https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset

Step1:Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import io
%cd "C:\Users\91866\Desktop\github_dataset\Default of Credit Card Clients Dataset"
```

```
C:\Users\91866\Desktop\github_dataset\Default of Credit Card Clients Dataset
```

In [3]:
```
dataset=pd.read_csv("UCI_Credit_Card.csv")
```

In [4]:
```
# Printing the first five rows of dataset
dataset.head()
```

Out[4]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	689.0	
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0	
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	1

5 rows × 25 columns

In [5]:
```
# Printing the last five rows of Dataset
dataset.tail()
```

Out[5]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AM
29995	29996	220000.0	1	3	1	39	0	0	0	0	...	88004.0	31237.0	15980.0	8500.0	20000
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	...	8979.0	5190.0	0.0	1837.0	3526
29997	29998	30000.0	1	2	2	37	4	3	2	-1	...	20878.0	20582.0	19357.0	0.0	0
29998	29999	80000.0	1	3	1	41	1	-1	0	0	...	52774.0	11855.0	48944.0	85900.0	3409
29999	30000	50000.0	1	2	1	46	0	0	0	0	...	36535.0	32428.0	15313.0	2078.0	1800

5 rows × 25 columns

In [6]:
```
# Extracting information of Datasets
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
```

```
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [7]:

```
# knowing the statistical measures
dataset.describe()
```

Out[7]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	...
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	-0.220667	...
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	1.169139	...
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000	...
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000	...
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000	...
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000	...
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000	...

8 rows × 25 columns

There are 3000 distinct credit card clients.

The average value for the amount of credit card limit is 1,67,484. The standard deviation is unusually large, max value being 10,00,000.

Education Level is mostly graduate school and university.

Most of the clients are either married or single .

Average age is 35.5 years, with a standard deviation of 9.2

As the value 0 for default payment means 'not default' and value 1 means 'default',the mean of 0.221 means that there are 22.1% of credit card contracts that will default next month (will verify this in the next sections of this analysis).

In [8]:
```
dataset.columns
```

Out[8]:
```
Index(['ID', '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',
       'default.payment.next.month'],
      dtype='object')
```

In [9]:
```
dataset.index = dataset['ID']
```

In [10]:
```
dataset.drop('ID',axis=1,inplace=True)
```

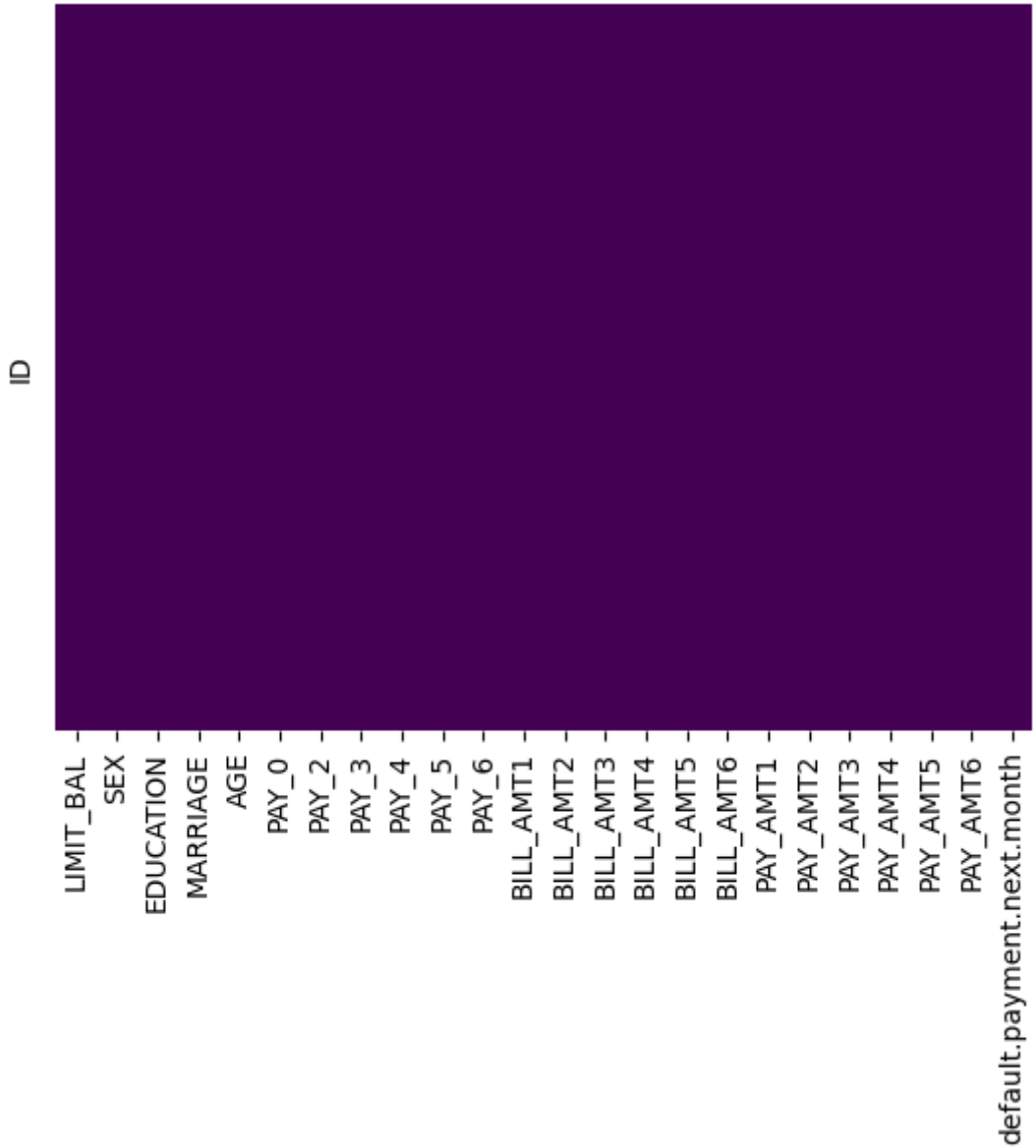
In [11]:
```
# Checking the number missing value of each columns
dataset.isnull().sum()
```

Out[11]:
```
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.payment.next.month  0
dtype: int64
```

In [12]:
```
sns.heatmap(dataset.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[12]:
```
<AxesSubplot: ylabel='ID'>
```



In [13]:
```
dataset.nunique()
```

Out[13]:
```
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.payment.next.month    2
dtype: int64
```

In [14]:
```
dataset['SEX'].value_counts(dropna=False)
```

Out[14]:
```
2    18112
1    11888
Name: SEX, dtype: int64
```

In [15]:
```
dataset['EDUCATION'].value_counts(dropna=False)
```

Out[15]:
```
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64
```

In [16]:
```
dataset['MARRIAGE'].value_counts(dropna=False)
```

Out[16]:
```
2    15964
1    13659
3      323
0       54
Name: MARRIAGE, dtype: int64
```

In [17]:
```
dataset = dataset.rename(columns={'default.payment.next.month': 'def_pay',
                        'PAY_0': 'PAY_1'})
dataset.head()
```

Out[17]:

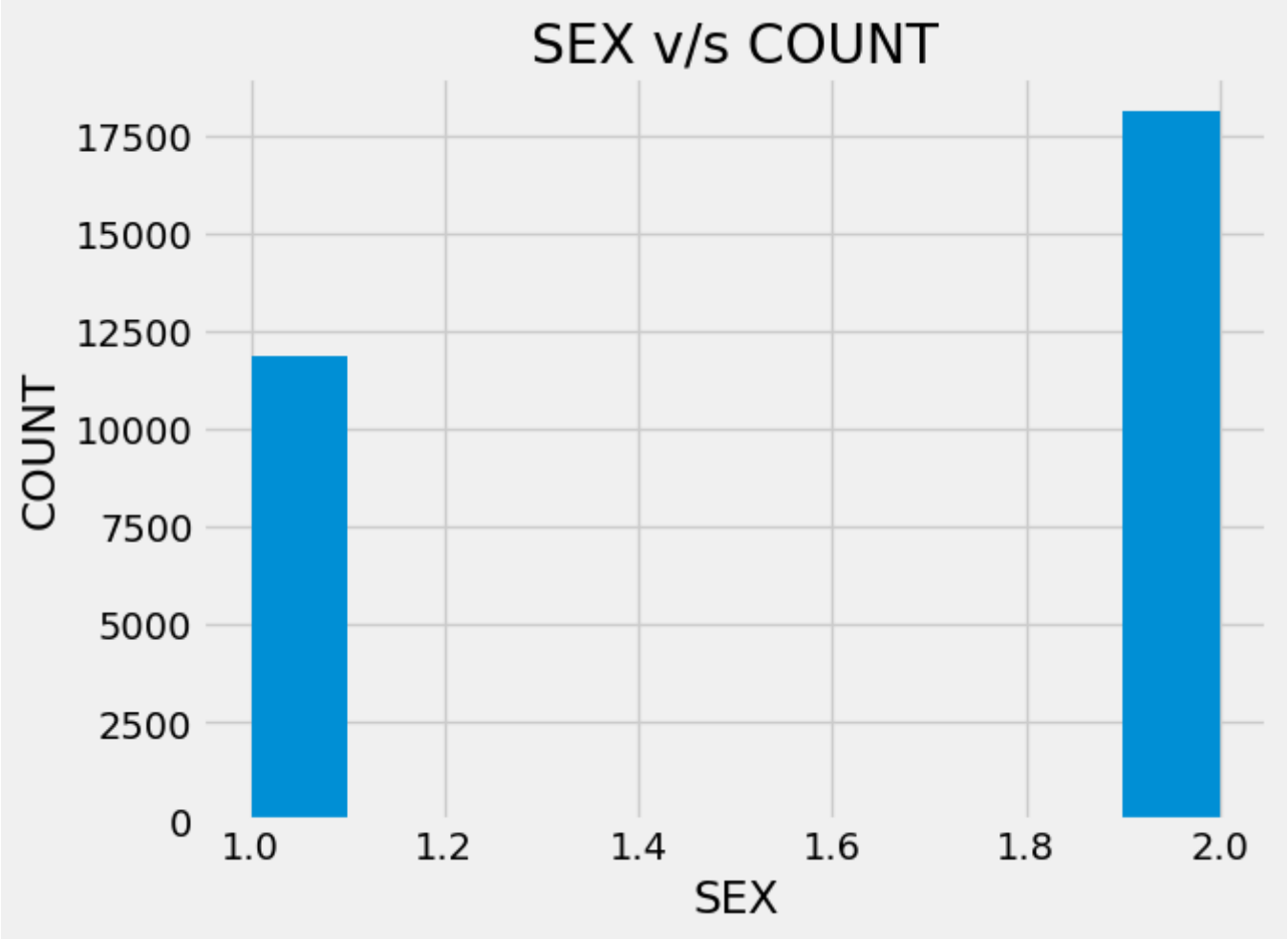
ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
1	20000.0	2	2	1	24	2	2	-1	-1	-2	...	0.0	0.0	0.0	0.0	689.0
2	120000.0	2	2	2	26	-1	2	0	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0
3	90000.0	2	2	2	34	0	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0
4	50000.0	2	2	1	37	0	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0
5	50000.0	1	2	1	57	-1	0	-1	0	0	...	20940.0	19146.0	19131.0	2000.0	36681.0

5 rows × 24 columns

Step2: Data Visualisation and Analysis

In [18]:
```
plt.style.use('fivethirtyeight')
dataset['SEX'].hist()
plt.xlabel('SEX')
plt.ylabel('COUNT')
plt.title('SEX v/s COUNT')
```

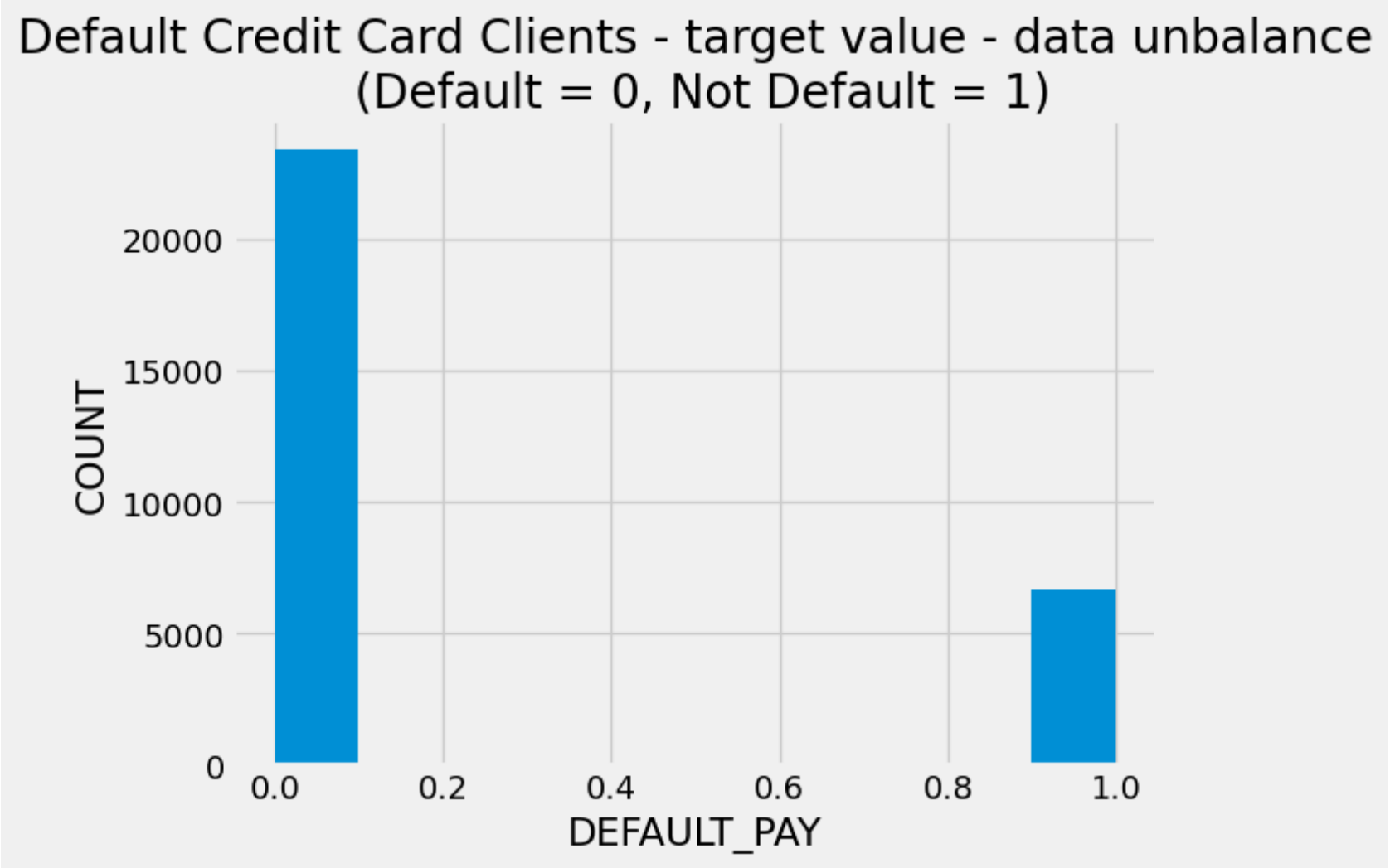
Out[18]:
```
Text(0.5, 1.0, 'SEX v/s COUNT')
```



Number of Male credit holder is less than Female

In [19]:
```
plt.style.use('fivethirtyeight')
dataset['def_pay'].hist()
plt.xlabel('DEFAULT_PAY')
plt.ylabel('COUNT')
plt.title('Default Credit Card Clients - target value - data unbalance\n (Default = 0, Not Default = 1)')
```

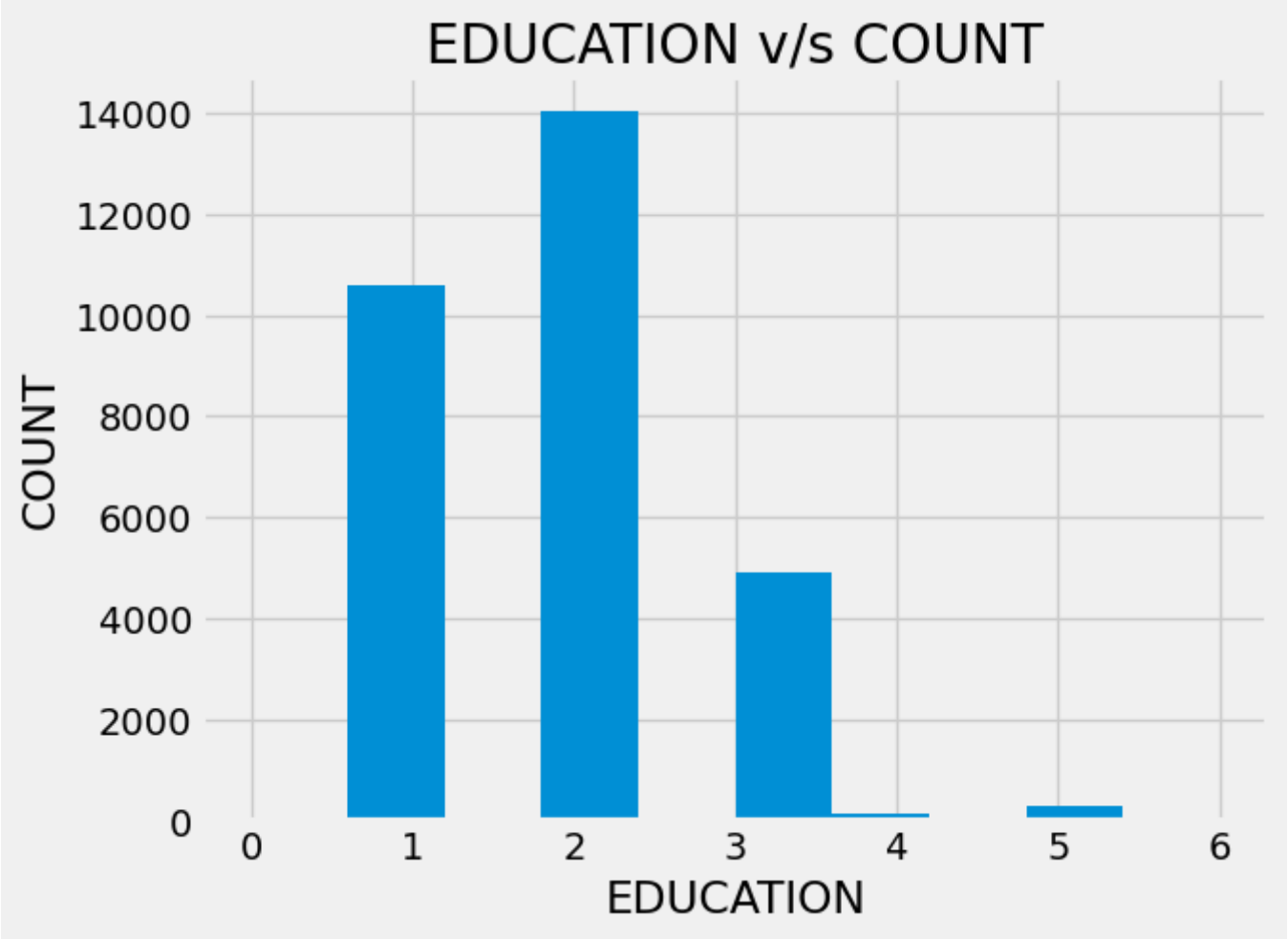
Out[19]:
```
Text(0.5, 1.0, 'Default Credit Card Clients - target value - data unbalance\n (Default = 0, Not Default = 1)')
```



Percentage of Defaulters are smaller than the Non Defaulters in the given dataset

In [20]:
```
dataset['EDUCATION'].hist()
plt.xlabel('EDUCATION')
plt.ylabel('COUNT')
plt.title('EDUCATION v/s COUNT')
```

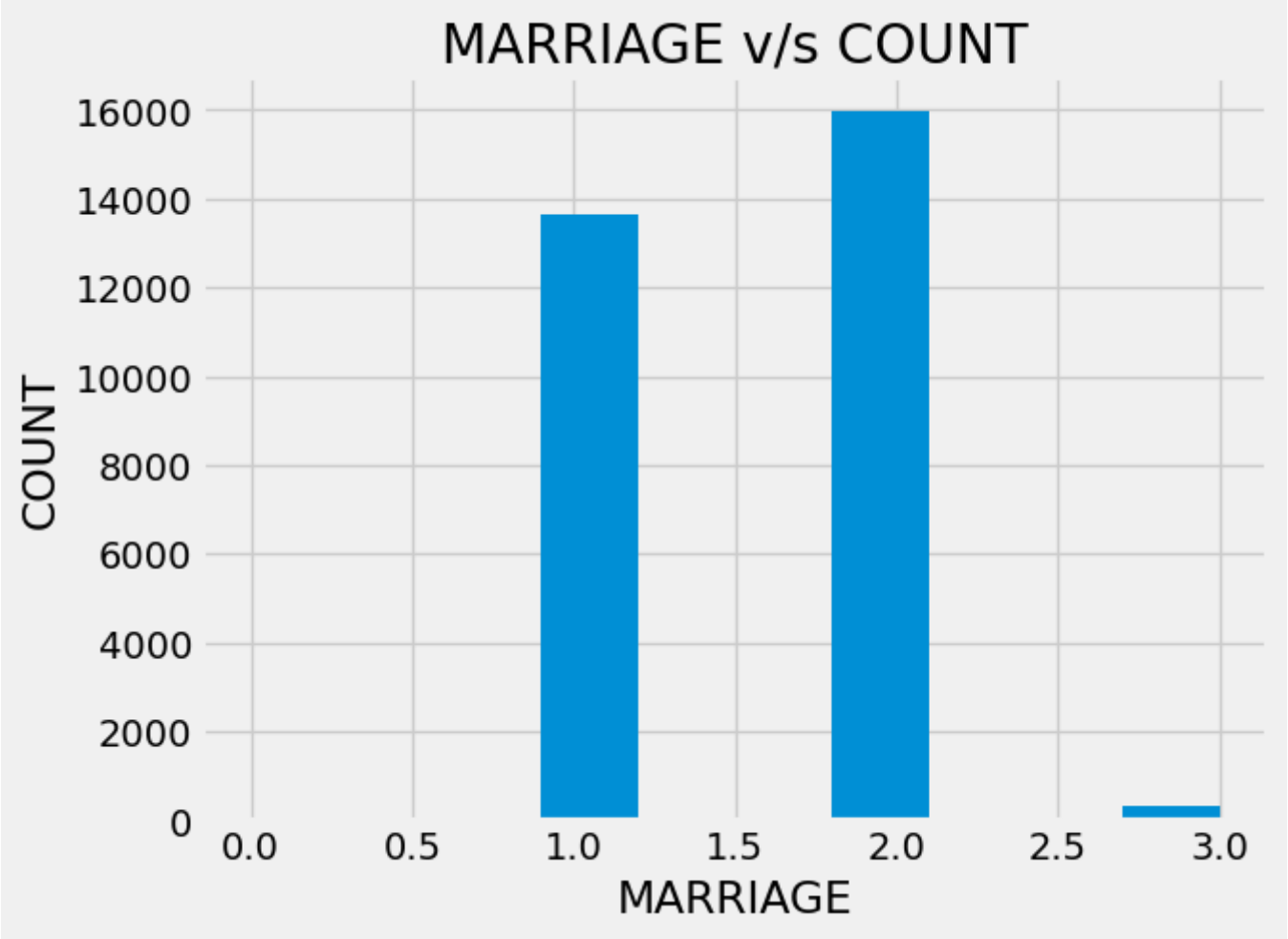
Out[20]: Text(0.5, 1.0, 'EDUCATION v/s COUNT')



More number of credit holders are university students followed by Graduates and then High school students

In [21]:
```
dataset['MARRIAGE'].hist()
plt.xlabel('MARRIAGE')
plt.ylabel('COUNT')
plt.title('MARRIAGE v/s COUNT')
```

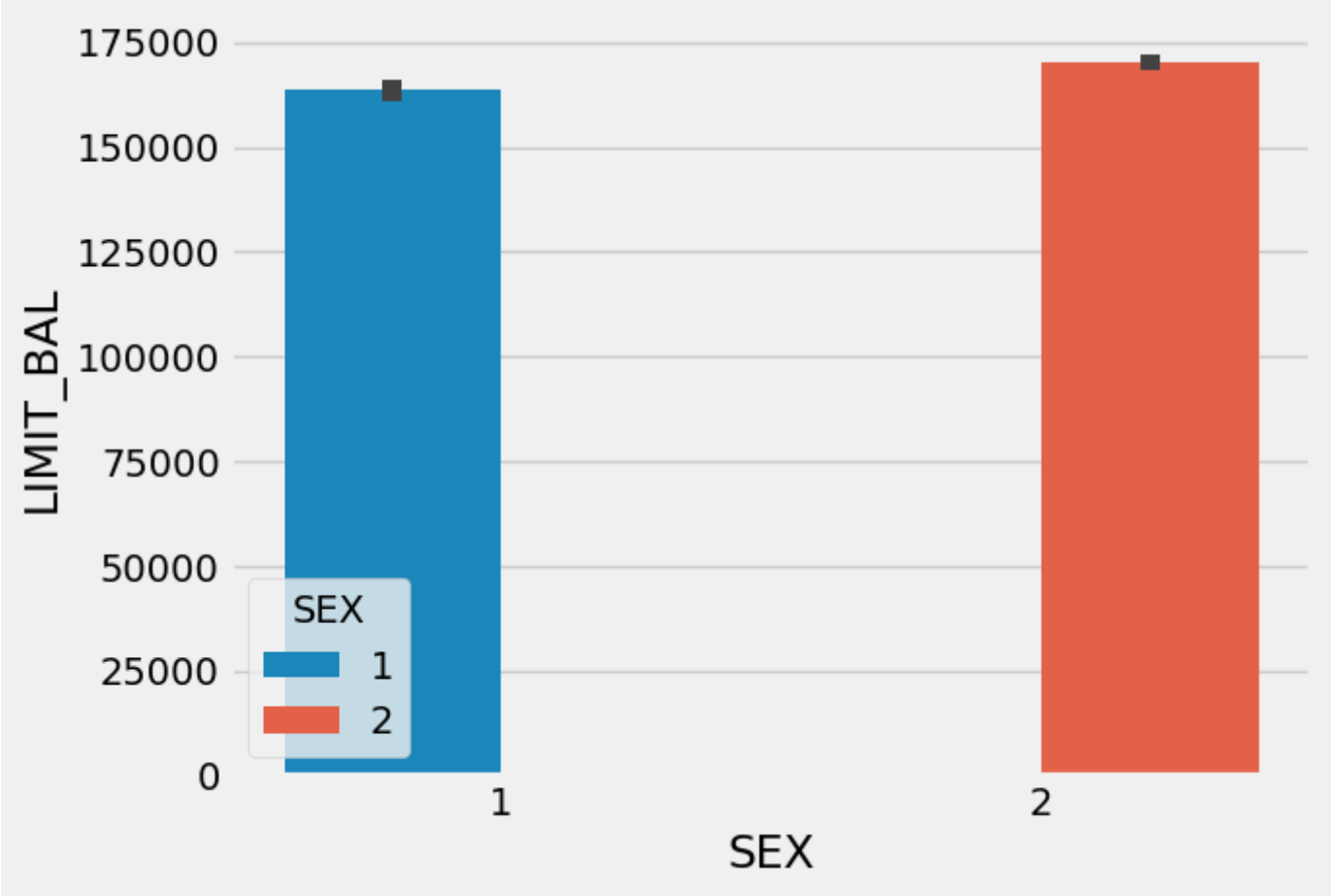
Out[21]:
```
Text(0.5, 1.0, 'MARRIAGE v/s COUNT')
```



More number of credit cards holder are Married

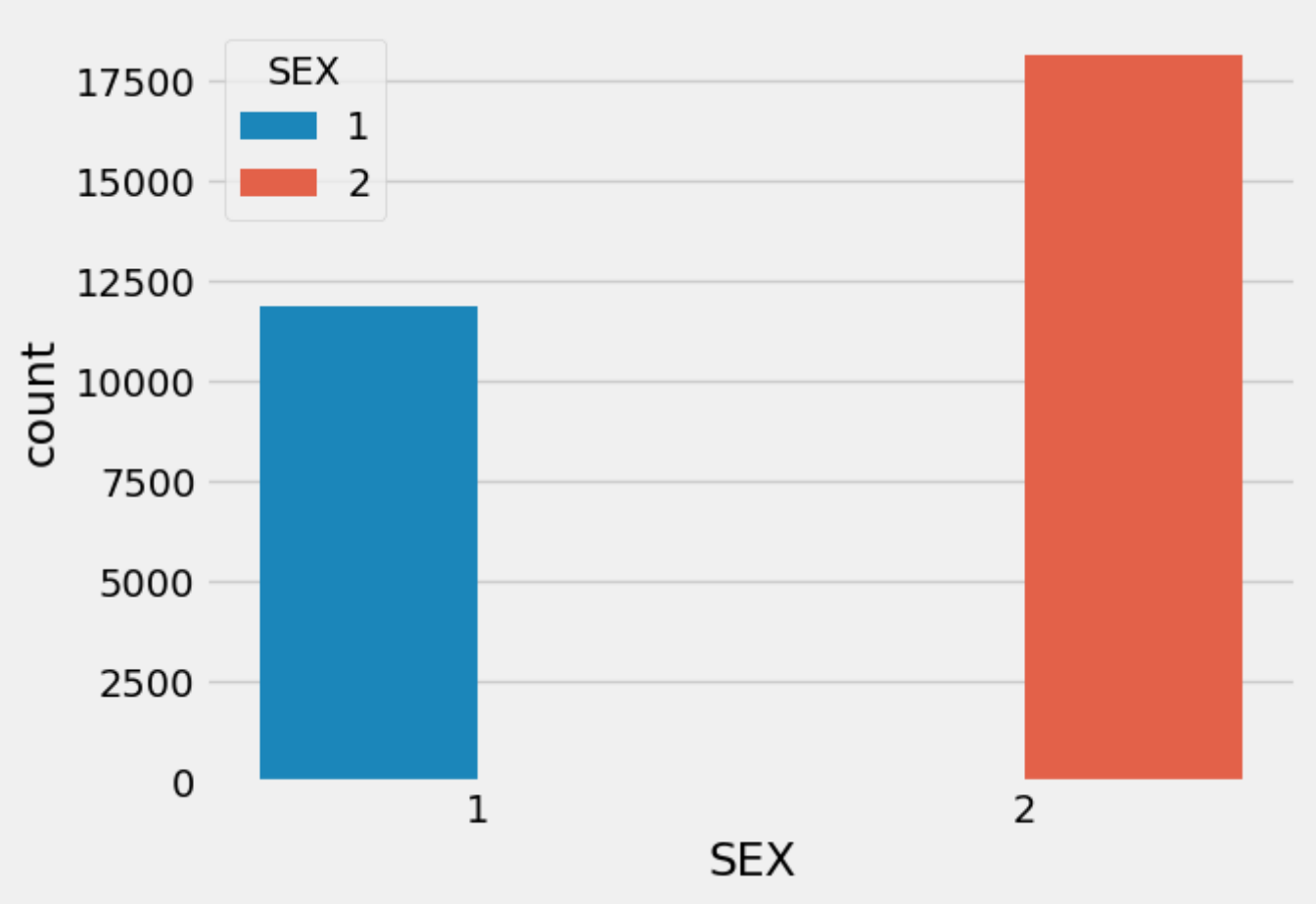
In [22]:
```
sns.barplot(x='SEX',y='LIMIT_BAL',data=dataset,hue='SEX')
```

Out[22]:
```
<AxesSubplot: xlabel='SEX', ylabel='LIMIT_BAL'>
```



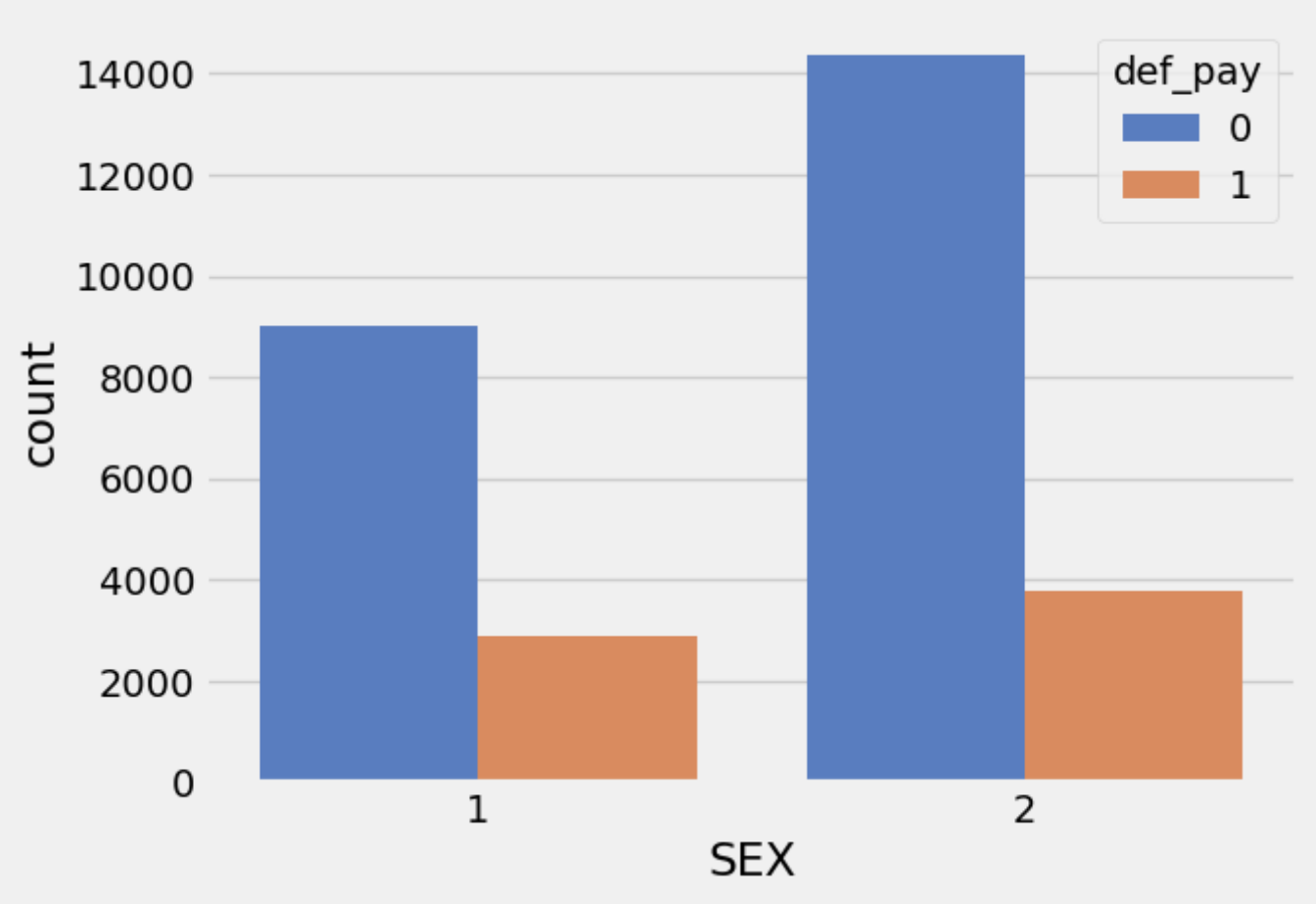
In [23]:
```
sns.countplot(x='SEX',data=dataset,hue = 'SEX')
```

Out[23]:
```
<AxesSubplot: xlabel='SEX', ylabel='count'>
```



In [24]:
```
# Checking the number of counts of defaulters and non defaulters sexwise
sns.countplot(x='SEX', data=dataset,hue="def_pay", palette="muted")
```

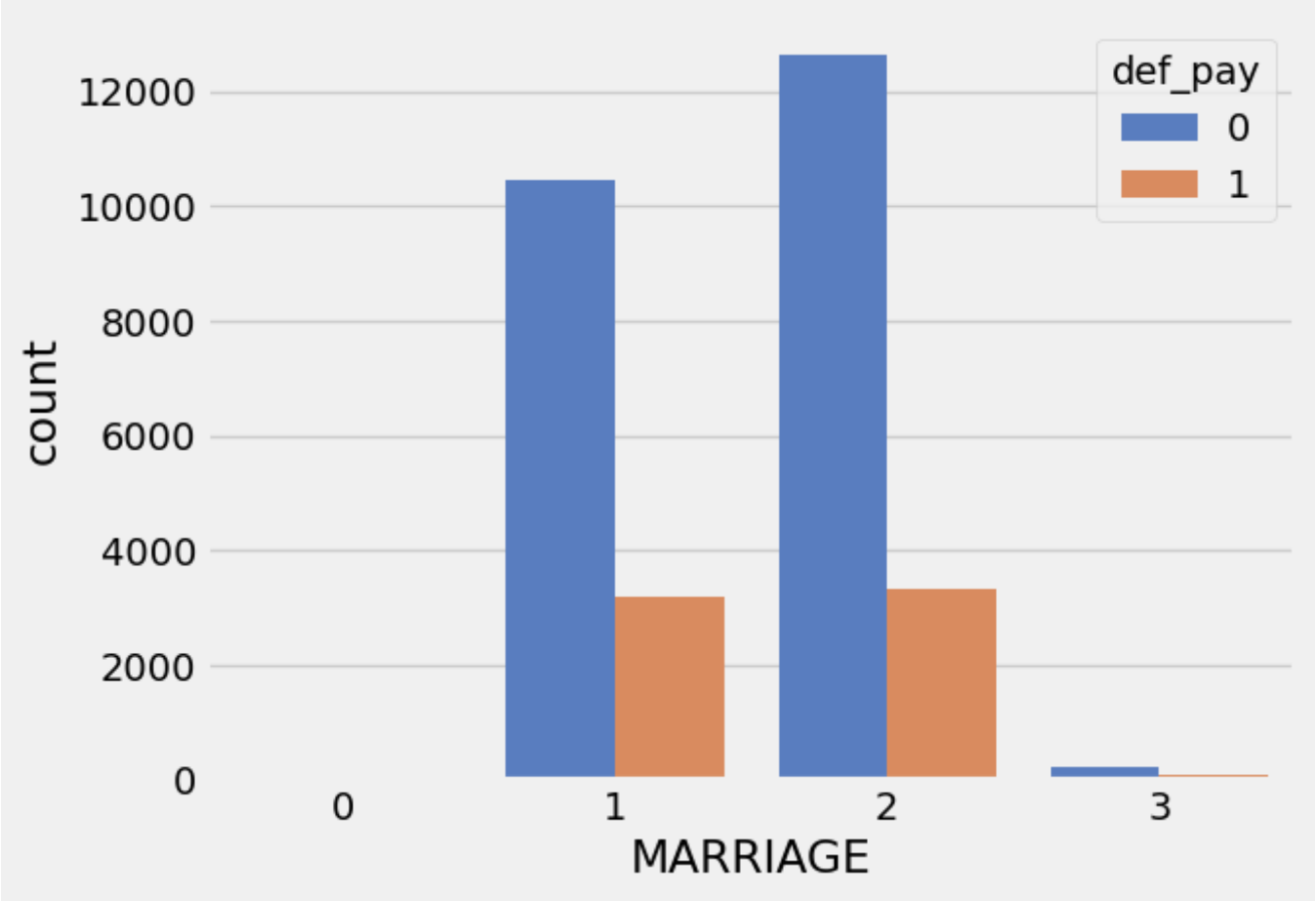
Out[24]:
```
<AxesSubplot: xlabel='SEX', ylabel='count'>
```



It is evident from the above output that females have overall less default payments wrt males

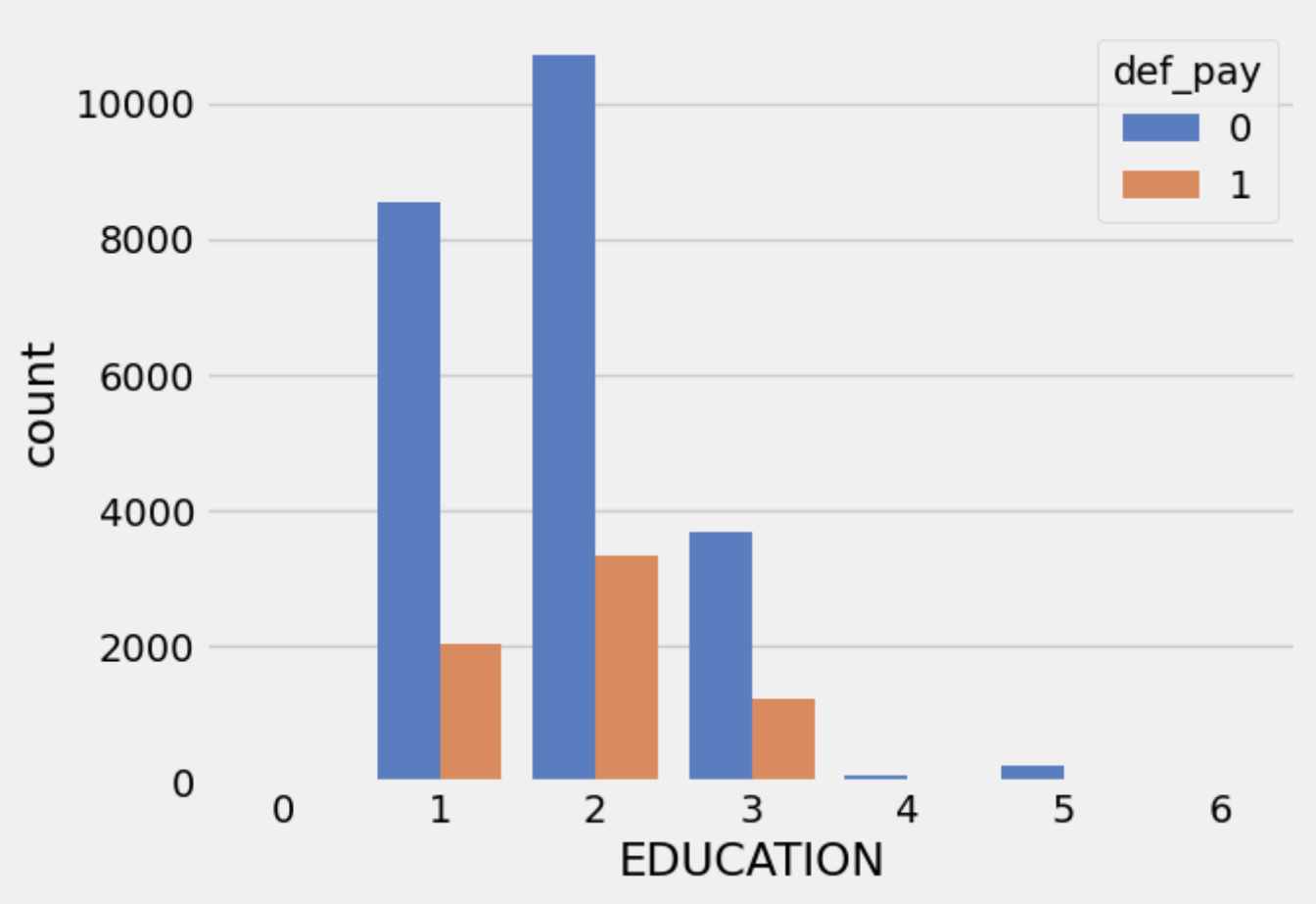
Non-Defaults have a higher proportion of Females (Sex=2)

In [25]:
```
g=sns.countplot(x="MARRIAGE", data=dataset,hue="def_pay", palette="muted")
```



From the above plot it is clear that those people who have marital status single have less default payment wrt married status people

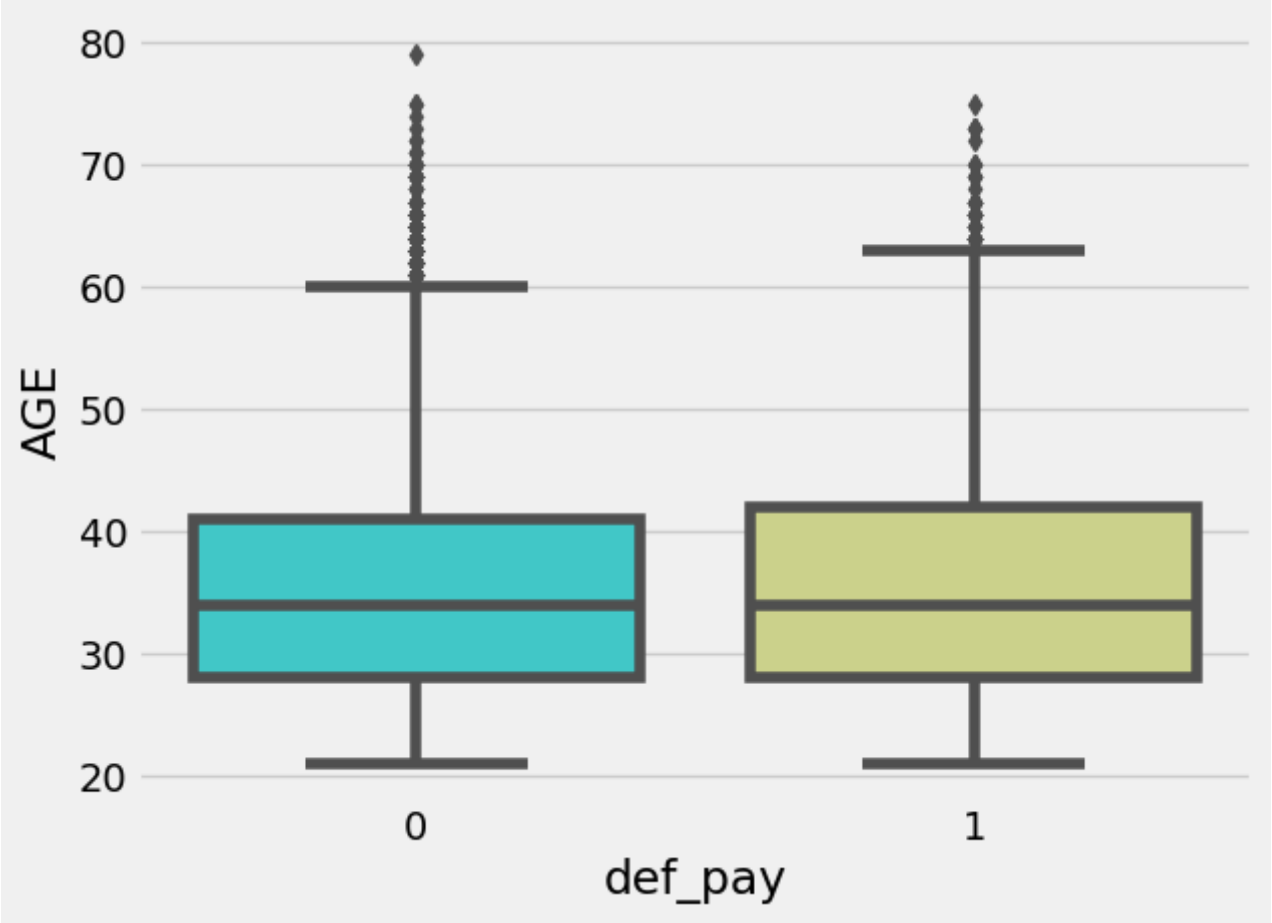
In [26]:
```
g=sns.countplot(x="EDUCATION", data=dataset,hue="def_pay", palette="muted")
```



From the above plot it is clear that those people who are university students have less default payment wrt graduates and high school people

In [27]:
```
sns.boxplot(x='def_pay',y='AGE',data=dataset,palette='rainbow')
```

Out[27]:
```
<AxesSubplot: xlabel='def_pay', ylabel='AGE'>
```

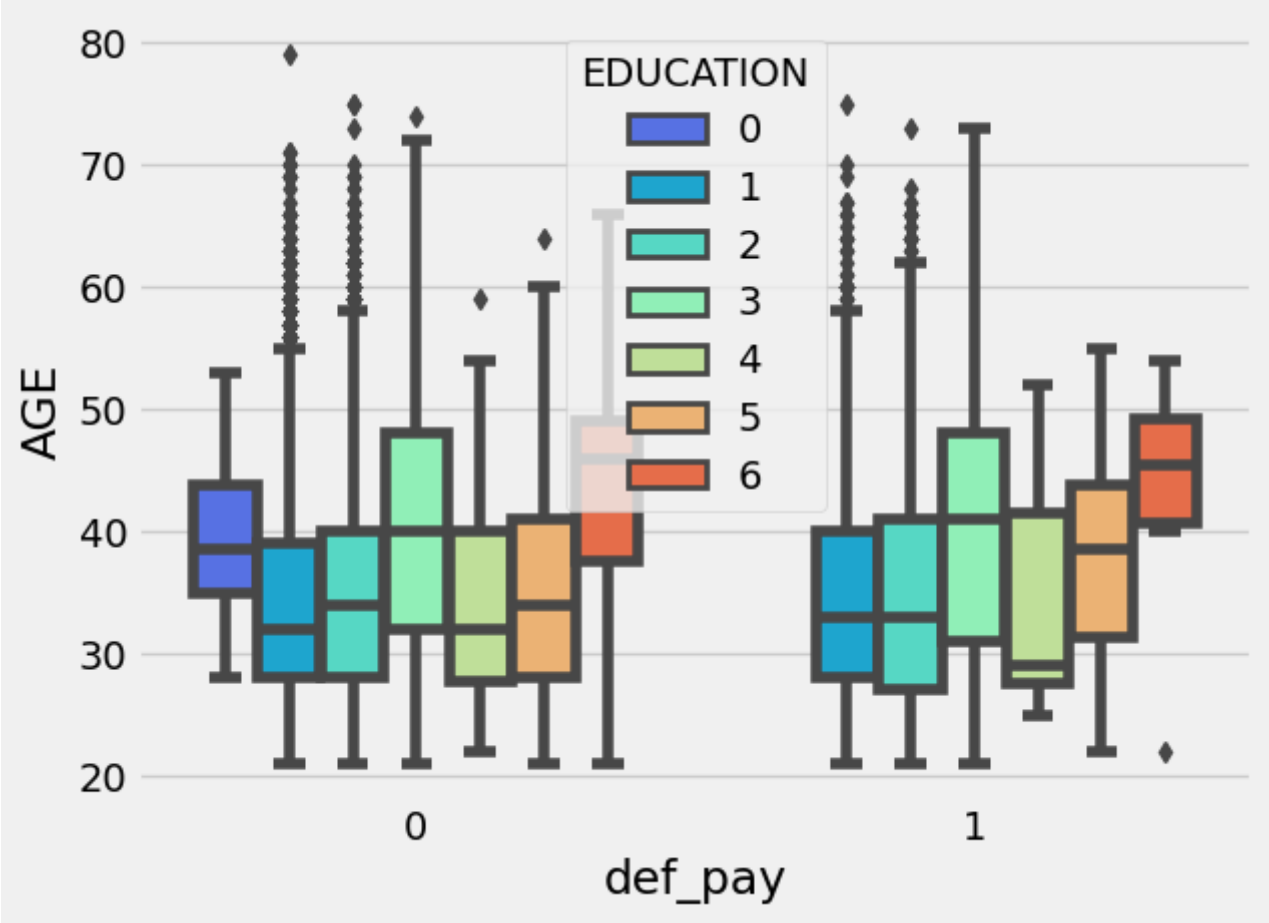


In [28]:
```
sns.boxplot(x='def_pay',hue='MARRIAGE', y='AGE',data=dataset,palette="rainbow")
```

Out[28]: <AxesSubplot: xlabel='def_pay', ylabel='AGE'>

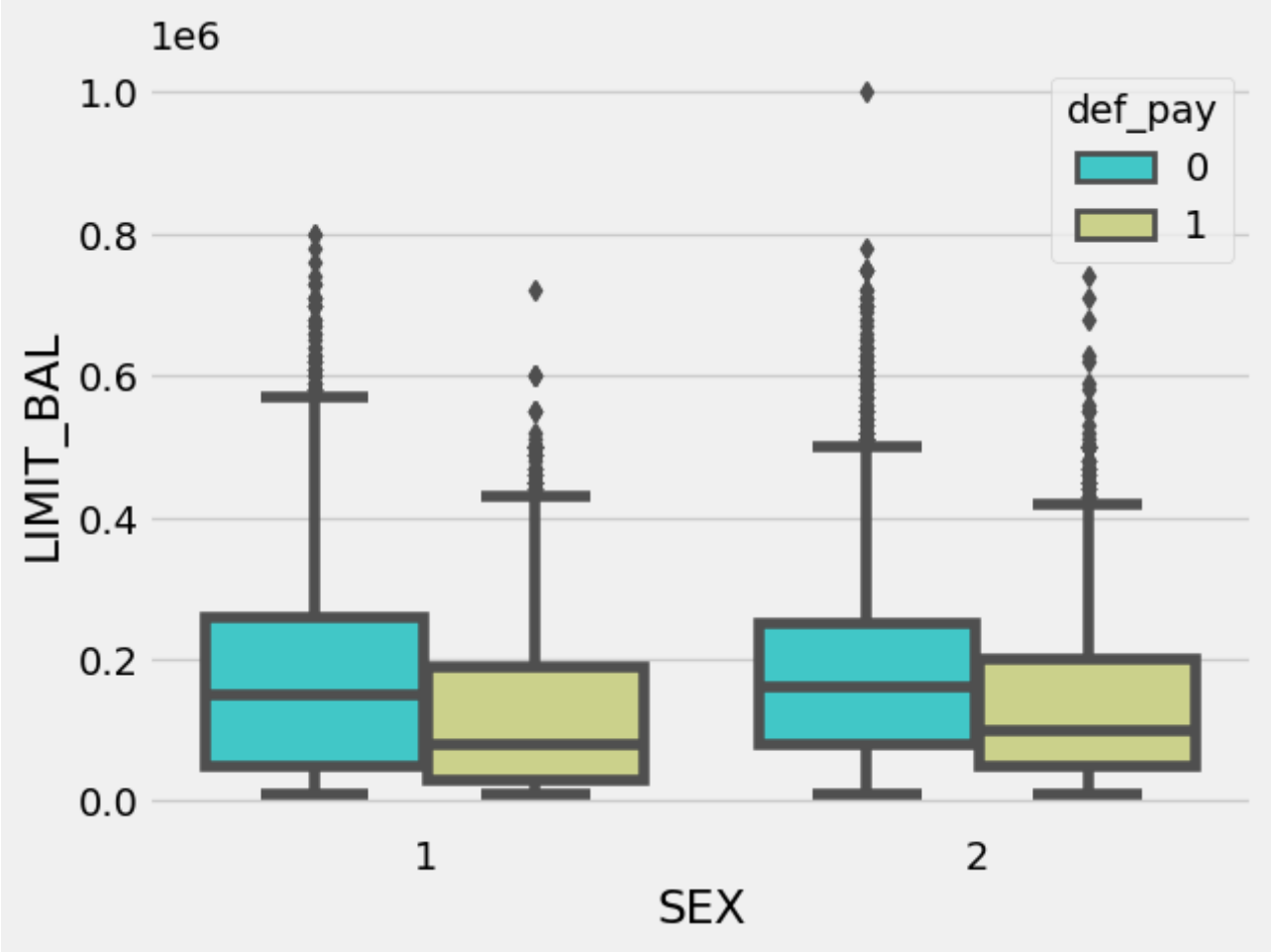
In [29]:
```
sns.boxplot(x='def_pay',hue='EDUCATION', y='AGE',data=dataset,palette="rainbow" )
```

Out[29]:
```
<AxesSubplot: xlabel='def_pay', ylabel='AGE'>
```



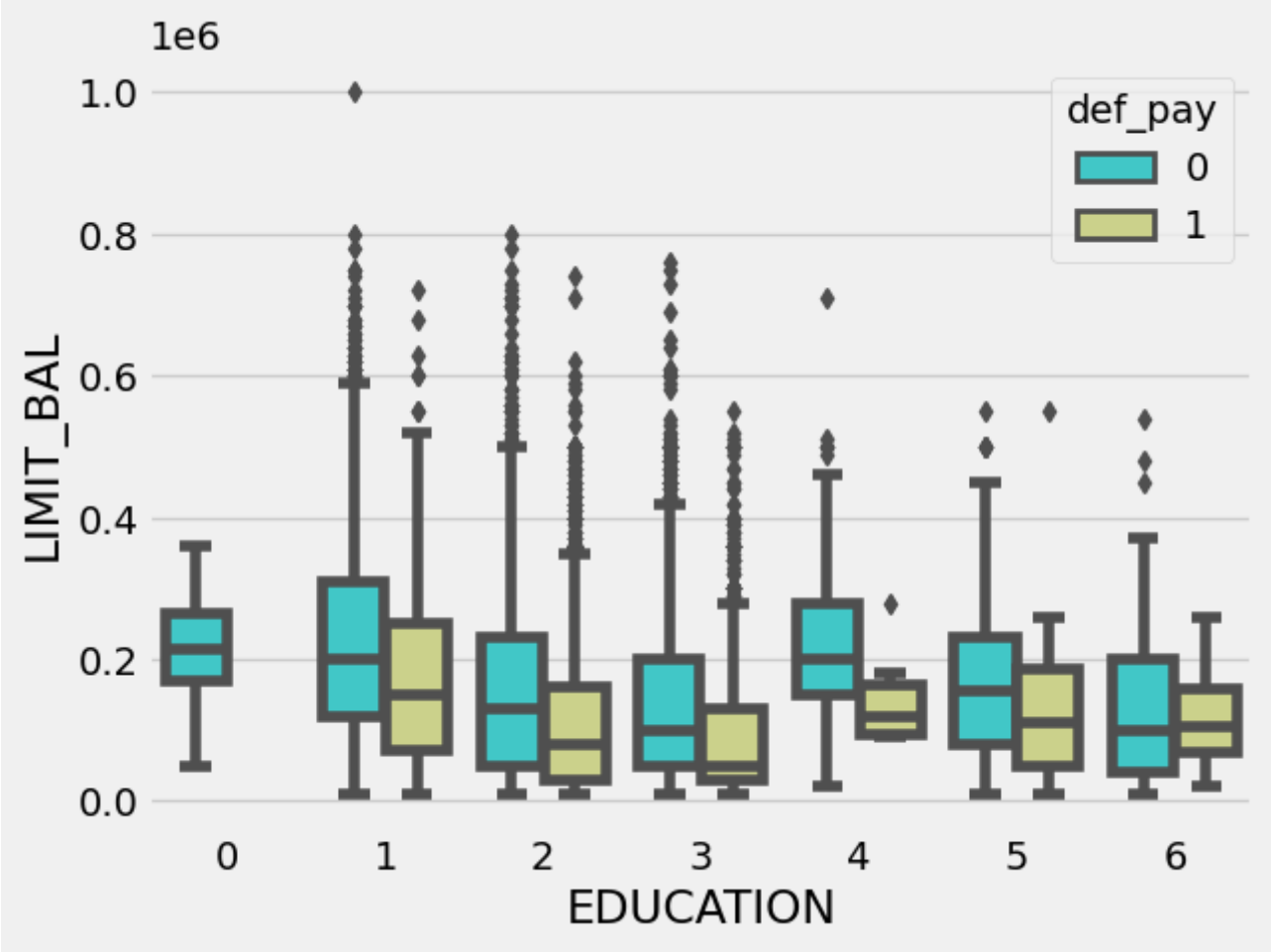
In [30]:
```
sns.boxplot(x='SEX',hue='def_pay', y='LIMIT_BAL',data=dataset,palette="rainbow")
```

Out[30]:
```
<AxesSubplot: xlabel='SEX', ylabel='LIMIT_BAL'>
```



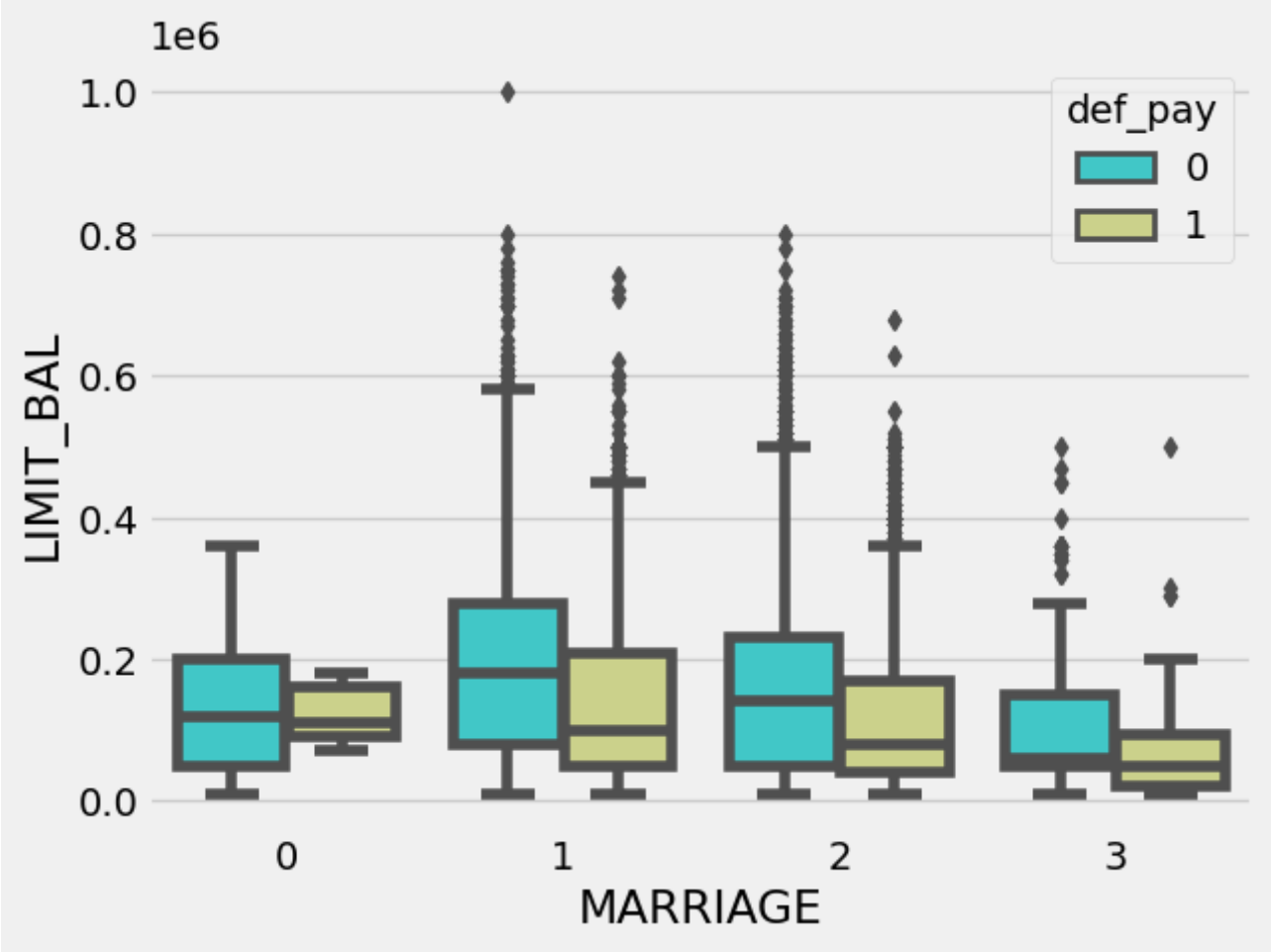
In [31]:
```
sns.boxplot(x='EDUCATION',hue='def_pay', y='LIMIT_BAL',data=dataset,palette="rainbow")
```

Out[31]:
```
<AxesSubplot: xlabel='EDUCATION', ylabel='LIMIT_BAL'>
```



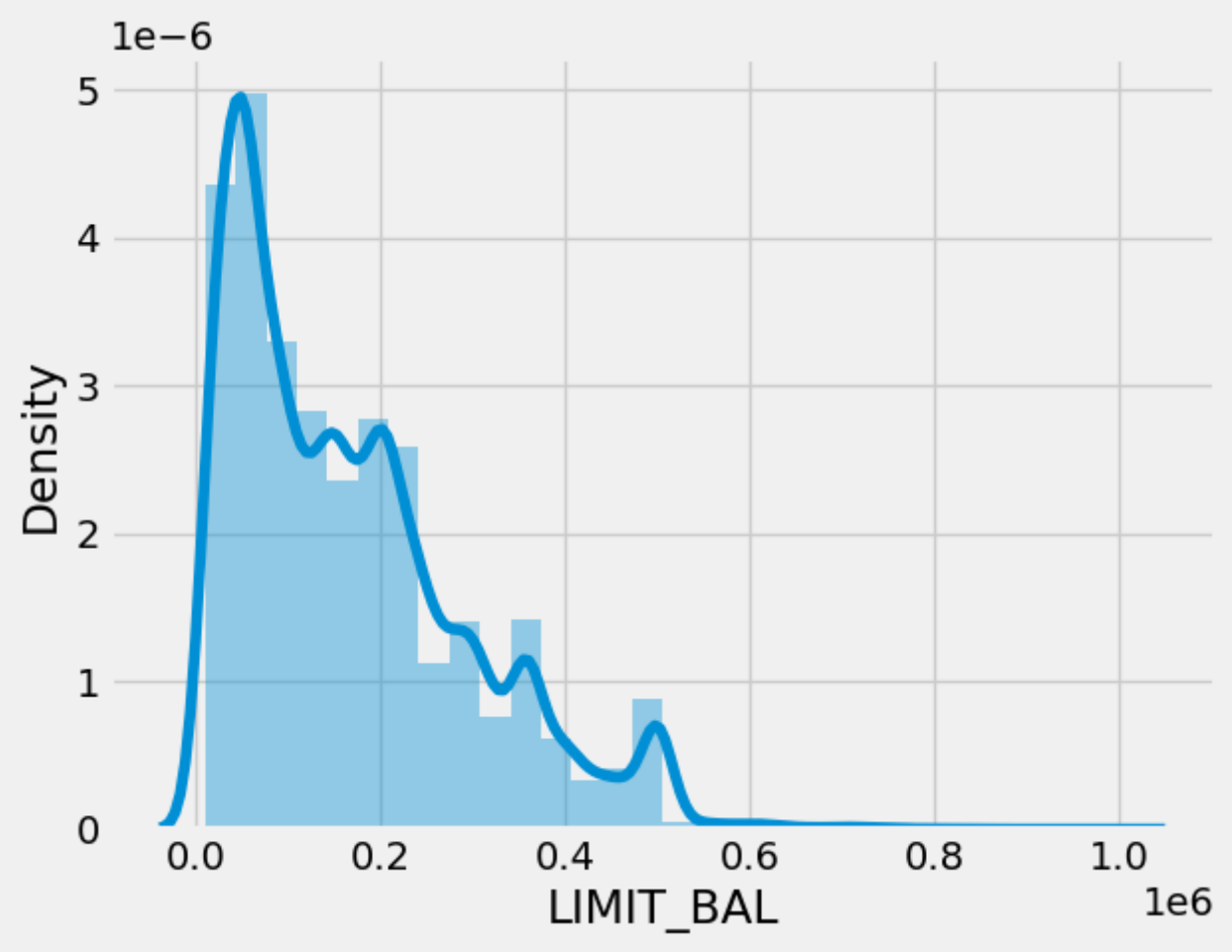
In [32]:
```
sns.boxplot(x='MARRIAGE',hue='def_pay', y='LIMIT_BAL',data=dataset,palette="rainbow")
```

Out[32]:
```
<AxesSubplot: xlabel='MARRIAGE', ylabel='LIMIT_BAL'>
```



In [33]:
```
sns.distplot(dataset['LIMIT_BAL'],kde=True,bins=30)
```

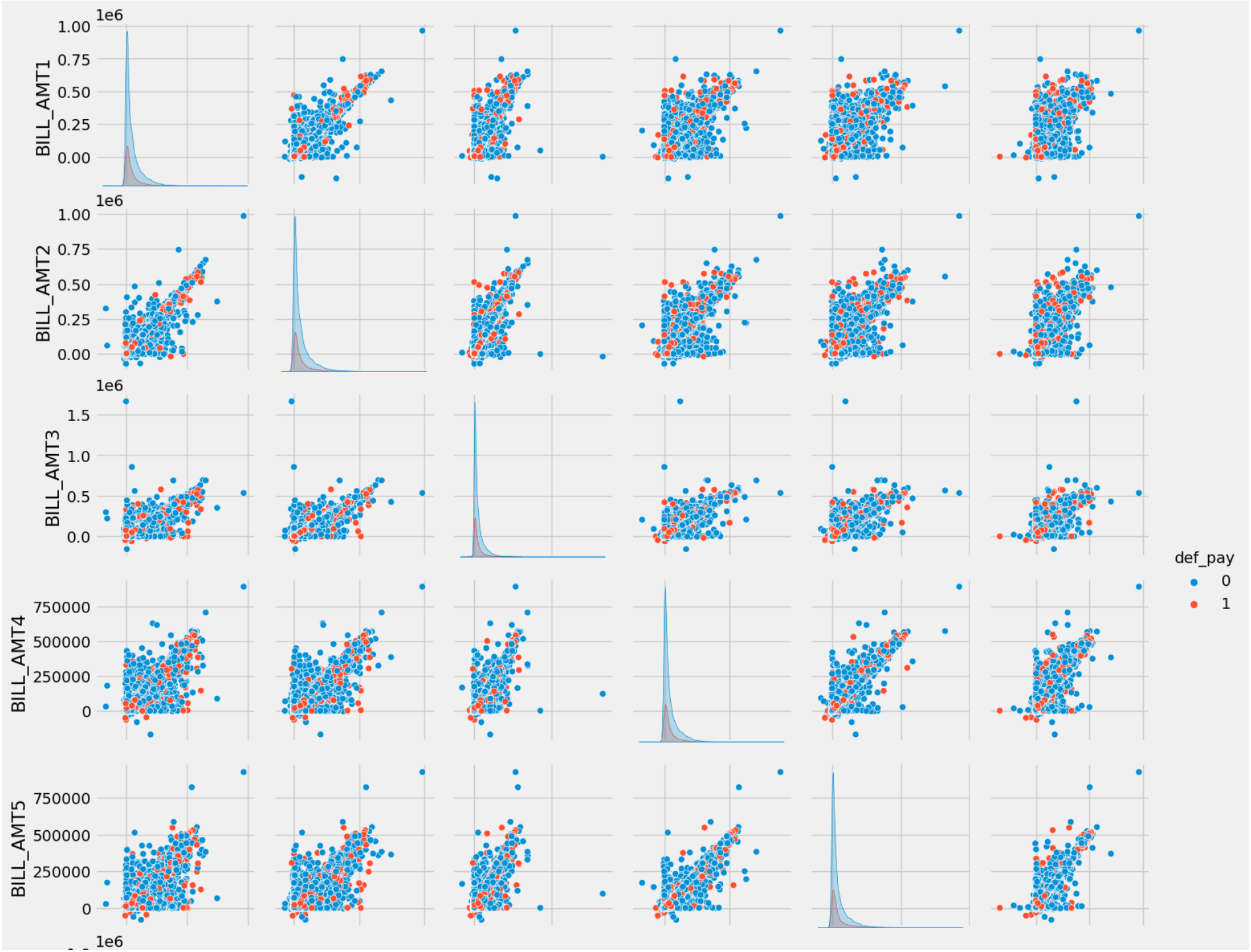
Out[33]:
```
<AxesSubplot: xlabel='LIMIT_BAL', ylabel='Density'>
```

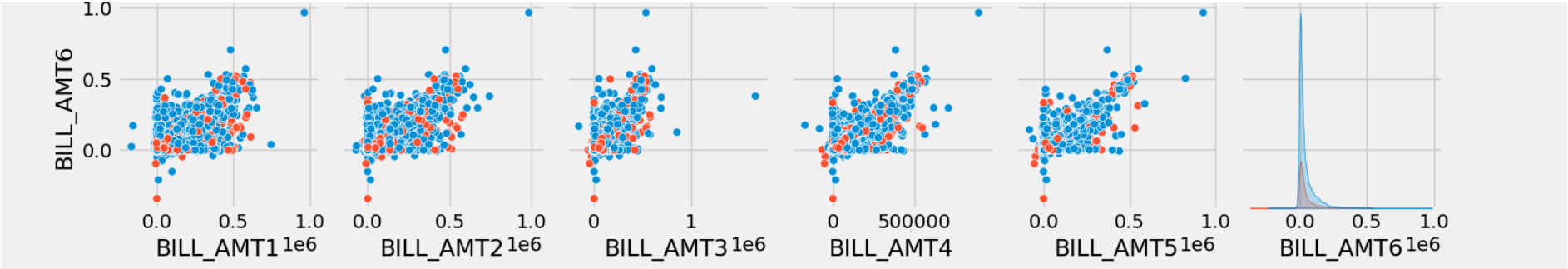


In [34]:
```
# plot columns with similar names to check the correlation

sns.pairplot(dataset, vars=dataset.columns[11:17], kind='scatter',hue= 'def_pay')
sns.pairplot(dataset, vars=dataset.columns[17:23],hue = 'def_pay')
```

Out[34]: <seaborn.axisgrid.PairGrid at 0x236e5e2fd60>


1e6
1.00
0.75
0.50
0.25
0.00
BILL_AMT1
BILL_AMT2
1.5
1.0
0.5
0.0
BILL_AMT3
750000
500000
250000
0
BILL_AMT4
BILL_AMT5
def_pay
0
1


BILL_AMT6
1.0
0.5
0.0
0.0
0.5
1.0
BILL_AMT1
1e6
0.0
0.5
1.0
BILL_AMT2
1e6
0
1
BILL_AMT3
1e6
0
500000
BILL_AMT4
0.0
0.5
1.0
BILL_AMT5
1e6
0.0
0.5
1.0
BILL_AMT6
1e6

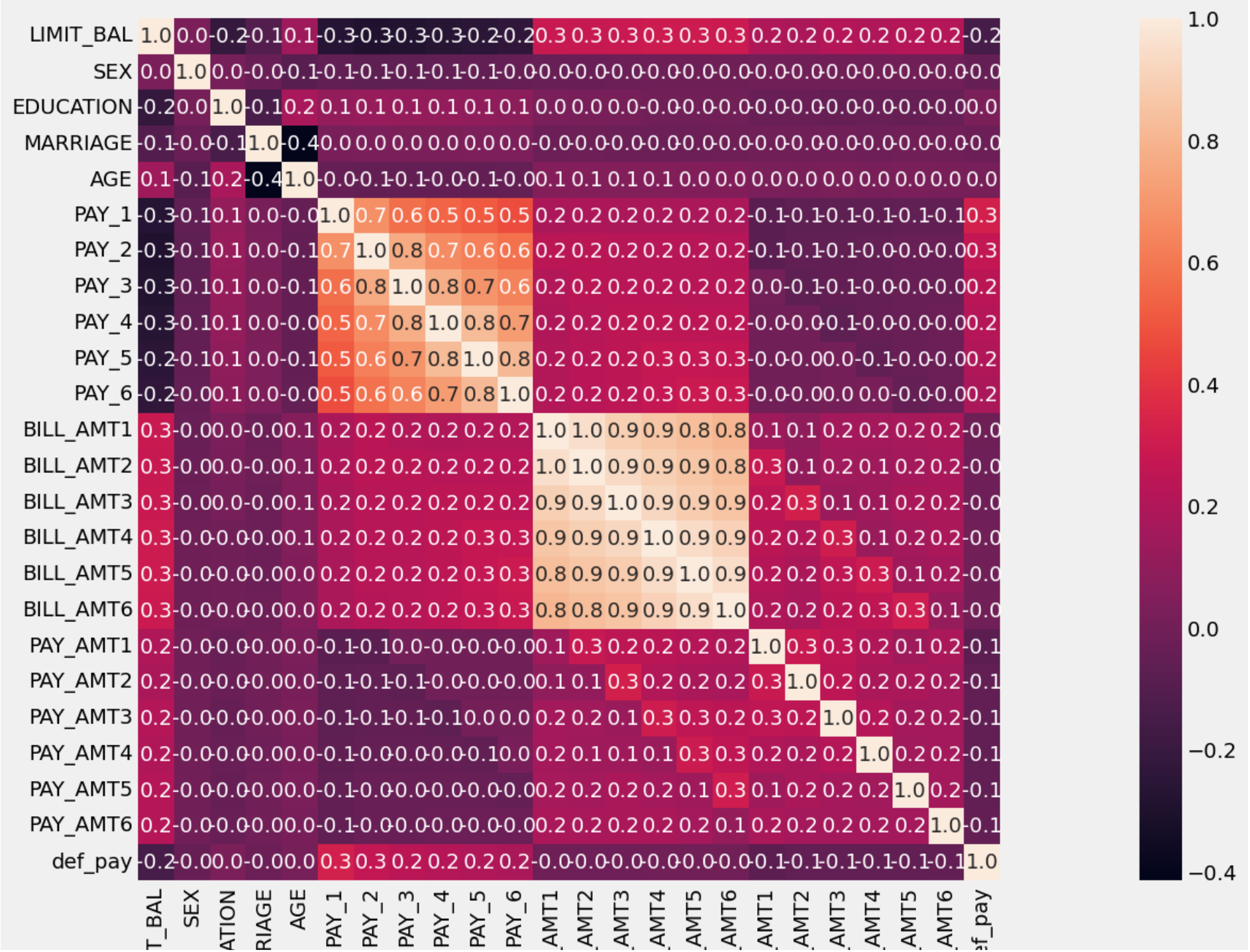

PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
1e6
def_pay
0
1

Step3: Observing Correlation between features of the Dataset

In [35]:
```
correlation = dataset.corr()
```

In [36]:
```
plt.subplots(figsize=(30,10))
sns.heatmap( correlation, square=True, annot=True,fmt=".1f" )
```

Out[36]: <AxesSubplot: >


LIMIT_BAL
SEX
EDUCATION
MARRIAGE
AGE
PAY_1
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
def_pay
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Step4: Data Cleaning

EDUCATION has category 5 and 6 that are unlabelled, moreover the category 0 is undocumented.

MARRIAGE has a label 0 that is undocumented

Data Transformation

The 0 in MARRIAGE can be safely categorized as 'Other' (thus 3).

The 0 (undocumented), 5 and 6 (label unknown) in EDUCATION can also be put in a 'Other' cathegory (thus 4)

Thus is a good occasion to learn how to use the .loc function

In [37]:
```
fil = (dataset.EDUCATION == 5) | (dataset.EDUCATION == 6) | (dataset.EDUCATION == 0)
dataset.loc[fil, 'EDUCATION'] = 4
dataset.EDUCATION.value_counts()
```

Out[37]:
```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

In [38]:
```
dataset.loc[dataset.MARRIAGE == 0, 'MARRIAGE'] = 3
dataset.MARRIAGE.value_counts()
```

Out[38]:
```
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: int64
```

Step5: Hypothesis Testing

In [39]:
```
from scipy import stats
from scipy.stats import chi2_contingency
```

In [40]:
```
chi2_stat, p_value, dof, ex = stats.chi2_contingency(pd.crosstab(dataset.SEX,dataset.def_pay))
print("--chi2_contingency hypothesis test--")
print("\n")
print("Chi2 Stat :",chi2_stat)
print("Degrees of Freedom :",dof)
print("P-Value :",p_value)
print("Contingency Table :",ex)

if p_value < 0.05:
    print(f"p-value={p_value:.3f}, Null hypothesis is rejected")
else:
    print(f"p-value={p_value:.3f} failed to reject null hypothesis.")
```

```
--chi2_contingency hypothesis test--


Chi2 Stat : 47.70879689062111
Degrees of Freedom : 1
P-Value : 4.944678999412044e-12
Contingency Table : [[ 9258.3744  2629.6256]
 [14105.6256  4006.3744]]
p-value=0.000, Null hypothesis is rejected
```

In [41]:
```
chi2_stat, p_value, dof, ex = stats.chi2_contingency(pd.crosstab(dataset.EDUCATION,dataset.def_pay))
print("--chi2_contingency hypothesis test--")
print("\n")
print("Chi2 Stat :",chi2_stat)
print("Degrees of Freedom :",dof)
print("P-Value :",p_value)
print("Contingency Table :",ex)

if p_value < 0.05:
    print(f"p-value={p_value:.3f}, Null hypothesis is rejected")
else:
    print(f"p-value={p_value:.3f} failed to reject null hypothesis.")
```

```
--chi2_contingency hypothesis test--


Chi2 Stat : 160.40995107224546
Degrees of Freedom : 3
P-Value : 1.4950645648106153e-34
Contingency Table : [[ 8243.598   2341.402 ]
```

```
 [10926.564   3103.436 ]
 [ 3829.3596  1087.6404]
 [  364.4784   103.5216]]
p-value=0.000, Null hypothesis is rejected
```

In [42]:

```
chi2_stat, p_value, dof, ex = stats.chi2_contingency(pd.crosstab(dataset.MARRIAGE,dataset.def_pay))
print("--chi2_contingency hypothesis test--")
print("\n")
print("Chi2 Stat :",chi2_stat)
print("Degrees of Freedom :",dof)
print("P-Value :",p_value)
print("Contingency Table :",ex)

if p_value < 0.05:
    print(f"p-value={p_value:.3f}, Null hypothesis is rejected")
else:
    print(f"p-value={p_value:.3f} failed to reject null hypothesis.")
```

```
--chi2_contingency hypothesis test--


Chi2 Stat : 28.13032464482199
Degrees of Freedom : 2
P-Value : 7.790720364202813e-07
Contingency Table : [[10637.6292  3021.3708]
 [12432.7632  3531.2368]
 [  293.6076    83.3924]]
p-value=0.000, Null hypothesis is rejected
```

Here in pay_1,pay_2,pay_3,pay_4,pay_5,pay_6 columns there are -1 and -2 value which does not represemnt any status so we change their status as 0 means pay on time .

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)

In [43]:
```
fil = (dataset.PAY_1 == -1) | (dataset.PAY_1==-2)
dataset.loc[fil,'PAY_1']=0
dataset.PAY_1.value_counts()
fil = (dataset.PAY_2 == -1) | (dataset.PAY_2==-2)
dataset.loc[fil,'PAY_2']=0
dataset.PAY_2.value_counts()
fil = (dataset.PAY_3 == -1) | (dataset.PAY_3==-2)
dataset.loc[fil,'PAY_3']=0
dataset.PAY_3.value_counts()
fil = (dataset.PAY_4 == -1) | (dataset.PAY_4==-2)
dataset.loc[fil,'PAY_4']=0
dataset.PAY_4.value_counts()
fil = (dataset.PAY_5 == -1) | (dataset.PAY_5==-2)
dataset.loc[fil,'PAY_5']=0
dataset.PAY_5.value_counts()
fil = (dataset.PAY_6 == -1) | (dataset.PAY_6==-2)
dataset.loc[fil,'PAY_6']=0
dataset.PAY_6.value_counts()
```

Out[43]:
```
0    26921
2     2766
3      184
4       49
7       46
6       19
5       13
8        2
Name: PAY_6, dtype: int64
```

In [44]:
```
dataset.head()
```

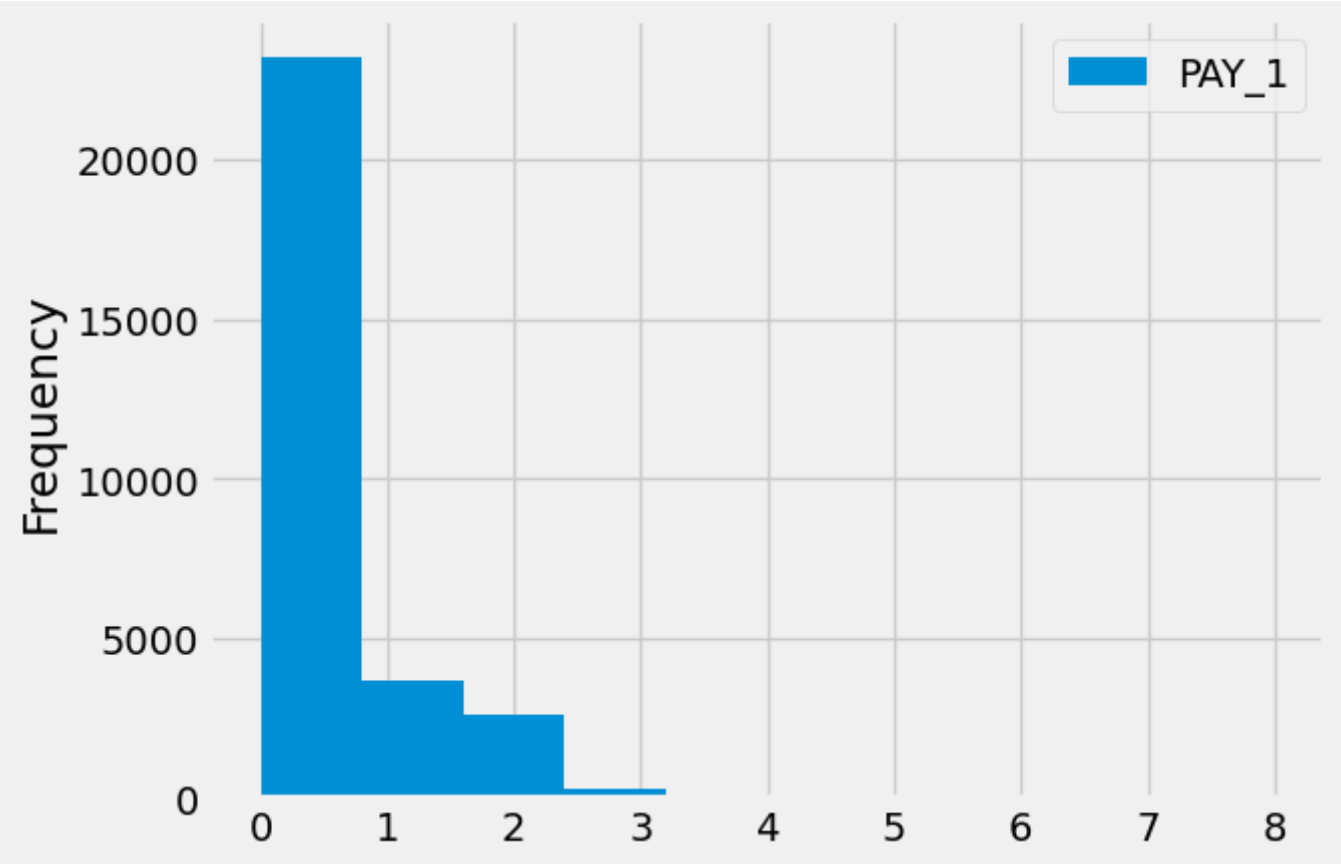
Out[44]:

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
1	20000.0	2	2	1	24	2	2	0	0	0	...	0.0	0.0	0.0	0.0	689.0
2	120000.0	2	2	2	26	0	2	0	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0
3	90000.0	2	2	2	34	0	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2
4	50000.0	2	2	1	37	0	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0
5	50000.0	1	2	1	57	0	0	0	0	0	...	20940.0	19146.0	19131.0	2000.0	36681.0

5 rows × 24 columns

In [45]:
```
dataset.plot(y = 'PAY_1',kind='hist')
plt.legend()
plt.show()
```



In [46]:
```
dataset['PAY_1'].describe()
```

Out[46]:
```
count    30000.000000
mean         0.356767
std          0.760594
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          8.000000
Name: PAY_1, dtype: float64
```

In [47]:
```
dataset.SEX.nunique()
```

Out[47]: 2

In [48]:
```
dataset[['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].describe()
```

Out[48]:

	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6
count	30000.000000	3.000000e+04	30000.00000	30000.000000	30000.000000	30000.000000
mean	5663.580500	5.921163e+03	5225.68150	4826.076867	4799.387633	5215.502567
std	16563.280354	2.304087e+04	17606.96147	15666.159744	15278.305679	17777.465775
min	0.000000	0.000000e+00	0.00000	0.000000	0.000000	0.000000
25%	1000.000000	8.330000e+02	390.00000	296.000000	252.500000	117.750000
50%	2100.000000	2.009000e+03	1800.00000	1500.000000	1500.000000	1500.000000
75%	5006.000000	5.000000e+03	4505.00000	4013.250000	4031.500000	4000.000000
max	873552.000000	1.684259e+06	896040.00000	621000.000000	426529.000000	528666.000000

In [49]:
```
dataset[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].describe()
```

Out[49]:

	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6
count	30000.000000	30000.000000	3.000000e+04	30000.000000	30000.000000	30000.000000
mean	51223.330900	49179.075167	4.701315e+04	43262.948967	40311.400967	38871.760400
std	73635.860576	71173.768783	6.934939e+04	64332.856134	60797.155770	59554.107537
min	-165580.000000	-69777.000000	-1.572640e+05	-170000.000000	-81334.000000	-339603.000000
25%	3558.750000	2984.750000	2.666250e+03	2326.750000	1763.000000	1256.000000
50%	22381.500000	21200.000000	2.008850e+04	19052.000000	18104.500000	17071.000000
75%	67091.000000	64006.250000	6.016475e+04	54506.000000	50190.500000	49198.250000
max	964511.000000	983931.000000	1.664089e+06	891586.000000	927171.000000	961664.000000

In [50]:
```
dataset.columns
```

Out[50]:
```
Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1', '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', 'def_pay'],
      dtype='object')
```

Step6: Spliting data into Categorical columns and Numerical columns

In [51]:
```
categorycols=dataset[['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6']]
```

In [52]:
```
categorycols.head()
```

Out[52]:

ID	SEX	EDUCATION	MARRIAGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
1	2	2	1	2	2	0	0	0	0
2	2	2	2	0	2	0	0	0	2
3	2	2	2	0	0	0	0	0	0

ID	SEX	EDUCATION	MARRIAGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
4	2	2	1	0	0	0	0	0	0
5	1	2	1	0	0	0	0	0	0

In [53]:
```
dataset=dataset.drop(['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6'],axis=1)
```

In [54]:
```
dataset.columns
```

Out[54]:
```
Index(['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'def_pay'],
      dtype='object')
```

In [55]:
```
numericcols=dataset[['LIMIT_BAL', 'AGE', '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 [56]:
```
dataset=dataset.drop(['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6'],axis=1)
```

Step7 : Label encoding for categorical variable

In [57]:
```
# Lable Encoding or Dummy Encoding objectcols
from sklearn.preprocessing import LabelEncoder
```

In [58]:
```
le=LabelEncoder()
```

In [59]:
```
categorycols_encode=categorycols.apply(le.fit_transform)
```

In [60]:
```
categorycols_encode.head()
```

Out[60]:

ID	SEX	EDUCATION	MARRIAGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
1	1	1	0	2	2	0	0	0	0
2	1	1	1	0	2	0	0	0	1
3	1	1	1	0	0	0	0	0	0
4	1	1	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0

In [61]:
```
categorycols_encode=pd.DataFrame(categorycols_encode,columns=['SEX','EDUCATION','MARRIAGE','PAY_1','PAY_2','PAY_3','PAY_4','PAY_5'
```

In [62]:
```
categorycols_encode=categorycols_encode.reset_index()
```

Step8: Feature Scaling of Numerical Attributes

In [63]:
```
from sklearn.preprocessing import StandardScaler
```

In [64]:
```
scaler=StandardScaler()
```

In [65]:
```
numericcols_scaled=scaler.fit_transform(numericcols)
```

In [66]:
```
numericcols_scaled=pd.DataFrame(numericcols_scaled,columns=['LIMIT_BAL', 'AGE', '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 [67]:
```
numericcols_scaled.head()
```

Out[67]:

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AM
0	-1.136720	-1.246020	-0.642501	-0.647399	-0.667993	-0.672497	-0.663059	-0.652724	-0.341942	-0.227086	-0.296801	-0.308063	-0.314
1	-0.365981	-1.029047	-0.659219	-0.666747	-0.639254	-0.621636	-0.606229	-0.597966	-0.341942	-0.213588	-0.240005	-0.244230	-0.314
2	-0.597202	-0.161156	-0.298560	-0.493899	-0.482408	-0.449730	-0.417188	-0.391630	-0.250292	-0.191887	-0.240005	-0.244230	-0.248
3	-0.905498	0.164303	-0.057491	-0.013293	0.032846	-0.232373	-0.186729	-0.156579	-0.221191	-0.169361	-0.228645	-0.237846	-0.244
4	-0.905498	2.334029	-0.578618	-0.611318	-0.161189	-0.346997	-0.348137	-0.331482	-0.221191	1.335034	0.271165	0.266434	-0.269

In [68]:
```
dataset.columns
```

Out[68]:
```
Index(['def_pay'], dtype='object')
```

In [69]:
```
dataset_df=pd.concat([numericcols_scaled,categorycols_encode],axis=1)
```

In [70]:
```
dataset_df.columns
```

Out[70]:
```
Index(['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'ID', 'SEX', 'EDUCATION',
       'MARRIAGE', 'PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6'],
      dtype='object')
```

In [71]:
```
dataset_df.head()
```

Out[71]:

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	...	ID	SEX	EDUCATION	MAR
0	-1.136720	-1.246020	-0.642501	-0.647399	-0.667993	-0.672497	-0.663059	-0.652724	-0.341942	-0.227086	...	1	1	1	
1	-0.365981	-1.029047	-0.659219	-0.666747	-0.639254	-0.621636	-0.606229	-0.597966	-0.341942	-0.213588	...	2	1	1	
2	-0.597202	-0.161156	-0.298560	-0.493899	-0.482408	-0.449730	-0.417188	-0.391630	-0.250292	-0.191887	...	3	1	1	
3	-0.905498	0.164303	-0.057491	-0.013293	0.032846	-0.232373	-0.186729	-0.156579	-0.221191	-0.169361	...	4	1	1	
4	-0.905498	2.334029	-0.578618	-0.611318	-0.161189	-0.346997	-0.348137	-0.331482	-0.221191	1.335034	...	5	0	1	

5 rows × 24 columns

In [72]:
```
X = dataset_df
y = dataset['def_pay']
```

Step9: Spiliting Dataset into training(70%) and test set(30%)

In [73]:
```
from sklearn.model_selection import train_test_split
```

In [74]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3,random_state = 1)
```

In [75]:
```
X_train.shape
```

Out[75]: (21000, 24)

In [76]:
```
X_test.shape
```

Out[76]: (9000, 24)

Step10: Applying Machine Learning Algorithm for Classification Problem

Logistic Regression

In Logistic Regression, we wish to model a dependent variable(Y) in terms of one or more independent variables(X). It is a method for classification. This algorithm is used for the dependent variable that is Categorical. Y is modeled using a function that gives output between 0 and 1 for all values of X. In Logistic Regression, the Sigmoid (aka Logistic) Function is used

In [77]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression(random_state=1)
logmodel.fit(X_train,y_train)
```

Out[77]: LogisticRegression(random_state=1)

In [78]:
```
y_pred = logmodel.predict(X_test)
```

In [79]:
```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

results = pd.DataFrame([['Logistic Regression', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results
```

Out[79]:

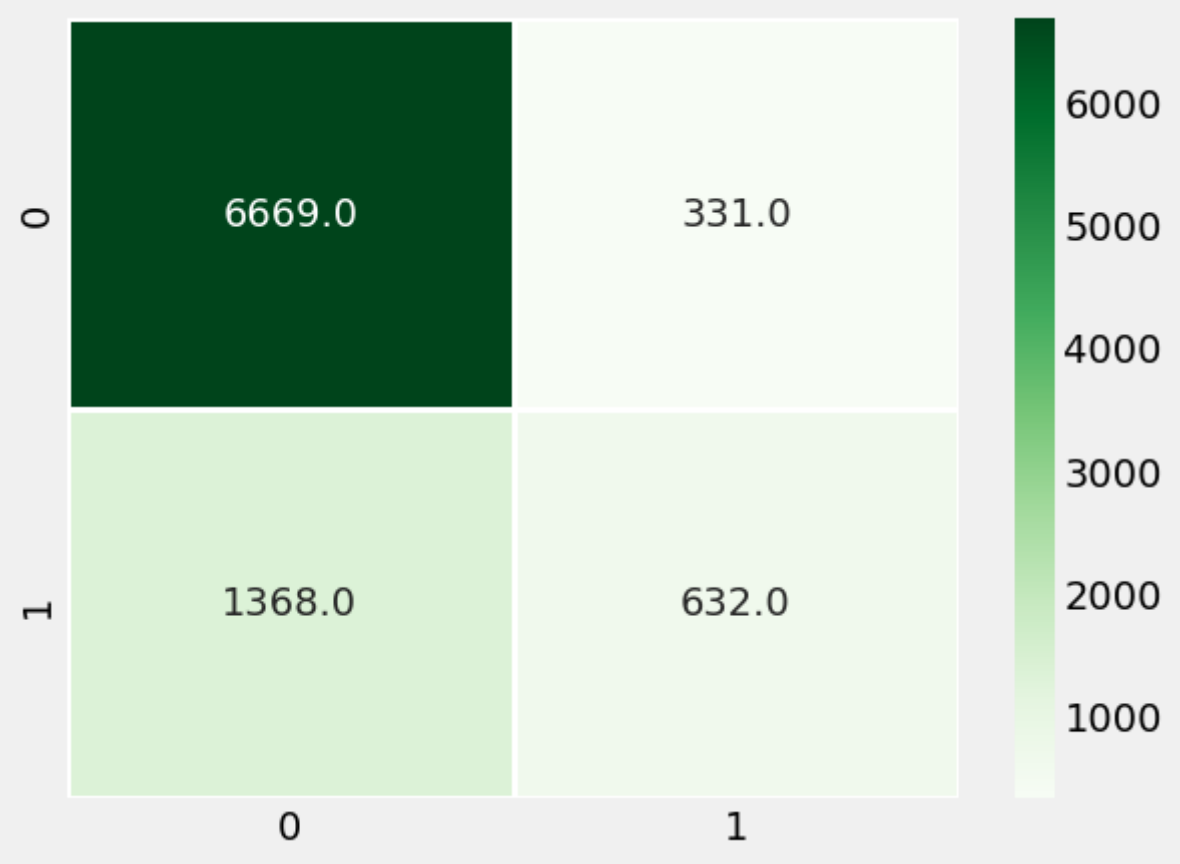
	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Logistic Regression	0.811222	0.656282	0.316	0.426595	0.634357

In [80]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[80]:
```
array([[6669,  331],
       [1368,  632]], dtype=int64)
```

In [81]:
```
sns.heatmap(cm,annot= True,linewidths=1,cmap="Greens",fmt=".1f")
```

Out[81]:
```
<AxesSubplot: >
```



Decision Tree Classification

The idea of a decision tree is to divide the data set into smaller data sets based on the descriptive features until you reach a small enough set that contains data points that fall under one label.

Advantages of Decision Trees Decision trees are easy to interpret. To build a decision tree requires little data preparation from the user- there is no need to normalize data

Disadvantages of Decision Trees Decision trees are likely to overfit noisy data. The probability of overfitting on noise increases as a tree gets deeper.

In [82]:
```
from sklearn.tree import DecisionTreeClassifier
dct = DecisionTreeClassifier(criterion = 'entropy',random_state = 0)
```

In [83]:
```
dct.fit(X_train,y_train)
```

Out[83]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

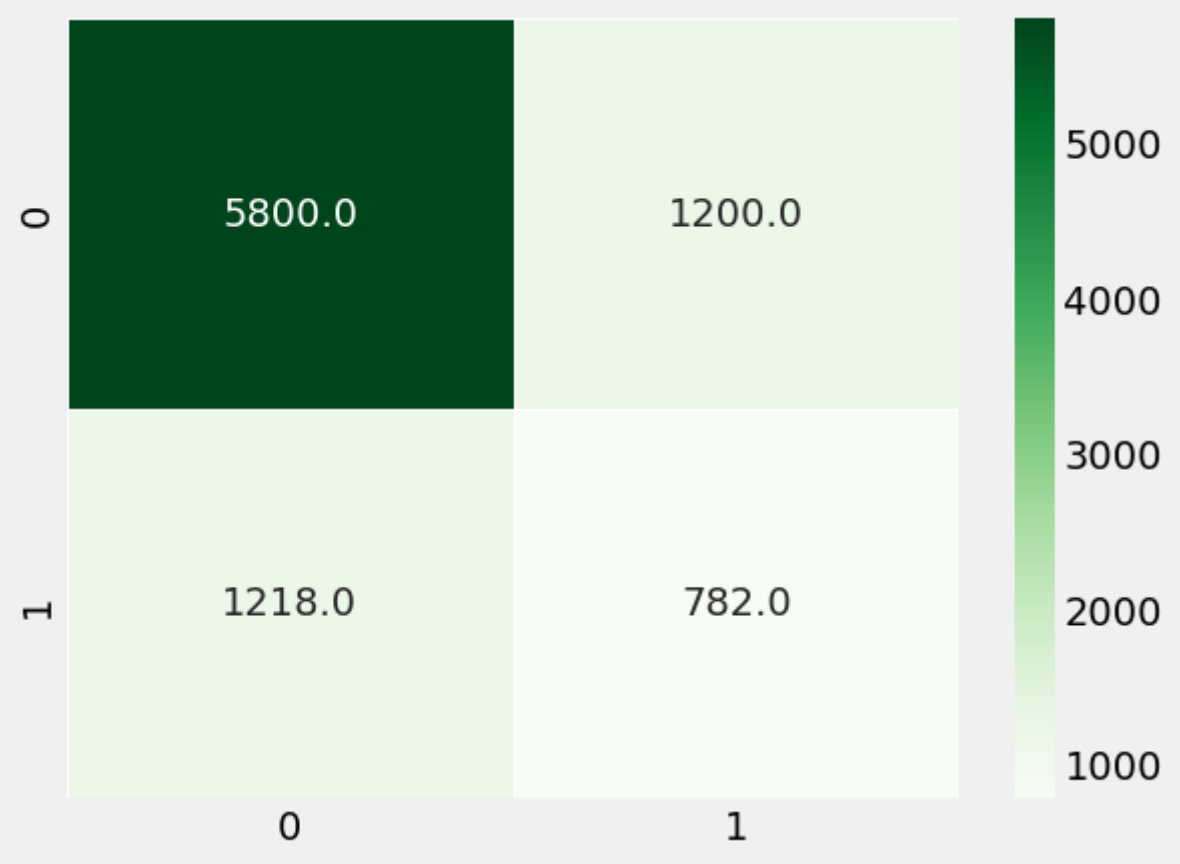
In [84]:
```
y_pred = dct.predict(X_test)
```

In [85]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[85]:
```
array([[5800, 1200],
       [1218,  782]], dtype=int64)
```

In [86]:
```
sns.heatmap(cm,annot= True,linewidths=0.5,cmap="Greens",fmt=".1f")
```

Out[86]:
```
<AxesSubplot: >
```



In [87]:

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model = pd.DataFrame([['Decision Tree Classifier', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
model
```

Out[87]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Decision Tree Classifier	0.731333	0.394551	0.391	0.392767	0.609786

Random Forest Classification

Random Forest is a supervised learning algorithm, it creates a forest and makes it somehow random. The "forest" it builds, is an ensemble of Decision Trees.

Step1:- Pick at random K data points from the training set

Step2:- Build the Decision tree associated to these K data points

Step3:- Choose the Number of trees(n) you want to build and repeat STEP1 and STEP2

Step4:- For a new data points make each one of your 'n' trees predict the category to which the data point belongs and assign the new data point to the category that wins the majority vote

In [88]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 100,criterion = 'entropy',random_state = 0)
```

In [89]:
```
rfc.fit(X_train,y_train)
```

Out[89]:
```
RandomForestClassifier(criterion='entropy', random_state=0)
```

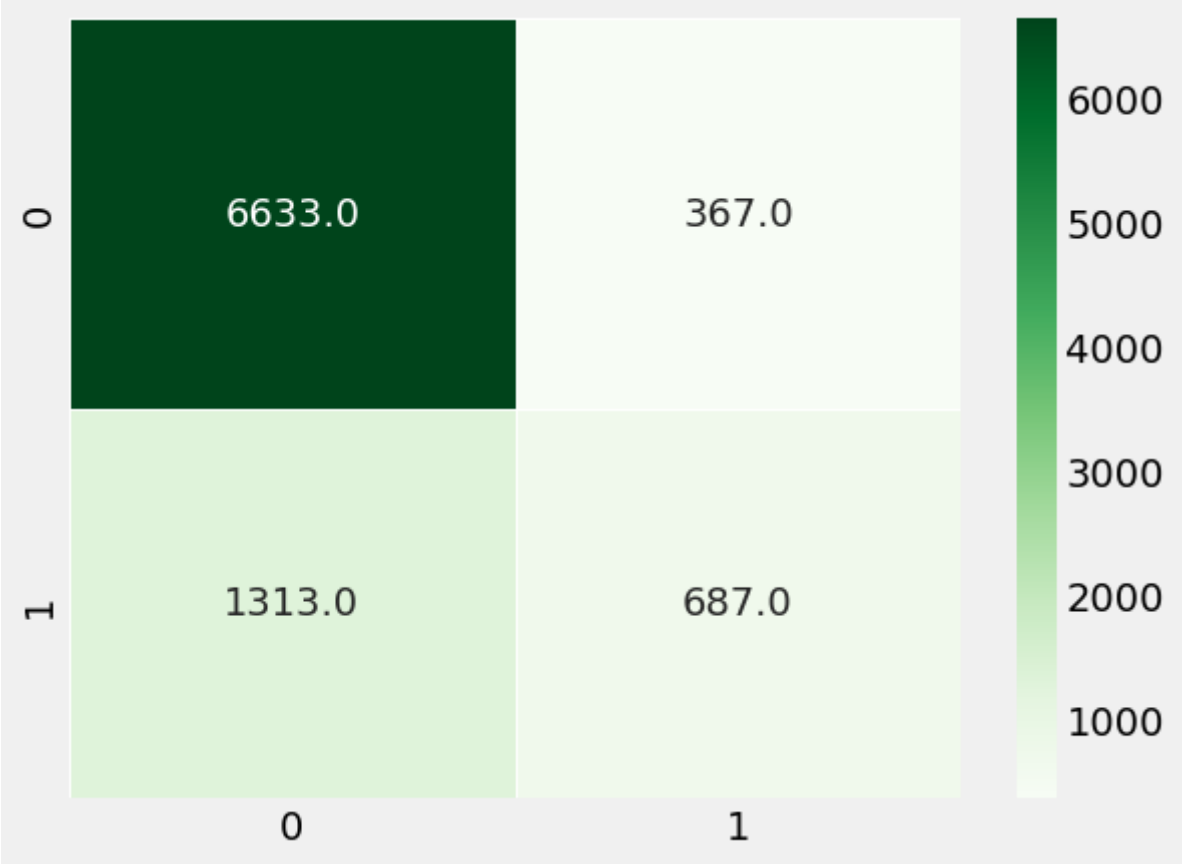
In [90]:
```
y_pred = rfc.predict(X_test)
```

In [91]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[91]:
```
array([[6633,  367],
       [1313,  687]], dtype=int64)
```

In [92]:
```
sns.heatmap(cm,annot= True,linewidths=0.5,cmap="Greens",fmt=".1f")
```

Out[92]:
```
<AxesSubplot: >
```



In [93]:

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

results = pd.DataFrame([['Random tree Classifier', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])

results
```

Out[93]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Random tree Classifier	0.813333	0.651803	0.3435	0.449902	0.645536

Support Vector Machine

Support Vector Machine SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems.The algorithm creates a line or a hyperplane which separates the data into classes using different kernel tricks like = 'linear','rbf' (gaussian).

Thus SVM tries to make a decision boundary in such a way that the separation between the two classes(that street) is as wide as possible.

In [94]:
```
# Fitting SVM to the Training set
from sklearn.svm import SVC
scv = SVC(kernel = 'rbf', random_state = 0)
scv.fit(X_train, y_train)
```

Out[94]:
```
SVC(random_state=0)
```

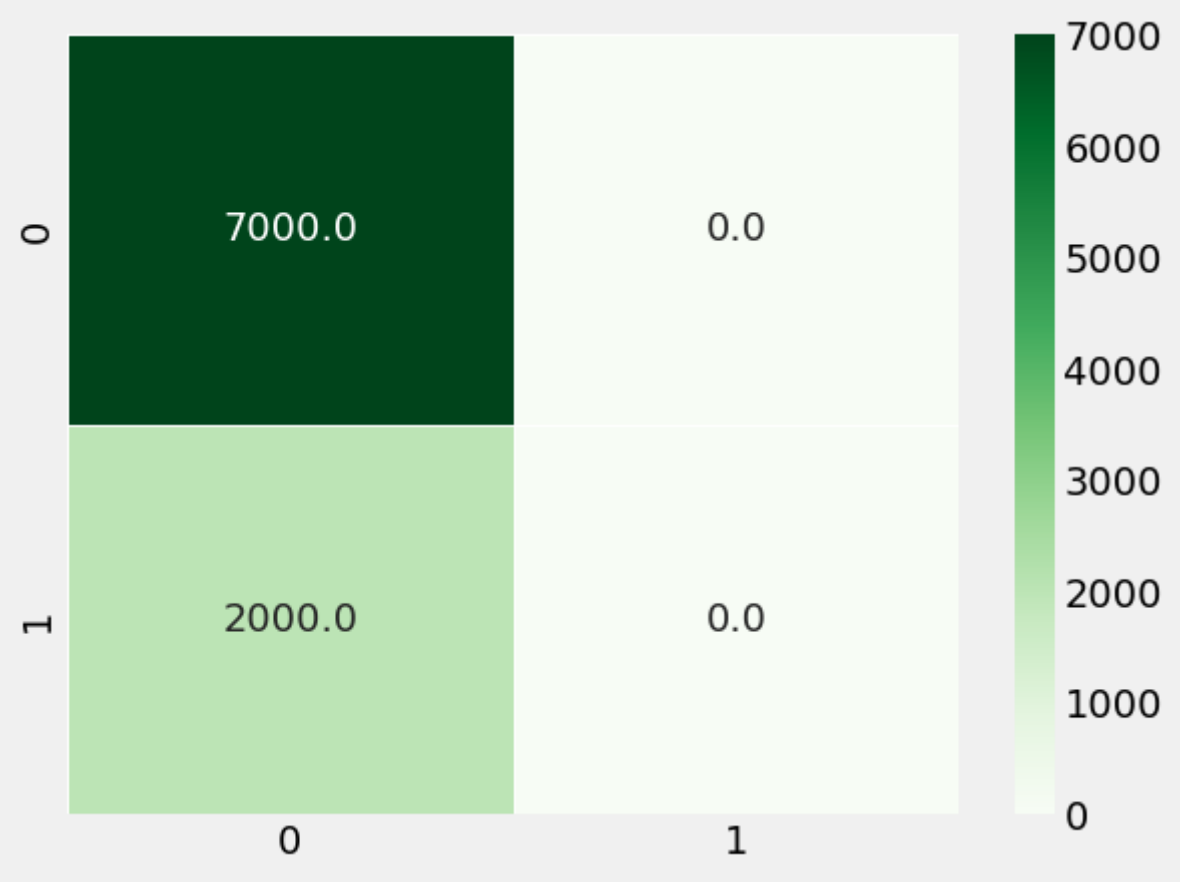
In [95]:
```
# Predicting the Test set results
y_pred = scv.predict(X_test)
```

In [96]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[96]:
```
array([[7000,    0],
       [2000,    0]], dtype=int64)
```

In [97]:
```
sns.heatmap(cm,annot= True,linewidths=0.5,cmap="Greens",fmt=".1f")
```

Out[97]:
```
<AxesSubplot: >
```



In [98]:

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model = pd.DataFrame([['Support Vector Machine', acc,prec,rec, f1,roc]],
                columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
model
```

Out[98]:

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Support Vector Machine	0.777778	0.0	0.0	0.0	0.5

Gaussian Naive Bayes

In [99]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train,y_train)
```

Out[99]:
```
GaussianNB()
```

In [100…
```
y_pred =naive_bayes.predict(X_test)
```

In [101…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[101…
```
array([[6069,  931],
       [ 972, 1028]], dtype=int64)
```

In [102…
```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model= pd.DataFrame([['Gaussian Naive Bayes', acc,prec,rec, f1,roc]],
                    columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
model
```

Out[102…

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Gaussian Naive Bayes	0.788556	0.524758	0.514	0.519323	0.6905

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.It is on of the Gradient Descent Algorithm.It uses only a single example (a batch size

of 1) per iteration. Given enough iterations, SGD works but is very noisy. The term "stochastic" indicates that the one example comprising each batch is chosen at random.

In [103…
```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss='log', penalty='l1', learning_rate='optimal',random_state=1)
sgd.fit(X_train, y_train)
```

Out[103…
```
SGDClassifier(loss='log', penalty='l1', random_state=1)
```

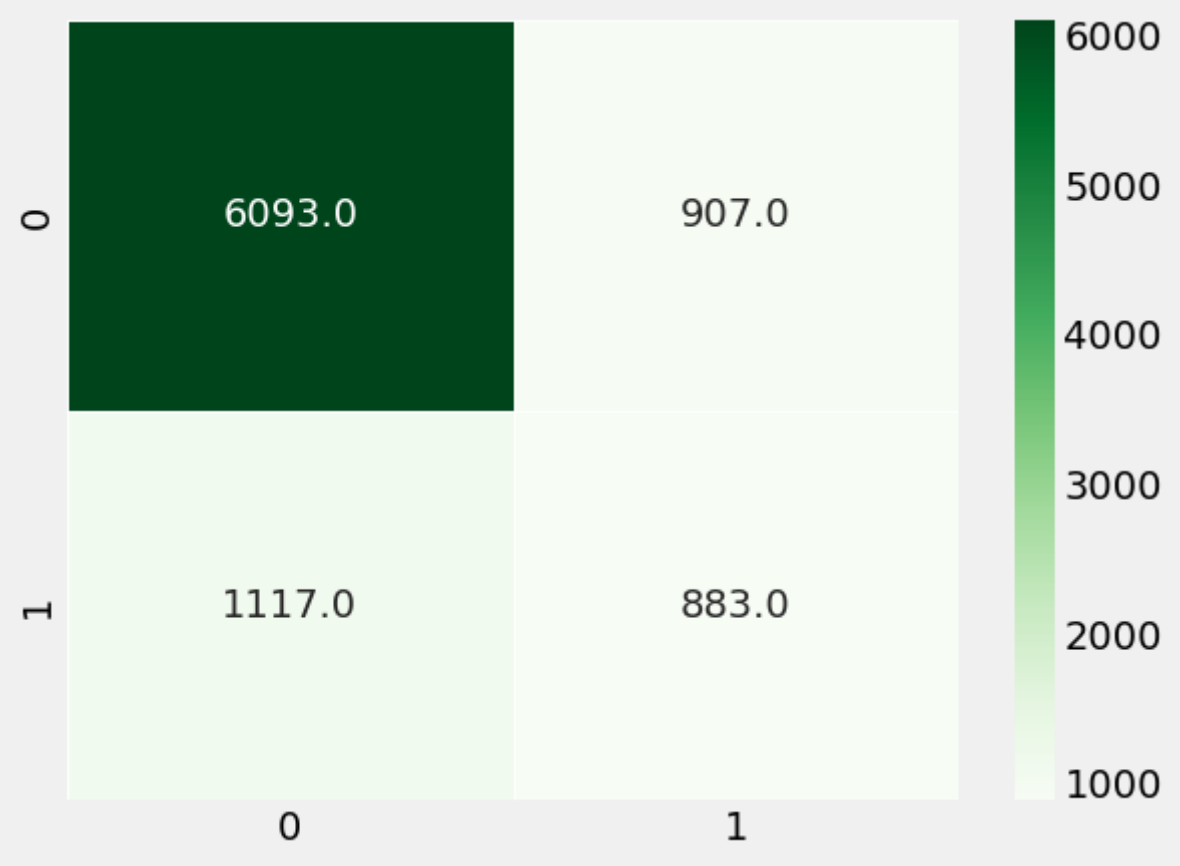
In [104…
```
y_pred = sgd.predict(X_test)
```

In [105…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[105…
```
array([[6093,  907],
       [1117,  883]], dtype=int64)
```

In [106…
```
sns.heatmap(cm,annot= True,linewidths=0.5,cmap="Greens",fmt=".1f")
```

Out[106…
```
<AxesSubplot: >
```



In [107…

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model = pd.DataFrame([['Stochastic Gradient Descent', acc,prec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
model
```

Out[107…

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	Stochastic Gradient Descent	0.775111	0.493296	0.4415	0.465963	0.655964

K-Nearest Neighbour

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

KNN focuses on easy implementation and good performance at the cost of computational time, but in our case the size of the dataset is considerably small so we can apply KNN.

We can implement a KNN model by following the below steps:

- Load the data
- Initialise the value of k
- For getting the predicted class, iterate from 1 to total number of training data points
- Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
- Sort the calculated distances in ascending order based on distance values
- Get top k rows from the sorted array
- Get the most frequent class of these rows
- Return the predicted class

Let's go ahead and use the elbow method to pick a good K Value!

Create a for loop that trains various KNN models with different k values, then keep track of the error_rate for each of these models with a list. Refer to the lecture if you are confused on this step.

In [108…

```
from sklearn.neighbors import KNeighborsClassifier
```

In [109…

```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i,n_jobs=-1)
    knn.fit(X_train,y_train)
```

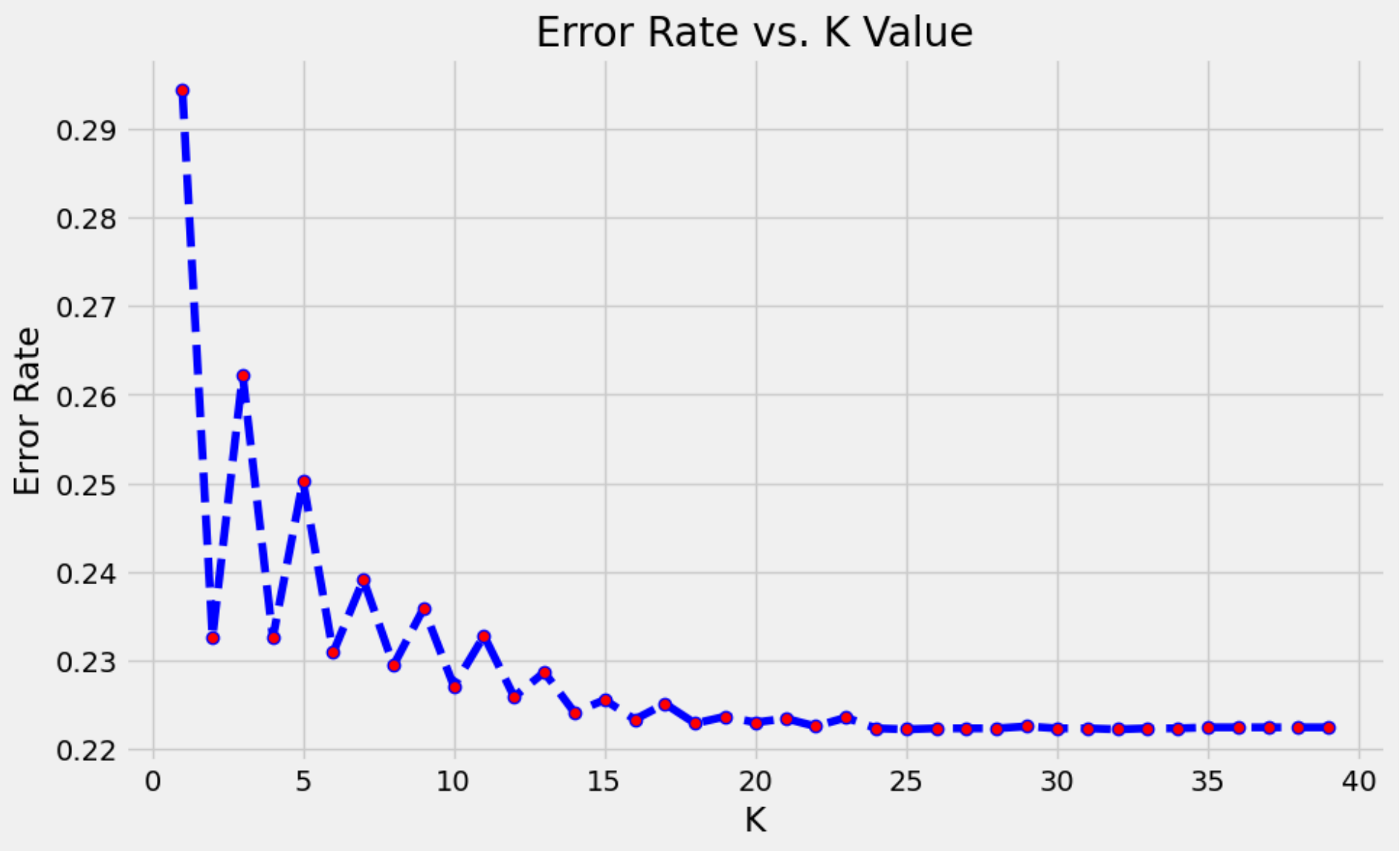
```
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [110…

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=6)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[110…

```
Text(0, 0.5, 'Error Rate')
```



In [111…

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
```

In [112…

```
knn.fit(X_train,y_train)
```

Out[112… KNeighborsClassifier(n_neighbors=7)

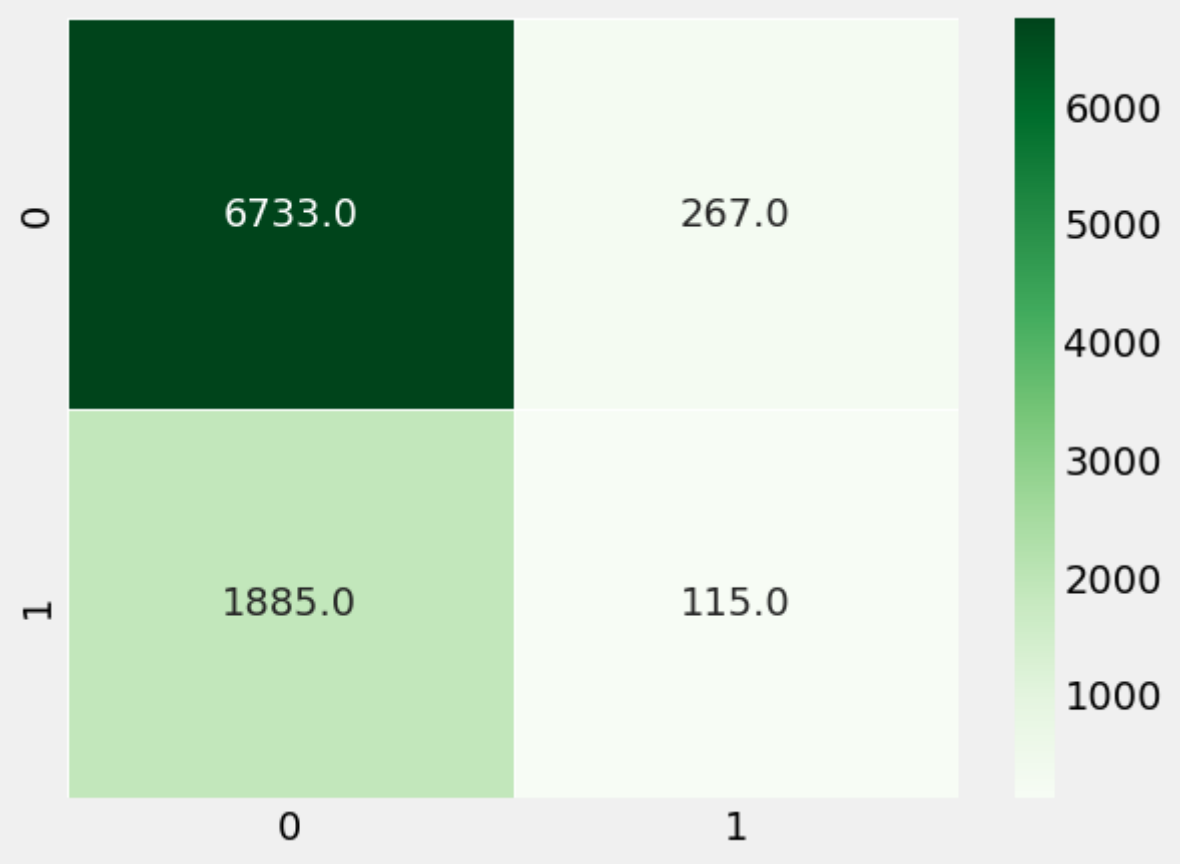
In [113…
```
y_pred = knn.predict(X_test)
```

In [114…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[114…
```
array([[6733,  267],
       [1885,  115]], dtype=int64)
```

In [115…
```
sns.heatmap(cm,annot= True,linewidths=0.5,cmap="Greens",fmt=".1f")
```

Out[115… <AxesSubplot: >



In [116...

```
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
roc=roc_auc_score(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

res = pd.DataFrame([['K-Nearest Neighbour', acc,prec,rec, f1,roc]],
                   columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
res
```

Out[116...

	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	K-Nearest Neighbour	0.760889	0.301047	0.0575	0.096558	0.509679

Conclusion

1)Using a Logistic Regression classifier, we can predict with 81.12% accuracy, whether a customer is likely to default next month.

2)Using a Decision Tree classifier, we can predict with 73.13% accuracy, whether a customer is likely to default next month.

3)Using a Random Forest classifier, we can predict with 81.33% accuracy, whether a customer is likely to default next month.

4)Using a Support Vector Machine classifier, we can predict with 77.77% accuracy, whether a customer is likely to default next month.

5)Using a Gaussian Naive Bayes classifier, we can predict with 78.85% accuracy, whether a customer is likely to default next month.

6)Using a Stochastic Gradient Descent classifier, we can predict with 77.51% accuracy, whether a customer is likely to default next month.

7)Using a K-Nearest Neighbour classifier, we can predict with 76.08% accuracy, whether a customer is likely to default next month.

In [ ]: