This project is to identify the attributes/characteristics that have high tendency of carring out fraudulent credit card transactions based on the dataset collected.

The hypothesis is that some of the demographic factors play an important clues to classify card users' attributes into different risk level of credit card default groupings. This can be done by feature importance.

Type I error is to predict a case is default but the actual is not. Type II error is to predict a case is not default but the actual is indeed a default case. This error is more costly than Type I error.

The model to be built by classifier aims to reduce the two types of errors. An effective model should be able to predict default and non default equally accurate.

As the data set is imbalanced with 78% of non default and 22% of default cases. The model created by the classifier is to be higher than the base accuracy, 78%.

Nevertheless, different python libraries will be used to test out the accuracy of the training model. So that, the feature importance generated by higher accuracy model will have more data credibility.

In this case study, Classifiers e.g. Logistic regression, Random Forest Classifier and Decision Tree Classifier, as well as KNeighbor Classifier are applied. The results show that Random Forest Classifer is the best classifier, followed by Decision Tree Classifier and then Logistric Regression because the feature columns are highly categorical in nature.

Dataset Information: This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005 The target column is the 'default.payment.next.month'

There are 25 variables: ID: ID of each client

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

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

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

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

AGE: Age in years

PAY_0: 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)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import csv
import scipy.stats as stats
import seaborn as sns
import pandas as pd
import json
from math import log
from sklearn.datasets import load_boston

%matplotlib inline
```

In [2]:

```
Credit= pd.read_csv('/Users/kaiengwee/Documents/GitHub/GA18Aug/Project4/UCI_Cr
edit_Card.csv')
Credit.head()
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_
0	1	20000.0	2	2	1	24	2	2	-1	-1
1	2	120000.0	2	2	2	26	-1	2	0	0
2	3	90000.0	2	2	2	34	0	0	0	0
3	4	50000.0	2	2	1	37	0	0	0	0
4	5	50000.0	1	2	1	57	-1	0	-1	0

5 rows × 25 columns

In [3]:

```
Credit.rename(columns={'default.payment.next.month':'default_nextMTH'},inplace
= True)
```

In [4]:

```
Credit.describe()
```

Out[4]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867
std	8660.398374	129747.661567	0.489129	0.790349	0.521970
min	1.000000	10000.000000	1.000000	0.000000	0.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000

8 rows × 25 columns

In [5]:

```
Credit['default_nextMTH'].describe()
```

Out[5]:

```
count    30000.000000
mean         0.221200
std          0.415062
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: default_nextMTH, dtype: float64
```

In [6]:

```
Credit.shape
```

Out[6]:

```
(30000, 25)
```

In [7]:

```
Credit.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 float64
SEX                  30000 non-null int64
EDUCATION            30000 non-null int64
MARRIAGE             30000 non-null int64
AGE                  30000 non-null int64
PAY_0                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_AMT1            30000 non-null float64
BILL_AMT2            30000 non-null float64
BILL_AMT3            30000 non-null float64
BILL_AMT4            30000 non-null float64
BILL_AMT5            30000 non-null float64
BILL_AMT6            30000 non-null float64
PAY_AMT1             30000 non-null float64
PAY_AMT2             30000 non-null float64
PAY_AMT3             30000 non-null float64
PAY_AMT4             30000 non-null float64
PAY_AMT5             30000 non-null float64
PAY_AMT6             30000 non-null float64
default_nextMTH      30000 non-null int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [8]:

```
Credit.isnull().sum()
```

Out[8]:

```
ID                0
LIMIT_BAL         0
SEX               0
EDUCATION         0
MARRIAGE          0
AGE               0
PAY_0             0
PAY_2             0
PAY_3             0
PAY_4             0
PAY_5             0
PAY_6             0
BILL_AMT1         0
BILL_AMT2         0
BILL_AMT3         0
BILL_AMT4         0
BILL_AMT5         0
BILL_AMT6         0
PAY_AMT1          0
PAY_AMT2          0
PAY_AMT3          0
PAY_AMT4          0
PAY_AMT5          0
PAY_AMT6          0
default_nextMTH   0
dtype: int64
```

In [9]:

```
Credit.isna().sum()
```

Out[9]:

```
ID                 0
LIMIT_BAL          0
SEX                0
EDUCATION          0
MARRIAGE           0
AGE                0
PAY_0              0
PAY_2              0
PAY_3              0
PAY_4              0
PAY_5              0
PAY_6              0
BILL_AMT1          0
BILL_AMT2          0
BILL_AMT3          0
BILL_AMT4          0
BILL_AMT5          0
BILL_AMT6          0
PAY_AMT1           0
PAY_AMT2           0
PAY_AMT3           0
PAY_AMT4           0
PAY_AMT5           0
PAY_AMT6           0
default_nextMTH    0
dtype: int64
```

In [10]:

```
Credit.nunique()
```

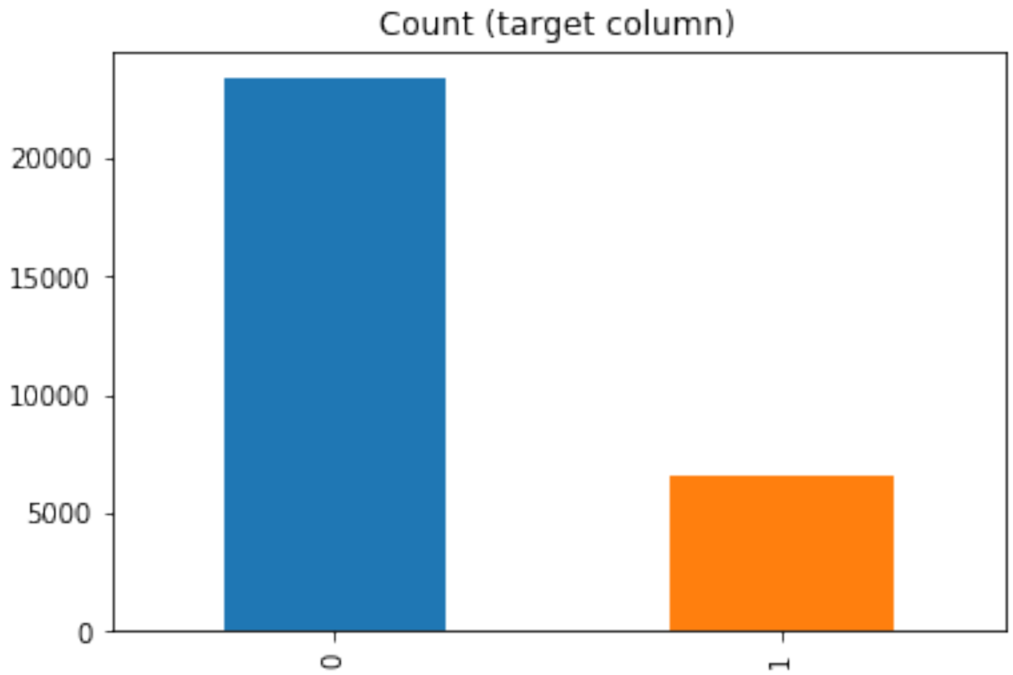
Out[10]:

```
ID                  30000
LIMIT_BAL              81
SEX                     2
EDUCATION               7
MARRIAGE                4
AGE                    56
PAY_0                  11
PAY_2                  11
PAY_3                  11
PAY_4                  11
PAY_5                  10
PAY_6                  10
BILL_AMT1           22723
BILL_AMT2           22346
BILL_AMT3           22026
BILL_AMT4           21548
BILL_AMT5           21010
BILL_AMT6           20604
PAY_AMT1             7943
PAY_AMT2             7899
PAY_AMT3             7518
PAY_AMT4             6937
PAY_AMT5             6897
PAY_AMT6             6939
default_nextMTH         2
dtype: int64
```

In [11]:

```
target_count = Credit['default_nextMTH'].value_counts()
print('No default in next month:', target_count[0])
print('Default in next month:', target_count[1])
print('Proportion of default cases in the dataset:', round(target_count[1] / t
arget_count[0], 2), ': 1')

target_count.plot(kind='bar', title='Count (target column)');
```

```
No default in next month: 23364
Default in next month: 6636
Proportion of default cases in the dataset: 0.28 : 1
```

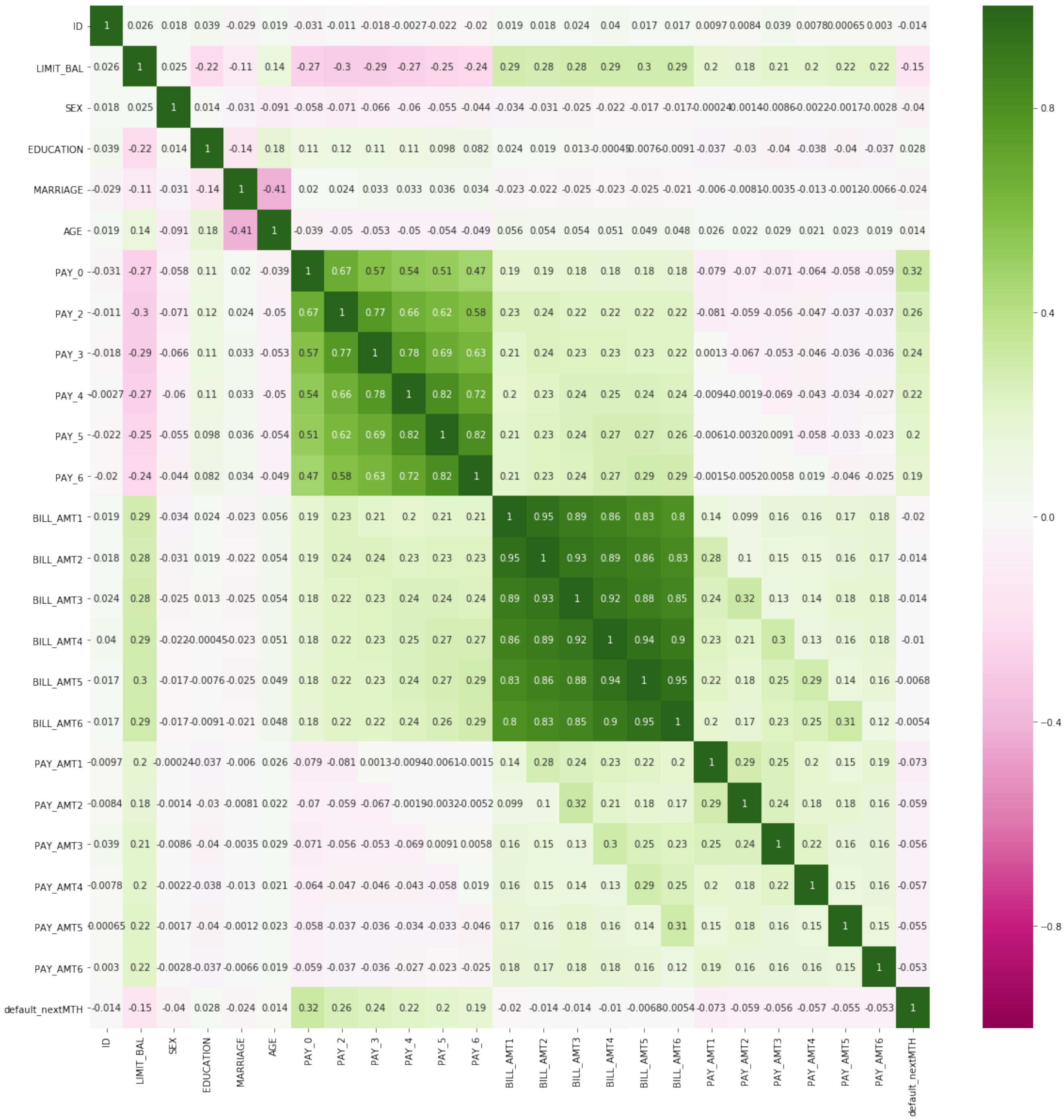


In [12]:

```
plt.subplots(figsize=(20,20))
sns.heatmap(Credit.corr(),  annot=True, vmin=-1, vmax= 1, cmap='PiYG', center=
0)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1088e8f60>
```



From the heatmap above, BILL_AMT1 to BILL_AMT6 are highly correlated within themselves. Since BILL_AMT1 is the latest billing month before default occured, I will drop out BILL_AMT2 to BILL_AMT6. In addition, ID and AGE features will also be removed since it doesn't mean anything in predicting the default occurence and lower correlated to the target.

In [13]:

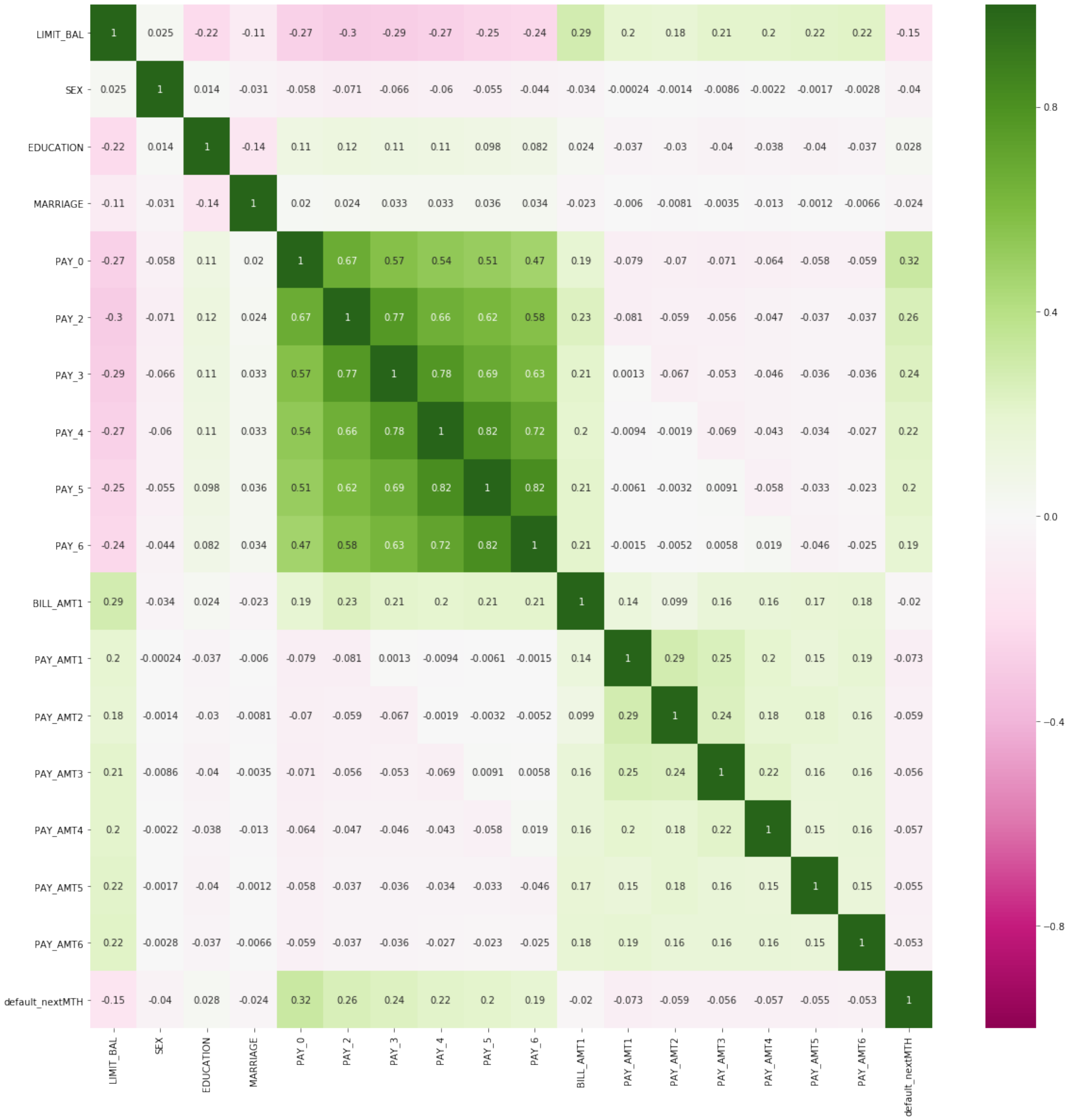
```
Credit.drop(['ID','AGE','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6'], axis = 1, inplace = True)
```

In [14]:

```
plt.subplots(figsize=(20,20))
sns.heatmap(Credit.corr(),  annot=True, vmin=-1, vmax= 1, cmap='PiYG', center=
0)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10924bc88>
```



In [15]:

```
count_d = 0
count_nd = 0

for i in Credit['default_nextMTH']:
    if i == 1:
        count_d += 1
    else:
        count_nd += 1

total= count_d + count_nd
print("Total cases under study", total)
print("Total default cases", count_d,"or ", round(count_d/total,2)*100,"%" )
print("Total non-default cases", count_nd,"or ", round(count_nd/total,2)*100,"
%" )
```

```
Total cases under study 30000
Total default cases 6636 or  22.0 %
Total non-default cases 23364 or  78.0 %
```

In [16]:

```
# The remaining features have 18 columns
Credit.shape
```

Out[16]:

```
(30000, 18)
```

In [17]:

```
Credit.head()
```

Out[17]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5
0	20000.0	2	2	1	2	2	-1	-1	-2
1	120000.0	2	2	2	-1	2	0	0	0
2	90000.0	2	2	2	0	0	0	0	0
3	50000.0	2	2	1	0	0	0	0	0
4	50000.0	1	2	1	-1	0	-1	0	0

In [18]:

```
# SEX: Gender (1=male, 2=female), the data set in SEX column only has either m
ale or female
Credit['SEX'].value_counts().sort_index()
```

Out[18]:

```
1    11888
2    18112
Name: SEX, dtype: int64
```

In [19]:

```
SEX_dummies= pd.get_dummies(Credit.SEX, prefix= 'SEX')
SEX_dummies.sample(n=5, random_state=1)
```

Out[19]:

	SEX_1	SEX_2
10747	1	0
12573	0	1
29676	1	0
8856	0	1
21098	1	0

In [20]:

```
# Drop SEX_2 since one column is enough to represent the two genders

SEX_dummies.drop(SEX_dummies.columns[-1], axis=1, inplace= True)
SEX_dummies.head()
```

Out[20]:

	SEX_1
0	0
1	0
2	0
3	0
4	1

In [21]:

```
# EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unkn
own, 6=unknown)
# value counts shows there is redundant categories under EDUCATION column i.e.
categories 0, and 6
Credit['EDUCATION'].value_counts().sort_index()
```

Out[21]:

```
0       14
1    10585
2    14030
3     4917
4      123
5      280
6       51
Name: EDUCATION, dtype: int64
```

In [22]:

```
# to replace the category 0 with NA

Credit.EDUCATION.replace(0, np.nan, inplace=True)
Credit.isna().sum()
```

Out[22]:

```
LIMIT_BAL          0
SEX                0
EDUCATION         14
MARRIAGE           0
PAY_0              0
PAY_2              0
PAY_3              0
PAY_4              0
PAY_5              0
PAY_6              0
BILL_AMT1          0
PAY_AMT1           0
PAY_AMT2           0
PAY_AMT3           0
PAY_AMT4           0
PAY_AMT5           0
PAY_AMT6           0
default_nextMTH    0
dtype: int64
```

In [23]:

```
# Check the column MARRIAGE after replacing the extra category 0 with NA

Credit['EDUCATION'].value_counts().sort_index()
```

Out[23]:

```
1.0    10585
2.0    14030
3.0     4917
4.0      123
5.0      280
6.0       51
Name: EDUCATION, dtype: int64
```

In [24]:

```
# Create EDUCATION DUMMY COLUMNS

EDUCATION_dummies= pd.get_dummies(Credit.EDUCATION, prefix= 'EDUCATION')
EDUCATION_dummies.sample(n=5, random_state=1)
```

Out[24]:

	EDUCATION_1.0	EDUCATION_2.0	EDUCATION_3.0	EDUCATION_4.0	EDUC
10747	0	0	1	0	0
12573	0	0	1	0	0
29676	0	1	0	0	0
8856	0	0	1	0	0
21098	1	0	0	0	0

In [25]:

```
# Drop EDUCATION_0 & EDUCATION_6, columns with unknown
# Drop EDUCATION_5 since EDUCATION 1, 2, 3, and 4 columns are enough to repres
ent all categories of EDUCATION

EDUCATION_dummies.drop(['EDUCATION_5.0','EDUCATION_6.0'], axis=1, inplace= Tru
e)
EDUCATION_dummies.head()
```

Out[25]:

	EDUCATION_1.0	EDUCATION_2.0	EDUCATION_3.0	EDUCATION_4.0
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0

In [26]:

```
# MARRIAGE: Marital status (1=married, 2=single, 3=others)
# value counts shows there is redundant categories under MARRIAGE column i.e.
categories 0

Credit['MARRIAGE'].value_counts().sort_index()
```

Out[26]:

```
0       54
1    13659
2    15964
3      323
Name: MARRIAGE, dtype: int64
```

In [27]:

```
# to replace the category 0 with NA

Credit.MARRIAGE.replace(0, np.nan, inplace=True)
Credit.isna().sum()
```

Out[27]:

```
LIMIT_BAL           0
SEX                 0
EDUCATION          14
MARRIAGE           54
PAY_0               0
PAY_2               0
PAY_3               0
PAY_4               0
PAY_5               0
PAY_6               0
BILL_AMT1           0
PAY_AMT1            0
PAY_AMT2            0
PAY_AMT3            0
PAY_AMT4            0
PAY_AMT5            0
PAY_AMT6            0
default_nextMTH     0
dtype: int64
```

In [28]:

```
# Check the column MARRIAGE after replacing the extra category 0 with NA

Credit['MARRIAGE'].value_counts().sort_index()
```

Out[28]:

```
1.0    13659
2.0    15964
3.0      323
Name: MARRIAGE, dtype: int64
```

In [29]:

```
MARRIAGE_dummies= pd.get_dummies(Credit.MARRIAGE, prefix= 'MARRIAGE')
MARRIAGE_dummies.sample(n=5, random_state=1)
```

Out[29]:

	MARRIAGE_1.0	MARRIAGE_2.0	MARRIAGE_3.0
10747	1	0	0
12573	1	0	0
29676	1	0	0
8856	1	0	0
21098	0	1	0

In [30]:

```
# Drop MARRIAGE_3 since MARRIAGE 1 and 2 are enough to represent all categorie
s of EDUCATION

MARRIAGE_dummies.drop(MARRIAGE_dummies.columns[-1], axis=1, inplace= True)
MARRIAGE_dummies.head()
```

Out[30]:

	MARRIAGE_1.0	MARRIAGE_2.0
0	1	0
1	0	1
2	0	1
3	1	0
4	1	0

In [31]:

```
# Concatenate the data set, Credit with three sets of dummies created namely S
EX, EDUCATION and MARRIAGE

Credit_dummies = pd.concat([Credit, SEX_dummies, EDUCATION_dummies, MARRIAGE_d
ummies], axis=1)

Credit_dummies.shape
```

Out[31]:

```
(30000, 25)
```

In [32]:

```
# Remove the rows with NA intentionally created for extra categories found in
columns, EDUCATION AND MARRIAGAE

Credit_dummies.dropna(inplace=True)
```

In [33]:

```
# Check if the dataset Credit_dummies dummies are clear from cell with NA

Credit_dummies.isna().sum()
```

Out[33]:

```
LIMIT_BAL          0
SEX                0
EDUCATION          0
MARRIAGE           0
PAY_0              0
PAY_2              0
PAY_3              0
PAY_4              0
PAY_5              0
PAY_6              0
BILL_AMT1          0
PAY_AMT1           0
PAY_AMT2           0
PAY_AMT3           0
PAY_AMT4           0
PAY_AMT5           0
PAY_AMT6           0
default_nextMTH    0
SEX_1              0
EDUCATION_1.0      0
EDUCATION_2.0      0
EDUCATION_3.0      0
EDUCATION_4.0      0
MARRIAGE_1.0       0
MARRIAGE_2.0       0
dtype: int64
```

In [34]:

```
# the latest shape of data set Credit_dummies has lower number of rows at 2993
2

Credit_dummies.shape
```

Out[34]:

```
(29932, 25)
```

In [35]:

```
# Remove the original columns of SEX, EDUCATION and MARRIAGE

Credit_dummies.drop(['SEX','EDUCATION','MARRIAGE'], axis = 1, inplace = True)
```

In [36]:

```
# Shift the target column ('default_nextMTH') to the last column

y= Credit_dummies.default_nextMTH
Credit_dummies.drop(['default_nextMTH'], axis=1, inplace= True)
Credit_dummies= pd.concat([Credit_dummies, y], axis=1)
```

In [37]:

```
# Show the columns in data set named as Credit_dummies

Credit_dummies.columns
```

Out[37]:

```
Index(['LIMIT_BAL', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', '
PAY_6',
       'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4'
, 'PAY_AMT5',
       'PAY_AMT6', 'SEX_1', 'EDUCATION_1.0', 'EDUCATION_2.0', 'EDU
CATION_3.0',
       'EDUCATION_4.0', 'MARRIAGE_1.0', 'MARRIAGE_2.0', 'default_n
extMTH'],
      dtype='object')
```

In [38]:

```
# Check the shape of Credit_dummies after removing the three original columns
of SEX, EDUCATION AND MARRIAGE

Credit_dummies.shape
```

Out[38]:

```
(29932, 22)
```

In [39]:

```
# Create feature matrix (X)
feature_cols= Credit_dummies.columns.drop(['default_nextMTH'])
X_orig= Credit_dummies[feature_cols]

# Create response vector (y)

y_orig= Credit_dummies.default_nextMTH
```

In [40]:

```
print((type(X_orig)))
print((type(X_orig.values)))
print((type(y_orig)))
print((type(y_orig.values)))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'numpy.ndarray'>
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
```

In [41]:

```
print((X_orig.shape))
print((y_orig.shape))
```

```
(29932, 21)
(29932,)
```

In [42]:

```
X_orig.to_excel("X_orig.xlsx")
```

In [43]:

```
# check if Credit_dummies contain any NA cell

Credit_dummies.isna().sum()
```

Out[43]:

```
LIMIT_BAL          0
PAY_0              0
PAY_2              0
PAY_3              0
PAY_4              0
PAY_5              0
PAY_6              0
BILL_AMT1          0
PAY_AMT1           0
PAY_AMT2           0
PAY_AMT3           0
PAY_AMT4           0
PAY_AMT5           0
PAY_AMT6           0
SEX_1              0
EDUCATION_1.0      0
EDUCATION_2.0      0
EDUCATION_3.0      0
EDUCATION_4.0      0
MARRIAGE_1.0       0
MARRIAGE_2.0       0
default_nextMTH    0
dtype: int64
```

In [45]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_pred = scaler.fit_transform(X_orig)
```

Split data set for Train and Test sets using Logistic Regression and check their accuracies

In [46]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_orig, y_orig, random_sta
te=123)

model = logreg.fit(X_train,y_train)

y_pred = logreg.predict(X_test)

print('Testing score: '  + str(logreg.score(X_test, y_test)))
print('Training score: '  + str(logreg.score(X_train, y_train)))
```

```
Testing score: 0.7815047440865963
Training score: 0.7774511114080805
```

The Training and Testing scores are almost equal to the mix (78%) of none default cases in the data set.

In [47]:

```
print(logreg.intercept_)
print(logreg.coef_)
coeff = pd.DataFrame(dict(zip(X_orig.columns,model.coef_[0])),index=[0])
coeff
```

```
[-3.96367037e-08]
[[-5.47746144e-06  8.93562845e-08  7.19795673e-08  6.68494763e-08
   6.38936164e-08  6.22160956e-08  5.99718706e-08  1.06980608e-06
  -2.48111388e-05 -1.97145876e-05 -1.14323384e-05 -7.87454872e-06
  -3.44798717e-06 -4.05374625e-06 -1.16204268e-08 -8.42674575e-09
  -2.08277600e-08 -8.18250041e-09 -4.57448283e-10 -8.02342842e-09
  -3.09205441e-08]]
```

Out[47]:

	LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_
0	-0.000005	8.935628e-08	7.197957e-08	6.684948e-08	6.389362e-08	6.221610e-08	5.997187 08

1 rows × 21 columns

The coefficient figures do not show any particular feature having significant importance, in other words, the features are having similar but very little weights on the target column.

This is typically so when the features and target column do not have linear relationship yet created by logistic regression that assumes features and target are linearly related.

In [48]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [49]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[49]:

	Predicted: No	Predicted: Yes
Actual: No	5848	0
Actual: Yes	1635	0

In [50]:

```
print(classification_report(y_test, y_pred))
```

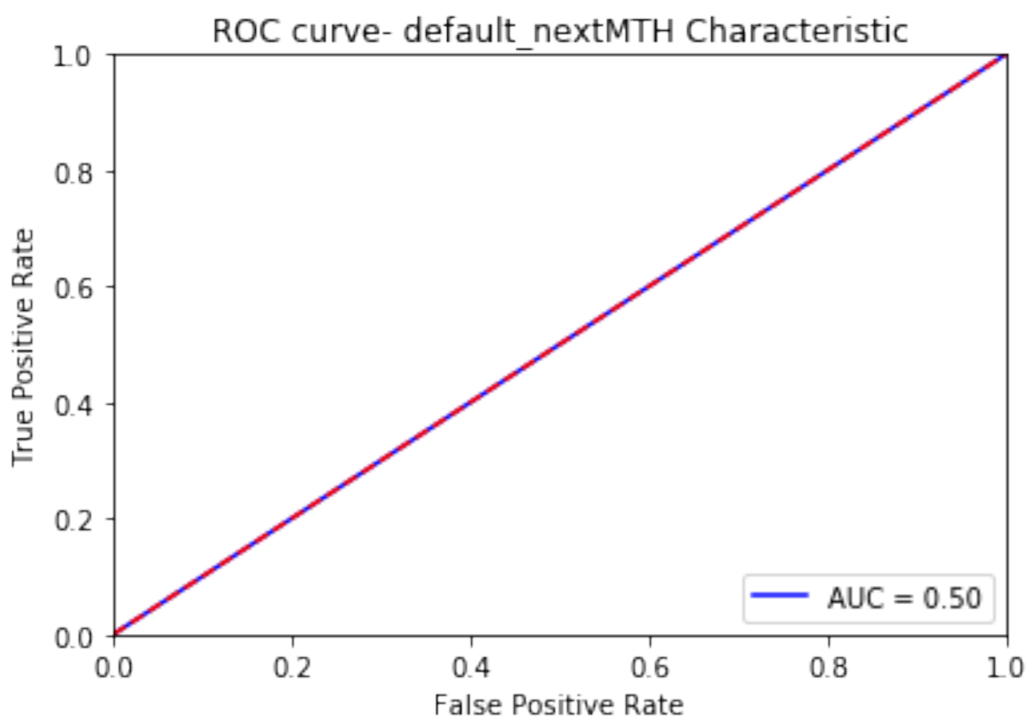
```
             precision    recall  f1-score   support

          0       0.78      1.00      0.88      5848
          1       0.00      0.00      0.00      1635

avg / total       0.61      0.78      0.69      7483


/Users/kaiengwee/anaconda3/lib/python3.6/site-packages/sklearn/met
rics/classification.py:1135: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples.
  'precision', 'predicted', average, warn_for)
```

The classification report simply shows the model prediction is only as good as predicting non-default cases at base accuracy of 78%.

We need to use Smote Tomek to improve from data imbalanced.

In [51]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('ROC curve- default_nextMTH Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Though the logistic regression classifier score is 78%, the AUC score is 0.50.

That means the classifier is not able to predict test data but only repeat the outcome in the actual data set that has 78% of data imbalanced.

Using another sklearn linear model to train and test the model

In [52]:

```
import pandas as pd
from sklearn import linear_model, model_selection, metrics

X_train, X_test, y_train, y_test = model_selection.train_test_split(X_orig, y_
orig, test_size= 0.25, random_state=46)
logit_simple = linear_model.LogisticRegression(C=1e9).fit(X_train, y_train)
```

In [53]:

```
1- y_train.mean()
```

Out[53]:

```
0.7784311105171723
```

In [54]:

```
# What is our accuracy on the test set?
print(np.mean(y_test == logit_simple.predict(X_test)))
```

```
0.7785647467593211
```

In [55]:

```
# Get probability predictions.
logit_pred_proba = logit_simple.predict_proba(X_test)[:,1]
```

In [56]:

```
metrics.confusion_matrix(y_true=y_test, y_pred=logit_pred_proba > .5)
```

Out[56]:

```
array([[5826,    0],
       [1657,    0]])
```

This sklearn linear model is built on logistic regression and hence, is showing the similar accuracy at about 78%

Use Decision Tree Classifier to build the model

In [57]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(random_state=123, max_depth=7)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_orig, y_orig, random_sta
te=123)

model= DT.fit(X_train, y_train)

y_pred= DT.predict(X_test)

print('Testing score: ', DT.score(X_test, y_test))
print('Training score: ', DT.score(X_train, y_train))
```

```
Testing score:  0.8276092476279567
Training score:  0.8263174306205177
```

In [58]:

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.84      0.96      0.90      5848
          1       0.71      0.36      0.48      1635

avg / total       0.81      0.83      0.81      7483
```

In [59]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [60]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[60]:

	Predicted: No	Predicted: Yes
Actual: No	5603	245
Actual: Yes	1045	590

This Decision Tree Classifier has a testing score of about 83%, however, the model performs poorly at 36% in identifying correctly over all default cases.

In addition, the more costly Type II error is 22% which means the model is unable to predict test data more accurately after training.

Apply Random Forest Classifier to get model accuracy

In [61]:

```
from sklearn.ensemble import RandomForestClassifier

# Train model
Random = RandomForestClassifier(n_estimators=10, max_depth= 10)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_orig, y_orig, random_sta
te=123)

model= Random.fit(X_train,y_train)

# Predict on training set
y_pred = Random.predict(X_test)

print('Testing score: ' + str(Random.score(X_test, y_test)))
print('Training score: '  + str(Random.score(X_train, y_train)))
list(zip(feature_cols, Random.feature_importances_))
```

```
Testing score: 0.8211947080048109
Training score: 0.8513964987304558
```

Out[61]:

```
[('LIMIT_BAL', 0.05274868450364332),
 ('PAY_0', 0.16707572606513138),
 ('PAY_2', 0.18851680445426136),
 ('PAY_3', 0.0552777755467845),
 ('PAY_4', 0.0689902141166802 7),
 ('PAY_5', 0.02708124771328950 3),
 ('PAY_6', 0.05729778452311564),
 ('BILL_AMT1', 0.06768000527905003),
 ('PAY_AMT1', 0.06014689104415414),
 ('PAY_AMT2', 0.04664700589747588),
 ('PAY_AMT3', 0.04630501666019361),
 ('PAY_AMT4', 0.039314317096305 5),
 ('PAY_AMT5', 0.04106071835143417 6),
 ('PAY_AMT6', 0.04381989192473266 5),
 ('SEX_1', 0.007507074059407611 5),
 ('EDUCATION_1.0', 0.006519112025743627),
 ('EDUCATION_2.0', 0.006445727792535586),
 ('EDUCATION_3.0', 0.005637775183700084),
 ('EDUCATION_4.0', 0.0003571555468891788 6),
 ('MARRIAGE_1.0', 0.00679262627373183),
 ('MARRIAGE_2.0', 0.004778443607070217)]
```

The feature importance check shows

The most importance feature is PAY_0 which makes sense as this is the most recent bill payment that reveals the payment capability.

The least important feature is EDUCATION category 4 which also makes sense as this is a minority category under EDUCATION column.

Random forest can work better with categorical features, than logistic regression.

Check the test data set accuracy using roc auc curve after modeled by Random forest

In [62]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [63]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[63]:

	Predicted: No	Predicted: Yes
Actual: No	5591	257
Actual: Yes	1081	554

In [64]:

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.84      0.96      0.89      5848
          1       0.68      0.34      0.45      1635

avg / total       0.80      0.82      0.80      7483
```

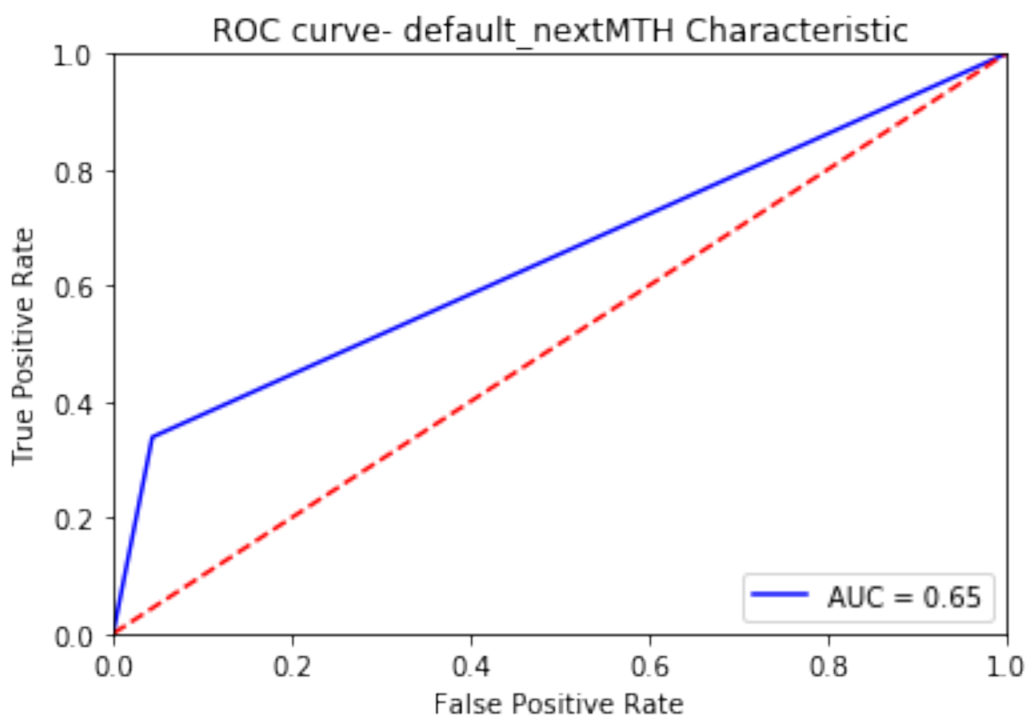
The Precision is good but the recall is very low for the minority Class 1 (default class) because the data set is imbalanced

Although the test set accuracy is 83% but only 37% defaulted cases are predicted correctly.

That also means this model is better in predicting non defaulted cases and not equally good in predicting default cases.

In [65]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('ROC curve- default_nextMTH Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Though the Random Forest classifier score is 83%, the AUC score is 0.66.

That means this model is better in predicting non defaulted cases but not equally good in predicting default cases because the data set has very high imbalanced data set at 78%.

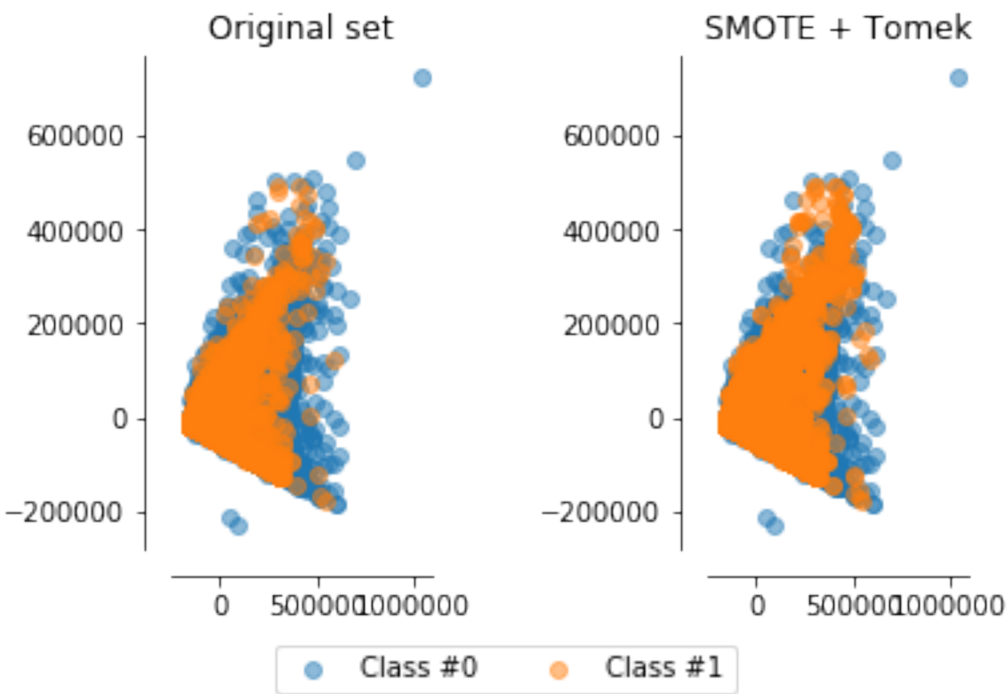
Apply Smote Tomek on imbalanced dataset

In [66]:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

from imblearn.combine import SMOTETomek

print(__doc__)

X= Credit_dummies[feature_cols]
y= Credit_dummies.default_nextMTH

# Generate the dataset
#X, y = make_classification(n_classes=2, class_sep=2, weights=[0.22, 0.78],
```

```
                                #n_informative=3, n_redundant=1, flip_y=0,

                                #n_features=18, n_clusters_per_class=1,
                                #n_samples=30000, random_state=10)

# Instanciate a PCA object for the sake of easy visualisation
pca = PCA(n_components= 2)
# Fit and transform x to visualise inside a 2D feature space
X_vis = pca.fit_transform(X)

# Apply SMOTE + Tomek links
sm = SMOTETomek()
X_resampled, y_resampled = sm.fit_sample(X, y)
X_res_vis = pca.transform(X_resampled)

# Two subplots, unpack the axes array immediately
f, (ax1, ax2) = plt.subplots(1, 2)

c0 = ax1.scatter(X_vis[y == 0, 0], X_vis[y == 0, 1], label="Class #0",
                 alpha=0.5)
c1 = ax1.scatter(X_vis[y == 1, 0], X_vis[y == 1, 1], label="Class #1",
                 alpha=0.5)
ax1.set_title('Original set')

ax2.scatter(X_res_vis[y_resampled == 0, 0], X_res_vis[y_resampled == 0, 1],
            label="Class #0", alpha=0.5)
ax2.scatter(X_res_vis[y_resampled == 1, 0], X_res_vis[y_resampled == 1, 1],
            label="Class #1", alpha=0.5)
ax2.set_title('SMOTE + Tomek')

# make nice plotting
for ax in (ax1, ax2):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()
    ax.spines['left'].set_position(('outward', 10))
    ax.spines['bottom'].set_position(('outward', 10))
    #ax.set_xlim([-6, 8])
    #ax.set_ylim([-6, 6])

plt.figlegend((c0, c1), ('Class #0', 'Class #1'), loc='lower center',
              ncol=2, labelspacing=0.)
plt.tight_layout(pad=3)
plt.show()
```

```
Automatically created module for IPython interactive environment
```



In [67]:

```
# To check the level of imbalanced data set after Smote Tomek application:
# The results show the Smote Tomek function has actually increased the minorit
y much more than reducing the majority
# The Smote Tomek function has provided a balanced data set at 50:50

count_d = 0
count_nd = 0

for i in y_resampled:
    if i == 1:
        count_d += 1
    else:
        count_nd += 1

total= count_d + count_nd
print("Total cases under study", total)
print("Total default cases", count_d,"or ", round(count_d/total,2)*100,"%" )
print("Total non-default cases", count_nd,"or ", round(count_nd/total,2)*100,"
%" )
```

```
Total cases under study 45100
Total default cases 22550 or  50.0 %
Total non-default cases 22550 or  50.0 %
```

In [68]:

```
# Create a new dataframe for all the columns after the Credit_dummies data set
was treated by Smote Tomek

X= pd.DataFrame(X_resampled)
y= pd.DataFrame(y_resampled)
Credit_ST= pd.concat([X, y], axis= 1)
Credit_ST.head()
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	...	12	1
0	20000.0	2.0	2.0	-1.0	-1.0	-2.0	-2.0	3913.0	0.0	689.0	...	0.0	0.0
1	120000.0	-1.0	2.0	0.0	0.0	0.0	2.0	2682.0	0.0	1000.0	...	0.0	2000
2	90000.0	0.0	0.0	0.0	0.0	0.0	0.0	29239.0	1518.0	1500.0	...	1000.0	5000
3	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	46990.0	2000.0	2019.0	...	1069.0	1000
4	50000.0	-1.0	0.0	-1.0	0.0	0.0	0.0	8617.0	2000.0	36681.0	...	689.0	679.(

5 rows × 22 columns

In [69]:

```
# Call out the headers of Credit_dummies to compare aginst the headers of X_re
sampled

Credit_dummies.head(1)
```

Out[69]:

	LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	PAY_AMT1
0	20000.0	2	2	-1	-1	-2	-2	3913.0	0.0

1 rows × 22 columns

In [70]:

```
# Put headers of Credit_dummies.columns back onto X_resampled and y_resampled

initialcol = Credit_dummies.columns
Credit_ST.columns = initialcol
```

In [71]:

```
Credit_ST.head(1)
```

Out[71]:

	LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	PAY_AMT1
0	20000.0	2.0	2.0	-1.0	-1.0	-2.0	-2.0	3913.0	0.0

1 rows × 22 columns

In [72]:

```
# Create feature matrix X from the columns treated by Smote Tomek
feature_cols= Credit_ST.columns.drop(['default_nextMTH'])
X= Credit_ST[feature_cols]

# Create response vector (y)

y= Credit_ST.default_nextMTH
```

In [73]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_pred = scaler.fit_transform(X)
```

The Credit_dummies data set has been treated by Smote Tomek. Now, the data set is renamed to Credit_ST.

Use Logistic Regression again to check the model accuracy

In [74]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123)

model = logreg.fit(X_train,y_train)

y_pred = logreg.predict(X_test)

print('Testing score: '  + str(logreg.score(X_test, y_test)))
print('Training score: '  + str(logreg.score(X_train, y_train)))
```

```
Testing score: 0.5260310421286031
Training score: 0.5336881005173688
```

In [75]:

```
print(logreg.intercept_)
print(logreg.coef_)
coeff = pd.DataFrame(dict(zip(X.columns,model.coef_[0])),index=[0])
coeff
```

```
[1.85714607e-08]
[[-8.36786870e-07  7.57003969e-08  5.82077936e-08  5.01246932e-08
   4.48667579e-08  3.92846087e-08  3.63546935e-08  3.68299699e-06
  -1.93134385e-05 -1.82145425e-05 -6.85088016e-06 -7.30606632e-06
  -6.10907006e-06 -5.69349191e-06  1.17159689e-08  3.41984986e-09
   1.11159501e-08  5.09535854e-09 -2.66891410e-10  1.23824066e-08
   5.72258197e-09]]
```

Out[75]:

	LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY
0	-8.367869e-07	7.570040e-08	5.820779e-08	5.012469e-08	4.486676e-08	3.928461e-08	3.63546 08

1 rows × 21 columns

The coefficient figures do not show any particular feature having significant importance, in other words, the features are having similar but very little weights on the target column.

This is typically so when the features and target column do not have linear relationship which is the basis applied in logistic regresion.

The train and test scores at about 52% only shows slightly better accuracy than the 50:50 data set treated by Smote Tomek

In [76]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [77]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[77]:

	Predicted: No	Predicted: Yes
Actual: No	4480	1104
Actual: Yes	4240	1451

In [78]:

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.51      0.80      0.63      5584
          1       0.57      0.25      0.35      5691

avg / total       0.54      0.53      0.49     11275
```

The Precision and the recall are having similar scores due to balanced data set after treated by Smote Tomek.

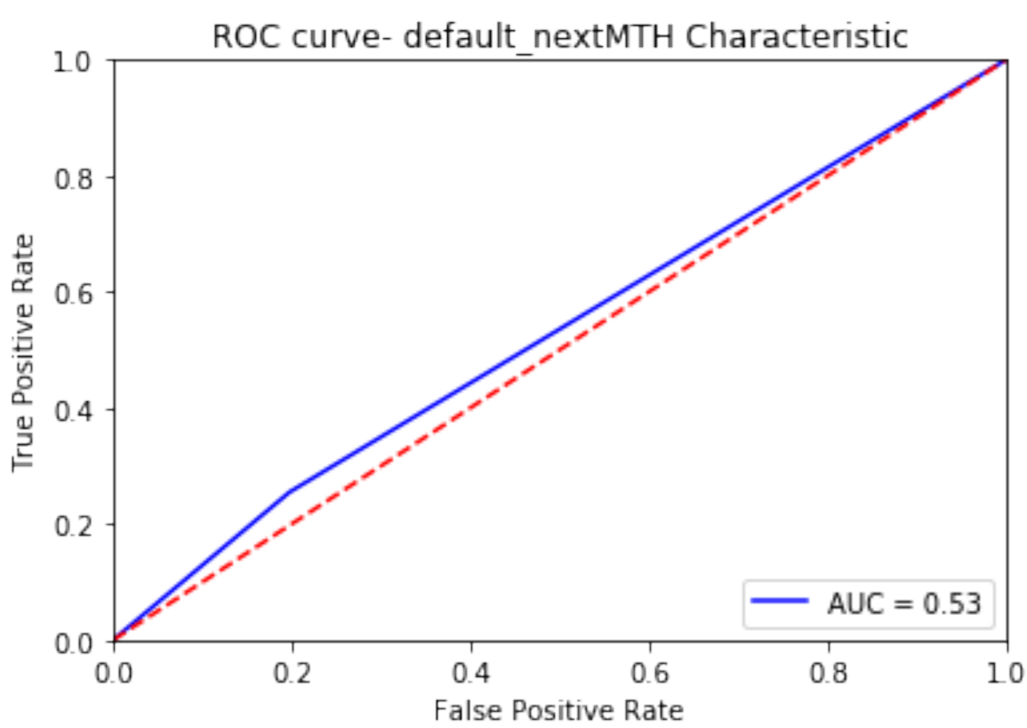
However, the accuray is only slightly more accurate than the base accuracy (50%) of the data set Credit_ST.

The model has poor predicting power on non default and default cases.

The low accuracy suggested the logistic regression classifier is not ideal for this type of highly categorical data set.

In [79]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('ROC curve- default_nextMTH Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



The Logistic classifier and the AUC scores are 53%.

That means this model is as good as guessing in predicting non defaulted and default cases.

Use another sklearn linear model to train and test the model

In [80]:

```
import pandas as pd
from sklearn import linear_model, model_selection, metrics

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test
_size= 0.25, random_state=46)
logit_simple = linear_model.LogisticRegression(C=1e9).fit(X_train, y_train)
```

In [81]:

```
1- y_train.mean()
```

Out[81]:

```
0.503089430894309
```

In [82]:

```
# What is our accuracy on the test set?
print(np.mean(y_test == logit_simple.predict(X_test)))
```

```
0.5250554323725055
```

In [83]:

```
# Get probability predictions.
logit_pred_proba = logit_simple.predict_proba(X_test)[:,1]
```

In [84]:

```
metrics.confusion_matrix(y_true=y_test, y_pred=logit_pred_proba > .5)
```

Out[84]:

```
array([[4637,  896],
       [4459, 1283]])
```

This sklearn linear model is built on logistic regression and hence, is showing the similar accuracy at about 52%

Use Decision Tree Classifier to build the test model

In [85]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier(random_state=123, max_depth=9)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123)

model= DT.fit(X_train, y_train)

y_pred= DT.predict(X_test)

print('Testing score: '  + str(DT.score(X_test, y_test)))
print('Training score: '  + str(DT.score(X_train, y_train)))
```

```
Testing score: 0.846740576496674
Training score: 0.8606060606060606
```

In [86]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [87]:

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.81      0.91      0.85      5584
          1       0.90      0.79      0.84      5691

avg / total       0.85      0.85      0.85     11275
```

In [88]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[88]:

	Predicted: No	Predicted: Yes
Actual: No	5061	523
Actual: Yes	1205	4486

This Decision Tree Classifier has a testing score of about 84%, in addition, the model can now perform equally good in identifying non default and default correctly over all types of cases.

In addition, the more costly Type II error is 11% which means the model is able to predict test data more accurately after training.

Apply Random Forest Classifier to build test model

In [123]:

```
from sklearn.ensemble import RandomForestClassifier

# Train model
Random = RandomForestClassifier(n_estimators=12, max_depth= 12)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123)

model= Random.fit(X_train,y_train)

# Predict on training set
y_pred = Random.predict(X_test)


print('Testing score: ' + str(Random.score(X_test, y_test)))
print('Training score: '  + str(Random.score(X_train, y_train)))
list(zip(feature_cols, Random.feature_importances_))
```

```
Testing score: 0.8704212860310421
Training score: 0.9035033259423503
```

Out[123]:

```
[('LIMIT_BAL', 0.021742646873163563),
 ('PAY_0', 0.2419106307338927),
 ('PAY_2', 0.07583938220836083),
 ('PAY_3', 0.03915175407196083),
 ('PAY_4', 0.0737127607005522),
 ('PAY_5', 0.024385138485233535),
 ('PAY_6', 0.04864200454128837),
 ('BILL_AMT1', 0.03859527573922469),
 ('PAY_AMT1', 0.037501167303126824),
 ('PAY_AMT2', 0.022520973321286608),
 ('PAY_AMT3', 0.019702948051378646),
 ('PAY_AMT4', 0.02155863252415359),
 ('PAY_AMT5', 0.01995658205300862),
 ('PAY_AMT6', 0.01643899980366313),
 ('SEX_1', 0.05012077793982415),
 ('EDUCATION_1.0', 0.05270421786833879 4),
 ('EDUCATION_2.0', 0.05035740446087555),
 ('EDUCATION_3.0', 0.0273236955176093),
 ('EDUCATION_4.0', 0.00012977666523859 13),
 ('MARRIAGE_1.0', 0.05847250321599216),
 ('MARRIAGE_2.0', 0.05923272792182734)]
```

In [124]:

```
# Features and thier respective coefficients
feature_importances_df = pd.DataFrame(list(zip(X.columns, Random.feature_impor
tances_)), columns = ['Features', 'Estimated_Coefficients'])
feature_importances_df.sort_values(by='Estimated_Coefficients', ascending=Fals
e)
```

Out[124]:

	Features	Estimated_Coefficients
1	PAY_0	0.241911
2	PAY_2	0.075839
4	PAY_4	0.073713
20	MARRIAGE_2.0	0.059233
19	MARRIAGE_1.0	0.058473
15	EDUCATION_1.0	0.052704
16	EDUCATION_2.0	0.050357
14	SEX_1	0.050121
6	PAY_6	0.048642
3	PAY_3	0.039152
7	BILL_AMT1	0.038595
8	PAY_AMT1	0.037501
17	EDUCATION_3.0	0.027324
5	PAY_5	0.024385
9	PAY_AMT2	0.022521
0	LIMIT_BAL	0.021743
11	PAY_AMT4	0.021559
12	PAY_AMT5	0.019957
10	PAY_AMT3	0.019703
13	PAY_AMT6	0.016439
18	EDUCATION_4.0	0.000130

The feature importance check shows

The most importance features are PAY_0 and PAY_2 which make sense as these are the most recent bill payments that reveal the payment capability.

The least important feature is EDUCATION category 4 which also makes sense as this is a minority category under EDUCATION column.

Random forest classifier score is as high as 87% and that means it can work better with categorical features, than logistic regression.

Check the test data set accuracy using confusion matric

In [90]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc
_auc_score, roc_curve, recall_score, classification_report
cm=metrics.confusion_matrix(y_test,y_pred)
```

In [91]:

```
list1 = ["Actual: No", "Actual: Yes"]
list2 = ["Predicted: No", "Predicted: Yes"]
pd.DataFrame(cm, list1,list2)
```

Out[91]:

	Predicted: No	Predicted: Yes
Actual: No	5154	430
Actual: Yes	1038	4653

In [92]:

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.83      0.92      0.88      5584
          1       0.92      0.82      0.86      5691

avg / total       0.87      0.87      0.87     11275
```

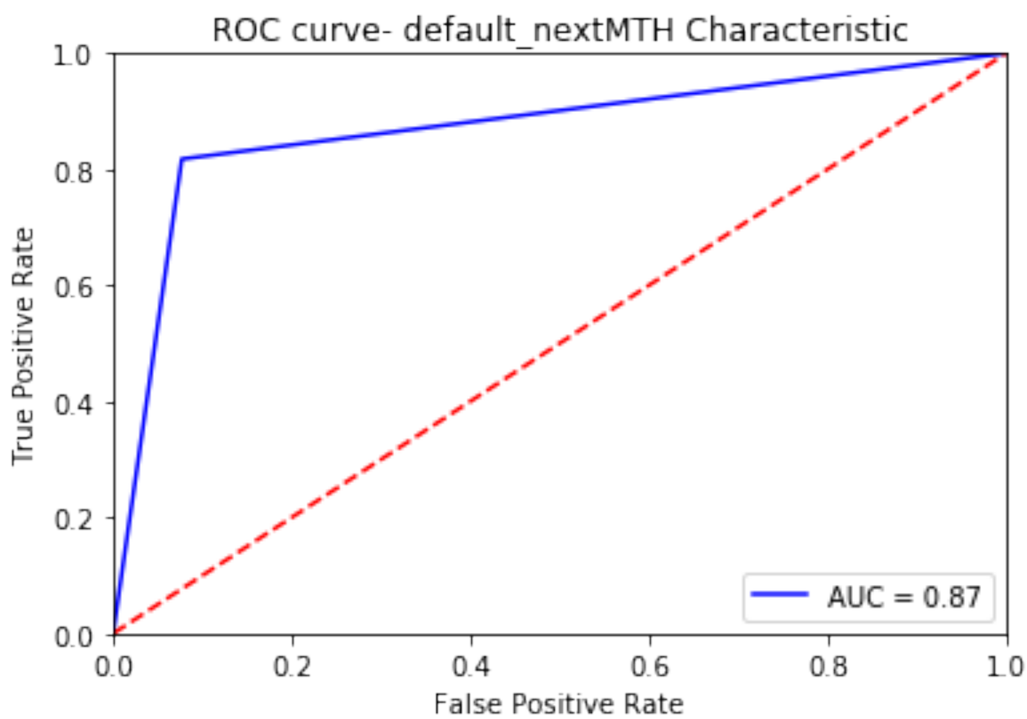
The Precision and the recall are having similarly high scores due to more suitable Classifier applied on balanced data set after treated by Smote Tomek.

The accuray is now much more accurate than the base accuracy (50%) of the data set Credit_ST.

The model created by Random Forest Classifier is now having very good predicting power on non default and default cases.

In [93]:

```
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('ROC curve- default_nextMTH Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



The Random Forest classifier and the AUC scores are 87%. That means this model is very good in predicting non defaulted and default cases equally.

The more costly Type II error is 9% which means the model is able to predict test data even more accurately after training.

If we can pre-empt and avoid the default accounts from materialise, we will cut down the impairment ratio from 22%

If we can allow the non default accounts to increase more spending, we will be able to optimise earning.

If only 9% that will default inadvertently that is beyond the model predictability, this credit card portfolio impairment ratio should be brought from 22% to about 9% remarkably!

With the comparisons over different classifiers, we can confirm that Random Forest Classifier appear to be the best classifier in predicting non defaut and default cases in this case study.

In [94]:

```
Credit_ST.to_excel("Credit_ST.xlsx")
```

In [95]:

```
X_orig.to_excel("X_orig(postST).xlsx")
```

In [108]:

```
X_orig.loc[219]
```

Out[108]:

```
LIMIT_BAL         310000.0
PAY_0                 -1.0
PAY_2                 -1.0
PAY_3                 -1.0
PAY_4                 -1.0
PAY_5                 -2.0
PAY_6                 -2.0
BILL_AMT1           1424.0
PAY_AMT1            4542.0
PAY_AMT2             126.0
PAY_AMT3               0.0
PAY_AMT4               0.0
PAY_AMT5               0.0
PAY_AMT6               0.0
SEX_1                  0.0
EDUCATION_1.0          1.0
EDUCATION_2.0          0.0
EDUCATION_3.0          0.0
EDUCATION_4.0          0.0
MARRIAGE_1.0           0.0
MARRIAGE_2.0           1.0
Name: 219, dtype: float64
```

In [98]:

```
len(X_orig)
```

Out[98]:

```
29932
```

In [105]:

```
X_orig.dtypes
```

Out[105]:

```
LIMIT_BAL          float64
PAY_0                int64
PAY_2                int64
PAY_3                int64
PAY_4                int64
PAY_5                int64
PAY_6                int64
BILL_AMT1          float64
PAY_AMT1           float64
PAY_AMT2           float64
PAY_AMT3           float64
PAY_AMT4           float64
PAY_AMT5           float64
PAY_AMT6           float64
SEX_1                uint8
EDUCATION_1.0        uint8
EDUCATION_2.0        uint8
EDUCATION_3.0        uint8
EDUCATION_4.0        uint8
MARRIAGE_1.0         uint8
MARRIAGE_2.0         uint8
dtype: object
```

In [ ]:

```
default= np.asarray(arraylist)
default=
```

In [162]:

```
import pickle
```

In [163]:

```
with open('model_pickle','wb') as f:
    pickle.dump(model,f)
```

In [164]:

```
with open('model_pickle','rb') as f:
    mp= pickle.load(f)
```

In [165]:

```
y_mp= mp.predict(X_orig)
```

In [166]:

```
count0=0
count1=0

for i in y_mp :
    if i == 0:
        count0 += 1
    else:
        count1 += 1

print('This is the predicted y column. ')
print("0: " + str(count0) + "\n" + "1: " + str(count1),'\n')
```

```
This is the predicted y column.
0: 25185
1: 4747
```

In [173]:

```
y_true= Credit_dummies.default_nextMTH
```

In [174]:

```
from sklearn import metrics

print ('Random Forest Scores : ' )
print ('\nConfusion Matrix' )
print (metrics.confusion_matrix(y_true=y_true, y_pred=y_mp) )      # confusion
matrix
print ('\nClassification Report' )
print (metrics.classification_report(y_true, y_mp))               # classific
ation report for RF
print ('-------------------------------------------------------')
print ('\n' )
```

```
Random Forest Scores :

Confusion Matrix
[[21975  1326]
 [ 3210  3421]]

Classification Report
             precision    recall  f1-score   support

          0       0.87      0.94      0.91     23301
          1       0.72      0.52      0.60      6631

avg / total       0.84      0.85      0.84     29932

-------------------------------------------------------
```

In [180]:

```
y_mp = mp.predict(X_orig)                              # predict the y
proba_mp =   mp.predict_proba(X_orig)                      # compute the probabil
ies..
proba_mp = proba_mp[:,1]
print ('y_mp shape :' , y_mp.shape)          #resulting y_pred_RF is a 1D num
py arrays of 0s and 1s.
print ('y_mp contents : ', y_mp)
print (' proba_mp contents : ', proba_mp)
print ('\n' )
```

```
y_mp shape : (29932,)
y_mp contents :  [1 0 0 ... 1 1 0]
 proba_mp contents :  [0.8067439  0.31723864 0.15975161 ... 0.7050
7085 0.78215979 0.22070461]
```

In [231]:

```
pd.DataFrame(proba_mp).inplace= True
```

In [232]:

```
pd.DataFrame(y_mp).inplace= True
```

In [291]:

```
pd.DataFrame(X_orig).inplace= True
```

In [233]:

```
pd.DataFrame(Credit_dummies.default_nextMTH).inplace= True
```

In [360]:

```
Credit_dummies.columns
```

Out[360]:

```
Index(['LIMIT_BAL', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', '
PAY_6',
       'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4'
, 'PAY_AMT5',
       'PAY_AMT6', 'SEX_1', 'EDUCATION_1.0', 'EDUCATION_2.0', 'EDU
CATION_3.0',
       'EDUCATION_4.0', 'MARRIAGE_1.0', 'MARRIAGE_2.0', 'default_n
extMTH'],
      dtype='object')
```

In [292]:

```
Credit_dummies.default_nextMTH.shape
```

Out[292]:

```
(29932,)
```

In [293]:

```
y_mp
```

Out[293]:

```
array([1, 0, 0, ..., 1, 1, 0])
```

In [300]:

```
columns = ['pickle_guess']
df1 = pd.DataFrame(y_mp,columns=columns)

df1.head()
df1.shape
```

Out[300]:

```
(29932, 1)
```

In [295]:

```
proba_mp
```

Out[295]:

```
array([0.8067439 , 0.31723864, 0.15975161, ..., 0.70507085, 0.7821
5979,
       0.22070461])
```

In [328]:

```
columns = ['proba']
df2 = pd.DataFrame(proba_mp,columns=columns)

df2.head()
df2.shape
```

Out[328]:

```
(29932, 1)
```

In [329]:

```
X_orig.shape
```

Out[329]:

```
(29932, 21)
```

In [340]:

```
columns = ['default_nextMTH']
df3 = pd.DataFrame(Credit_dummies.default_nextMTH,columns=columns)

df3.head()
df3.shape
```

Out[340]:

```
(29932, 1)
```

In [344]:

```
df4= np.concatenate([df3, X_orig], axis=1)
```

In [345]:

```
df4.shape
```

Out[345]:

```
(29932, 22)
```

In [351]:

```
df4= pd.DataFrame(df4)
```

In [361]:

```
df4.columns= ['default_nextMTH','LIMIT_BAL', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4
', 'PAY_5', 'PAY_6',
       'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5'
,
       'PAY_AMT6', 'SEX_1', 'EDUCATION_1.0', 'EDUCATION_2.0', 'EDUCATION_3.0',
       'EDUCATION_4.0', 'MARRIAGE_1.0', 'MARRIAGE_2.0']
```

In [362]:

```
df4
```

Out[362]:

	default_nextMTH	LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	E
0	1.0	20000.0	2.0	2.0	-1.0	-1.0	-2.0	-2.0	3
1	1.0	120000.0	-1.0	2.0	0.0	0.0	0.0	2.0	2
2	0.0	90000.0	0.0	0.0	0.0	0.0	0.0	0.0	2
3	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	4
4	0.0	50000.0	-1.0	0.0	-1.0	0.0	0.0	0.0	8
5	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	6
6	0.0	500000.0	0.0	0.0	0.0	0.0	0.0	0.0	3

7	0.0	100000.0	0.0	-1.0	-1.0	0.0	0.0	-1.0	1
8	0.0	140000.0	0.0	0.0	2.0	0.0	0.0	0.0	1
9	0.0	20000.0	-2.0	-2.0	-2.0	-2.0	-1.0	-1.0	0
10	0.0	200000.0	0.0	0.0	2.0	0.0	0.0	-1.0	1
11	0.0	260000.0	-1.0	-1.0	-1.0	-1.0	-1.0	2.0	1
12	0.0	630000.0	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	1
13	1.0	70000.0	1.0	2.0	2.0	0.0	0.0	2.0	6
14	0.0	250000.0	0.0	0.0	0.0	0.0	0.0	0.0	7
15	0.0	50000.0	1.0	2.0	0.0	0.0	0.0	0.0	5
16	1.0	20000.0	0.0	0.0	2.0	2.0	2.0	2.0	1
17	0.0	320000.0	0.0	0.0	0.0	-1.0	-1.0	-1.0	2
18	0.0	360000.0	1.0	-2.0	-2.0	-2.0	-2.0	-2.0	0
19	0.0	180000.0	1.0	-2.0	-2.0	-2.0	-2.0	-2.0	0
20	0.0	130000.0	0.0	0.0	0.0	0.0	0.0	-1.0	3
21	1.0	120000.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	3
22	1.0	70000.0	2.0	0.0	0.0	2.0	2.0	2.0	4
23	1.0	450000.0	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	5
24	0.0	90000.0	0.0	0.0	0.0	-1.0	0.0	0.0	4
25	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	4
26	1.0	60000.0	1.0	-2.0	-1.0	-1.0	-1.0	-1.0	-
27	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	2
28	0.0	50000.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	6
29	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	..
29902	0.0	360000.0	-1.0	-1.0	-1.0	0.0	0.0	-1.0	3
29903	0.0	80000.0	0.0	0.0	0.0	0.0	0.0	0.0	6
29904	0.0	190000.0	0.0	0.0	0.0	0.0	0.0	-1.0	2
29905	1.0	230000.0	1.0	-2.0	-2.0	-2.0	-2.0	-2.0	0
29906	1.0	50000.0	1.0	2.0	2.0	2.0	0.0	0.0	1
29907	0.0	220000.0	0.0	0.0	-1.0	-1.0	-2.0	-2.0	4
29908	1.0	40000.0	2.0	2.0	3.0	2.0	2.0	2.0	5
29909	0.0	420000.0	0.0	0.0	0.0	0.0	0.0	0.0	1
29910	0.0	310000.0	0.0	0.0	0.0	0.0	0.0	0.0	2

29911	0.0	180000.0	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	0
29912	0.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	4
29913	0.0	50000.0	1.0	2.0	2.0	2.0	0.0	0.0	3
29914	1.0	90000.0	0.0	0.0	0.0	0.0	0.0	0.0	7
29915	0.0	20000.0	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	1
29916	0.0	30000.0	-1.0	-1.0	-2.0	-1.0	-1.0	-1.0	3
29917	0.0	240000.0	-2.0	-2.0	-2.0	-2.0	-2.0	-2.0	0
29918	0.0	360000.0	-1.0	-1.0	-2.0	-2.0	-2.0	-2.0	2
29919	0.0	130000.0	0.0	0.0	0.0	0.0	0.0	0.0	2
29920	0.0	250000.0	0.0	0.0	0.0	0.0	0.0	0.0	2
29921	0.0	150000.0	-1.0	-1.0	-1.0	-1.0	-1.0	-2.0	3
29922	0.0	140000.0	0.0	0.0	0.0	0.0	0.0	0.0	1
29923	1.0	210000.0	3.0	2.0	2.0	2.0	2.0	2.0	2
29924	0.0	10000.0	0.0	0.0	0.0	-2.0	-2.0	-2.0	8
29925	0.0	100000.0	0.0	-1.0	-1.0	0.0	0.0	0.0	3
29926	1.0	80000.0	2.0	2.0	2.0	2.0	2.0	2.0	7
29927	0.0	220000.0	0.0	0.0	0.0	0.0	0.0	0.0	1
29928	0.0	150000.0	-1.0	-1.0	-1.0	-1.0	0.0	0.0	1
29929	1.0	30000.0	4.0	3.0	2.0	-1.0	0.0	0.0	3
29930	1.0	80000.0	1.0	-1.0	0.0	0.0	0.0	-1.0	-
29931	1.0	50000.0	0.0	0.0	0.0	0.0	0.0	0.0	4

29932 rows × 22 columns

In [363]:

```
frames= [df1, df2, df4]
```

In [364]:

```
result= pd.concat(frames, axis=1, ignore_index= True)
```

In [365]:

```
result.shape
```

Out[365]:

```
(29932, 24)
```

In [369]:

```
result.columns= ['pickle_guess', 'Proba','default_nextMTH','LIMIT_BAL', 'PAY_0
', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
       'BILL_AMT1', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5'
,
       'PAY_AMT6', 'SEX_1', 'EDUCATION_1.0', 'EDUCATION_2.0', 'EDUCATION_3.0',
       'EDUCATION_4.0', 'MARRIAGE_1.0', 'MARRIAGE_2.0']
```

In [370]:

```
result.to_excel("CardDefault_pred.xlsx")
```

Apply KNeighbors Classifier to build test model

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
# Calculate TRAINING ACCURACY and TESTING ACCURACY for K=1 through 100.

k_range = list(range(1, 101))
training_accuracy = []
testing_accuracy = []

# Find test accuracy for all values of K between 1 and 100 (inclusive).
for k in k_range:

    # Instantiate the model with the current K value.
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Calculate training accuracy
    y_pred_class = knn.predict(X)
    training_accuracyy = metrics.accuracy_score(y, y_pred_class)
    training_accuracy.append(training_accuracyy)

    # Calculate testing accuracy.
    y_pred_class = knn.predict(X_test)
    testing_accuracyy = metrics.accuracy_score(y_test, y_pred_class)
    testing_accuracy.append(testing_accuracyy)

print('Testing score: '  + str(knn.score(X_test, y_test)))
print('Training score: '  + str(knn.score(X_train, y_train)))
```

In [ ]:

```
# Allow plots to appear in the notebook.
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

# Create a DataFrame of K, training accuracy, and testing accuracy.
column_dict = {'K': k_range, 'training accuracy':training_accuracy, 'testing a
ccuracy':testing_accuracy}
df = pd.DataFrame(column_dict).set_index('K').sort_index(ascending=False)
df
```

In [ ]:

```
# Plot the relationship between K (HIGH TO LOW) and TESTING ERROR.
df.plot(y='testing accuracy');
plt.xlabel('Value of K for KNN');
plt.ylabel('Accuracy (higher is better)');
```

In [ ]:

```
# Find the minimum testing accuracy and the associated K value.
df.sort_values('testing accuracy').head()
```

In [ ]:

```
# Find the maximum testing accuracy and the associated K value.
df.sort_values('testing accuracy').tail()
```

In [ ]:

```
# Alternative method:
max(list(zip(training_accuracy, testing_accuracy,k_range)))
```

The highest testing accuracy obtained from KNN Classifier is 0.836 when K = 1, however, this score is much lower than the training score of 0.956. That means the model is over-fitted.

The next K numbers without overfitting issue is when K = 5. Testing score is 75%, Training score is 81%.

Since the testing score is still lower than 87% obtained from Random Forest Classifier, we will stick to use Random Forest Classifier model.

Please ignore SVC classifier below. It is only in research stage and has nothing to do with this case study submission.

In [ ]:

```
import numpy as np

from sklearn.svm import SVC

svc = SVC(gamma='auto', random_state=123)
SVC(C=1.0, cache_size=20, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=1, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=123, shrinking=True,
    tol=0.001, verbose=False)


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123)

model= svc.fit(X_train, y_train)

y_pred= svc.predict(X_test)

print('Testing score: '  + str(svc.score(X_test, y_test)))
print('Training score: '  + str(svc.score(X_train, y_train)))
```