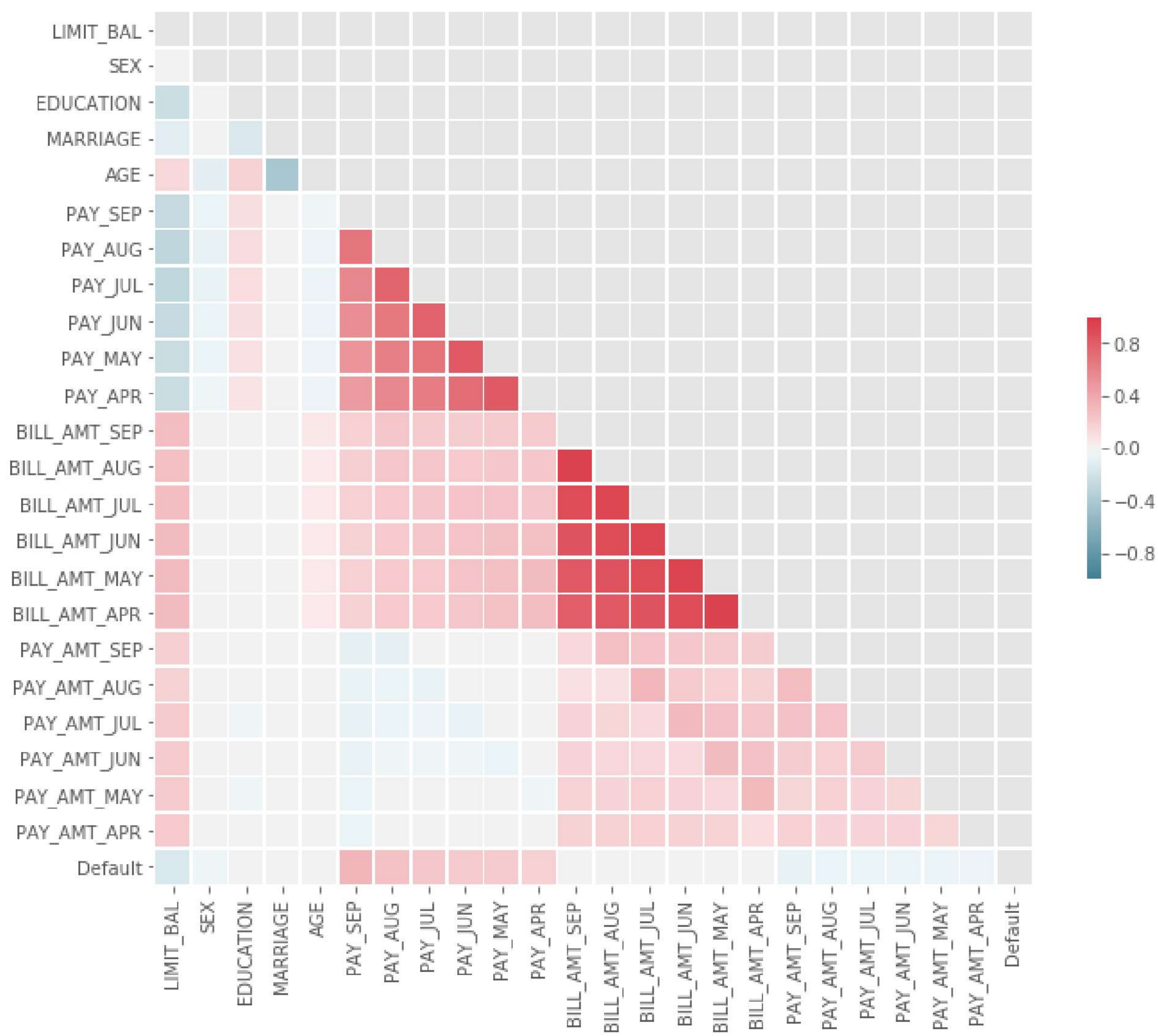


We can see that bulk of the target variable do not default, implying we may need to apply methods to make sample more balanced

Visualing the correlation matrix to understand the linear relationship between the variables

In [6]:
```
corr = df_withoutID.corr() #Computing correlation matrix
mask = np.zeros_like(corr, dtype=np.bool) #Generating a mask for upper triangle
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9)) #Setting up matplotlib figure
cmap = sns.diverging_palette(220, 10, as_cmap=True) #Generating a custom divergin
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, vmin=-1, center=0, linewidths=.7,
```

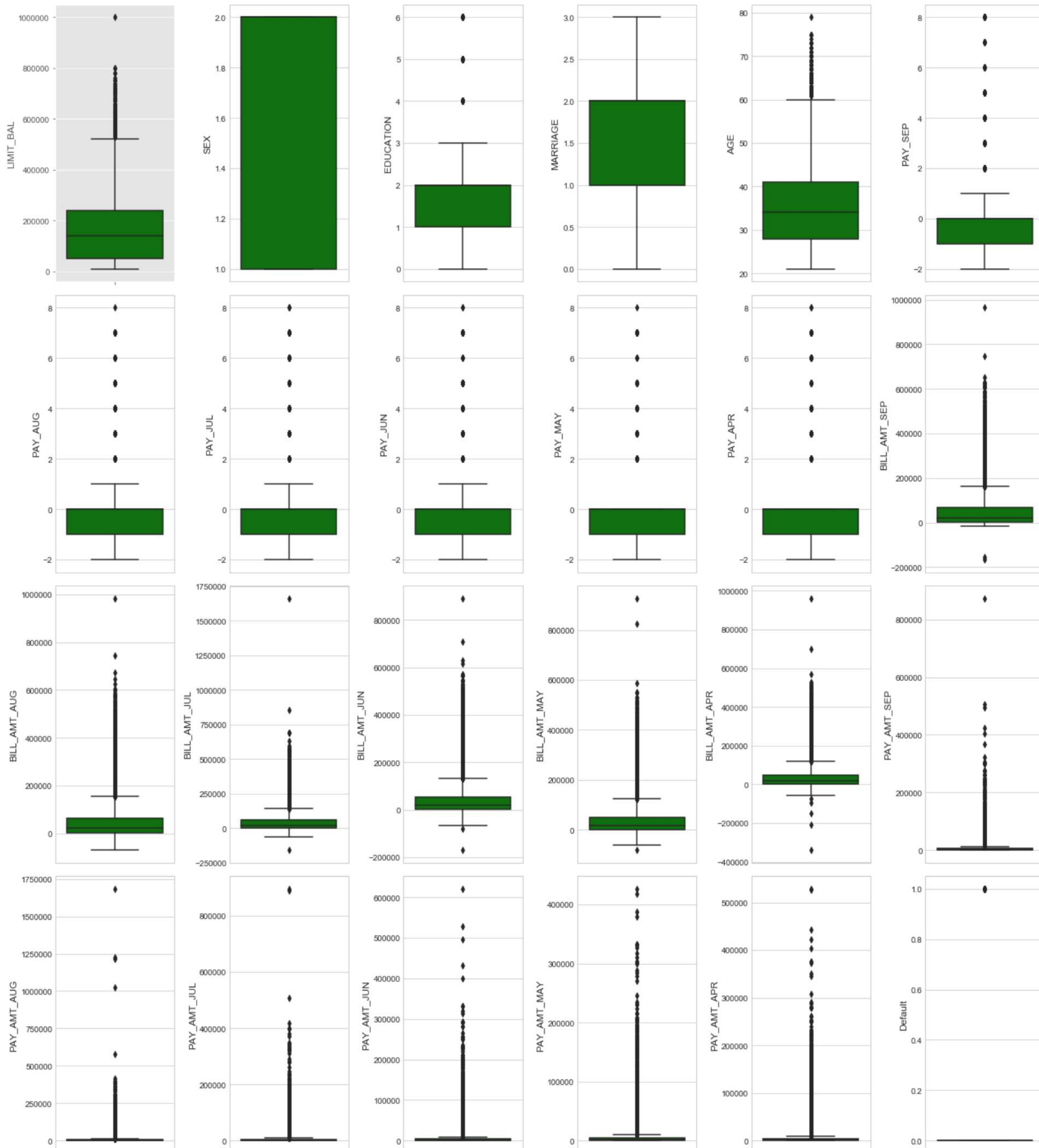
Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f63a436cc0>
```



From the correlation heat map, we can see that the limit balance and the amount the customers paid has a negative correlation with default in the next month; and the history of past payment has a positive correlation with default in the next month.

Generating barplots to understand the distribution of the variables and to look for potential outliers

In [7]:
```
l = df_withoutID.columns.values
number_of_columns=6
number_of_rows = len(l)-1/number_of_columns
plt.figure(figsize=(3*number_of_columns,5*number_of_rows))
for i in range(0,len(l)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    sns.set_style('whitegrid')
    sns.boxplot(df_withoutID[l[i]],color='green',orient='v')
    plt.tight_layout()
```

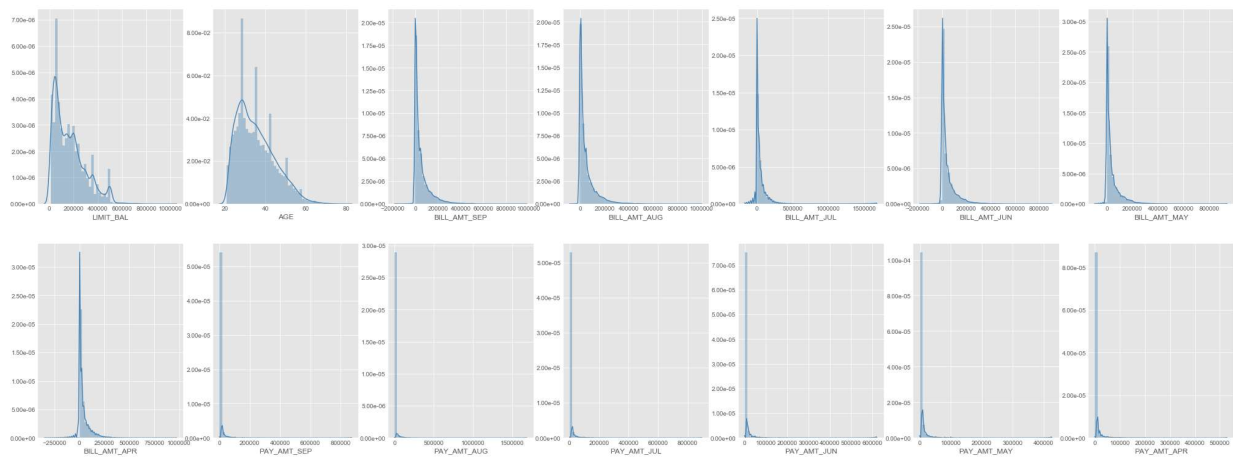


Generating distribution graphs for continuous variables

In [132]:
```
import matplotlib.ticker as mtick
df_Cont_Variables = df1[['LIMIT_BAL','AGE','BILL_AMT_SEP', 'BILL_AMT_AUG', 'BILL_
l = df_Cont_Variables.columns.values
number_of_columns= 7
number_of_rows = len(l)-1/number_of_columns
plt.figure(figsize=(5*number_of_columns,7*number_of_rows))
for i in range(0,len(l)):
    plt.subplot(number_of_rows + 1,number_of_columns,i+1)
    b = sns.distplot(df_Cont_Variables[l[i]], kde = True, color ='steelblue')
    b.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
```

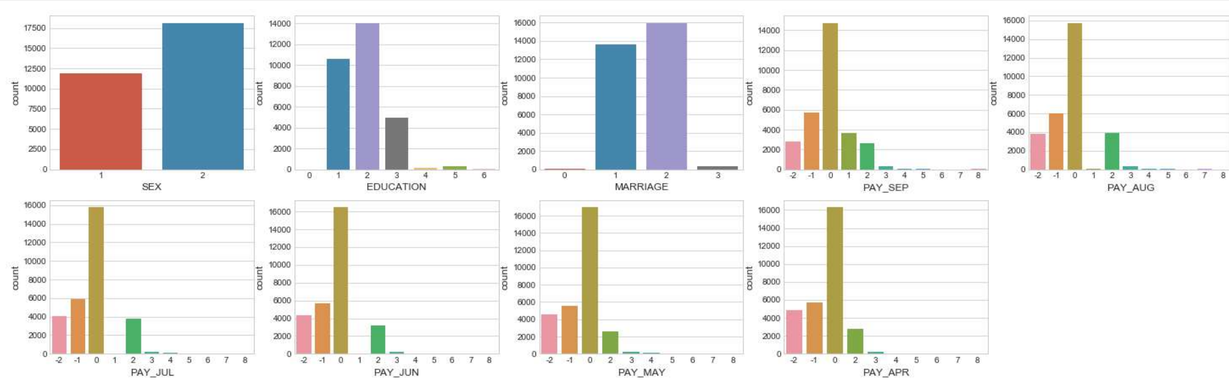
```
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

```
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\65918\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserW
arning: The 'normed' kwarg is deprecated, and has been replaced by the 'densit
y' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



Generating barplots for categorical variables

In [138]:
```python
df_Cat_Variables = df1[['SEX','EDUCATION','MARRIAGE','PAY_SEP','PAY_AUG', 'PAY_JU
l = df_Cat_Variables.columns.values
number_of_columns= 5
number_of_rows = len(l)-1/number_of_columns
plt.figure(figsize=(5*number_of_columns,4*number_of_rows))
for i in range(0,len(l)):
     plt.subplot(number_of_rows + 1,number_of_columns,i+1)
     sns.countplot(l[i], data=df_Cat_Variables)
```



From the plots, we can see that continuous variables are skewed to the right.

Data Preprocessing

Data Validation and Transformation

In [74]:
```python
import pandas as pd
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import train_test_split

df1 = pd.read_csv('card.csv')
print(df1.head())
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1  PAY_2  PAY_3  PAY_4  \
0   1      20000    2          2         1   24      2      2     -1     -1   
1   2     120000    2          2         2   26     -1      2      0      0   
2   3      90000    2          2         2   34      0      0      0      0   
3   4      50000    2          2         1   37      0      0      0      0   
4   5      50000    1          2         1   57     -1      0     -1      0   

              ...              BILL_AMT_JUN  BILL_AMT_MAY  BILL_AMT_APR  \
0             ...                         0             0             0   
1             ...                      3272          3455          3261   
2             ...                     14331         14948         15549   
3             ...                     28314         28959         29547   
4             ...                     20940         19146         19131   

   PAY_AMT_SEP  PAY_AMT_AUG  PAY_AMT_JUL  PAY_AMT_JUN  PAY_AMT_MAY  \
0            0          689            0            0            0   
1            0         1000         1000         1000            0   
2         1518         1500         1000         1000         1000   
3         2000         2019         1200         1100         1069   
4         2000        36681        10000         9000          689   

   PAY_AMT_APR  default payment next month  
0            0                           1  
1         2000                           1  
2         5000                           0  
3         1000                           0  
4          679                           0  

[5 rows x 25 columns]
```

In [75]:
```
#check for Null values --> No null values
print(df1.isnull().values.any())
# check if values are correct
for columns in list(df1):
  break
  #print(df1.groupby([columns])[columns].count())  #check no. of records in each g

#All other categories are OK
#Education: 0,4,5,6 --> can be just 1 category = 4 because all values are unlabel
fil = (df1.EDUCATION == 5) | (df1.EDUCATION == 6) | (df1.EDUCATION == 0)
df1.loc[fil, 'EDUCATION'] = 4
print(df1.EDUCATION.value_counts())

#Payment Status (X6-X11) -2/-1/0 should all be 0
for i in range(1,7):
  column_name = "PAY_{}".format(i)
  condition = (df1[column_name] == -2) | (df1[column_name] == -1)
  df1.loc[condition, column_name] = 0
  print(df1[column_name].value_counts())
```

```
False
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
0    23182
1     3688
2     2667
3      322
4       76
5       26
8       19
6       11
7        9
Name: PAY_1, dtype: int64
0    25562
2     3927
3      326
4       99
1       28
5       25
7       20
6       12
8        1
Name: PAY_2, dtype: int64
0    25787
2     3819
3      240
4       76
7       27
6       23
5       21
1        4
8        3
Name: PAY_3, dtype: int64
```

```
0    26490
2     3159
3      180
4       69
7       58
5       35
6        5
8        2
1        2
Name: PAY_4, dtype: int64
0    27032
2     2626
3      178
4       84
7       58
5       17
6        4
8        1
Name: PAY_5, dtype: int64
0    26921
2     2766
3      184
4       49
7       46
6       19
5       13
8        2
Name: PAY_6, dtype: int64
```

In [76]:
```
#Standardising the data, except categorical data
from sklearn.preprocessing import StandardScaler

categorical_attributes = df1.iloc[:,2:12]
continuous_attributes = df1.drop(df1.iloc[:,2:12], axis = 1)
continuous_attributes = continuous_attributes.drop(labels = ["ID","default paymen
scaler = StandardScaler()
scaler.fit(continuous_attributes)
scaled_continuousdf = pd.DataFrame(scaler.transform(continuous_attributes.values)

#add back categorical columns
final_df = pd.concat((df1.iloc[:,0],categorical_attributes, scaled_continuousdf,d
print(final_df.head())

df1 = final_df
df1.head()
```

```
   ID  SEX  EDUCATION  MARRIAGE  AGE  PAY_1  PAY_2  PAY_3  PAY_4  PAY_5  \
0   1    2          2         1   24      2      2      0      0      0
1   2    2          2         2   26      0      2      0      0      0
2   3    2          2         2   34      0      0      0      0      0
3   4    2          2         1   37      0      0      0      0      0
4   5    1          2         1   57      0      0      0      0      0

             ...          BILL_AMT_JUN  BILL_AMT_MAY  BILL_AMT_APR  \
0            ...             -0.672497     -0.663059     -0.652724
1            ...             -0.621636     -0.606229     -0.597966
2            ...             -0.449730     -0.417188     -0.391630
3            ...             -0.232373     -0.186729     -0.156579
4            ...             -0.346997     -0.348137     -0.331482

   PAY_AMT_SEP  PAY_AMT_AUG  PAY_AMT_JUL  PAY_AMT_JUN  PAY_AMT_MAY  \
0    -0.341942    -0.227086    -0.296801    -0.308063    -0.314136
1    -0.341942    -0.213588    -0.240005    -0.244230    -0.314136
2    -0.250292    -0.191887    -0.240005    -0.244230    -0.248683
3    -0.221191    -0.169361    -0.228645    -0.237846    -0.244166
4    -0.221191     1.335034     0.271165     0.266434    -0.269039

   PAY_AMT_APR  default payment next month
0    -0.293382                           1
1    -0.180878                           1
2    -0.012122                           0
3    -0.237130                           0
4    -0.255187                           0

[5 rows x 25 columns]
```

Out[76]:

	ID	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_A
0	1	2	2	1	24	2	2	0	0	0	...	-
1	2	2	2	2	26	0	2	0	0	0	...	-

	ID	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_A
2	3	2	2	2	34	0	0	0	0	0	...	-
3	4	2	2	1	37	0	0	0	0	0	...	-
4	5	1	2	1	57	0	0	0	0	0	...	-

5 rows × 25 columns

Splitting the Dataset

We will split the dataset into 2/3 for training and 1/3 for testing

In [77]:
```
import random
random.seed(42)
x = df1.loc[df1["default payment next month"] == 1]["default payment next month"]
y = df1.loc[df1["default payment next month"] == 0]["default payment next month"]
X = df1.drop(labels = ["default payment next month"], axis = 1)
y = df1.iloc[:,24]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
train = pd.concat([X_train,y_train], axis = 1)
test = pd.concat([X_test,y_test], axis = 1)
test.head()
test.to_csv("card_test.csv")
```

Oversampling our Training Data after checking the class counts

We can see that the classes in the dataset is higly skewed such that Class 1 has more records than class 0. There is a need to preprocess imbalanced data before training a model, if not the model will always be biased to the class with more records.

Thus we will conduct 2 sampling methods to make the training set more balanced. Test set will not be oversampled.

1. SMOTE
2. Random Oversampling

1. Synthetic Minority Over-Sampling (SMOTE)

SMOTE creates new (artificial) training examples based on the original training examples. For instance, if it sees two examples (of the same class) near each other, it creates a third artificial one, in the middle of the original two. Instead of simply duplicating,entries SMOTE creates entries that are interpolations of the minority class,as well as undersamples the majority class.

In [78]:
```
X_resampled, y_resampled = SMOTE().fit_resample(X_train, y_train)
print(X_resampled)
print(y_resampled)
X_df= pd.DataFrame(data=X_resampled,columns = None,)
Y_df = pd.DataFrame(data=y_resampled, columns = None)
smote_df = pd.concat([X_df,Y_df], axis = 1)
smote_df.columns = list(df1)
print(smote_df.head())
smote_df.to_csv("smote_train.csv")
```

```
[[ 1.68320000e+04  1.00000000e+00  3.00000000e+00 ...  2.14729025e-01
  -1.30505935e-02 -2.32573727e-01]
 [ 4.22300000e+03  1.00000000e+00  1.00000000e+00 ... -2.14866504e-01
   2.30399887e+00  2.69137190e-01]
 [ 8.73700000e+03  2.00000000e+00  2.00000000e+00 ... -2.43080653e-01
  -2.35592068e-01 -2.53443191e-01]
 ...
 [ 2.76221490e+04  2.00000000e+00  2.00000000e+00 ... -2.62599581e-01
  -2.09905539e-01 -2.84746108e-01]
 [ 2.60406740e+04  2.00000000e+00  3.00000000e+00 ... -2.59394657e-01
  -3.14136117e-01 -2.93382058e-01]
 [ 5.39518228e+02  2.00000000e+00  2.15104178e+00 ... -3.08062562e-01
  -2.19961656e-01 -2.12446663e-01]]
[1 0 1 ... 1 1 1]
        ID  SEX  EDUCATION  MARRIAGE   AGE  PAY_1  PAY_2  PAY_3  PAY_4  PAY_5  \
0  16832.0  1.0        3.0       1.0  49.0    0.0    0.0    0.0    0.0    0.0
1   4223.0  1.0        1.0       2.0  38.0    2.0    0.0    0.0    0.0    0.0
2   8737.0  2.0        2.0       2.0  39.0    0.0    0.0    0.0    0.0    0.0
3  27881.0  2.0        3.0       1.0  26.0    0.0    0.0    2.0    2.0    2.0
4  29291.0  1.0        3.0       2.0  26.0    2.0    0.0    0.0    0.0    0.0

               ...              BILL_AMT_JUN  BILL_AMT_MAY  BILL_AMT_APR  \
0              ...                 -0.660373     -0.528346     -0.575482
1              ...                  0.219845      0.097368      0.577472
2              ...                 -0.084265     -0.155641     -0.181531
3              ...                  1.396931      1.428505      1.548151
4              ...                 -0.168951     -0.389061     -0.367164

   PAY_AMT_SEP  PAY_AMT_AUG  PAY_AMT_JUL  PAY_AMT_JUN  PAY_AMT_MAY  \
0    -0.109858    -0.157209    -0.252500     0.214729    -0.013051
1    -0.100440    -0.039980     0.157572    -0.214867     2.303999
2    -0.221191    -0.170186    -0.228645    -0.243081    -0.235592
3     0.322189    -0.144145     0.214369    -0.308063     0.078584
4    -0.218353    -0.008384    -0.222966    -0.231846    -0.273751

   PAY_AMT_APR  default payment next month
0    -0.232574                           1
1     0.269137                           0
2    -0.253443                           1
3    -0.012122                           0
4    -0.256818                           1

[5 rows x 25 columns]
```

2. Random Oversampling

Random oversampling just increases the size of the training data set through repetition of the original examples. It does not cause any increase in the variety of training examples.

In [79]:
```
#import package
import random
random.seed(42)
import imblearn
from imblearn.over_sampling import RandomOverSampler

#random oversampling
ros = RandomOverSampler(random_state=42)
X_resampled, Y_resampled = ros.fit_resample(X_train,y_train)

# using Counter to display results of naive oversampling
from collections import Counter
print(sorted(Counter(Y_resampled).items()))

df_X = pd.DataFrame(data=X_resampled, columns=None,)
df_Y = pd.DataFrame(data=Y_resampled, columns=None)

rsample_df = pd.concat([df_X,df_Y], axis = 1)
rsample_df.columns = list(df1)

rsample_df.to_csv('random_sampled_train.csv')
```

```
[(0, 15622), (1, 15622)]
```

Random Forest Classifier

We will apply the data to the Random Forest classifier. Random Forest classifier is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees for training data and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random decision forests correct for decision trees' habit of overfitting to their training set.

Packages

In [259]:

```
# Pandas is used for reading in csv files
import pandas as pd
# Use numpy to convert to arrays
import numpy as np
# pprint is for printing
from pprint import pprint
# For splitting train, test set
from sklearn.model_selection import train_test_split
# Import the random forest model
from sklearn.ensemble import RandomForestClassifier
# For PCA
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.decomposition import PCA
# for time
import time
# Importing module
# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
# Accuracy metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
# feature selection
from sklearn.feature_selection import SelectFromModel
```

Blind Testing

We will fit the raw data to the Random Forest classifier.

In [260]:
```
# Read in data and display first 5 rows of data
original_data = pd.read_csv('card.csv')
original_data.head(5)
```

Out[260]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
0	1	20000	2	2	1	24	2	2	-1	-1	...
1	2	120000	2	2	2	26	-1	2	0	0	...
2	3	90000	2	2	2	34	0	0	0	0	...
3	4	50000	2	2	1	37	0	0	0	0	...
4	5	50000	1	2	1	57	-1	0	-1	0	...

5 rows × 25 columns

In [261]:
```
# Labels are the values we want to predict
labels = np.array(original_data['default payment next month'])
labels
```

Out[261]:
```
array([1, 1, 0, ..., 1, 1, 1])
```

In [262]:
```
# Remove the labels, ID columns from the features
# axis 1 refers to the columns
features = original_data.drop('default payment next month', axis =
1)
features = features.drop('ID', axis = 1)
features
```

Out[262]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PA'
0	20000	2	2	1	24	2	2	-1	-1	
1	120000	2	2	2	26	-1	2	0	0	
2	90000	2	2	2	34	0	0	0	0	
3	50000	2	2	1	37	0	0	0	0	
4	50000	1	2	1	57	-1	0	-1	0	
...	...	...	...	...	...	...	...	...	...	
29995	220000	1	3	1	39	0	0	0	0	
29996	150000	1	3	2	43	-1	-1	-1	-1	
29997	30000	1	2	2	37	4	3	2	-1	
29998	80000	1	3	1	41	1	-1	0	0	
29999	50000	1	2	1	46	0	0	0	0	

30000 rows × 23 columns

In [263]:
```
# Saving feature names for later use
feature_list = list(features.columns)
feature_list
```

Out[263]:
```
['LIMIT_BAL',
 'SEX',
 'EDUCATION',
 'MARRIAGE',
 'AGE',
 'PAY_0',
 'PAY_2',
 'PAY_3',
 'PAY_4',
 'PAY_5',
 'PAY_6',
 'BILL_AMT1',
 'BILL_AMT2',
 'BILL_AMT3',
 'BILL_AMT4',
 'BILL_AMT5',
 'BILL_AMT6',
 'PAY_AMT1',
 'PAY_AMT2',
 'PAY_AMT3',
 'PAY_AMT4',
 'PAY_AMT5',
 'PAY_AMT6']
```

In [264]:
```
# split the df into train and test, it is important these two do no
t communicate during the training
original_training, original_testing, original_training_labels, orig
inal_testing_labels = train_test_split(features, labels, test_size=
1/3, random_state=42)
# this means we will train on 2/3 of the data and test on the remai
ning 1/3.
```

Establish Baseline

Before we can make and evaluate predictions, we need to establish a baseline, a sensible measure that we hope to beat with our model.

In [265]:
```
def most_frequent(List):
    counter = 0
    num = List[0]

    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency> counter):
            counter = curr_frequency
            num = i

    return (num, counter)

result = most_frequent(list(original_training_labels))
print(result)
print("baseline accuracy is", result[1]/ len(original_training_labe
ls)*100, "%")

# 0 class is majority class
# 15,546 instances in 18,000 rows
# baseline accuracy is 77.73 %
```

```
(0, 15546)
baseline accuracy is 77.73 %
```

Train the model

After getting the training and testing set, we will proceed to train and fit models with Scikit-learn.

We will instantiate the model, and fit (scikit-learn's name for training) the model on the training data.

We will set the random state at 42 for reproducible results.

In [314]:
```
# Instantiate model with 1000 decision trees
rf1 = RandomForestClassifier(n_estimators = 100, bootstrap = True,
random_state = 42)

# Train the model on training data
rf1.fit(original_training, original_training_labels);

# Test accuracy on test data
predictions = rf1.predict(original_testing)
print(accuracy_score(original_testing_labels, predictions, normaliz
e=True))
print(confusion_matrix(original_testing_labels, predictions))
print(f1_score(original_testing_labels, predictions))
print(classification_report(original_testing_labels, predictions))
```

```
0.8153
[[7356  462]
 [1385  797]]
0.4632374309793665
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      7818
           1       0.63      0.37      0.46      2182

    accuracy                           0.82     10000
   macro avg       0.74      0.65      0.68     10000
weighted avg       0.80      0.82      0.80     10000
```

Randomised Search

The most efficient way to find an optimal set of hyperparameters for a machine learning model is to use random search. The randomized search meta-estimator is an algorithm that trains and evaluates a series of models by taking random draws from a predetermined set of hyperparameter distributions. The algorithm picks the most successful version of the model it’s seen after training N different versions of the model with different randomly selected hyperparameter combinations, leaving you with a model trained on a near-optimal set of hyperparameters.

In [273]:
```python
# Number of trees in random forest
n_estimators = [80, 100, 120, 140, 160, 180, 200]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [5, 10, 20, 30]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap,
               'criterion' :['gini', 'entropy']
             }

from pprint import pprint
pprint(param_grid)
```

```
{'bootstrap': [True, False],
 'criterion': ['gini', 'entropy'],
 'max_depth': [5, 10, 20, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [80, 100, 120, 140, 160, 180, 200]}
```

In [278]:
```python
RFmodel = RandomForestClassifier()
rf_original_RS = RandomizedSearchCV(estimator = RFmodel, param_dist
ributions = param_grid, random_state=42)

print("Randomized search..")
search_time_start = time.time()
rf_original_RS.fit(original_training, original_training_labels)
print("Randomized search time:", time.time() - search_time_start)
```

```
Randomized search..

/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection
/_split.py:1978: FutureWarning: The default value of cv will chang
e from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)

Randomized search time: 169.6366798877716
```

In [279]:
```
print(rf_original_RS.best_score_)
pprint(rf_original_RS.best_params_)
```

```
0.81895
{'bootstrap': True,
 'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 2,
 'min_samples_split': 10,
 'n_estimators': 200}
```

In [280]:
```
rf_original_RS.best_params_
```

Out[280]:
```
{'n_estimators': 200,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 10,
 'criterion': 'entropy',
 'bootstrap': True}
```

In [313]:
```
rf2 = RandomForestClassifier(n_estimators = 200, bootstrap = True,
random_state = 42, criterion = 'entropy',
                             max_depth = None, min_samples_leaf = 2
, min_samples_split = 10,
                             max_features = 'sqrt')

# Train the model on training data
rf2.fit(original_training, original_training_labels);

# Test accuracy on test data
predictions = rf2.predict(original_testing)
print(accuracy_score(original_testing_labels, predictions, normaliz
e=True))
print(confusion_matrix(original_testing_labels, predictions))
print(f1_score(original_testing_labels, predictions))
print(classification_report(original_testing_labels, predictions))
```

```
0.8214
[[7406  412]
 [1374  808]]
0.4750146972369194
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7818
           1       0.66      0.37      0.48      2182

    accuracy                           0.82     10000
   macro avg       0.75      0.66      0.68     10000
weighted avg       0.80      0.82      0.80     10000
```

Training and fitting the model with balanced data

We now fit the synthetically sampled data to train our random forest classifier, and we expect the accuracy of our classifier to improve since our sampled data is more balanced.

In [283]:
```
# Reading in data
new_trainingR_data = pd.read_csv('random_sampled_train.csv')
new_trainingS_data = pd.read_csv('smote_train.csv')
new_test_data = pd.read_csv('card_test.csv')
```

In [284]:
```
# Labels are the values we want to predict
new_trainingR_labels = np.array(new_trainingR_data['default payment
next month'])
new_trainingR_labels
```

Out[284]: array([1, 0, 1, ..., 1, 1, 1])

In [285]:
```
# Labels are the values we want to predict
new_trainingS_labels = np.array(new_trainingS_data['default payment
next month'])
new_trainingS_labels
```

Out[285]: array([1, 0, 1, ..., 1, 1, 1])

In [286]:
```
# Remove the labels from the features
new_trainingR = new_trainingR_data.drop('default payment next month
', axis = 1)
new_trainingR = new_trainingR.drop('ID', axis = 1)
new_trainingR
```

Out[286]:

	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
0	1	3	1	49	0	0	0	0	0	0
1	1	1	2	38	2	0	0	0	0	0
2	2	2	2	39	0	0	0	0	0	0
3	2	3	1	26	0	0	2	2	2	0
4	1	3	2	26	2	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
31239	2	2	1	38	2	0	0	0	2	0
31240	1	3	1	45	0	0	2	0	0	0
31241	2	2	1	32	2	2	2	2	2	0
31242	2	1	2	34	1	0	0	0	0	0
31243	1	2	1	48	2	0	0	0	0	0

31244 rows × 23 columns

In [287]:
```
# Remove the labels from the features
new_trainingS = new_trainingS_data.drop('default payment next month
', axis = 1)
new_trainingS = new_trainingS.drop('ID', axis = 1)
new_trainingS
```

Out[287]:

	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4
0	1.0	3.000000	1.000000	49.000000	0.000000	0.000000	0.000000	0.000000
1	1.0	1.000000	2.000000	38.000000	2.000000	0.000000	0.000000	0.000000
2	2.0	2.000000	2.000000	39.000000	0.000000	0.000000	0.000000	0.000000
3	2.0	3.000000	1.000000	26.000000	0.000000	0.000000	2.000000	2.000000
4	1.0	3.000000	2.000000	26.000000	2.000000	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...	...
31239	1.0	1.475296	1.524704	32.049408	2.574112	2.524704	2.000000	2.000000
31240	2.0	2.286933	1.000000	50.356534	0.000000	0.000000	0.000000	0.000000
31241	2.0	2.000000	1.865684	22.134316	0.000000	0.000000	0.000000	0.000000
31242	2.0	3.000000	1.000000	42.000000	1.524853	1.524853	1.524853	1.524853
31243	2.0	2.151042	1.000000	46.179688	1.575521	0.000000	0.000000	1.726563

31244 rows × 23 columns

In [290]:
```
# Saving feature names for later use
new_feature_list = list(new_trainingS.columns)
new_feature_list
```

Out[290]:
```
['SEX',
 'EDUCATION',
 'MARRIAGE',
 'AGE',
 'PAY_1',
 'PAY_2',
 'PAY_3',
 'PAY_4',
 'PAY_5',
 'PAY_6',
 'LIMIT_BAL',
 'BILL_AMT_SEP',
 'BILL_AMT_AUG',
 'BILL_AMT_JUL',
 'BILL_AMT_JUN',
 'BILL_AMT_MAY',
 'BILL_AMT_APR',
 'PAY_AMT_SEP',
 'PAY_AMT_AUG',
 'PAY_AMT_JUL',
 'PAY_AMT_JUN',
 'PAY_AMT_MAY',
 'PAY_AMT_APR']
```

In [289]:
```
# Labels are the values we want to predict
new_testing_labels = np.array(new_test_data['default payment next m
onth'])
new_testing_labels
```

Out[289]: array([0, 0, 0, ..., 0, 0, 0])

In [292]:
```
new_testing = new_test_data.drop('default payment next month', axis
= 1)
new_testing = new_testing.drop('ID', axis = 1)
new_testing
```

Out[292]:

	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	.
0	1	2	2	25	0	0	0	0	0	0	.
1	2	1	2	26	0	0	0	0	0	0	.
2	2	3	1	32	0	0	0	0	0	0	.
3	1	3	2	49	0	0	0	0	0	0	.
4	2	2	2	36	0	0	0	0	0	2	.
...	...	...	...	...	...	...	...	...	...	...	.
9895	2	1	2	26	0	0	0	0	0	0	.
9896	1	2	1	32	0	0	0	0	0	2	.
9897	2	1	1	41	0	0	0	0	0	0	.
9898	1	2	2	30	0	0	0	0	0	0	.
9899	2	2	2	24	0	0	0	0	0	0	.

9900 rows × 23 columns

Train the model

SMOTE sampled data

In [315]:

```
# Instantiate model with 100 decision trees
rf_S1 = RandomForestClassifier(n_estimators = 100, bootstrap = True
, random_state = 42)
# Train the model on training data
rf_S1.fit(new_trainingS, new_trainingS_labels);

# Test accuracy on test data
predictions = rf_S1.predict(new_testing)
print(accuracy_score(new_testing_labels, predictions, normalize=Tru
e))
print(confusion_matrix(new_testing_labels, predictions))
print(f1_score(new_testing_labels, predictions))
print(classification_report(new_testing_labels, predictions))
```

```
0.8109090909090909
[[7146  596]
 [1276  882]]
0.4851485148514851
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      7742
           1       0.60      0.41      0.49      2158

    accuracy                           0.81      9900
   macro avg       0.72      0.67      0.68      9900
weighted avg       0.79      0.81      0.80      9900
```

In [295]:

```
# features selection
feature_importances_S = pd.DataFrame(rf_S1.feature_importances_,
                                  index = new_feature_list,
                                  columns = ['importance']).sort_va
lues('importance', ascending = False)
feature_importances_S
```

Out[295]:

	importance
PAY_1	0.154369
PAY_2	0.095312
EDUCATION	0.056633
PAY_3	0.056318
MARRIAGE	0.055590
BILL_AMT_SEP	0.043815
LIMIT_BAL	0.041654
AGE	0.039668
PAY_AMT_SEP	0.037128
BILL_AMT_AUG	0.035346
PAY_AMT_AUG	0.034061
PAY_4	0.032985
BILL_AMT_JUL	0.031897
PAY_AMT_JUL	0.031676
BILL_AMT_JUN	0.030767
BILL_AMT_MAY	0.030556
PAY_AMT_MAY	0.030310
PAY_6	0.030278
PAY_AMT_JUN	0.030256
BILL_AMT_APR	0.030218
PAY_AMT_APR	0.030082
PAY_5	0.026090
SEX	0.014990

Random Sampled data

In [316]:
```
# Instantiate model with 100 decision trees
rf_R1 = RandomForestClassifier(n_estimators = 100, bootstrap = True
, random_state = 42)
# Train the model on training data
rf_R1.fit(new_trainingR, new_trainingR_labels);

# Test accuracy on test data
predictions = rf_R1.predict(new_testing)
print(accuracy_score(new_testing_labels, predictions, normalize=Tru
e))
print(confusion_matrix(new_testing_labels, predictions))
print(f1_score(new_testing_labels, predictions))
print(classification_report(new_testing_labels, predictions))
```

```
0.8098989898989899
[[7086  656]
 [1226  932]]
0.49759743726641753
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      7742
           1       0.59      0.43      0.50      2158

    accuracy                           0.81      9900
   macro avg       0.72      0.67      0.69      9900
weighted avg       0.79      0.81      0.80      9900
```

In [297]:
```
# features selection
feature_importances_R = pd.DataFrame(rf_R1.feature_importances_,
                                 index = new_feature_list,
                                 columns = ['importance']).sort_va
lues('importance', ascending = False)
feature_importances_R
```

Out[297]:

	importance
PAY_1	0.093401
LIMIT_BAL	0.065572
BILL_AMT_SEP	0.064521
AGE	0.064043
BILL_AMT_AUG	0.056321
PAY_AMT_SEP	0.054431
BILL_AMT_JUL	0.053749
PAY_AMT_AUG	0.052077
BILL_AMT_MAY	0.051247
BILL_AMT_JUN	0.050924
PAY_AMT_JUL	0.050794
BILL_AMT_APR	0.050116
PAY_AMT_APR	0.049921
PAY_AMT_JUN	0.046870
PAY_AMT_MAY	0.046106
PAY_2	0.044130
PAY_3	0.021448
EDUCATION	0.021023
PAY_4	0.015112
MARRIAGE	0.012900
PAY_6	0.012291
SEX	0.011948
PAY_5	0.011054

Randomised Search

SMOTE sampled data

In [298]:
```
# Feature selection
new_trainingS_FS = new_trainingS[['PAY_1','PAY_2', 'EDUCATION', 'PA
Y_3', 'MARRIAGE',
                                  'BILL_AMT_SEP']]
new_trainingS_FS
```

Out[298]:

	PAY_1	PAY_2	EDUCATION	PAY_3	MARRIAGE	BILL_AMT_SEP
0	0.000000	0.000000	3.000000	0.000000	1.000000	0.926421
1	2.000000	0.000000	1.000000	0.000000	2.000000	0.251019
2	0.000000	0.000000	2.000000	0.000000	2.000000	-0.074888
3	0.000000	0.000000	3.000000	2.000000	1.000000	0.952075
4	2.000000	0.000000	3.000000	0.000000	2.000000	-0.021448
...	...	...	...	...	...	...
31239	2.574112	2.524704	1.475296	2.000000	1.524704	-0.495967
31240	0.000000	0.000000	2.286933	0.000000	1.000000	-0.648828
31241	0.000000	0.000000	2.000000	0.000000	1.865684	-0.388794
31242	1.524853	1.524853	3.000000	1.524853	1.000000	-0.608854
31243	1.575521	0.000000	2.151042	0.000000	1.000000	-0.275138

31244 rows × 6 columns

In [303]:
```
RFmodel = RandomForestClassifier()
rf_S2 = RandomizedSearchCV(estimator = RFmodel, param_distributions
= param_grid, random_state=42)

print("Randomized search..")
search_time_start = time.time()
rf_S2 .fit(new_trainingS_FS, new_trainingS_labels)
print("Randomized search time:", time.time() - search_time_start)
```

```
Randomized search..

/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection
/_split.py:1978: FutureWarning: The default value of cv will chang
e from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)

Randomized search time: 103.82698583602905
```

In [304]:
```
rf_S2.best_params_
```

Out[304]:
```
{'n_estimators': 180,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 10,
 'criterion': 'gini',
 'bootstrap': False}
```

In [301]:
```
rf_S2.best_score_
```
Out[301]:
```
0.8366406350019203
```

In [305]:
```
# Test data
new_testingS_FS = new_testing[['PAY_1','PAY_2', 'EDUCATION', 'PAY_3
', 'MARRIAGE',
                                                'BILL_AMT_SEP']]
new_testingS_FS
```
Out[305]:

	PAY_1	PAY_2	EDUCATION	PAY_3	MARRIAGE	BILL_AMT_SEP
0	0	0	2	0	2	-0.575264
1	0	0	1	0	2	1.161310
2	0	0	3	0	1	0.256655
3	0	0	3	0	2	-0.414823
4	0	0	2	0	2	0.584028
...	...	...	...	...	...	...
9895	0	0	1	0	2	-0.338840
9896	0	0	2	0	1	0.800909
9897	0	0	1	0	1	0.568641
9898	0	0	2	0	2	-0.288320
9899	0	0	2	0	2	0.175443

9900 rows × 6 columns

In [317]:
```
# Test accuracy on test data
predictions = rf_S2.predict(new_testingS_FS)
print(accuracy_score(new_testing_labels, predictions, normalize=Tru
e))
print(confusion_matrix(new_testing_labels, predictions))
print(f1_score(new_testing_labels, predictions))
print(classification_report(new_testing_labels, predictions))
```
```
0.8150505050505050
[[7201  541]
 [1290  868]]
0.4866834875245305
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      7742
           1       0.62      0.40      0.49      2158

    accuracy                           0.82      9900
   macro avg       0.73      0.67      0.69      9900
weighted avg       0.80      0.82      0.80      9900
```

Random Sampled data

In [310]:
```
# Feature selection
new_trainingR_FS = new_trainingR[['PAY_1','LIMIT_BAL', 'BILL_AMT_SE
P', 'AGE', 'BILL_AMT_AUG',
                                  'PAY_AMT_SEP', 'BILL_AMT_JUL',  '
PAY_AMT_AUG', 'BILL_AMT_MAY', 'BILL_AMT_JUN',
                                  'PAY_AMT_JUL', 'BILL_AMT_APR', 'P
AY_AMT_APR',  'PAY_AMT_JUN', 'PAY_AMT_MAY',
                                 'PAY_2']]
new_trainingR_FS
```

Out[310]:

	PAY_1	LIMIT_BAL	BILL_AMT_SEP	AGE	BILL_AMT_AUG	PAY_AMT_SEP	BILL_AM'
0	0	-0.365980	0.926421	49	-0.636974	-0.109858	-0.6
1	2	-1.059646	0.251019	38	0.319293	-0.100440	0.2
2	0	-0.597202	-0.074888	39	-0.058085	-0.221191	-0.0
3	0	-0.288907	0.952075	26	1.118576	0.322189	1.1
4	2	-0.905498	-0.021448	26	0.642156	-0.218353	-0.0
...	...	...	...	...	...	...	
31239	2	-0.674276	0.154800	38	0.076349	-0.148257	-0.0
31240	0	-0.674276	-0.585191	45	-0.536739	-0.160815	-0.5
31241	2	0.096463	-0.515210	32	-0.472529	-0.172890	-0.4
31242	1	0.250611	-0.724338	34	-0.720672	-0.341942	-0.7
31243	2	-0.288907	0.953161	48	1.038699	0.020312	1.1

31244 rows × 16 columns

In [311]:

```
# Feature selection
new_testingR_FS = new_testing[['PAY_1','LIMIT_BAL', 'BILL_AMT_SEP',
'AGE', 'BILL_AMT_AUG',
                                  'PAY_AMT_SEP', 'BILL_AMT_JUL',  '
PAY_AMT_AUG', 'BILL_AMT_MAY', 'BILL_AMT_JUN',
                                  'PAY_AMT_JUL', 'BILL_AMT_APR', 'P
AY_AMT_APR',  'PAY_AMT_JUN', 'PAY_AMT_MAY',
                                 'PAY_2']]
new_testingR_FS
```

Out[311]:

	PAY_1	LIMIT_BAL	BILL_AMT_SEP	AGE	BILL_AMT_AUG	PAY_AMT_SEP	BILL_AMT_
0	0	-1.059646	-0.575264	25	-0.549609	-0.251378	-0.51
1	0	-0.134759	1.161310	26	1.074458	-0.071097	1.00
2	0	-0.751350	0.256655	32	0.279615	-0.195169	0.31
3	0	-0.288907	-0.414823	49	-0.424645	-0.244737	-0.44
4	0	-0.905498	0.584028	36	-0.021695	-0.221191	-0.06
...	...	...	...	...	...	...	
9895	0	-0.520128	-0.338840	26	-0.692009	-0.341942	-0.67
9896	0	-0.443054	0.800909	32	0.822380	-0.109435	0.53
9897	0	0.019389	0.568641	41	0.644418	-0.130627	0.74
9898	0	-1.059646	-0.288320	30	-0.271018	-0.229885	-0.26
9899	0	0.019389	0.175443	24	0.072611	-0.160815	0.12

9900 rows × 16 columns

In [222]:

```
RFmodel = RandomForestClassifier()
rf_new_RS_R = RandomizedSearchCV(estimator = RFmodel, param_distrib
utions = param_grid, random_state=42)

print("Randomized search..")
search_time_start = time.time()
rf_new_RS_R.fit(new_trainingR, new_trainingR_labels)
print("Randomized search time:", time.time() - search_time_start)
```

```
Randomized search..

/opt/anaconda3/lib/python3.7/site-packages/sklearn/model_selection
/_split.py:1978: FutureWarning: The default value of cv will chang
e from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)

Randomized search time: 233.99393796920776
```

In [226]:

```
print(rf_new_RS_R.best_score_)
```

```
0.9433171168864422
```

In [227]:
```
rf_original_RS.best_params_
```

Out[227]:
```
{'n_estimators': 160,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 30,
 'criterion': 'gini',
 'bootstrap': True}
```

In [321]:
```
rf_R2 = RandomForestClassifier(n_estimators = 160, bootstrap = True
, random_state = 42, criterion = 'gini',
                              max_depth = 30, min_samples_leaf = 2,
min_samples_split = 2, max_features = 'sqrt')

# Train the model on training data
rf_R2.fit(new_trainingR_FS, new_trainingR_labels)

# Test accuracy on test data
predictions = rf_R2.predict(new_testingR_FS)
print(accuracy_score(new_testing_labels, predictions, normalize=Tru
e))
print(confusion_matrix(new_testing_labels, predictions))
print(f1_score(new_testing_labels, predictions, average = 'macro'))
print(classification_report(new_testing_labels, predictions))
```

```
0.8052525252525252
[[6979  763]
 [1165  993]]
0.693022288146426
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      7742
           1       0.57      0.46      0.51      2158

    accuracy                           0.81      9900
   macro avg       0.71      0.68      0.69      9900
weighted avg       0.79      0.81      0.80      9900
```

In [ ]:

Logistic Regression

Load data

In [2]:

```
import pandas as pd
import os
card = pd.read_csv(r"C:\Users\yingr\OneDrive\Desktop\BT2101\card.csv")
new_trainingS = pd.read_csv(r"C:\Users\yingr\Downloads\smote_train.csv")
new_trainingR = pd.read_csv(r"C:\Users\yingr\Downloads\random_sampled_train.csv")
new_test = pd.read_csv(r"C:\Users\yingr\Downloads\card_test.csv")
```

In [3]:

```
card.columns = card.iloc[0]
card = card[1:]
card = card.rename(columns={'default payment next month': 'def_pay','PAY_0': 'PAY_1'})
```

In [4]:

```
new_trainingS = new_trainingS.rename(columns={'default payment next month': 'def_pay'})
new_trainingR = new_trainingR.rename(columns={'default payment next month': 'def_pay'})
new_test = new_test.rename(columns={'default payment next month': 'def_pay'})
```

Blind Testing

In [5]:

```
## Blinding ##

from sklearn.model_selection import train_test_split
# X contains all features and y contains the target variable
X_train, X_test, y_train, y_test = train_test_split(card.drop(['ID','def_pay'],axis=1),
                                                     card['def_pay'], test_size=0.33, randon
## fit the model on the original training dataset
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train, y_train)

## make prediction on the X_test dataset
prediction = model.predict(X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(y_test,prediction))
print(confusion_matrix(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      7742
           1       0.00      0.00      0.00      2158

    accuracy                           0.78      9900
   macro avg       0.39      0.50      0.44      9900
weighted avg       0.61      0.78      0.69      9900

[[7742    0]
 [2158    0]]
```

Randomized Search

In [6]:

```
# define the parameters grid
from sklearn.model_selection import RandomizedSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }

rs_model = RandomizedSearchCV(model, param_grid, n_iter=20,n_jobs=1, verbose=0,
                               cv=5,scoring='accuracy', refit=True, random_state=42)

rs_model.fit(X_train, y_train)
best_score = rs_model.best_score_
best_params = rs_model.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.7771144278606965
Best params: 
C: 0.001
```

In [7]:

```
## change the parameters and fit the model on the original training dataset
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver = 'liblinear', C = 0.001)
model.fit(X_train, y_train)

## make prediction on the X_test dataset
prediction = model.predict(X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(y_test,prediction))
print(confusion_matrix(y_test,prediction))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      7742
           1       0.00      0.00      0.00      2158

    accuracy                           0.78      9900
   macro avg       0.39      0.50      0.44      9900
weighted avg       0.61      0.78      0.69      9900

[[7742    0]
 [2158    0]]
```

#

Training & fitting model with balanced data

SMOTE

In [8]:

```
## Cleaned (SMOTE) ##

from sklearn.linear_model import LogisticRegression
from sklearn import metrics
#create an instance and fit the model on the trainSmote dataset
y_trainSmote = new_trainingS['def_pay'].copy()
features = ['LIMIT_BAL','SEX','EDUCATION','MARRIAGE','AGE',
            'PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6',
            'BILL_AMT_SEP','BILL_AMT_AUG','BILL_AMT_JUL','BILL_AMT_JUN','BILL_AMT_MAY','BIL
            'PAY_AMT_SEP','PAY_AMT_AUG','PAY_AMT_JUL','PAY_AMT_JUN','PAY_AMT_MAY','PAY_AMT_
X_trainSmote = new_trainingS[features].copy()
logmodelS = LogisticRegression(solver = 'liblinear')
logmodelS.fit(X_trainSmote, y_trainSmote)

new_y_test = new_test['def_pay'].copy()
new_X_test = new_test[features].copy()
## make prediction on the card_test dataset
predictionS = logmodelS.predict(new_X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionS))
print(confusion_matrix(new_y_test,predictionS))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      7742
           1       0.49      0.57      0.53      2158

    accuracy                           0.78      9900
   macro avg       0.68      0.70      0.69      9900
weighted avg       0.79      0.78      0.78      9900

[[6471 1271]
 [ 932 1226]]
```

Feature Selection

In [9]:

```
## feature selection: drop variables that have p-value > 0.05
import statsmodels.api as sm
logit_model = sm.Logit(y_trainSmote,X_trainSmote)
result = logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.557962
         Iterations 6
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.195
Dependent Variable: def_pay          AIC:              34911.9592
Date:               2019-11-22 01:41 BIC:              35103.9996
No. Observations:   31244            Log-Likelihood:   -17433.
Df Model:           22               LL-Null:          -21657.
Df Residuals:       31221            LLR p-value:      0.0000
Converged:          1.0000           Scale:            1.0000
No. Iterations:     6.0000
-----------------------------------------------------------------
                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-----------------------------------------------------------------
LIMIT_BAL        -0.2088   0.0171 -12.2282 0.0000 -0.2423 -0.1754
SEX              -0.1613   0.0239  -6.7550 0.0000 -0.2081 -0.1145
EDUCATION        -0.0811   0.0192  -4.2199 0.0000 -0.1188 -0.0434
MARRIAGE         -0.2647   0.0210 -12.6117 0.0000 -0.3058 -0.2235
AGE               0.0003   0.0012   0.2170 0.8282 -0.0020  0.0025
PAY_1             1.0922   0.0261  41.8416 0.0000  1.0411  1.1434
PAY_2             0.0835   0.0260   3.2072 0.0013  0.0325  0.1346
PAY_3             0.2052   0.0264   7.7608 0.0000  0.1534  0.2571
PAY_4             0.1726   0.0297   5.8081 0.0000  0.1143  0.2308
PAY_5             0.0950   0.0330   2.8765 0.0040  0.0303  0.1598
PAY_6             0.2154   0.0277   7.7805 0.0000  0.1611  0.2696
BILL_AMT_SEP     -0.1786   0.0637  -2.8034 0.0051 -0.3034 -0.0537
BILL_AMT_AUG      0.1641   0.0835   1.9650 0.0494  0.0004  0.3277
BILL_AMT_JUL      0.1944   0.0739   2.6315 0.0085  0.0496  0.3391
BILL_AMT_JUN      0.0390   0.0682   0.5715 0.5677 -0.0948  0.1728
BILL_AMT_MAY     -0.1919   0.0754  -2.5464 0.0109 -0.3396 -0.0442
BILL_AMT_APR     -0.0236   0.0595  -0.3961 0.6920 -0.1403  0.0931
PAY_AMT_SEP      -0.1344   0.0245  -5.4752 0.0000 -0.1825 -0.0863
PAY_AMT_AUG      -0.1874   0.0349  -5.3708 0.0000 -0.2558 -0.1190
PAY_AMT_JUL      -0.0548   0.0257  -2.1332 0.0329 -0.1052 -0.0045
PAY_AMT_JUN      -0.0261   0.0217  -1.2015 0.2295 -0.0686  0.0165
PAY_AMT_MAY      -0.0395   0.0219  -1.8061 0.0709 -0.0824  0.0034
PAY_AMT_APR      -0.0460   0.0189  -2.4280 0.0152 -0.0831 -0.0089
=================================================================
```

In [10]:

```
## choose variables with p-value <0.05
## drop AGE, BILL_AMT_JUN, BILL_AMT_APR, PAY_AMT_JUN, PAY_AMT_MAY
selectedS = ['LIMIT_BAL','SEX','EDUCATION','MARRIAGE',
            'PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6',
            'BILL_AMT_SEP','BILL_AMT_AUG','BILL_AMT_JUL','BILL_AMT_MAY',
            'PAY_AMT_SEP','PAY_AMT_AUG','PAY_AMT_JUL','PAY_AMT_APR']
X_trainSmote = new_trainingS[selectedS].copy()
```

Randomized Search

In [11]:

```
## fit the model again after feature selection
logmodelS.fit(X_trainSmote, y_trainSmote)

# hyperparamter tuning; define the parameters grid
from sklearn.model_selection import RandomizedSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }

rs_modelS = RandomizedSearchCV(logmodelS, param_grid, n_iter=20,n_jobs=1, verbose=0,
                               cv=5,scoring='accuracy', refit=True, random_state=42)

rs_modelS.fit(X_trainSmote, y_trainSmote)
best_score = rs_modelS.best_score_
best_params = rs_modelS.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.7297721162463193
Best params:
C: 0.001
```

In [12]:

```
## change the parameters and fit the model on the original training dataset
from sklearn.linear_model import LogisticRegression
logmodelS = LogisticRegression(solver = 'liblinear', C = 0.001)
logmodelS.fit(X_trainSmote, y_trainSmote)

new_X_test = new_test[selectedS].copy()
## make prediction on the card_test dataset
predictionS = logmodelS.predict(new_X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionS))
print(confusion_matrix(new_y_test,predictionS))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      7742
           1       0.50      0.56      0.53      2158

    accuracy                           0.78      9900
   macro avg       0.68      0.70      0.69      9900
weighted avg       0.79      0.78      0.78      9900

[[6518 1224]
 [ 951 1207]]
```

Random Over-sampling

In [13]:

```
## Cleaned (RANDOM) ##

from sklearn.linear_model import LogisticRegression
from sklearn import metrics
#create an instance and fit the model on the trainRandom dataset
y_trainRandom = new_trainingR['def_pay'].copy()
X_trainRandom = new_trainingR[features].copy()
logmodelR = LogisticRegression(solver = 'liblinear')
logmodelR.fit(X_trainRandom, y_trainRandom)

new_y_test = new_test['def_pay'].copy()
new_X_test = new_test[features].copy()
## make prediction on the card_test dataset
predictionR = logmodelR.predict(new_X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionR))
print(confusion_matrix(new_y_test,predictionR))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      7742
           1       0.49      0.56      0.53      2158

    accuracy                           0.78      9900
   macro avg       0.68      0.70      0.69      9900
weighted avg       0.79      0.78      0.78      9900

[[6485 1257]
 [ 939 1219]]
```

Feature Selection

In [14]:

```
## feature selection: drop variables that have p-value > 0.05
import statsmodels.api as sm
logit_model = sm.Logit(y_trainRandom, X_trainRandom)
result = logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.578124
         Iterations 6
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.166
Dependent Variable: def_pay          AIC:              36171.8309
Date:               2019-11-22 01:41 BIC:              36363.8713
No. Observations:   31244            Log-Likelihood:   -18063.
Df Model:           22               LL-Null:          -21657.
Df Residuals:       31221            LLR p-value:      0.0000
Converged:          1.0000           Scale:            1.0000
No. Iterations:     6.0000
-----------------------------------------------------------------
                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-----------------------------------------------------------------
LIMIT_BAL        -0.2027   0.0158 -12.7965 0.0000 -0.2337 -0.1716
SEX              -0.1711   0.0225  -7.5950 0.0000 -0.2152 -0.1269
EDUCATION        -0.0564   0.0177  -3.1922 0.0014 -0.0911 -0.0218
MARRIAGE         -0.2085   0.0196 -10.6618 0.0000 -0.2468 -0.1702
AGE               0.0011   0.0011   0.9818 0.3262 -0.0011  0.0032
PAY_1             0.9058   0.0228  39.7179 0.0000  0.8611  0.9505
PAY_2             0.0751   0.0232   3.2434 0.0012  0.0297  0.1205
PAY_3             0.1415   0.0241   5.8652 0.0000  0.0942  0.1888
PAY_4             0.1418   0.0268   5.2948 0.0000  0.0893  0.1943
PAY_5             0.0493   0.0297   1.6587 0.0972 -0.0090  0.1076
PAY_6             0.1927   0.0250   7.7128 0.0000  0.1437  0.2417
BILL_AMT_SEP     -0.0891   0.0565  -1.5771 0.1148 -0.1999  0.0216
BILL_AMT_AUG      0.0344   0.0757   0.4547 0.6493 -0.1139  0.1828
BILL_AMT_JUL      0.2496   0.0670   3.7229 0.0002  0.1182  0.3809
BILL_AMT_JUN     -0.0066   0.0628  -0.1052 0.9162 -0.1298  0.1165
BILL_AMT_MAY     -0.1750   0.0690  -2.5378 0.0112 -0.3101 -0.0398
BILL_AMT_APR      0.0152   0.0550   0.2757 0.7828 -0.0927  0.1230
PAY_AMT_SEP      -0.1139   0.0224  -5.0748 0.0000 -0.1579 -0.0699
PAY_AMT_AUG      -0.2055   0.0328  -6.2648 0.0000 -0.2699 -0.1412
PAY_AMT_JUL      -0.0285   0.0216  -1.3176 0.1876 -0.0709  0.0139
PAY_AMT_JUN      -0.0136   0.0187  -0.7252 0.4684 -0.0503  0.0231
PAY_AMT_MAY      -0.0384   0.0197  -1.9459 0.0517 -0.0771  0.0003
PAY_AMT_APR      -0.0390   0.0173  -2.2610 0.0238 -0.0728 -0.0052
=================================================================
```

In [15]:

```
## choose variables with p-value < 0.05
## drop AGE, PYA_5, BILL_AMT_SEP, BILL_AMT_AUG, BILL_AMT_JUN, BILL_AMT_APR, PAY_AMT_JUL, PA
selectedR = ['LIMIT_BAL','SEX','EDUCATION','MARRIAGE',
            'PAY_1','PAY_2','PAY_3','PAY_4','PAY_6',
            'BILL_AMT_JUL','BILL_AMT_MAY',
            'PAY_AMT_SEP','PAY_AMT_AUG','PAY_AMT_APR']
X_trainRandom = new_trainingR[selectedR].copy()
```

Randomized Search

In [16]:

```
## fit the model again after feature selection
logmodelR.fit(X_trainRandom, y_trainRandom)

# hyperparamter tuning; define the parameters grid
from sklearn.model_selection import RandomizedSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }

rs_modelR = RandomizedSearchCV(logmodelR, param_grid, n_iter=20,n_jobs=1, verbose=0,
                               cv=5,scoring='accuracy', refit=True, random_state=42)

rs_modelR.fit(X_trainRandom, y_trainRandom)
best_score = rs_modelR.best_score_
best_params = rs_modelR.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.7017667392139291
Best params:
C: 0.001
```

In [17]:

```
## change the parameters and fit the model on the original training dataset
from sklearn.linear_model import LogisticRegression
logmodelR = LogisticRegression(solver = 'liblinear', C = 0.001)
logmodelR.fit(X_trainRandom, y_trainRandom)

new_X_test = new_test[selectedR].copy()
## make prediction on the card_test dataset
predictionR = logmodelR.predict(new_X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionR))
print(confusion_matrix(new_y_test,predictionR))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      7742
           1       0.49      0.56      0.53      2158

    accuracy                           0.78      9900
   macro avg       0.68      0.70      0.69      9900
weighted avg       0.79      0.78      0.78      9900

[[6506 1236]
 [ 949 1209]]
```

In [ ]:

SVM

In [114]:

```
# Load the data


card = pd.read_csv("card.csv")
trainingS = pd.read_csv("smote_train.csv")
trainingR = pd.read_csv("random_sampled_train.csv")
testingDF = pd.read_csv("card_test.csv")
```

As a first step, let's have a look if there are missing or anomalous data

In [115]:

```
card = card.rename(columns={'default payment next month': 'def_pay',
                        'PAY_0': 'PAY_1'})
card.head()
```

Out[115]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...	BI
0	1	20000	2	2	1	24	2	2	-1	-1	...	
1	2	120000	2	2	2	26	-1	2	0	0	...	
2	3	90000	2	2	2	34	0	0	0	0	...	
3	4	50000	2	2	1	37	0	0	0	0	...	
4	5	50000	1	2	1	57	-1	0	-1	0	...	

5 rows × 25 columns

In [116]:

```
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.model_selection import train_test_split
X = card.drop(['ID','def_pay'], axis = 1)
y= card['def_pay'].copy()
# split the df into train and test, it is important these two do not communicate
during the training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33333, ran
dom_state=42)
# this means we will train on 0.667% of the data and test on the remaining 20%.
```

Blind Testing

In [117]:

```
from sklearn.svm import SVC
#blind test without tuning
classifier = SVC(kernel='rbf', random_state = 1, gamma = "auto")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(y_test, y_pred))
print(pd.crosstab(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      7818
           1       0.50      0.01      0.03      2182

    accuracy                           0.78     10000
   macro avg       0.64      0.50      0.45     10000
weighted avg       0.72      0.78      0.69     10000

col_0       0   1
def_pay
0        7788  30
1        2152  30
```

Randomized Search

As the RandomizedSearchCV takes a long time to run, we will only do 2 iterations.

In [20]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
#DIRTY DATA TRAIN + TUNE
classifier = SVC(kernel='rbf', random_state = 1, gamma = "auto")
classifier.fit(X_train, y_train)

np.random.seed(123)
g_range = np.random.uniform(0.0, 0.3, 5).astype(float)
C_range = np.random.normal(1, 0.1, 5).astype(float)

hyperparameters = {'gamma': list(g_range),
                   'C': list(C_range)}
model = RandomizedSearchCV(SVC(kernel='rbf', ), param_distributions=hyperparamet
ers, n_iter=2)

model.fit(X_train, y_train)
best_score = model.best_score_
best_params = model.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
/Users/jcwu/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:1978: FutureWarning: The default value of cv will ch
ange from 3 to 5 in version 0.22. Specify it explicitly to silence t
his warning.
  warnings.warn(CV_WARNING, FutureWarning)


Best score: 0.7776
Best params:
C: 0.9948482279060615
gamma: 0.08584180048511383
```

In [63]:

```
rbfSVM = SVC(kernel='rbf', C=0.9948482279060615, gamma=0.08584180048511383)
rbfSVM.fit(X_train, y_train)
y_pred = rbfSVM.predict(X_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(y_test, y_pred))
print(pd.crosstab(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      7818
           1       0.52      0.01      0.02      2182

    accuracy                           0.78     10000
   macro avg       0.65      0.50      0.45     10000
weighted avg       0.73      0.78      0.69     10000

col_0       0   1
def_pay
0        7793  25
1        2155  27
```

Training & fitting model with balanced data

SMOTE

In [100]:

```
cleany_trainS = trainingS["default payment next month"]
trainingS.drop(["default payment next month"], axis = 1, inplace = True)
cleanx_trainS = trainingS
cleany_test = testingDF["default payment next month"]
testingDF.drop(["default payment next month"], axis = 1, inplace = True)
cleanx_test = testingDF
```

In [103]:

```
classifierS = SVC(kernel='rbf', random_state = 1, gamma = "auto")
classifierS.fit(cleanx_trainS, cleany_trainS)
y_predS = classifierS.predict(cleanx_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(cleany_test, y_predS))
print(pd.crosstab(cleany_test, y_predS))
```

```
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      7742
           1       0.25      0.19      0.21      2158

    accuracy                           0.70      9900
   macro avg       0.52      0.51      0.51      9900
weighted avg       0.67      0.70      0.68      9900

col_0                         0     1
default payment next month
0                          6470  1272
1                          1745   413
```

In [104]:

```
# define the parameters grid
# Designate distributions to sample hyperparameters from
np.random.seed(123)
g_range = np.random.uniform(0.0, 0.3, 5).astype(float)
C_range = np.random.normal(1, 0.1, 5).astype(float)

hyperparameters = {'gamma': list(g_range),
                   'C': list(C_range)}
modelS = RandomizedSearchCV(SVC(kernel='rbf', ), param_distributions=hyperparame
ters, n_iter=2)

modelS.fit(cleanx_trainS, cleany_trainS)
best_score = modelS.best_score_
best_params = modelS.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
/Users/jcwu/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:1978: FutureWarning: The default value of cv will ch
ange from 3 to 5 in version 0.22. Specify it explicitly to silence t
his warning.
  warnings.warn(CV_WARNING, FutureWarning)

Best score: 0.8141723210856484
Best params:
C: 0.9948482279060615
gamma: 0.08584180048511383
```

In [107]:

```
# Identify optimal hyperparameter values

rbfSVMS = SVC(kernel='rbf', C=0.8141723210856484, gamma= 0.08584180048511383)
rbfSVMS.fit(cleanx_trainS, cleany_trainS)
y_predS = rbfSVMS.predict(cleanx_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(cleany_test, y_predS))
print(pd.crosstab(cleany_test, y_predS))
```

```
              precision    recall  f1-score   support

           0       0.78      0.92      0.85      7742
           1       0.24      0.09      0.13      2158

    accuracy                           0.74      9900
   macro avg       0.51      0.51      0.49      9900
weighted avg       0.67      0.74      0.69      9900

col_0                          0    1
default payment next month
0                           7135  607
1                           1965  193
```

Random Over-sampling

In [90]:

```
cleany_train = trainingR["default payment next month"]
trainingR.drop(["default payment next month"], axis = 1, inplace = True)
cleanx_train = trainingR
cleany_test = testingDF["default payment next month"]
testingDF.drop(["default payment next month"], axis = 1, inplace = True)
cleanx_test = testingDF
```

In [95]:

```
classifierR = SVC(kernel='rbf', random_state = 1, gamma = "auto")
classifierR.fit(cleanx_train, cleany_train)
y_predR = classifierR.predict(cleanx_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(cleany_test, y_predR))
print(pd.crosstab(cleany_test, y_predR))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.84      7742
           1       0.26      0.11      0.15      2158

    accuracy                           0.74      9900
   macro avg       0.52      0.51      0.50      9900
weighted avg       0.67      0.74      0.69      9900

col_0                          0    1
default payment next month
0                           7073  669
1                           1926  232
```

In [34]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
```

In [35]:

```
# define the parameters grid
# Designate distributions to sample hyperparameters from
np.random.seed(123)
g_range = np.random.uniform(0.0, 0.3, 5).astype(float)
C_range = np.random.normal(1, 0.1, 5).astype(float)

hyperparameters = {'gamma': list(g_range),
                   'C': list(C_range)}
model = RandomizedSearchCV(SVC(kernel='rbf', ), param_distributions=hyperparamet
ers, n_iter=2)

model.fit(cleanx_train, cleany_train)
best_score = model.best_score_
best_params = model.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
/Users/jcwu/anaconda3/lib/python3.7/site-packages/sklearn/model_sele
ction/_split.py:1978: FutureWarning: The default value of cv will ch
ange from 3 to 5 in version 0.22. Specify it explicitly to silence t
his warning.
  warnings.warn(CV_WARNING, FutureWarning)

Best score: 0.9792280117782615
Best params:
C: 0.9795799035361106
gamma: 0.2089407556793585
```

In [94]:

```
# Identify optimal hyperparameter values

rbfSVM = SVC(kernel='rbf', C=0.9795799035361106, gamma= 0.2089407556793585)
rbfSVM.fit(cleanx_train, cleany_train)
y_pred = rbfSVM.predict(cleanx_test)


from sklearn.metrics import classification_report
#check accuracy of model for predictions
print(classification_report(cleany_test, y_pred))
print(pd.crosstab(cleany_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.99      0.88      7742
           1       0.22      0.01      0.01      2158

    accuracy                           0.78      9900
   macro avg       0.50      0.50      0.44      9900
weighted avg       0.66      0.78      0.69      9900

col_0                        0   1
default payment next month
0                         7702  40
1                         2147  11
```

XGBoost

In [6]:

```
# Import basic libraries
import numpy as np
import pandas as pd

# Import visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from ggplot import *

from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import average_precision_score

from xgboost.sklearn import XGBClassifier
from xgboost import plot_importance, to_graphviz
```

Load data

In [7]:

```
card = pd.read_csv("card.csv")
new_trainingS = pd.read_csv("smote_train.csv")
new_trainingR = pd.read_csv("random_sampled_train.csv")
new_test = pd.read_csv("card_test.csv")
```

In [8]:

```
card.columns = card.iloc[0]
card = card[1:]
card = card.rename(columns={'default payment next month': 'def_pay','PAY_0': 'PAY_1'})
card.head(5)
```

Out[8]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	...
0	1	20000	2	2	1	24	2	2	-1	-1	...
1	2	120000	2	2	2	26	-1	2	0	0	...
2	3	90000	2	2	2	34	0	0	0	0	...
3	4	50000	2	2	1	37	0	0	0	0	...
4	5	50000	1	2	1	57	-1	0	-1	0	...

5 rows × 25 columns

In [9]:

```
new_trainingS = new_trainingS.rename(columns={'default payment next month': 'def_pay'})
new_trainingR = new_trainingR.rename(columns={'default payment next month': 'def_pay'})
new_test = new_test.rename(columns={'default payment next month': 'def_pay'})

y = card['def_pay'].copy()
X = card.drop(['ID','def_pay'],axis=1)
```

Blind Testing

In [10]:

```
## Blinding ##

from sklearn.model_selection import train_test_split
# X contains all features and y contains the target variable
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y, test_size=0.33, random_state=42)

## fit the model on the original training dataset
clf = XGBClassifier()
clf.fit(X_train, y_train)

## make prediction on the X_test dataset
predictions = clf.predict(X_test)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
#check accuracy of model for predictions
print(classification_report(y_test,predictions))
print(pd.crosstab(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7742
           1       0.67      0.36      0.47      2158

    accuracy                           0.82      9900
   macro avg       0.76      0.66      0.68      9900
weighted avg       0.81      0.82      0.80      9900

col_0       0    1
def_pay
0        7366  376
1        1380  778
```

Randomized Search

In [44]:

```python
# define the parameters grid
from sklearn.model_selection import RandomizedSearchCV
param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
        'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [1.0, 5.0, 10.0, 50.0],
        'n_estimators': [100]}

clf_model = RandomizedSearchCV(clf, param_grid, n_iter=20,
                            n_jobs=1, verbose=0, cv=5,
                            scoring='accuracy', refit=True, random_state=42)

clf_model.fit(X_train, y_train)
best_score = clf_model.best_score_
best_params = clf_model.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.8208457711442786
Best params:
gamma: 1.0
learning_rate: 0.01
max_depth: 6
min_child_weight: 5.0
n_estimators: 100
reg_lambda: 10.0
silent: False
subsample: 0.7
```

In [11]:

```
## change the parameters and fit the model on the original training dataset
clf_optimized = XGBClassifier(gamma = 1.0, learning_rate = 0.01, max_depth = 6, min_chi
ld_weight = 5.0, n_estimators = 100, reg_lambda = 10.0, silent = False, subsample = 0.7
)

clf_optimized.fit(X_train, y_train)

## make prediction on the X_test dataset
predictions = clf_optimized.predict(X_test)
#check accuracy of model for predictions
print(classification_report(y_test,predictions))
print(pd.crosstab(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7742
           1       0.67      0.36      0.47      2158

    accuracy                           0.82      9900
   macro avg       0.76      0.66      0.68      9900
weighted avg       0.81      0.82      0.80      9900

col_0       0    1
def_pay
0        7367  375
1        1384  774
```

Training & fitting model with balanced data

SMOTE

In [12]:

```
## Cleaned (SMOTE) ##

from sklearn import metrics
#create an instance and fit the model on the trainSmote dataset
y_trainSmote = new_trainingS['def_pay'].copy()
features = ['LIMIT_BAL','SEX','EDUCATION','MARRIAGE','AGE',
            'PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6',
            'BILL_AMT_SEP','BILL_AMT_AUG','BILL_AMT_JUL','BILL_AMT_JUN','BILL_AMT_MAY',
'BILL_AMT_APR',
            'PAY_AMT_SEP','PAY_AMT_AUG','PAY_AMT_JUL','PAY_AMT_JUN','PAY_AMT_MAY','PAY_
AMT_APR']
X_trainSmote = new_trainingS[features].copy()
clfS = XGBClassifier()
clfS.fit(X_trainSmote, y_trainSmote)

new_y_test = new_test['def_pay'].copy()
new_X_test = new_test[features].copy()
## make prediction on the card_test dataset
predictionS = clfS.predict(new_X_test)
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionS))
print(pd.crosstab(new_y_test,predictionS))
```

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      7742
           1       0.64      0.39      0.48      2158

    accuracy                           0.82      9900
   macro avg       0.74      0.66      0.69      9900
weighted avg       0.80      0.82      0.80      9900

col_0       0    1
def_pay
0        7262  480
1        1317  841
```

Feature Selection

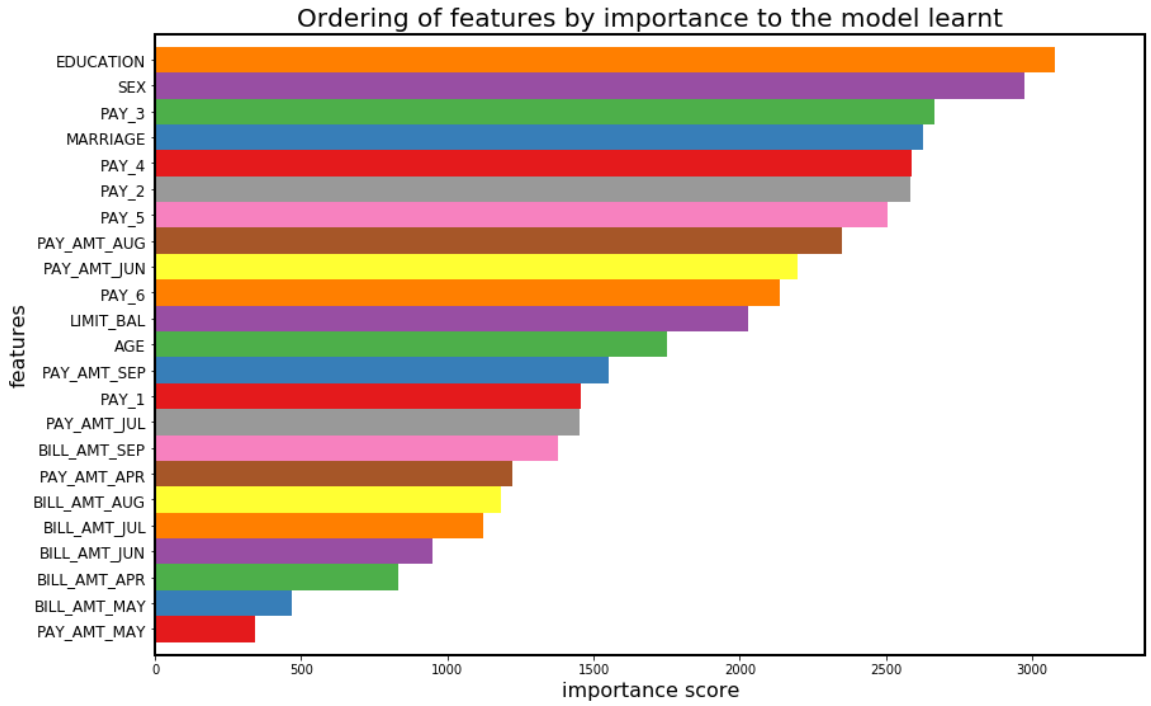
In [47]:

```
## feature selection
fig = plt.figure(figsize = (14, 9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0, 1, 9))

ax = plot_importance(clfS, height = 1, color = colours, grid = False, \
                     show_values = False, importance_type = 'cover', ax = ax);
for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size = 16);
ax.set_ylabel('features', size = 16);
ax.set_yticklabels(ax.get_yticklabels(), size = 12);
ax.set_title('Ordering of features by importance to the model learnt', size = 20);
```



In [13]:

```
## select attributes with importance score > 1500
selectedS = ['LIMIT_BAL','SEX','EDUCATION','MARRIAGE', 'AGE',
             'PAY_2','PAY_3','PAY_4','PAY_5','PAY_6',
             'PAY_AMT_SEP','PAY_AMT_AUG', 'PAY_AMT_JUN']
X_trainSmote = new_trainingS[selectedS].copy()
```

Randomized Search

In [50]:

```
## fit the model again after feature selection
clfS.fit(X_trainSmote, y_trainSmote)

# hyperparamter tuning; define the parameters grid
param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
        'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [1.0, 5.0, 10.0, 50.0],
        'n_estimators': [100]}

clf_modelS = RandomizedSearchCV(clfS, param_grid, n_iter=20,
                            n_jobs=1, verbose=0, cv=5,
                            scoring='accuracy', refit=True, random_state=42)

clf_modelS.fit(X_trainSmote, y_trainSmote)
best_score = clf_modelS.best_score_
best_params = clf_modelS.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.8380809115350147
Best params:
gamma: 0.25
learning_rate: 0.3
max_depth: 6
min_child_weight: 1.0
n_estimators: 100
reg_lambda: 50.0
silent: False
subsample: 1.0
```

In [14]:

```
## change the parameters and fit the model on the original training dataset
clfS_optimized = XGBClassifier(gamma = 0.25, learning_rate = 0.3, max_depth = 6, min_ch
ild_weight = 1.0, n_estimators = 100, reg_lambda = 50.0, silent = False, subsample = 1.
0)

clfS_optimized.fit(X_trainSmote, y_trainSmote)

new_X_test = new_test[selectedS].copy()
## make prediction on the X_test dataset
predictionS = clfS_optimized.predict(new_X_test)
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionS))
print(pd.crosstab(new_y_test,predictionS))
```

```
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      7742
           1       0.59      0.29      0.39      2158

    accuracy                           0.80      9900
   macro avg       0.71      0.62      0.63      9900
weighted avg       0.77      0.80      0.77      9900

col_0       0    1
def_pay
0        7303  439
1        1537  621
```

Random Over-sampling

In [15]:

```
## Cleaned (RANDOM) ##

#create an instance and fit the model on the trainRandom dataset
y_trainRandom = new_trainingR['def_pay'].copy()
X_trainRandom = new_trainingR[features].copy()

clfR = XGBClassifier()
clfR.fit(X_trainRandom, y_trainRandom)

new_y_test = new_test['def_pay'].copy()
new_X_test = new_test[features].copy()
## make prediction on the card_test dataset
predictionR = clfR.predict(new_X_test)
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionR))
print(pd.crosstab(new_y_test,predictionR))
```

```
              precision    recall  f1-score   support

           0       0.88      0.79      0.83      7742
           1       0.45      0.63      0.53      2158

    accuracy                           0.75      9900
   macro avg       0.67      0.71      0.68      9900
weighted avg       0.79      0.75      0.77      9900

col_0       0     1
def_pay
0        6118  1624
1         807  1351
```

Feature Selection

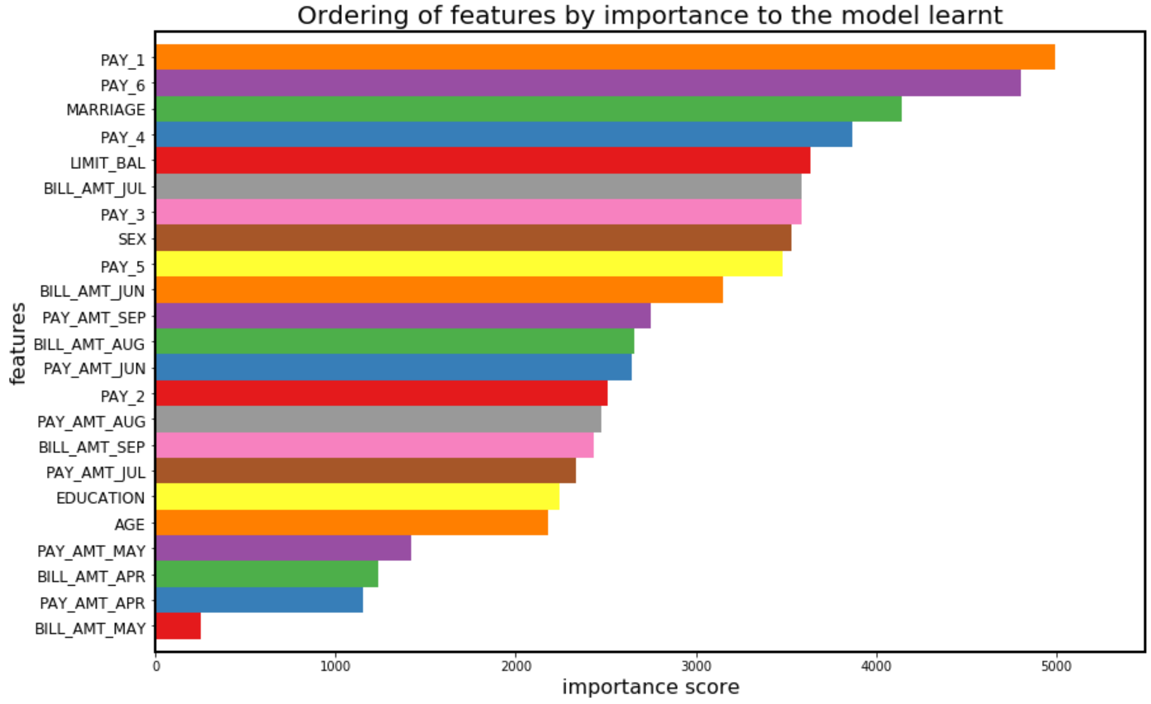
In [53]:

```
## feature selection
fig = plt.figure(figsize = (14, 9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0, 1, 9))

ax = plot_importance(clfR, height = 1, color = colours, grid = False, \
                     show_values = False, importance_type = 'cover', ax = ax);
for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size = 16);
ax.set_ylabel('features', size = 16);
ax.set_yticklabels(ax.get_yticklabels(), size = 12);
ax.set_title('Ordering of features by importance to the model learnt', size = 20);
```



In [16]:

```
## select attributes with importance score > 3000
selectedR = ['LIMIT_BAL','SEX','MARRIAGE',
             'PAY_1','PAY_3','PAY_4','PAY_5','PAY_6',
             'BILL_AMT_JUL','BILL_AMT_JUN']
X_trainRandom = new_trainingR[selectedR].copy()
```

Randomized Search

In [18]:

```
## fit the model again after feature selection
clfR.fit(X_trainRandom, y_trainRandom)
from sklearn.model_selection import RandomizedSearchCV
# hyperparamter tuning; define the parameters grid
param_grid = {
        'silent': [False],
        'max_depth': [6, 10, 15, 20],
        'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3],
        'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
        'min_child_weight': [0.5, 1.0, 3.0, 5.0],
        'gamma': [0, 0.25, 0.5, 1.0],
        'reg_lambda': [1.0, 5.0, 10.0, 50.0],
        'n_estimators': [100]}

clf_modelR = RandomizedSearchCV(clfR, param_grid, n_iter=20,
                            n_jobs=1, verbose=0, cv=5,
                            scoring='accuracy', refit=True, random_state=42)

clf_modelR.fit(X_trainRandom, y_trainRandom)
best_score = clf_modelR.best_score_
best_params = clf_modelR.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))
```

```
Best score: 0.8733516835232364
Best params:
gamma: 0.5
learning_rate: 0.2
max_depth: 20
min_child_weight: 1.0
n_estimators: 100
reg_lambda: 5.0
silent: False
subsample: 0.7
```

In [19]:

```
## change the parameters and fit the model on the original training dataset
clfR_optimized = XGBClassifier(gamma = 0.5, learning_rate = 0.2, max_depth = 20, min_ch
ild_weight = 1.0, n_estimators = 100, reg_lambda = 5.0, silent = False, subsample = 0.7
)

clfR_optimized.fit(X_trainRandom, y_trainRandom)

new_X_test = new_test[selectedR].copy()
## make prediction on the X_test dataset
predictionR = clfR_optimized.predict(new_X_test)
#check accuracy of model for predictions
print(classification_report(new_y_test,predictionR))
print(pd.crosstab(new_y_test,predictionR))
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85      7742
           1       0.45      0.47      0.46      2158

    accuracy                           0.76      9900
   macro avg       0.65      0.66      0.66      9900
weighted avg       0.76      0.76      0.76      9900

col_0       0     1
def_pay
0        6508  1234
1        1133  1025
```

Room for Improvement

Feature selection with Principal Component Analysis (PCA)

We will conduct PCA analysis on our raw data to do feature selection.

In [3]:
```
# Pandas is used for reading in csv files
import pandas as pd
# Use numpy to convert to arrays
import numpy as np
# For PCA
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.decomposition import PCA
```

In [4]:
```
# Read in data and display first 5 rows of data
original_data = pd.read_csv('card.csv')
original_data.head(5)
```

Out[4]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
0	1	20000	2	2	1	24	2	2	-1	-1	...
1	2	120000	2	2	2	26	-1	2	0	0	...
2	3	90000	2	2	2	34	0	0	0	0	...
3	4	50000	2	2	1	37	0	0	0	0	...
4	5	50000	1	2	1	57	-1	0	-1	0	...

5 rows × 25 columns

In [5]:
```
original_data_features = original_data.drop('default payment next m
onth', axis = 1)
original_data_features = original_data_features.drop('ID', axis = 1
)
original_data_features
```

Out[5]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PA'
0	20000	2	2	1	24	2	2	-1	-1	
1	120000	2	2	2	26	-1	2	0	0	
2	90000	2	2	2	34	0	0	0	0	
3	50000	2	2	1	37	0	0	0	0	
4	50000	1	2	1	57	-1	0	-1	0	
...	...	...	...	...	...	...	...	...	...	
29995	220000	1	3	1	39	0	0	0	0	
29996	150000	1	3	2	43	-1	-1	-1	-1	
29997	30000	1	2	2	37	4	3	2	-1	
29998	80000	1	3	1	41	1	-1	0	0	
29999	50000	1	2	1	46	0	0	0	0	

30000 rows × 23 columns

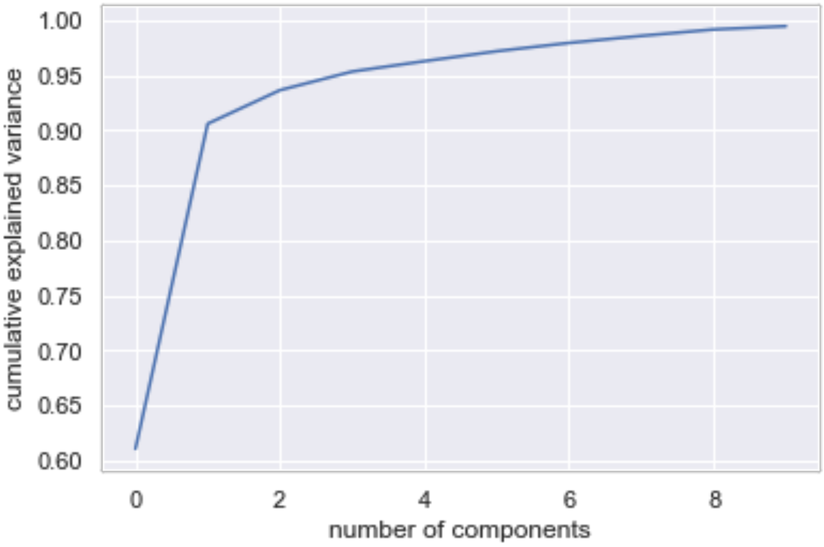
In [6]:
```
pca = PCA(n_components=10)
pca.fit(original_data_features)
```

Out[6]:
```
PCA(copy=True, iterated_power='auto', n_components=10, random_stat
e=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [7]:
```
print(pca.explained_variance_ratio_)
```

```
[0.61043701 0.29535381 0.03052419 0.01692859 0.00942042 0.00904175
 0.00754446 0.00638481 0.00583709 0.00296671]
```

In [8]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



In [9]:
```
pca = PCA(0.90).fit(original_data_features)
pca.n_components_

# The top 2 components account for 90% of the variance in our data
```

Out[9]: 2

In [10]:
```
print(pca.components_)

# Col 2,3,4,5,6,7,8,9,10,11 have very small coefficients
```

```
[[ 4.91590659e-01 -3.52873014e-08 -3.67290605e-07 -1.92469255e-07
   5.56879962e-06  3.42455214e-07  5.68458344e-07  5.81779941e-07
   6.64584131e-07  7.59373260e-07  8.36871693e-07  3.88453549e-01
   3.81356126e-01  3.72179448e-01  3.46397504e-01  3.22920046e-01
   3.08577267e-01  2.65676097e-02  3.12865310e-02  2.68185282e-02
   2.21681253e-02  2.22044122e-02  2.48098976e-02]
 [ 8.69022684e-01  1.76100266e-07 -1.49972284e-06 -4.01516316e-07
   8.31332276e-06 -3.80340392e-06 -4.62245984e-06 -4.49819779e-06
  -4.25959745e-06 -4.03177492e-06 -3.98438769e-06 -2.21364316e-01
  -2.26375798e-01 -2.16534865e-01 -1.94048190e-01 -1.76775713e-01
  -1.67365250e-01  5.71625946e-03  1.07848222e-02  1.09685628e-02
   1.03644900e-02  1.16931055e-02  1.53341329e-02]]
```

In [ ]: