Prediction of Credit Card Default

Table of Contents

1 | Introduction

Welcome to the Default of Credit Card Dataset Prediction Notebook! This comprehensive dataset provides information about default payments of credit card clients in Taiwan. The idea is to use this dataset to improve basic skills of data cleaning, data analysis, data visualization and machine learning.

2| Objectives

- Analyze data to identify significant factors that impact credit card default probabilities.
- Predict the likelihood of credit card default for customers of the Bank.

3 | About Dataset

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.

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)

4 | Understanding the Credit Card Default System

How does a credit card system works?

- Every month, you receive a bill (X) reflecting your credit card expenses.
- You make a payment (Y), typically the minimum amount due, by the due date mentioned on the billing statement
- The next month's bill includes the remaining balance from the previous month (X - Y) plus any new expenses (X') incurred during that month.
- You make another payment (Y') to cover part of the new bill.
- This cycle repeats, with each month's bill incorporating previous balances, new expenses, and subtracting payments.

Missing the minimum payment due date leads to a late payment, often accompanied by late fees. In addition, continued delay might lead to default

What is defaulter?

A person is considered a defaulter by the bank when they fail to make the required payments on their credit card or loan as per the agreed-upon terms. In the context of credit card, defaulting typically occurs when the cardholder misses making the minimum payment by the due date specified in the billing statement.

5 | Importing Packages and Loading Data

In [97]:
```
# here we will import the libraries used for machine learning
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas import set_option
pd.set_option('display.max_columns', None) # for showing maximum columns
plt.style.use('ggplot') # nice plots

from sklearn.model_selection import train_test_split # to split the data into two parts
from sklearn.model_selection import GridSearchCV # for tuning parameter
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, make_scorer, f1_score
from sklearn import metrics
```

In [98]:
```
df = pd.read_csv('UCI_Credit_Card.csv')
df.head()
```

Out[98]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1
0	1	20000.0	2	2	1	24	2	2	-1	-1	-2	-2	3913.0
1	2	120000.0	2	2	2	26	-1	2	0	0	0	2	2682.0
2	3	90000.0	2	2	2	34	0	0	0	0	0	0	29239.0
3	4	50000.0	2	2	1	37	0	0	0	0	0	0	46990.0
4	5	50000.0	1	2	1	57	-1	0	-1	0	0	0	8617.0

Let us understand the credit card system with one example from our dataset

Consider the example of person with ID 1.

- LIMIT_BAL: 20000

  ```
  This person's credit limit (maximum amount they can borrow) is 20,000 Taiwanese dollars.
  ```
- SEX: 2

  ```
  This person's gender is female (since 2 represents female in this dataset).
  ```
- EDUCATION: 2

  ```
  This person has a university-level education.
  ```
- MARRIAGE: 1

  ```
  This person is married.
  ```
- AGE: 24

  ```
  This person is 24 years old.
  ```

- PAY_0 to PAY_6 (Repayment Status for Different Months):

```
PAY_0: -1
PAY_2: -1
PAY_3: -2
PAY_4: -2
PAY_5: 0
PAY_6: 0
```

  These values indicate the repayment status for different months. There is ambiguity in the in the data here as -2 is not documented in the desciption

- BILL_AMT1 to BILL_AMT6 (Bill Statements for Different Months):

```
BILL_AMT1: 3913 (Bill amount in September, 2005)
BILL_AMT2: 3102 (Bill amount in August, 2005)
BILL_AMT3: 689 (Bill amount in July, 2005)
BILL_AMT4: 0 (Bill amount in June, 2005)
BILL_AMT5: 0 (Bill amount in May, 2005)
BILL_AMT6: 0 (Bill amount in April, 2005)
```

  These values represent the amount of money owed on the credit card bill for different months.

- PAY_AMT1 to PAY_AMT6 (Previous Payments for Different Months):

```
PAY_AMT1: 0 (No previous payment in September, 2005)
PAY_AMT2: 689 (Previous payment in August, 2005)
PAY_AMT3: 0 (No previous payment in July, 2005)
PAY_AMT4: 0 (No previous payment in June, 2005)
PAY_AMT5: 0 (No previous payment in May, 2005)
PAY_AMT6: 0 (No previous payment in April, 2005)
```

  These values represent the amount of money the person paid toward their credit card bill in previous months.

- default.payment.next.month: 1

  This person defaulted on their credit card payment in the next month (default.payment.next.month = 1)

In summary, this row of data provides information about a female individual who is 24 years old, with a university education, and is married. She has a credit limit of 20,000 Taiwanese dollars. Her repayment and bill statement history indicates some on-time payments, minor payment delays, and some months with no consumption. Despite her previous payment behavior, she defaulted on her credit card payment in the following month.

6 | Data Exploration

In [99]:

```
df.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
```

-> No null values and datatypes are also correct

In [100…

```
# Categorical variables description
df[['SEX', 'EDUCATION', 'MARRIAGE']].describe()
```

Out[100]:

	SEX	EDUCATION	MARRIAGE
count	30000.000000	30000.000000	30000.000000
mean	1.603733	1.853133	1.551867
std	0.489129	0.790349	0.521970
min	1.000000	0.000000	0.000000
25%	1.000000	1.000000	1.000000
50%	2.000000	2.000000	2.000000
75%	2.000000	2.000000	2.000000
max	2.000000	6.000000	3.000000

-> For the "EDUCATION" feature, category 5 and 6 are unlabelled and category 0 is undocumented
-> Similarly, the "MARRIAGE" feature includes an undocumented label 0

In [101…

```
# Payment delay description
df[['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']].describe()
```

Out[101]:

	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	-0.016700	-0.133767	-0.166200	-0.220667	-0.266200	-0.291100

std	1.123802	1.197186	1.196868	1.169139	1.133187	1.149988
min	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000

```
-> All coulumns present an undocumented label -2.
-> If 1,2,3, etc are the months of delay, 0 should be labeled 'pay duly' and
every negative value should be seen as a 0. But we will get to that later
```

In [102…

```
# Bill Statement description
df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].descr
```

Out[102]:

	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

```
-> Can Negative values be interpreted as credit? Need to investigate furthur
```

In [103…

```
#Previous Payment Description
df[['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].describe()
```

Out[103]:

	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 [104…

```
# How many records
print('There are',df.shape[0],'records in the data')
```

```
There are 30000 records in the data
```

7 | Data Cleaning

In [105…

```
# renaming columns
df.rename(columns={"default.payment.next.month": "def_pay",
                   'PAY_0': 'PAY_1'}, inplace=True)
df.head()
```

Out[105]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1
0	1	20000.0	2	2	1	24	2	2	-1	-1	-2	-2	3913.0
1	2	120000.0	2	2	2	26	-1	2	0	0	0	2	2682.0
2	3	90000.0	2	2	2	34	0	0	0	0	0	0	29239.0
3	4	50000.0	2	2	1	37	0	0	0	0	0	0	46990.0
4	5	50000.0	1	2	1	57	-1	0	-1	0	0	0	8617.0

Already we saw that some categories are mislabeled or undocumented. Before proceeding, it is time to fix it.

The 0 in MARRIAGE can be 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)

In [106…

```
# changing labels 0,5,6 to 4 which represent other category
df['EDUCATION'] = df['EDUCATION'].apply(lambda x: 4 if x in [0, 5, 6] else x)
```

In [107…

```
# replacing 0 to 3
df['MARRIAGE'] = df['MARRIAGE'].replace(0, 3)
```

The "PAY_n" variables signify the count of months a payment is delayed, with "-1" indicating "pay duly." However, the interpretation of "-2" and "0" is unclear. Adjusting the label to consider "pay duly" as 0 seems appropriate to enhance clarity in understanding the payment status progression.

In [108…

```
def replace_to_zero(col):
    fil = (df[col] == -2) | (df[col] == -1) | (df[col] == 0)
    df.loc[fil, col] = 0

for i in ['PAY_1','PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']:
    replace_to_zero(i)
```

Now updated categories are as following:

SEX: Gender

```
1 = male
2 = female
```

EDUCATION:

```
1 = graduate school
2 = university
3 = high school
4 = others
```

MARRIAGE:

```
1 = married
2 = single
3 = others
```

PAY_0,2,3,4,5,6:

```
0 = 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
```

8 | Exploratory Data Analysis (EDA)

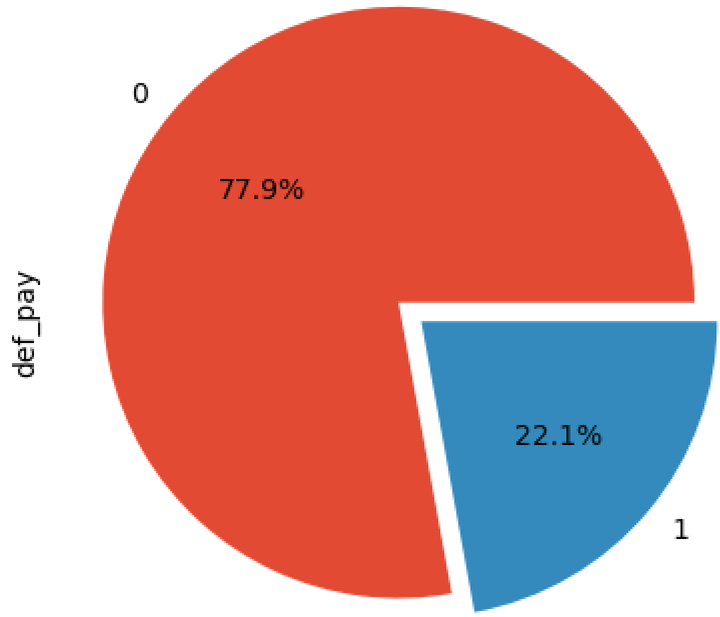
In [109…

```
# How many defaulters
perc_default = df.def_pay.sum() / len(df.def_pay)
print(f'The percentage of defaulters in the data is {perc_default*100} %')
df['def_pay'].value_counts().plot(kind='pie',explode=[0.1,0],autopct="%1.1f%%")
plt.plot()
```

```
The percentage of defaulters in the data is 22.12 %
```

Out[109]:

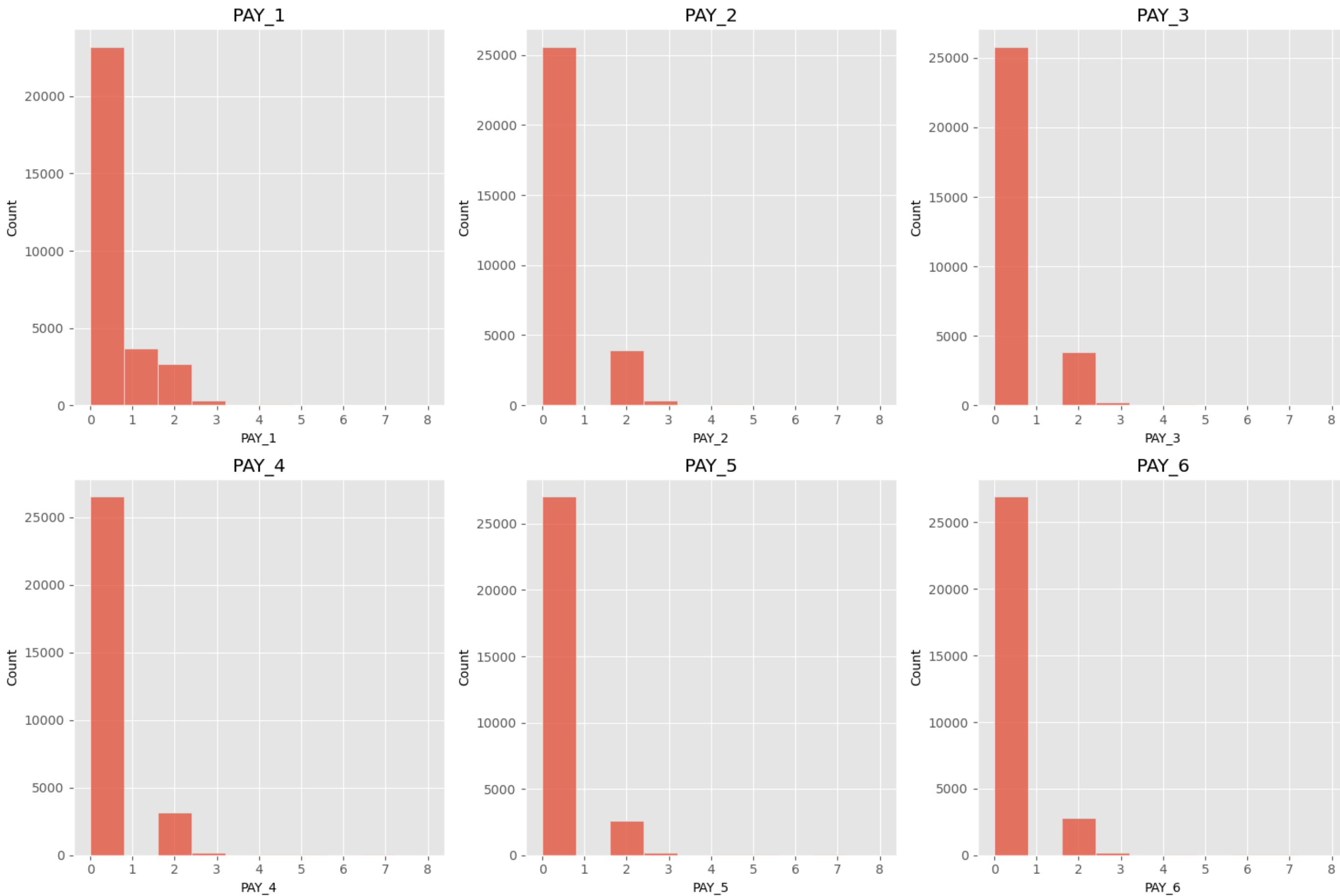
```
[]
```



8.1 | Payment Status (PAY_X)

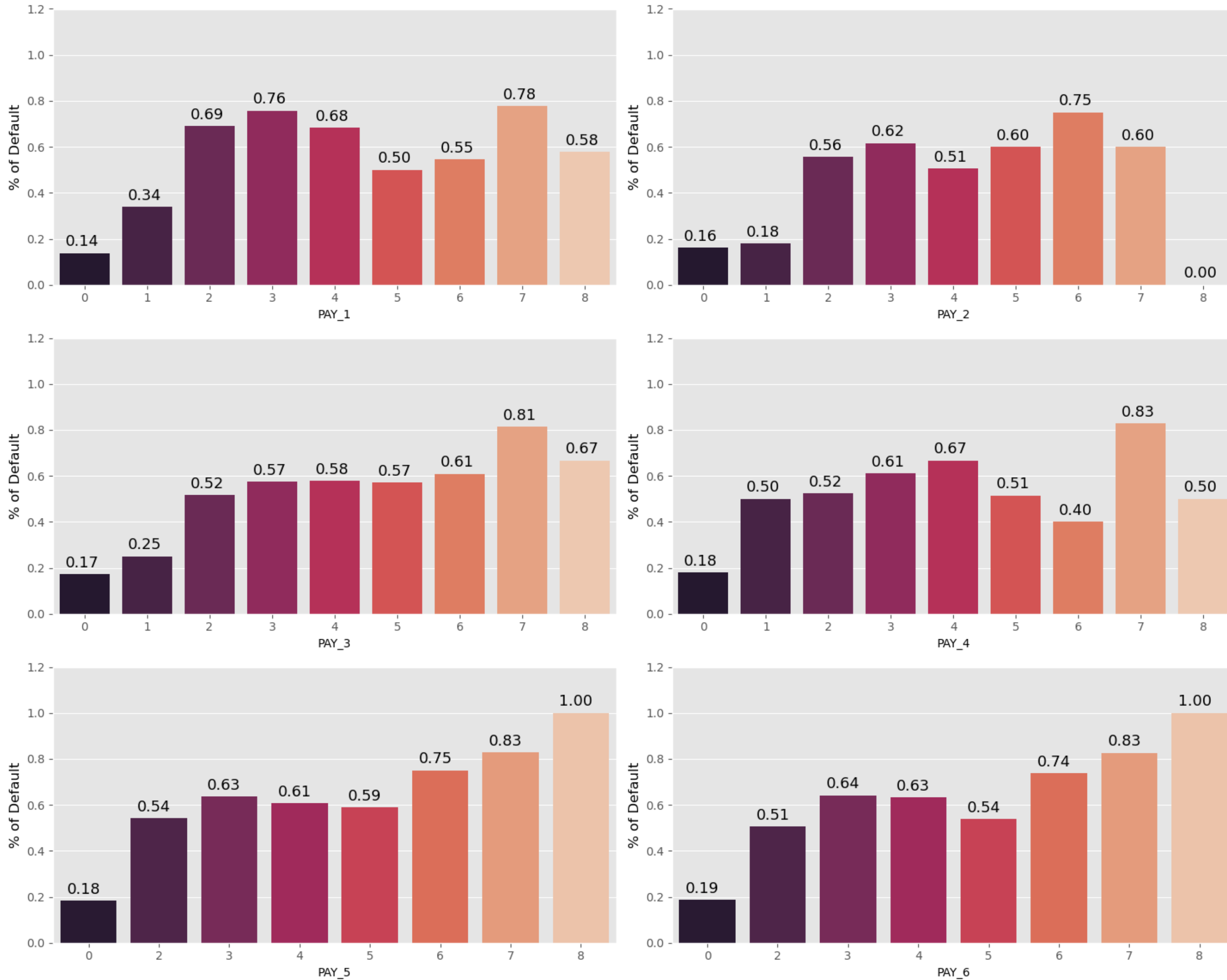
In [110…
```
def draw_histograms(df, variables, n_rows, n_cols, n_bins):
    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(15, 10))
    for i, var_name in enumerate(variables):
        row = i // n_cols
        col = i % n_cols
        sns.histplot(data=df, x=var_name, bins=n_bins, ax=axes[row, col])
        axes[row, col].set_title(var_name)
    fig.tight_layout()
    plt.show()
```

In [111…
```
late = df[['PAY_1','PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']]
draw_histograms(late, late.columns, 2, 3, 10)
```



In [113…

```
pay_x_fts = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
plt.figure(figsize=(15,12))

for i,col in enumerate(pay_x_fts):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "def_pay", data = df, palette = 'rocket', errorbar = N
    plt.ylabel("% of Default", fontsize= 12)
    plt.ylim(0,1.2)
    plt.tight_layout()

    for p in ax.patches:
        ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.09, p.get_height()+0.03),font

plt.show()
```



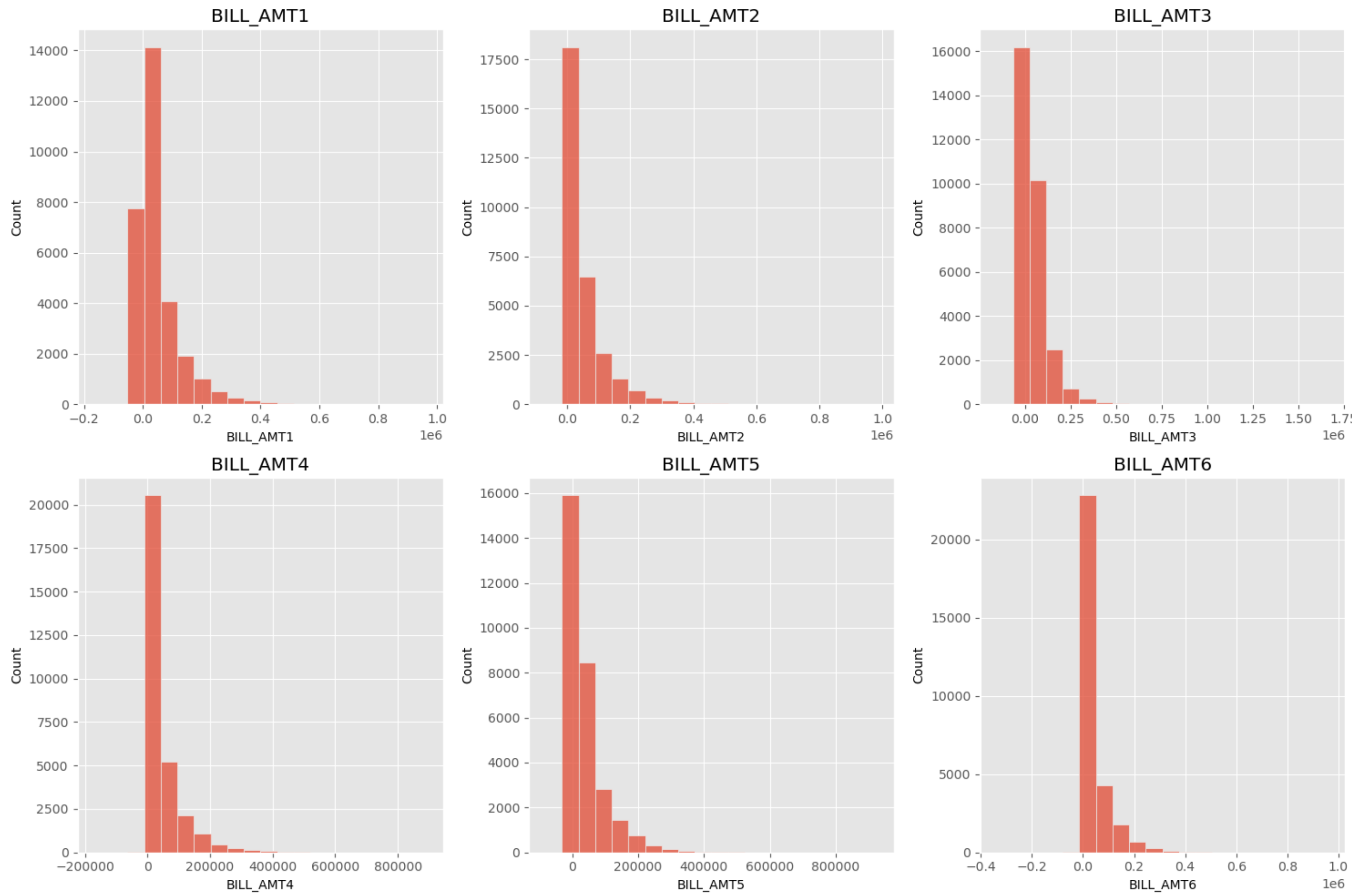
-> Most customers are duly paying their credit card bills. And it's pretty clear that their likelihood of default are much lower than the rest.

-> Credit card holders who consistently delay their payments for more than 3 months are significantly more likely to face defaults, with an approximate likelihood of 70%.

8.2 | Amount of Bill Statement (BILL_AMTX)

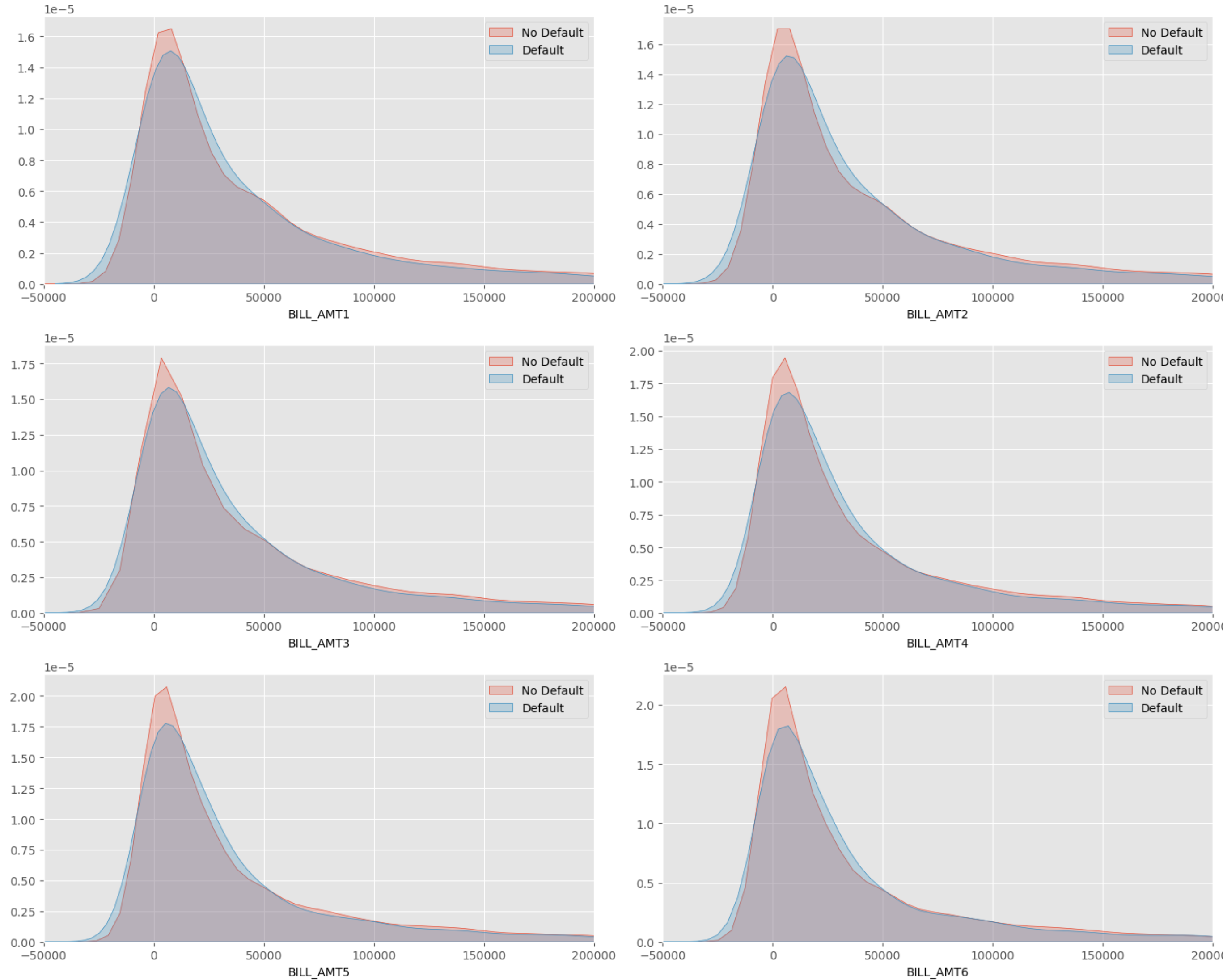
In [118…

```
bill_amtx_fts = ['BILL_AMT1','BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_A
bills = df[bill_amtx_fts]
draw_histograms(bills, bills.columns, 2, 3, 20)
```



In [119…

```
plt.figure(figsize=(15,12))

for i,col in enumerate(bill_amtx_fts):
    plt.subplot(3,2,i + 1)
    sns.kdeplot(df.loc[(df['def_pay'] == 0), col], label = 'No Default',fill = True)
    sns.kdeplot(df.loc[(df['def_pay'] == 1), col], label = 'Default', fill = True)
    plt.xlim(-50000,200000)
    plt.ylabel('')
    plt.legend()
    plt.tight_layout()

plt.show()
```



In [120…

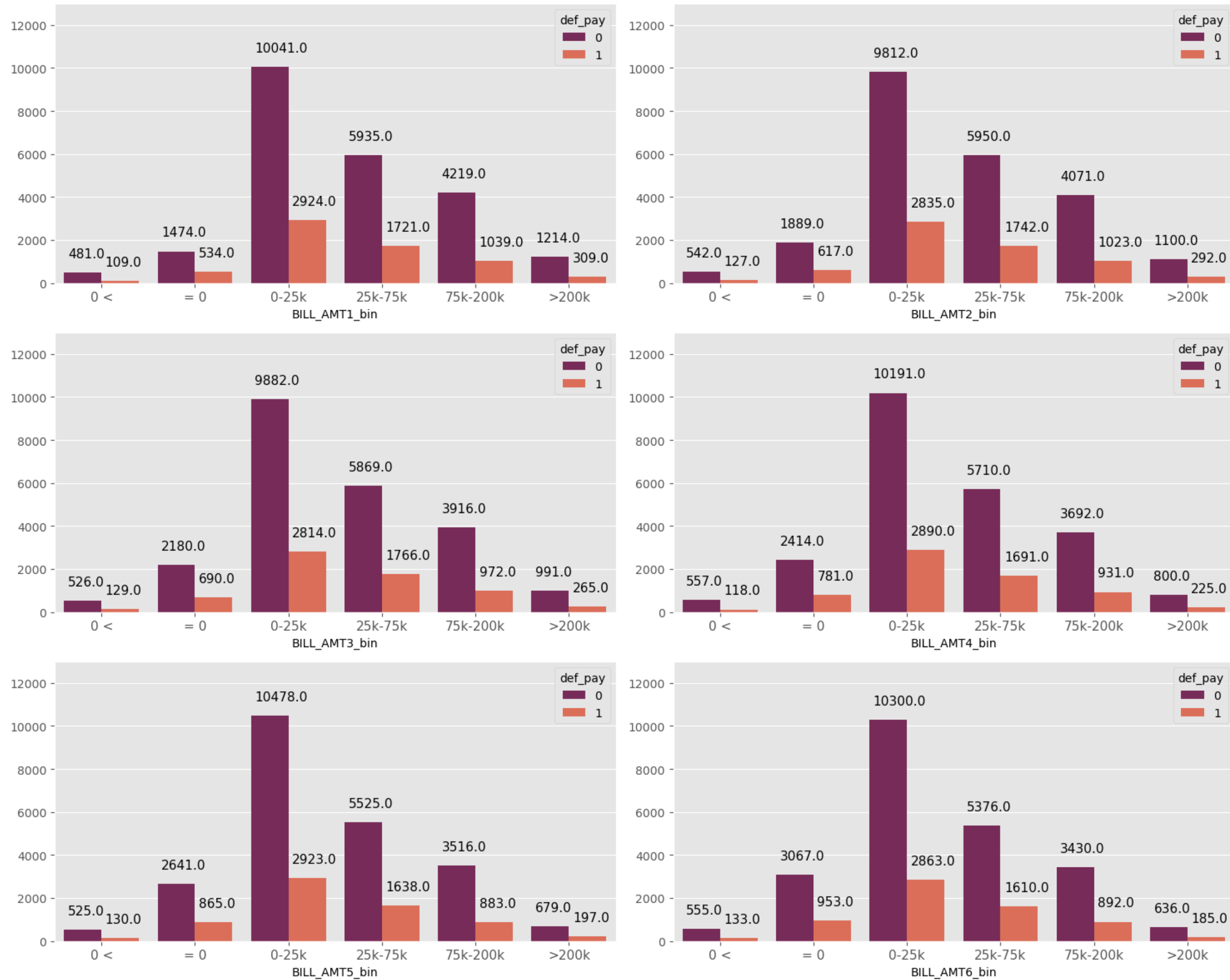
```
df['BILL_AMT1_bin'] = df['BILL_AMT1'].copy()
df['BILL_AMT2_bin'] = df['BILL_AMT2'].copy()
df['BILL_AMT3_bin'] = df['BILL_AMT3'].copy()
df['BILL_AMT4_bin'] = df['BILL_AMT4'].copy()
df['BILL_AMT5_bin'] = df['BILL_AMT5'].copy()
df['BILL_AMT6_bin'] = df['BILL_AMT6'].copy()
```

In [121…

```
bill_amtx_bins = ['BILL_AMT1_bin', 'BILL_AMT2_bin', 'BILL_AMT3_bin', 'BILL_AMT4_bin', 'B

for i, col in enumerate (bill_amtx_bins):
    df[col] = pd.cut(df[bill_amtx_fts[i]],[-350000,-1,0,25000, 75000, 200000, 2000000])
```

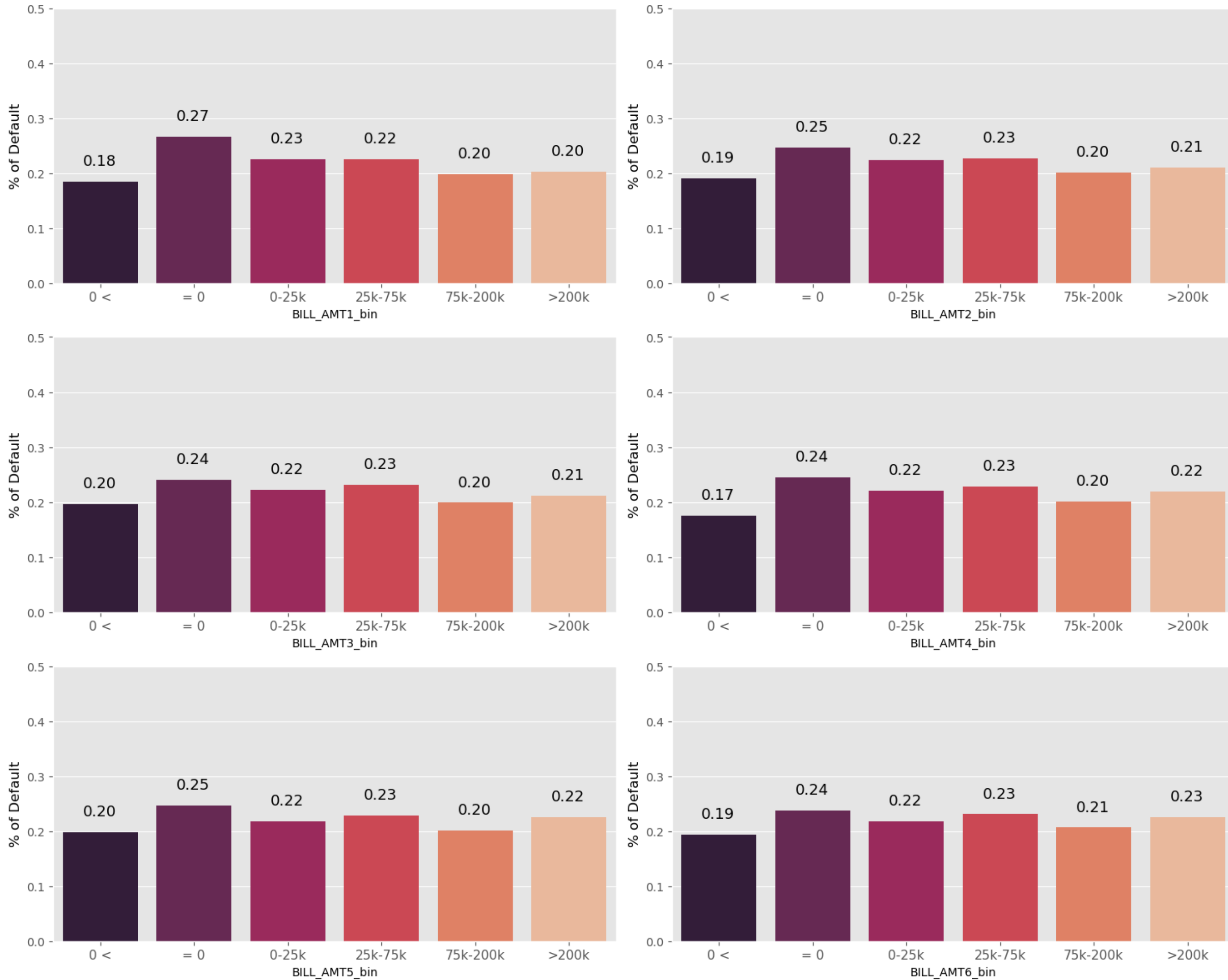
In [122…

```
plt.figure(figsize=(15, 12))
for i,col in enumerate(bill_amtx_bins):
    plt.subplot(3,2,i + 1)
    ax = sns.countplot(data = df, x = col, hue="def_pay", palette = 'rocket')
    plt.ylim(0,13000)
    plt.ylabel('')
    plt.xticks([0,1,2,3,4,5],['0 <', '= 0', '0-25k', '25k-75k', '75k-200k', '>200k'], fo
    plt.tight_layout()

    for p in ax.patches:
        ax.annotate((p.get_height()), (p.get_x()+0.04, p.get_height()+700), fontsize = 1

plt.show()
```



In [123…

```
plt.figure(figsize=(15,12))

for i,col in enumerate(bill_amtx_bins):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "def_pay", data = df, palette = 'rocket', errorbar = N
    plt.ylabel("% of Default", fontsize= 12)
    plt.ylim(0,0.5)
    plt.xticks([0,1,2,3,4,5],['0 <', '= 0', '0-25k', '25k-75k', '75k-200k', '>200k'], fo
    plt.tight_layout()

    for p in ax.patches:
        ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.21, p.get_height()+0.03),font

plt.show()
```

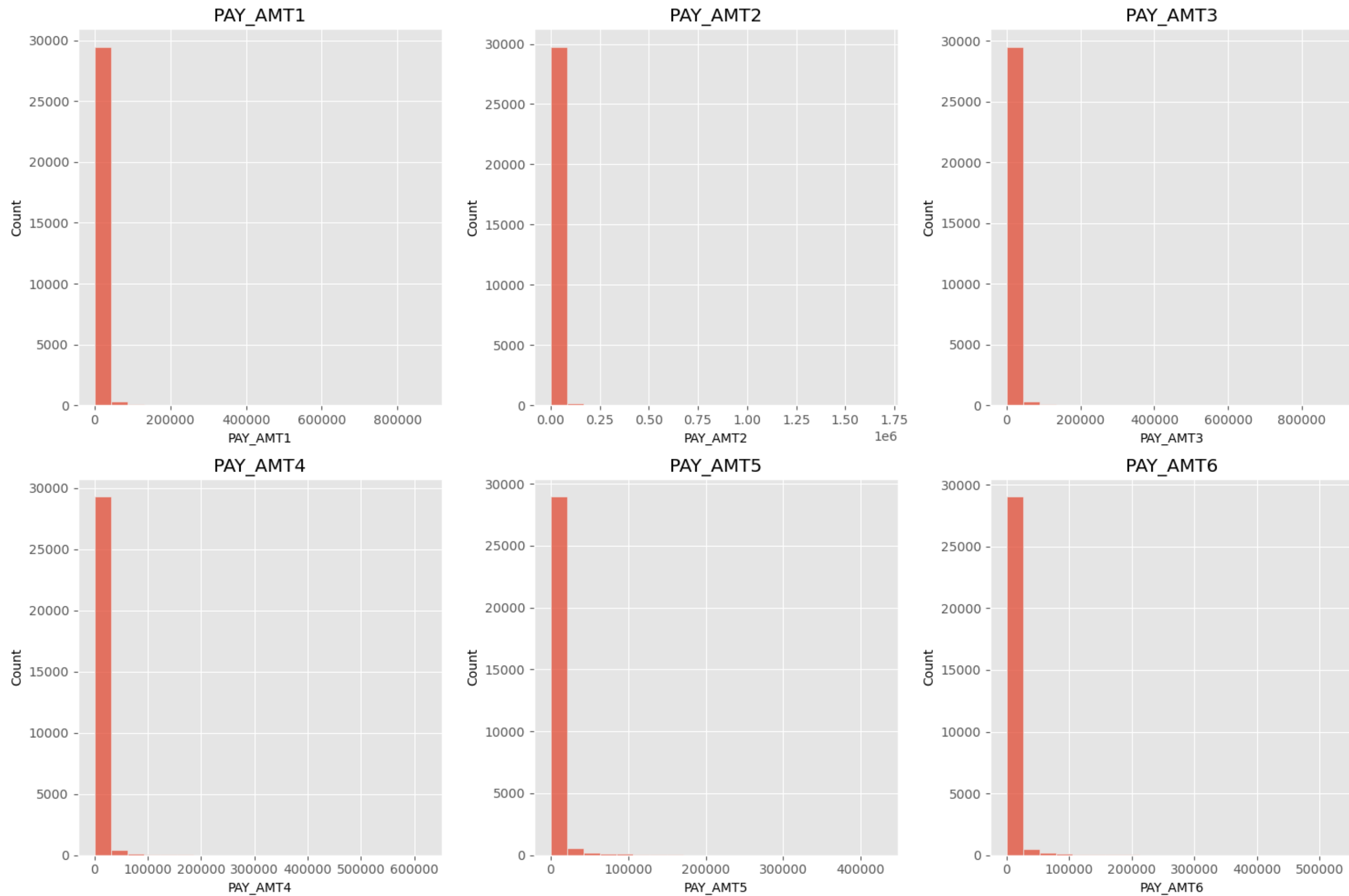


-> Those who have a negative bill statement have a lower chance of default than the rest. What stands out is that there is a little higher chance of default for those who didn't have a bill in the previous months.

8.3 | Amount of Previous Payment (PAY_AMTX)

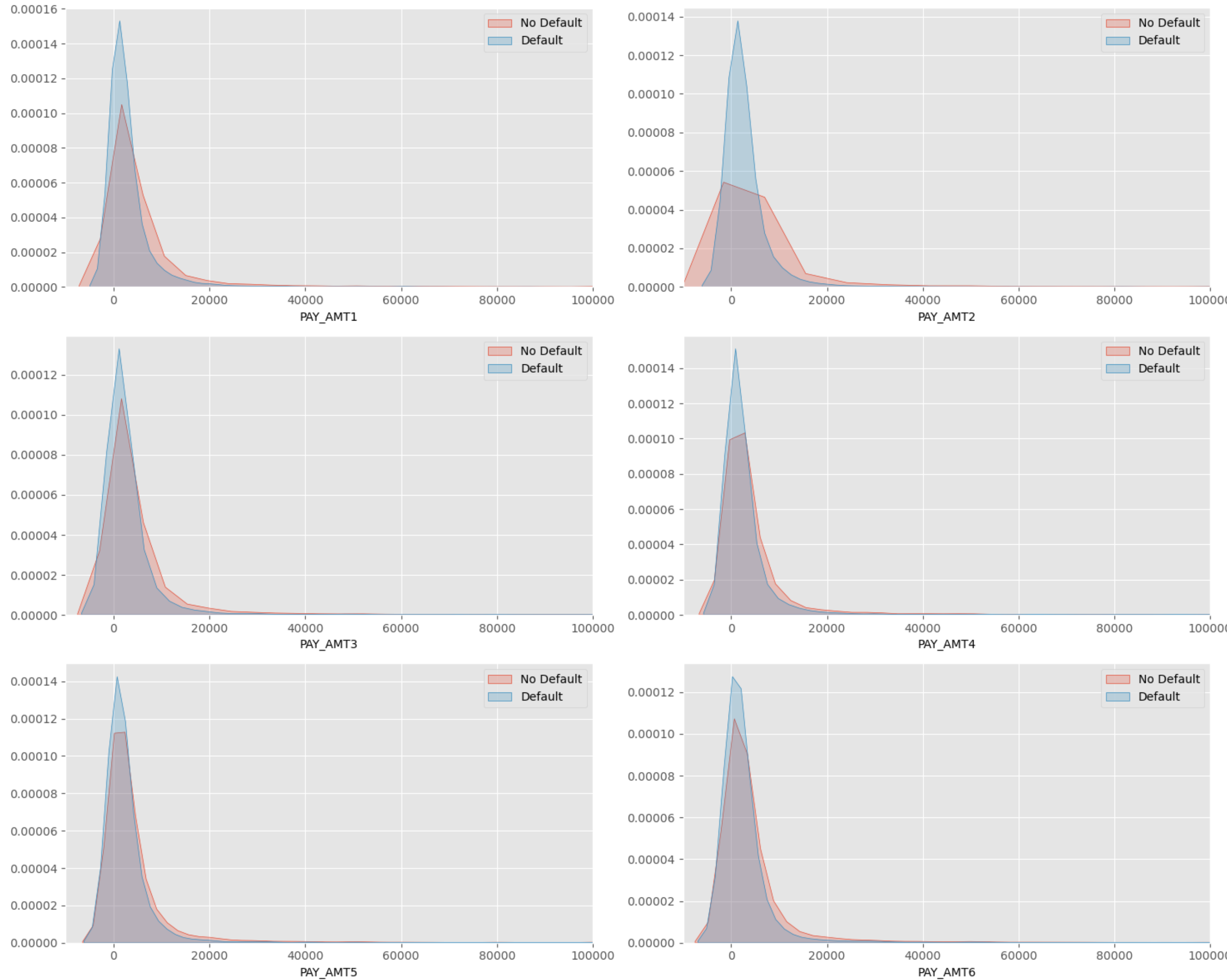
In [124…

```
pay_amtx_fts = ['PAY_AMT1','PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
pay = df[pay_amtx_fts]
draw_histograms(pay, pay.columns, 2, 3, 20)
```



In [125…

```
plt.figure(figsize=(15,12))

for i,col in enumerate(pay_amtx_fts):
    plt.subplot(3,2,i + 1)
    sns.kdeplot(df.loc[(df['def_pay'] == 0), col], label = 'No Default', fill = True)
    sns.kdeplot(df.loc[(df['def_pay'] == 1), col], label = 'Default', fill = True)
    plt.xlim(-10000,100000)
    plt.ylabel('')
    plt.legend()
    plt.tight_layout()

plt.show()
```



In [126...

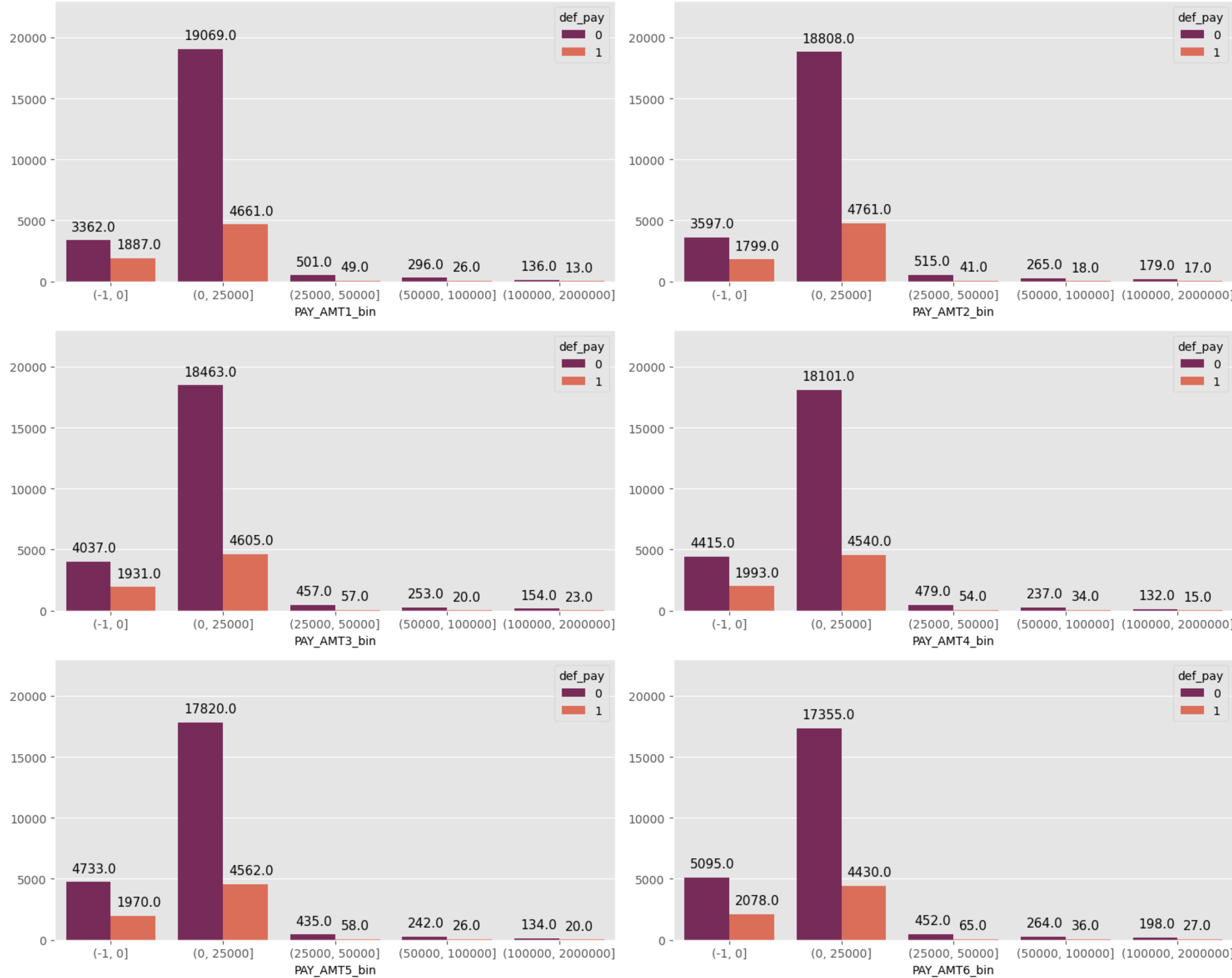
```
df['PAY_AMT1_bin'] = df['PAY_AMT1'].copy()
df['PAY_AMT2_bin'] = df['PAY_AMT2'].copy()
df['PAY_AMT3_bin'] = df['PAY_AMT3'].copy()
df['PAY_AMT4_bin'] = df['PAY_AMT4'].copy()
df['PAY_AMT5_bin'] = df['PAY_AMT5'].copy()
df['PAY_AMT6_bin'] = df['PAY_AMT6'].copy()
```

In [127...

```
pay_amtx_bins = ['PAY_AMT1_bin', 'PAY_AMT2_bin', 'PAY_AMT3_bin', 'PAY_AMT4_bin', 'PAY_AM

for i, col in enumerate (pay_amtx_bins):
    df[col] = pd.cut(df[pay_amtx_fts[i]],[-1, 0, 25000, 50000, 100000, 2000000])
```

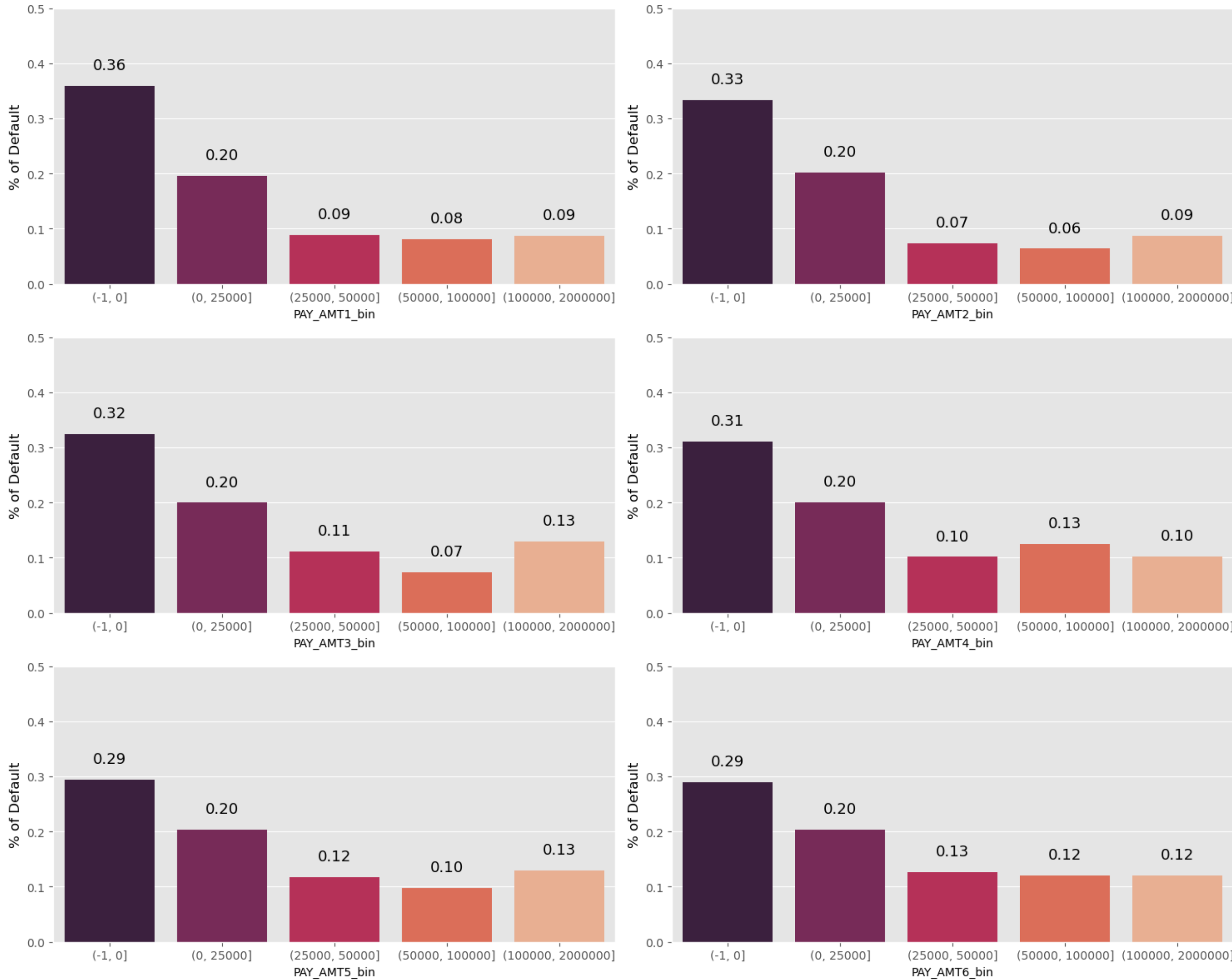
In [128...

```
plt.figure(figsize=(15,12))

for i,col in enumerate(pay_amtx_bins):
    plt.subplot(3,2,i + 1)
    ax = sns.countplot(data = df, x = col, hue="def_pay", palette = 'rocket')
    plt.ylim(0,23000)
    plt.ylabel('')
    plt.tight_layout()

    for p in ax.patches:
        ax.annotate((p.get_height()), (p.get_x()+0.05, p.get_height()+800), fontsize=11)

plt.show()
```



In [129…

```
plt.figure(figsize=(15,12))

for i,col in enumerate(pay_amtx_bins):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "def_pay", data = df, palette = 'rocket', errorbar = N
    plt.ylabel("% of Default", fontsize= 12)
    plt.ylim(0,0.5)
    plt.tight_layout()

    for p in ax.patches:
        ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.25, p.get_height()+0.03),font

plt.show()
```



-> There is a higher default rate among those who paid nothing in previous months and lower rates among those paid over 25k of NT dollars.

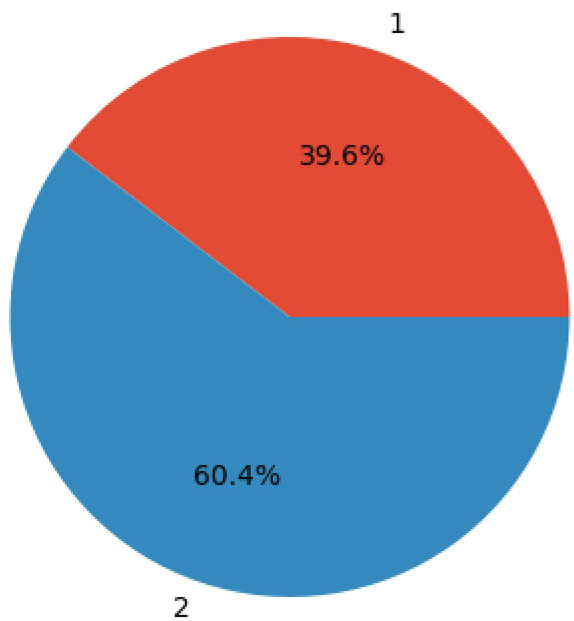
8.4 | Categorical Columns (SEX, EDUCATION, MARRIAGE)

In [130…

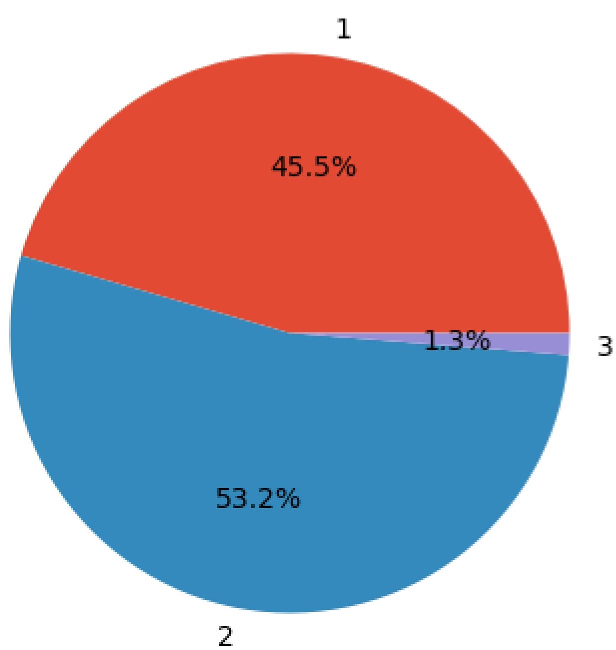
```
def show_value_counts(col):
    print(col)
    value_counts = df[col].value_counts()
    percentage = value_counts / len(df) * 100
    result_df = pd.DataFrame({'Value': value_counts.index, 'Count': value_counts, 'Perce
    result_df = result_df.sort_values(by='Value')
    print(result_df)
    print('--------------------------')
    generate_pie_plot(result_df)


def generate_pie_plot(data_frame):
    plt.figure(figsize=(6, 4))
    plt.pie(data_frame['Count'], labels=data_frame['Value'], autopct='%1.1f%%')
    plt.axis('equal')
    plt.show()
```

```
show_value_counts('SEX')
show_value_counts('MARRIAGE')
show_value_counts('EDUCATION')
```

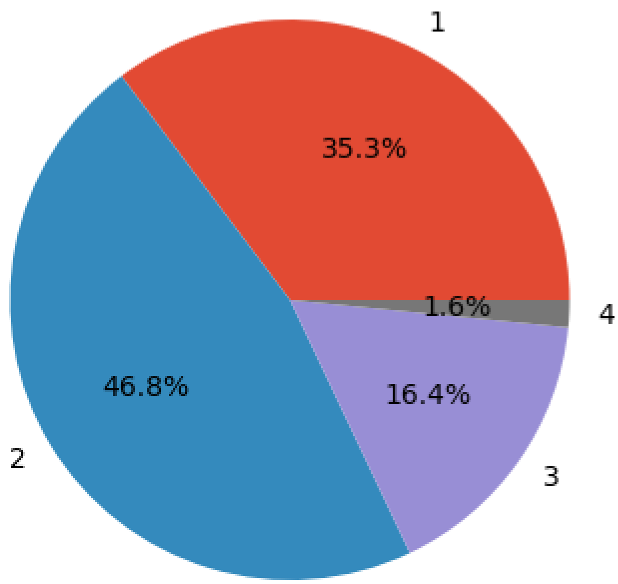
```
SEX
   Value  Count  Percentage
1      1  11888   39.626667
2      2  18112   60.373333
--------------------------
```




```
MARRIAGE
   Value  Count  Percentage
1      1  13659   45.530000
2      2  15964   53.213333
3      3    377    1.256667
--------------------------
```




```
EDUCATION
   Value  Count  Percentage
1      1  10585   35.283333
2      2  14030   46.766667
3      3   4917   16.390000
4      4    468    1.560000
--------------------------
```




```
-> There are significantly more women than men
-> Others category occupies very less percentage
```

In [131…

```
pd.crosstab(df.SEX, df.def_pay).style.background_gradient(cmap='summer_r')
```

Out[131]:

def_pay	0	1
SEX		
1	9015	2873
2	14349	3763

In [132…

```
pd.crosstab(df.MARRIAGE, df.def_pay).style.background_gradient(cmap='summer_r')
```

Out[132]:

def_pay	0	1
MARRIAGE		
1	10453	3206
2	12623	3341
3	288	89

In [133…

```
pd.crosstab(df.EDUCATION,df.def_pay).style.background_gradient(cmap='summer_r')
```

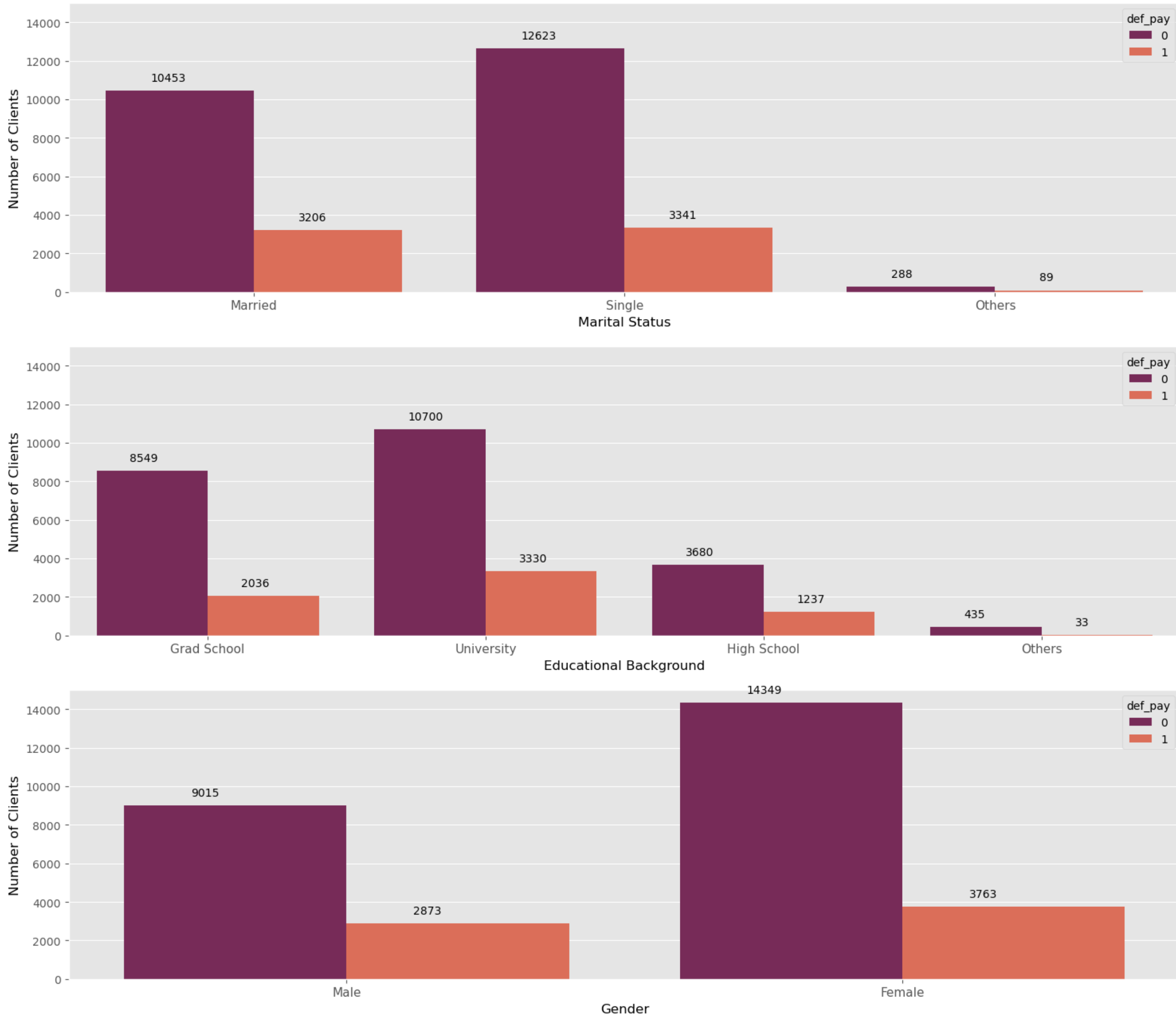
Out[133]:

def_pay	0	1
EDUCATION		
1	8549	2036
2	10700	3330
3	3680	1237
4	435	33

In [134…

```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(15, 13))

# Count plot for MARRIAGE
```

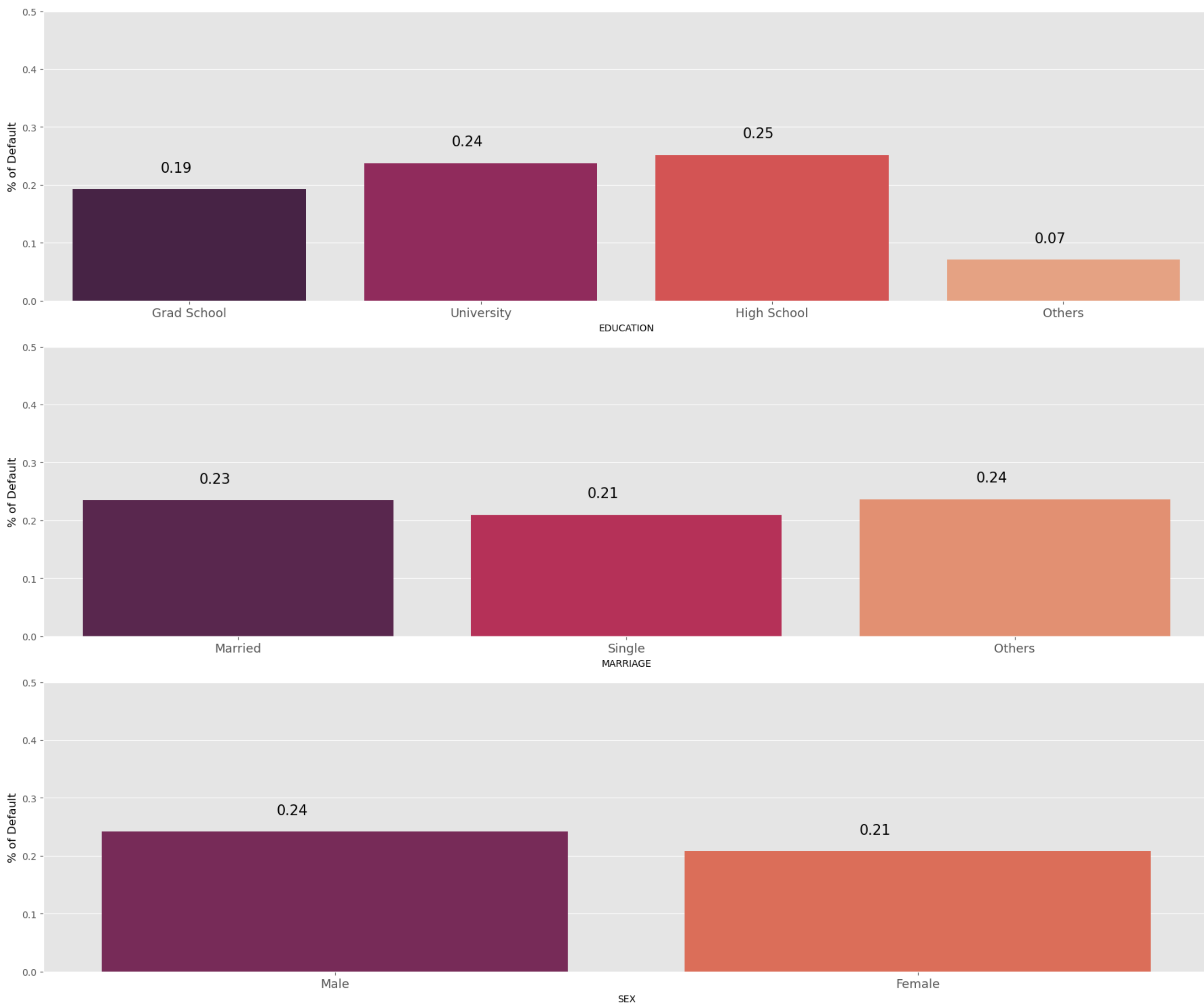
```python
ax1 = sns.countplot(data=df, x='MARRIAGE', hue='def_pay', palette='rocket', ax=axes[0])
ax1.set_xlabel("Marital Status", fontsize=12)
ax1.set_ylabel("Number of Clients", fontsize=12)
ax1.set_ylim(0, 15000)
ax1.set_xticks([0, 1, 2])
ax1.set_xticklabels(['Married', 'Single', 'Others'], fontsize=11)
for p in ax1.patches:
    ax1.annotate(int(p.get_height()), (p.get_x() + 0.12, p.get_height() + 500))

# Count plot for EDUCATION
ax2 = sns.countplot(data=df, x='EDUCATION', hue='def_pay', palette='rocket', ax=axes[1])
ax2.set_xlabel("Educational Background", fontsize=12)
ax2.set_ylabel("Number of Clients", fontsize=12)
ax2.set_ylim(0, 15000)
ax2.set_xticks([0, 1, 2, 3])
ax2.set_xticklabels(['Grad School', 'University', 'High School', 'Others'], fontsize=11)
for p in ax2.patches:
    ax2.annotate(int(p.get_height()), (p.get_x() + 0.12, p.get_height() + 500))

# Count plot for SEX
ax3 = sns.countplot(data=df, x='SEX', hue='def_pay', palette='rocket', ax=axes[2])
ax3.set_xlabel("Gender", fontsize=12)
ax3.set_ylabel("Number of Clients", fontsize=12)
ax3.set_ylim(0, 15000)
ax3.set_xticks([0, 1])
ax3.set_xticklabels(['Male', 'Female'], fontsize=11)
for p in ax3.patches:
    ax3.annotate(int(p.get_height()), (p.get_x() + 0.12, p.get_height() + 500))

plt.tight_layout()
plt.show()
```



In [135…

```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(18, 15))

# Bar plot for EDUCATION
ax1 = sns.barplot(x="EDUCATION", y="def_pay", data=df, palette='rocket', errorbar=None,
ax1.set_ylabel("% of Default", fontsize=12)
ax1.set_ylim(0, 0.5)
ax1.set_xticks([0, 1, 2, 3])
ax1.set_xticklabels(['Grad School', 'University', 'High School', 'Others'], fontsize=13)
for p in ax1.patches:
    ax1.annotate("%.2f" % (p.get_height()), (p.get_x() + 0.30, p.get_height() + 0.03), f

# Bar plot for MARRIAGE
ax2 = sns.barplot(x="MARRIAGE", y="def_pay", data=df, palette='rocket', errorbar=None, a
ax2.set_ylabel("% of Default", fontsize=12)
ax2.set_ylim(0, 0.5)
ax2.set_xticks([0,1,2])
ax2.set_xticklabels(['Married', 'Single', 'Others'], fontsize=13)
for p in ax2.patches:
    ax2.annotate("%.2f" % (p.get_height()), (p.get_x() + 0.30, p.get_height() + 0.03), f

# Bar plot for SEX
ax3 = sns.barplot(x="SEX", y="def_pay", data=df, palette='rocket', errorbar=None, ax=axe
ax3.set_ylabel("% of Default", fontsize=12)
ax3.set_ylim(0, 0.5)
ax3.set_xticks([0, 1])
ax3.set_xticklabels(['Male', 'Female'], fontsize=13)
for p in ax3.patches:
```

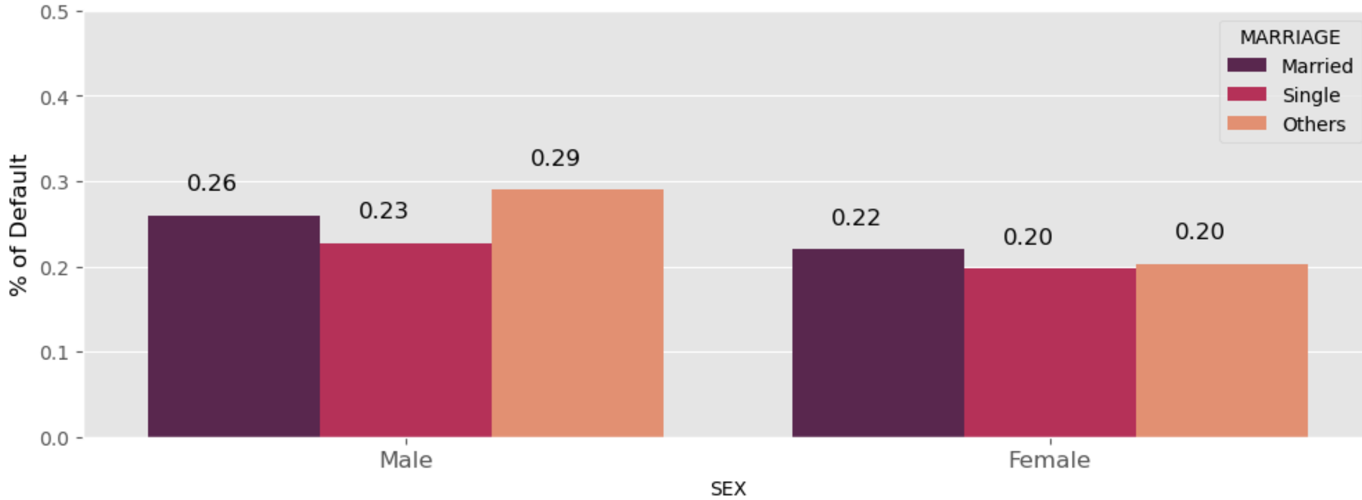
```
    ax3.annotate("%.2f" % (p.get_height()), (p.get_x() + 0.30, p.get_height() + 0.03), f

plt.tight_layout()
plt.show()
```



-> The likelihood of being a defaulter decreases as your education level increases.
-> Married and other marital statuses (possibly including divorced) have an approximately 0.24 probability of being defaulters, whereas single individuals have a lower likelihood at 0.21.
-> Despite a smaller number of males in the dataset compared to females, males exhibit a higher likelihood of being defaulters.

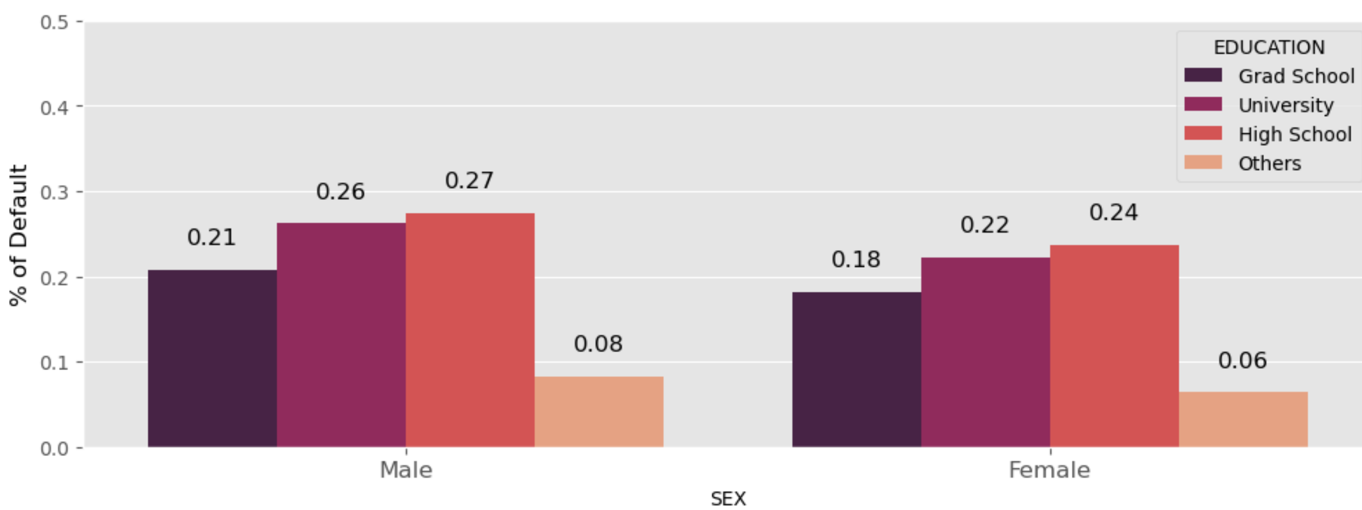
In [136…

```
plt.figure(figsize=(12,4))

ax = sns.barplot(x = "SEX", y = "def_pay", hue = "MARRIAGE", data = df, palette = 'rocke

plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)
plt.xticks([0,1],['Male', 'Female'], fontsize = 12)
plt.legend(['Married', 'Single','Others'], title = 'MARRIAGE')

for p in ax.patches:
    ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.06, p.get_height()+0.03),fontsize

plt.show()
```



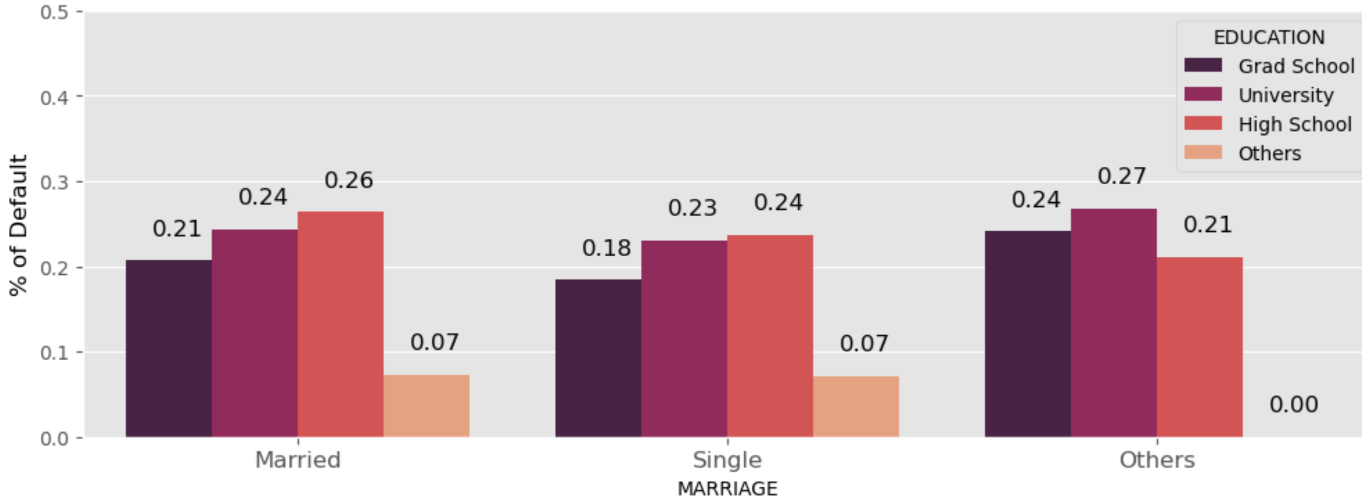
In [137…

```
plt.figure(figsize=(12,4))

ax = sns.barplot(x = "SEX", y = "def_pay", hue = "EDUCATION", data = df, palette = 'rock

plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)
plt.xticks([0,1],['Male', 'Female'], fontsize = 12)
plt.legend(['Grad School', 'University', 'High School', 'Others'], title = 'EDUCATION')

for p in ax.patches:
    ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.06, p.get_height()+0.03),fontsize

plt.show()
```



In [138…

```
plt.figure(figsize=(12,4))

ax = sns.barplot(x = "MARRIAGE", y = "def_pay", hue = "EDUCATION", data = df, palette =

plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)
plt.xticks([0,1,2],['Married', 'Single','Others'], fontsize = 12)
plt.legend(['Grad School', 'University', 'High School', 'Others'], title = 'EDUCATION')

for p in ax.patches:
    ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.06, p.get_height()+0.03),fontsize

plt.show()
```

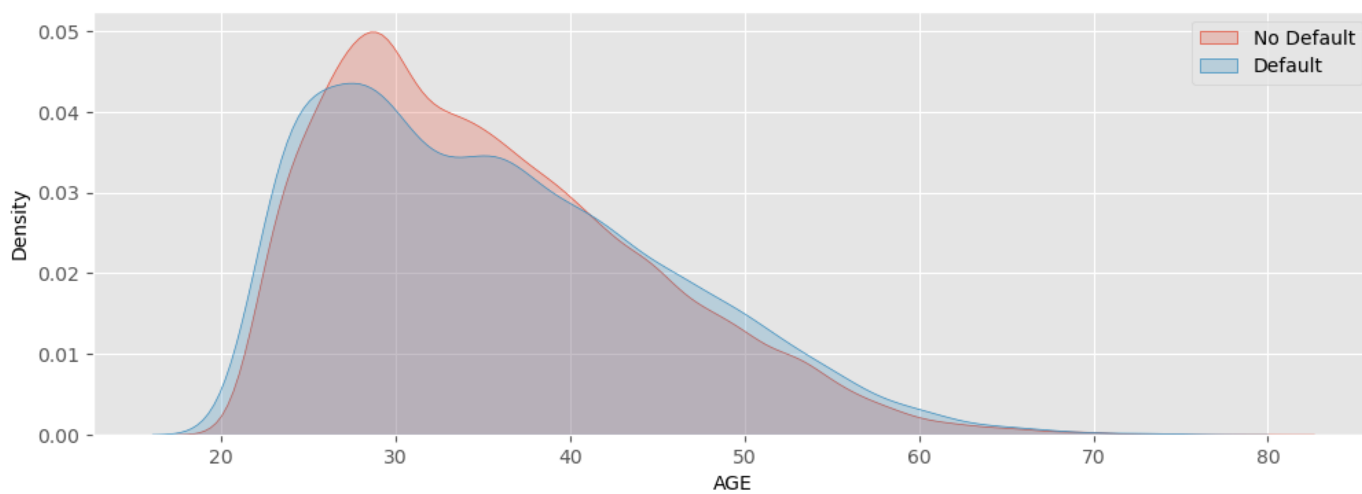


-> Being male, married, and having a high school education seems to increase the likelihood of being a defaulter.
-> People who are marked as "Others" in their marital status (likely indicating divorced individuals) have a notable probability of around 0.29 for facing defaults, which is a relatively higher occurrence.

8.5 | AGE COLUMN

In [139…

```
plt.figure(figsize=(12,4))

sns.kdeplot(df.loc[(df['def_pay'] == 0), 'AGE'], label = 'No Default', fill = True)
sns.kdeplot(df.loc[(df['def_pay'] == 1), 'AGE'], label = 'Default', fill = True)
plt.legend()
plt.show()
```



-> The age group between 20 to 30 years old appears to have a higher propensity for defaults.

Let us make categories for the age column which will make this more clear

In [140…

```
df['AgeBin'] = pd.cut(df['AGE'],[20, 25, 30, 35, 40, 50, 60, 80])
print(df['AgeBin'].value_counts())
```

```
(25, 30]    7142
(40, 50]    6005
(30, 35]    5796
(35, 40]    4917
(20, 25]    3871
(50, 60]    1997
(60, 80]     272
Name: AgeBin, dtype: int64
```

In [141…

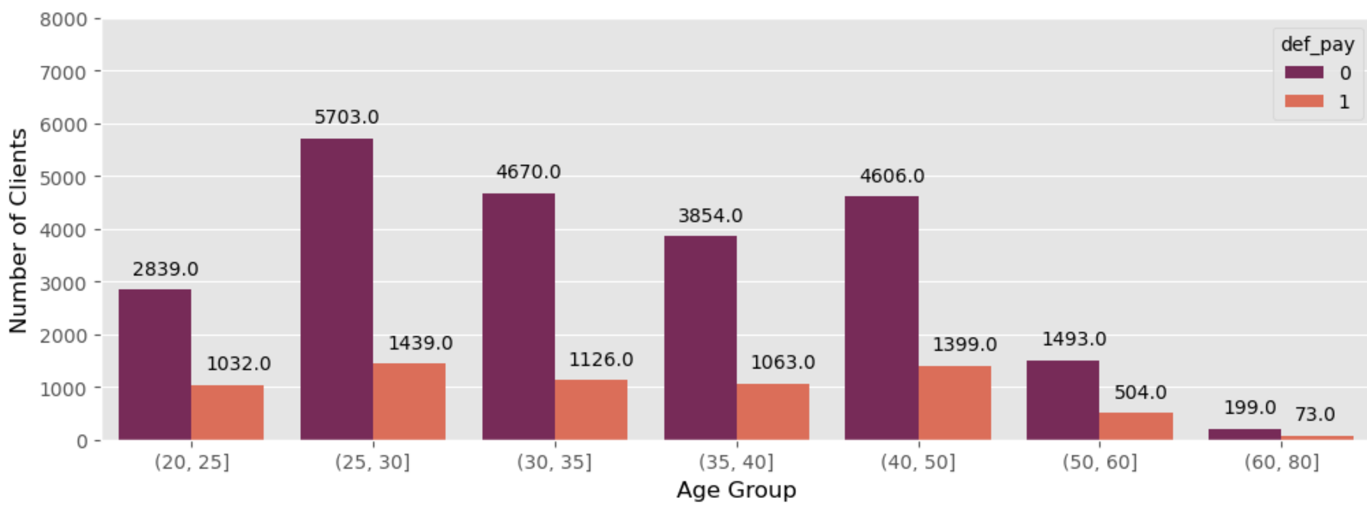
```python
df['def_pay'].groupby(df['AgeBin']).value_counts(normalize = True)
```

Out[141]:

```
AgeBin    def_pay
(20, 25]  0          0.733402
          1          0.266598
(25, 30]  0          0.798516
          1          0.201484
(30, 35]  0          0.805728
          1          0.194272
(35, 40]  0          0.783811
          1          0.216189
(40, 50]  0          0.767027
          1          0.232973
(50, 60]  0          0.747621
          1          0.252379
(60, 80]  0          0.731618
          1          0.268382
Name: def_pay, dtype: float64
```

In [142…

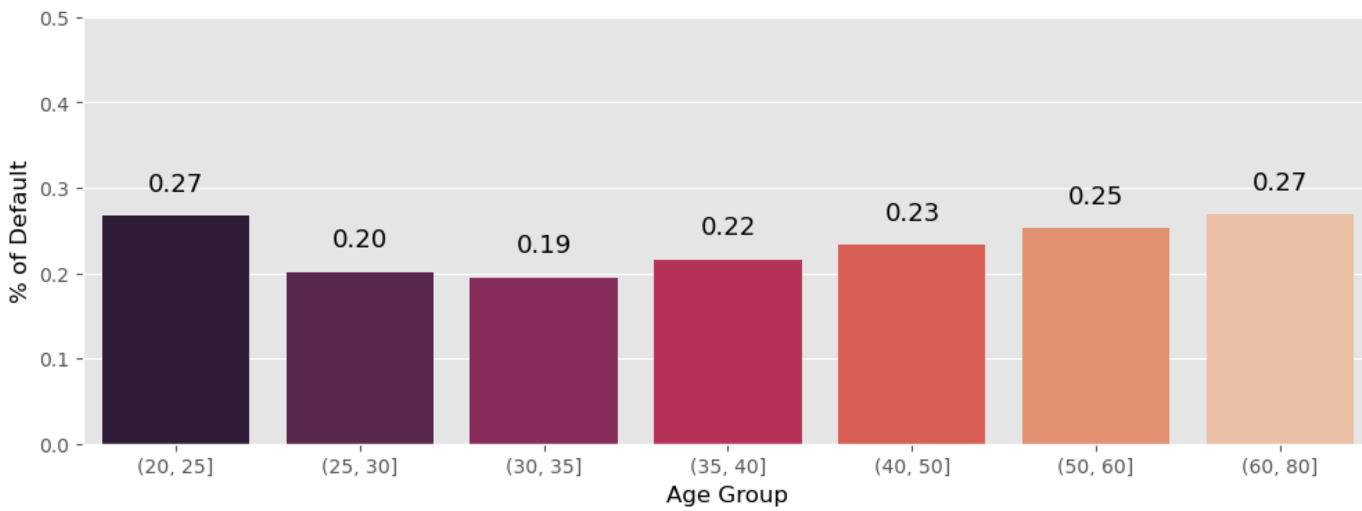
```python
plt.figure(figsize=(12,4))

df['AgeBin'] = df['AgeBin'].astype('str')
AgeBin_order = ['(20, 25]', '(25, 30]', '(30, 35]', '(35, 40]', '(40, 50]', '(50, 60]',

ax = sns.countplot(data = df, x = 'AgeBin', hue="def_pay", palette = 'rocket', order = A

plt.xlabel("Age Group", fontsize= 12)
plt.ylabel("Number of Clients", fontsize= 12)
plt.ylim(0,8000)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.075, p.get_height()+300))

plt.show()
```



In [143…

```python
plt.figure(figsize=(12,4))

ax = sns.barplot(x = "AgeBin", y = "def_pay", data = df, palette = 'rocket', errorbar = 

plt.xlabel("Age Group", fontsize= 12)
```

```
plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)

for p in ax.patches:
    ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.25, p.get_height()+0.03),fontsize

plt.show()
```

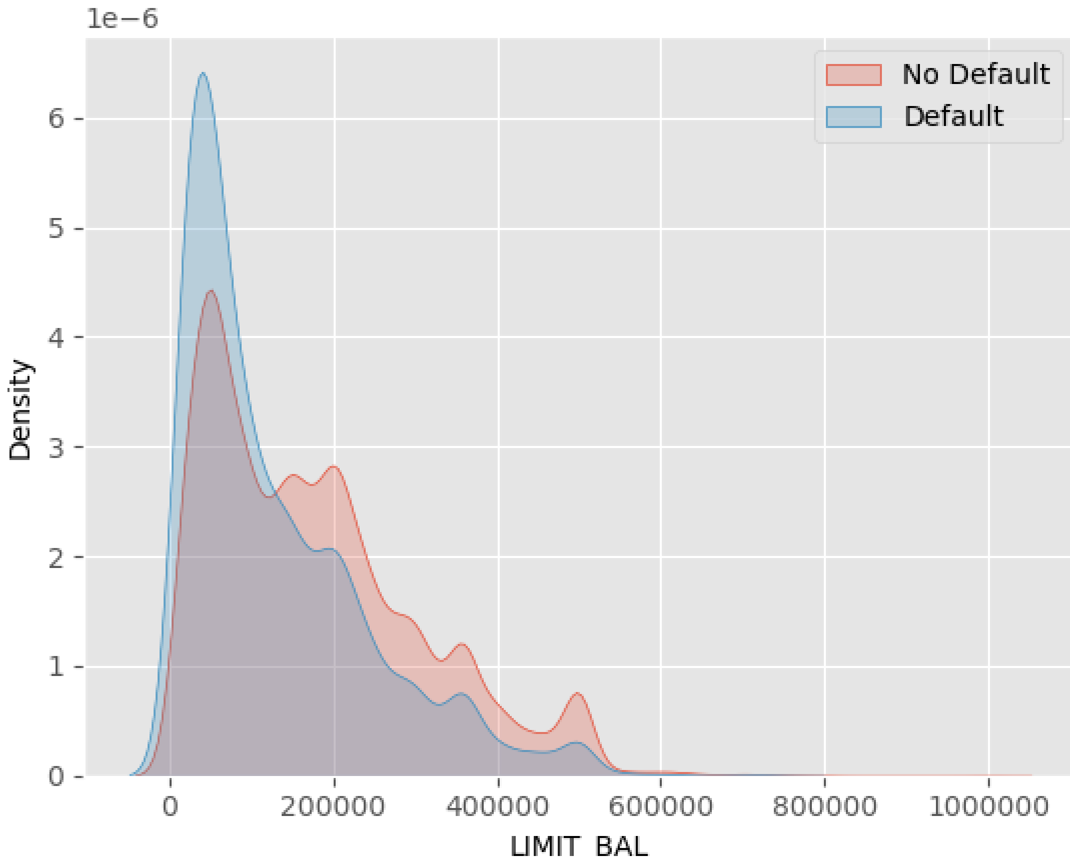


-> Individuals aged between 20 and 25, as well as those above 50, are more prone to default on their credit card payments. In contrast, individuals within other age ranges show lower tendencies for default.

8.6 | LIMIT BALANCE COLUMN (LIMIT_BAL)

In [144…

```
sns.kdeplot(df.loc[(df['def_pay'] == 0), 'LIMIT_BAL'], label = 'No Default', fill = True
sns.kdeplot(df.loc[(df['def_pay'] == 1), 'LIMIT_BAL'], label = 'Default', fill = True)
plt.ticklabel_format(style='plain', axis='x')
plt.legend()
plt.show()
```



In [146…
```
df['LIMIT_BAL'].describe()
```

Out[146]:
```
count      30000.000000
mean      167484.322667
std       129747.661567
min        10000.000000
25%        50000.000000
50%       140000.000000
75%       240000.000000
max      1000000.000000
Name: LIMIT_BAL, dtype: float64
```

    -> The majority of individuals possess a credit limit below 250000 NT dollars,
    although the upper limit extends to one million NT dollars

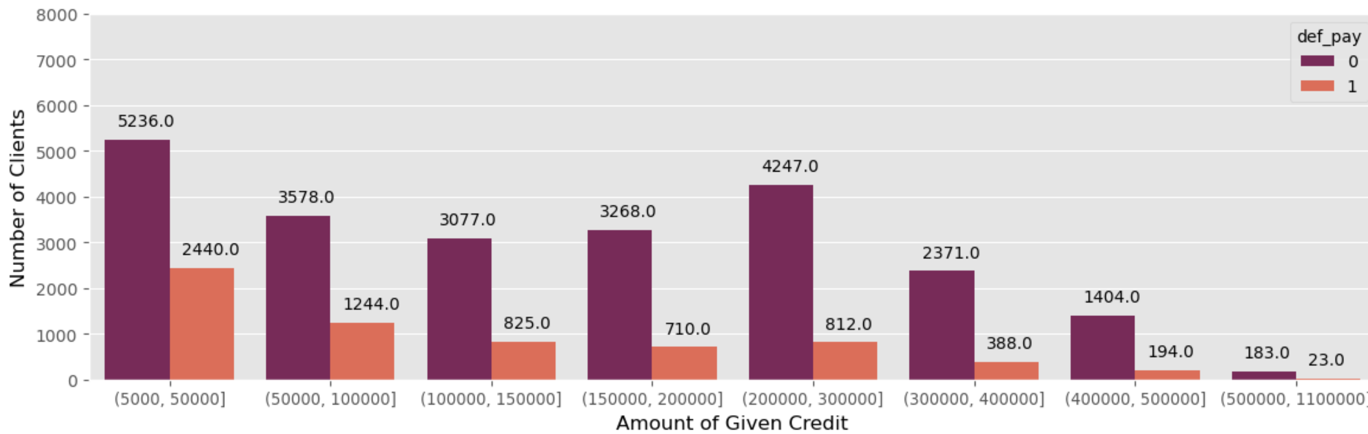
Let us make categories for the LIMTT BALANCE which will furthur help in analysis

In [147…
```
df['LimitBin'] = pd.cut(df['LIMIT_BAL'],[5000, 50000, 100000, 150000, 200000, 300000, 40
print(df['LimitBin'].value_counts())
```

```
(5000, 50000]        7676
(200000, 300000]     5059
(50000, 100000]      4822
(150000, 200000]     3978
(100000, 150000]     3902
(300000, 400000]     2759
(400000, 500000]     1598
(500000, 1100000]     206
Name: LimitBin, dtype: int64
```

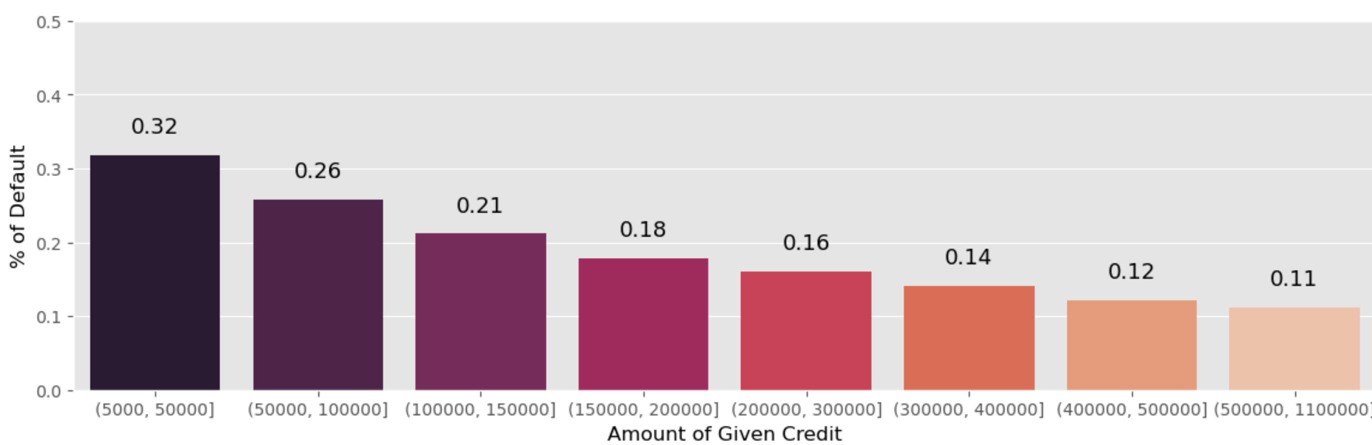
In [148…
```
plt.figure(figsize=(14,4))

df['LimitBin'] = df['LimitBin'].astype('str')
LimitBin_order = ['(5000, 50000]', '(50000, 100000]', '(100000, 150000]', '(150000, 2000
                 '(200000, 300000]', '(300000, 400000]', '(400000, 500000]', '(500000, 11
```

```
ax = sns.countplot(data = df, x = 'LimitBin', hue="def_pay", palette = 'rocket', order =

plt.xlabel("Amount of Given Credit", fontsize= 12)
plt.ylabel("Number of Clients", fontsize= 12)
plt.ylim(0,8000)
ax.tick_params(axis="x", labelsize= 9.5)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.075, p.get_height()+300))

plt.show()
```



In [149…

```
plt.figure(figsize=(14,4))

ax = sns.barplot(x = "LimitBin", y = "def_pay", data = df, palette = 'rocket', errorbar =

plt.xlabel("Amount of Given Credit", fontsize= 12)
plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)

for p in ax.patches:
    ax.annotate("%.2f" %(p.get_height()), (p.get_x()+0.25, p.get_height()+0.03),fontsize

plt.show()
```




```
-> There is a significant rate of default (over 30%) from customers with 50k or
less of credit limit.
-> Nearly 60 percent of defaulters have lower credit limits, specifically under
100k NT dollars.
-> The higher the limit, the lower is the chance of defaulting.
```

Let us find the relation between Limit Balance Variable with other variables

In [150…

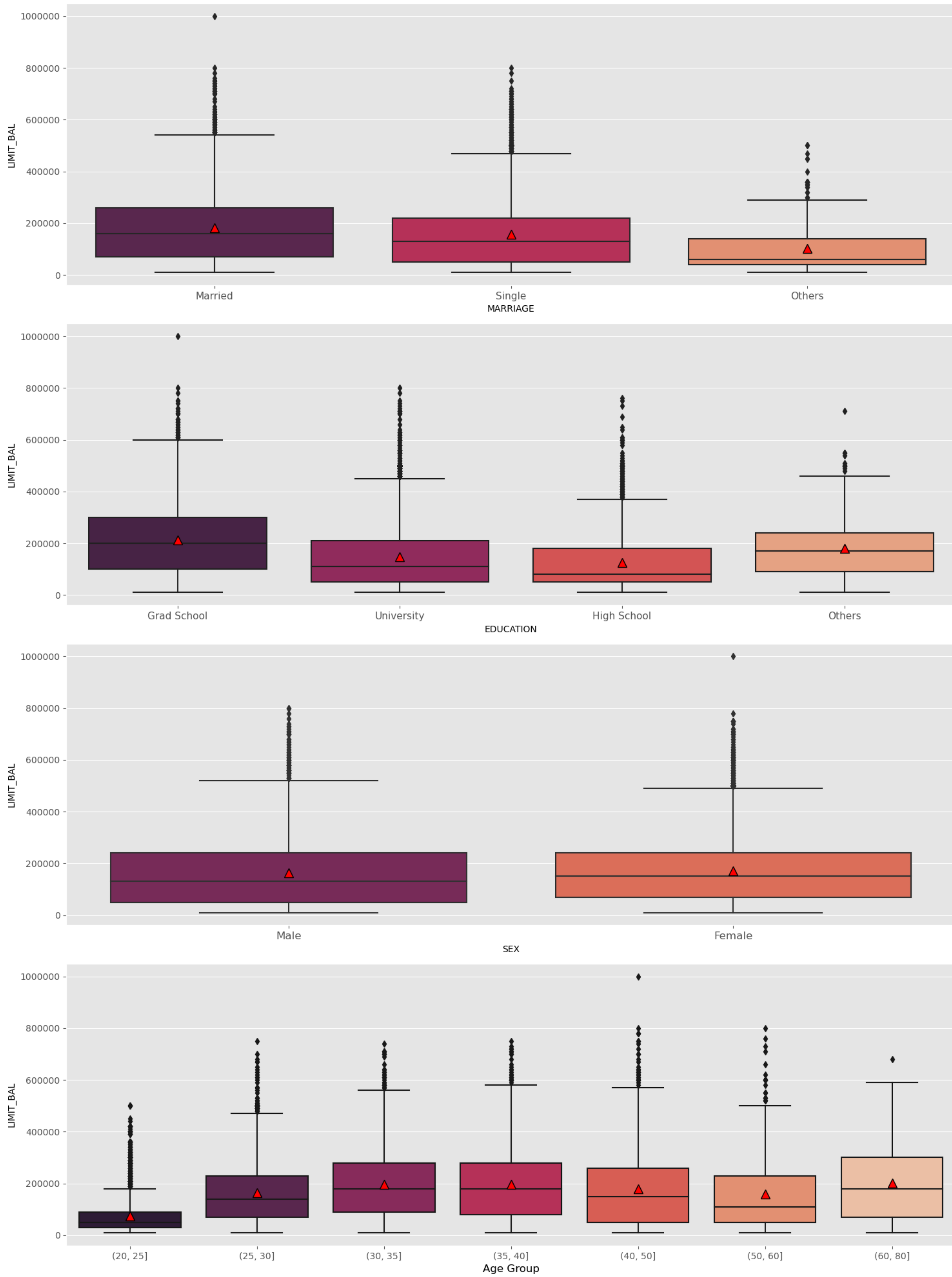
```python
# Group by and calculate mean for each category
mean_by_sex = df.groupby('SEX')['LIMIT_BAL'].mean()
mean_by_education = df.groupby('EDUCATION')['LIMIT_BAL'].mean()
mean_by_marriage = df.groupby('MARRIAGE')['LIMIT_BAL'].mean()
mean_by_age_bin = df.groupby('AgeBin')['LIMIT_BAL'].mean()


print("Mean LIMIT_BAL by SEX:")
print(mean_by_sex)

print('----------------------')
print("\nMean LIMIT_BAL by EDUCATION:")
print(mean_by_education)

print('----------------------')
print("\nMean LIMIT_BAL by MARRIAGE:")
print(mean_by_marriage)

print('----------------------')
print("\nMean LIMIT_BAL by AGE_BIN:")
print(mean_by_age_bin)
```

```
Mean LIMIT_BAL by SEX:
SEX
1    163519.825034
2    170086.462014
Name: LIMIT_BAL, dtype: float64
----------------------

Mean LIMIT_BAL by EDUCATION:
EDUCATION
1    212956.069910
2    147062.437634
3    126550.270490
4    181316.239316
Name: LIMIT_BAL, dtype: float64
----------------------

Mean LIMIT_BAL by MARRIAGE:
MARRIAGE
1    182200.893184
2    156413.660737
3    103076.923077
Name: LIMIT_BAL, dtype: float64
----------------------

Mean LIMIT_BAL by AGE_BIN:
AgeBin
(20, 25]     73763.885301
(25, 30]    164320.918510
(30, 35]    197688.060732
(35, 40]    196780.557250
(40, 50]    179680.213156
(50, 60]    159349.023535
(60, 80]    201617.647059
Name: LIMIT_BAL, dtype: float64
```

In [151…

```python
plt.figure(figsize=(15, 20))

plt.subplot(4, 1, 1)
sns.boxplot(x="MARRIAGE", y="LIMIT_BAL", data=df, palette='rocket', showmeans=True,
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xticks([0, 1, 2], ['Married', 'Single', 'Others'], fontsize=11)
```

```
plt.subplot(4, 1, 2)
sns.boxplot(x="EDUCATION", y="LIMIT_BAL", data=df, palette='rocket', showmeans=True,
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xticks([0, 1, 2, 3], ['Grad School', 'University', 'High School', 'Others'], fontsiz


plt.subplot(4, 1, 3)
sns.boxplot(x="SEX", y="LIMIT_BAL", data=df, palette='rocket', showmeans=True,
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xticks([0, 1], ['Male', 'Female'], fontsize=12)

plt.subplot(4, 1, 4)
sns.boxplot(x="AgeBin", y="LIMIT_BAL", data=df, palette='rocket', showmeans=True, order=
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xlabel("Age Group", fontsize=12)

plt.tight_layout()
plt.show()
```

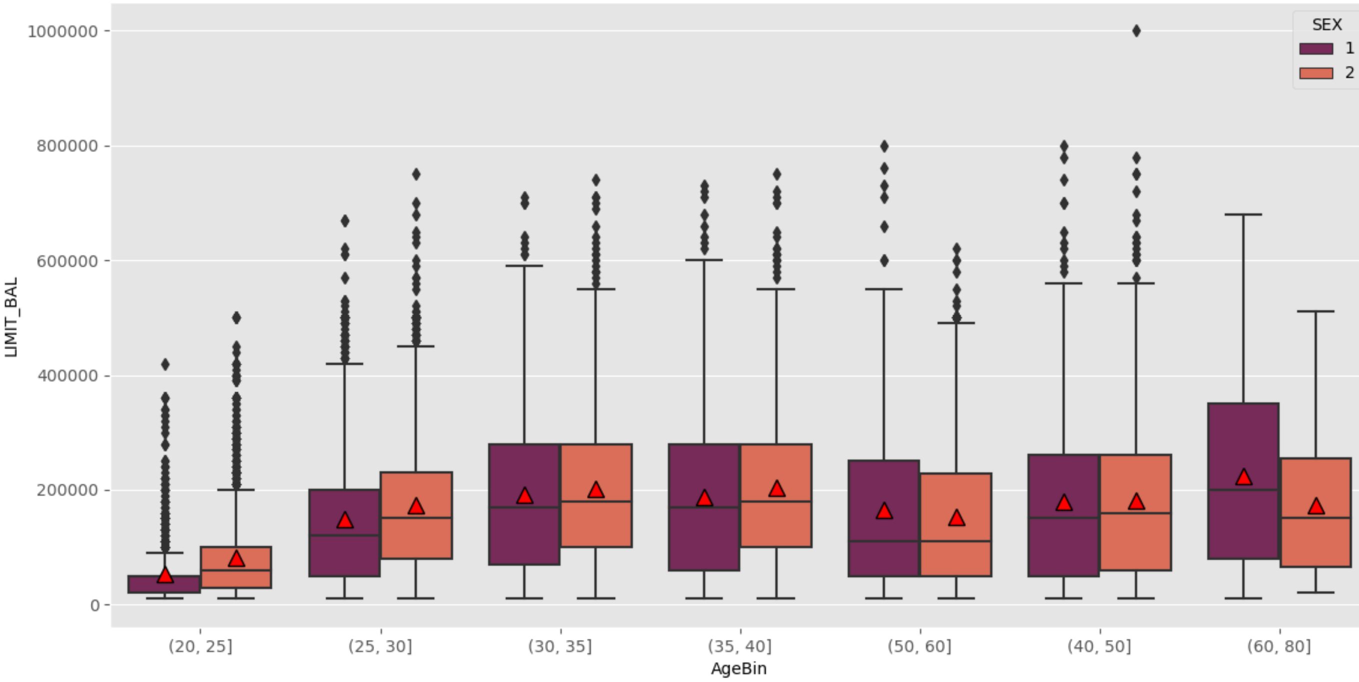


-> The Person with Highest Credit Limit (i.e. 1 million) is a female, married and belongs to 40 to 50 age group

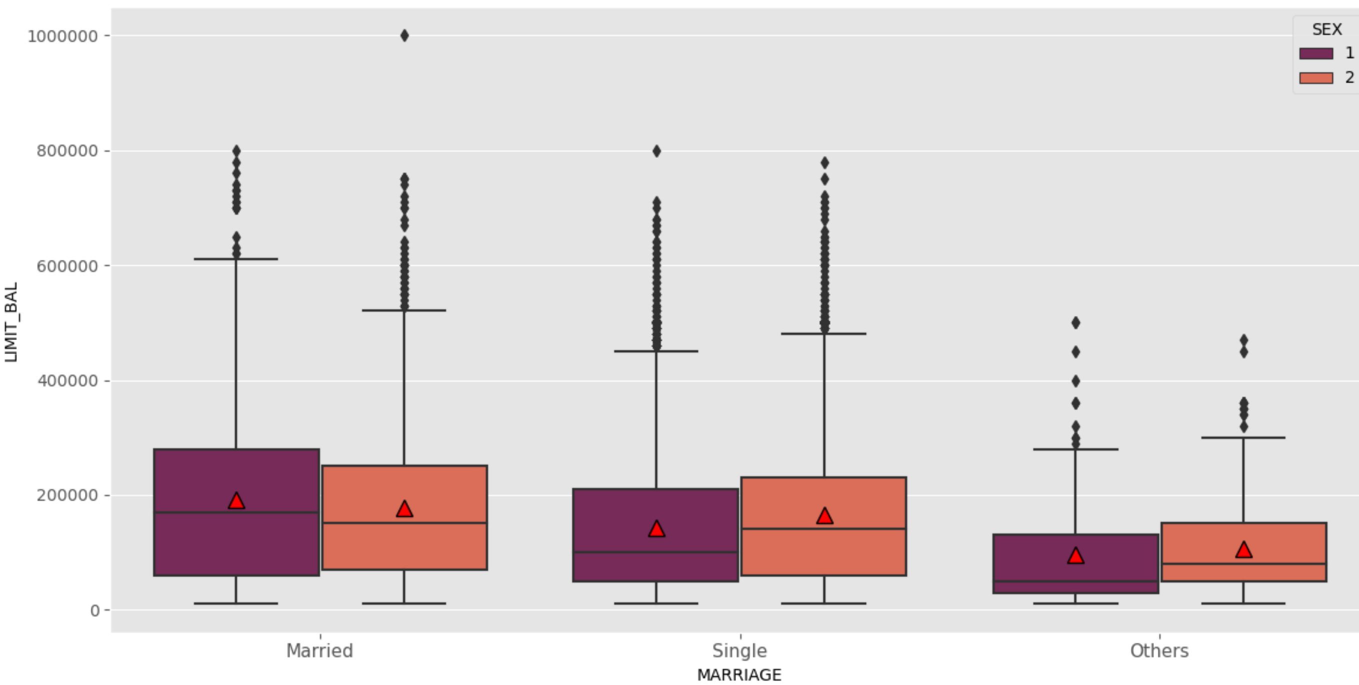
In [152…

```
plt.figure(figsize=(12, 18))
```

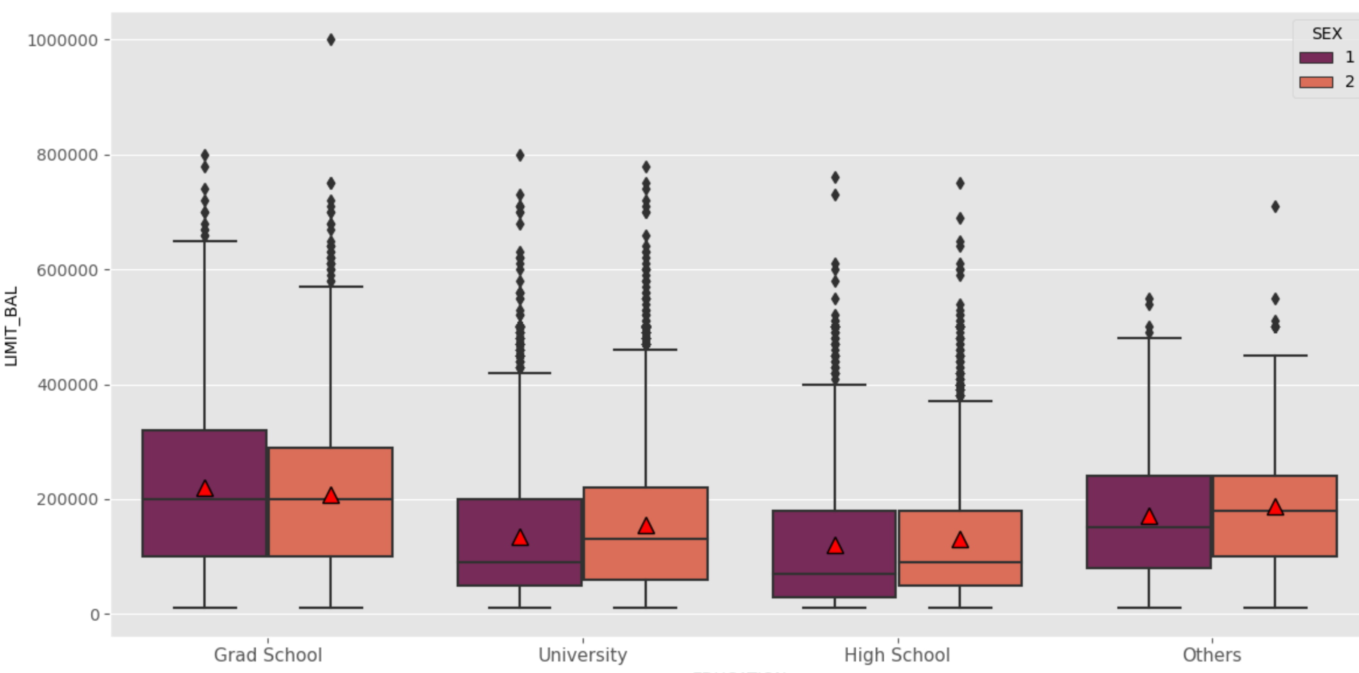
```
# Subplot for AgeBin
plt.subplot(3, 1, 1)
sns.boxplot(x="AgeBin", y="LIMIT_BAL", hue='SEX', data=df, palette='rocket', showmeans=T
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')


# Subplot for Marriage
plt.subplot(3, 1, 2)
sns.boxplot(x="MARRIAGE", y="LIMIT_BAL", hue='SEX', data=df, palette='rocket', showmeans
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xticks([0, 1, 2], ['Married', 'Single','Others'], fontsize=11)


# Subplot for Education
plt.subplot(3, 1, 3)
sns.boxplot(x="EDUCATION", y="LIMIT_BAL", hue='SEX', data=df, palette='rocket', showmean
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xticks([0, 1, 2, 3], ['Grad School', 'University', 'High School', 'Others'], fontsiz


plt.tight_layout()
plt.show()
```



SEX
1
2
LIMIT_BAL
1000000
800000
600000
400000
200000
0
(20, 25]
(25, 30]
(30, 35]
(35, 40]
(50, 60]
(40, 50]
(60, 80]
AgeBin



SEX
1
2
LIMIT_BAL
1000000
800000
600000
400000
200000
0
Married
Single
Others
MARRIAGE



SEX
1
2
LIMIT_BAL
1000000
800000
600000
400000
200000
0
Grad School
University
High School
Others
EDUCATION

The average given credit for women was slightly higher than for men. That still holds up for several combinations of categories, except among customers that:

```
Have a grad school diploma;
Are married;
Are 50+ years old.
```

8.7 | Correlation

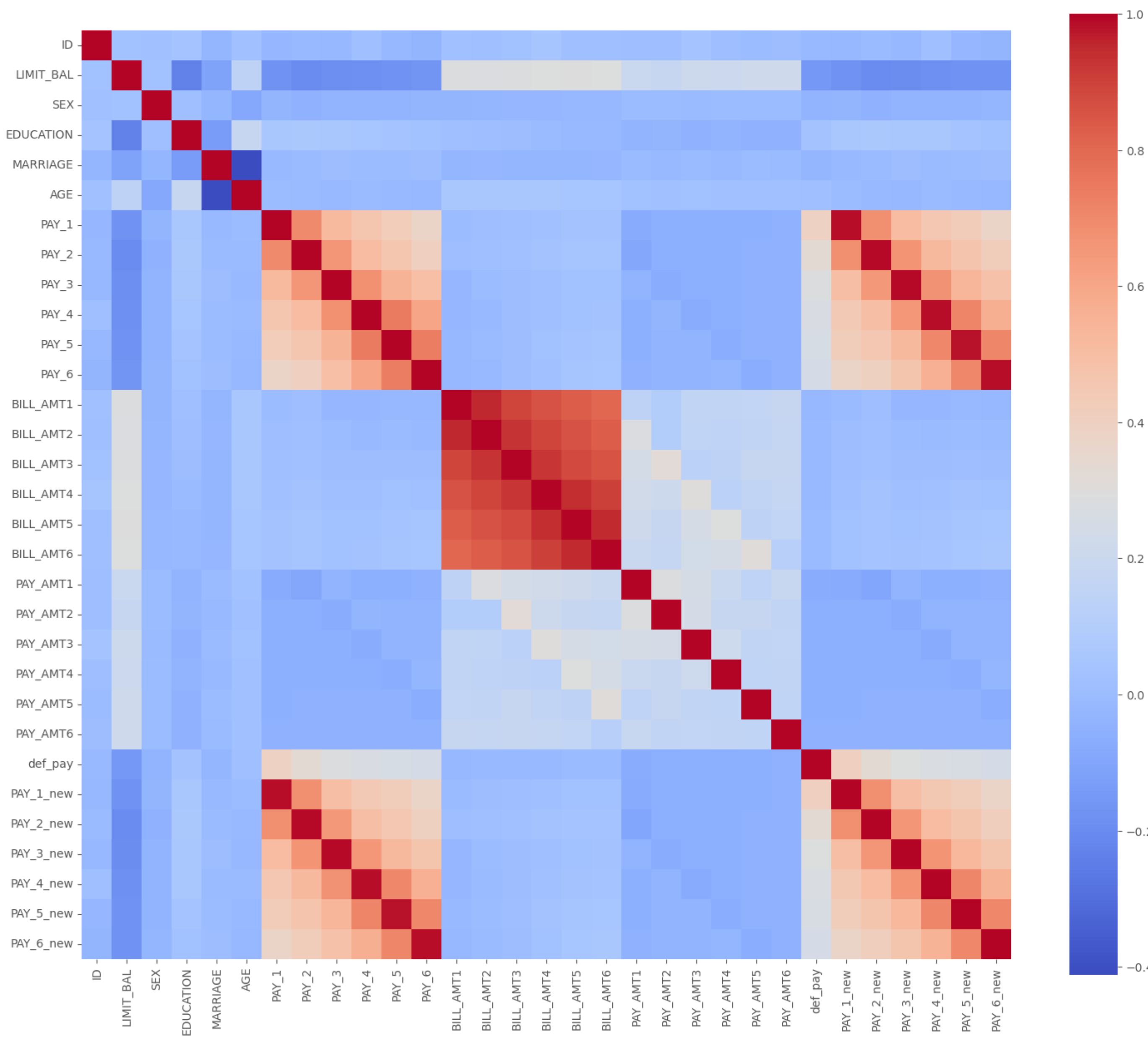
In [153…

```
plt.figure(figsize=(18,15))
sns.heatmap(df.corr(),square=True,cmap='coolwarm')
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_5484\3825302499.py:2: FutureWarning: The defa
ult value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only to sile
nce this warning.
  sns.heatmap(df.corr(),square=True,cmap='coolwarm')
```

Out[153]:

```
<Axes: >
```



```
-> The heatmat shows that features are correlated with each other
(collinearity), such us like PAY_0,2,3,4,5,6 and BILL_AMT1,2,3,4,5,6. In those
cases, the correlation is positive.
```

9 | Analysis Summary

There are 30,000 credit card clients.

The average value for the amount of credit card limit is 167,484 NT dollars. The standard deviation is 129,747 NT dollars, ranging from 10,000 to 1M NT dollars.

Education level is mostly graduate school and university.

Most of the clients are either married or single (less frequent the other status).

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.

The following is the behaviour of dataset columns with default column:

Repayment Behavior:

Individuals with a history of payment delays for more than 4 months have a significantly high chance of default, approximately 70%.

Bill Statement:

Individuals with negative bill statements (credit balance) are less likely to default

Previous Payment Amounts:

Individuals with very low previous payment amounts, nearly 0, have a higher likelihood of default, around 30%.

Education Level:

As the education level decreases, the limit balance also decreases, and the chance of default increases

Marital Status:

Individuals with marital status "Others" (possibly divorced) have a notably higher chance of default, approximately 30

Age Group:

People belonging to the age group of 20 to 25 and above 50 have a higher likelihood of default, around 27%.

Credit Limit:

Individuals with higher credit limits are less prone to default, while those with credit limits below 50k dollars have a high likelihood of default, almost 32%

10 | Feature Engineering

In [154…

```
data = df.drop(['ID','PAY_1_new', 'PAY_2_new', 'PAY_3_new',
       'PAY_4_new', 'PAY_5_new', 'PAY_6_new', 'PAY_AMT1_bin', 'PAY_AMT2_bin',
       'PAY_AMT3_bin', 'PAY_AMT4_bin', 'PAY_AMT5_bin', 'PAY_AMT6_bin',
       'BILL_AMT1_bin', 'BILL_AMT2_bin', 'BILL_AMT3_bin', 'BILL_AMT4_bin',
       'BILL_AMT5_bin', 'BILL_AMT6_bin','AgeBin','LimitBin'],axis=1)
```

In [155…

```
data[['SEX','MARRIAGE','EDUCATION']] = data[['SEX','MARRIAGE','EDUCATION']].astype('obje

#One Hot encoding
data = pd.get_dummies(data)
data.head()
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_5484\3524924288.py:4: FutureWarning: In a fut
ure version, the Index constructor will not infer numeric dtypes when passed object-dtyp
e sequences (matching Series behavior)
  data = pd.get_dummies(data)
C:\Users\DELL\AppData\Local\Temp\ipykernel_5484\3524924288.py:4: FutureWarning: In a fut
ure version, the Index constructor will not infer numeric dtypes when passed object-dtyp
e sequences (matching Series behavior)
  data = pd.get_dummies(data)
C:\Users\DELL\AppData\Local\Temp\ipykernel_5484\3524924288.py:4: FutureWarning: In a fut
ure version, the Index constructor will not infer numeric dtypes when passed object-dtyp
e sequences (matching Series behavior)
  data = pd.get_dummies(data)
```

Out[155]:

	LIMIT_BAL	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4
0	20000.0	24	2	2	0	0	0	0	3913.0	3102.0	689.0	0.0
1	120000.0	26	0	2	0	0	0	2	2682.0	1725.0	2682.0	3272.0
2	90000.0	34	0	0	0	0	0	0	29239.0	14027.0	13559.0	14331.0
3	50000.0	37	0	0	0	0	0	0	46990.0	48233.0	49291.0	28314.0
4	50000.0	57	0	0	0	0	0	0	8617.0	5670.0	35835.0	20940.0

11 | Machine Learning: Classification Models

Spliting the data: train and test

In [156…

```
X = data.drop(['def_pay'], axis=1)
y = data['def_pay']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, stratify=y, rand
```

Decision Tree Classifier

In [157…

```
classifier = DecisionTreeClassifier(max_depth=10, random_state=14)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

accuracy_tree = accuracy_score(y_test, y_pred)
conf_matrix_tree = confusion_matrix(y_test, y_pred)
classification_rep_tree = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_tree}")
print('-------------------')
print("Confusion Matrix:")
print(conf_matrix_tree)
print('-------------------')
print("Classification Report:")
print(classification_rep_tree)
```

```
Accuracy: 0.8053333333333333
-------------------
Confusion Matrix:
[[6512  497]
 [1255  736]]
-------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.93      0.88      7009
           1       0.60      0.37      0.46      1991

    accuracy                           0.81      9000
   macro avg       0.72      0.65      0.67      9000
weighted avg       0.78      0.81      0.79      9000
```

In [158…

```
param_grid = {'max_depth': np.arange(3, 10),
             'criterion' : ['gini','entropy'],
             'max_leaf_nodes': [5,10,20,100],
             'min_samples_split': [2, 5, 10, 20]}

grid_tree = GridSearchCV(DecisionTreeClassifier(), param_grid, cv = 5, scoring= 'accurac
grid_tree.fit(X_train, y_train)

print(grid_tree.best_estimator_)
print(np.abs(grid_tree.best_score_))

accuracy_tree = accuracy_score(y_test, y_pred)
conf_matrix_tree = confusion_matrix(y_test, y_pred)
classification_rep_tree = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_tree}")
print('-------------------')
print("Confusion Matrix:")
print(conf_matrix_tree)
print('-------------------')
print("Classification Report:")
print(classification_rep_tree)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, max_leaf_nodes=10)
0.8221904761904761
Accuracy: 0.8053333333333333
```

```
--------------------
Confusion Matrix:
[[6512  497]
 [1255  736]]
--------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.93      0.88      7009
           1       0.60      0.37      0.46      1991

    accuracy                           0.81      9000
   macro avg       0.72      0.65      0.67      9000
weighted avg       0.78      0.81      0.79      9000
```

Logistic Regression

In [159…

```python
# Creating a Logistic Regression model
logreg_model = LogisticRegression()

# Training the model
logreg_model.fit(X_train, y_train)

# Making predictions
y_pred = logreg_model.predict(X_test)

# Evaluating the model
accuracy_lr = accuracy_score(y_test, y_pred)
conf_matrix_lr = confusion_matrix(y_test, y_pred)
classification_rep_lr = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_lr}")
print('--------------------')
print("Confusion Matrix:")
print(conf_matrix_lr)
print('--------------------')
print("Classification Report:")
print(classification_rep_lr)
```

```
Accuracy: 0.7788888888888889
--------------------
Confusion Matrix:
[[7009    0]
 [1990    1]]
--------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.78      1.00      0.88      7009
           1       1.00      0.00      0.00      1991

    accuracy                           0.78      9000
   macro avg       0.89      0.50      0.44      9000
weighted avg       0.83      0.78      0.68      9000
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:460: Converg
enceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Hyperparameter Tuning

In [160…

```python
# Define the parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10],           # Regularization parameter
    'penalty': ['l1', 'l2'],          # Regularization type
    'solver': ['liblinear', 'lbfgs'] # Optimization algorithm
}

# Create a Logistic Regression model
logreg_model = LogisticRegression()

# Initialize GridSearchCV
grid_search = GridSearchCV(logreg_model, param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV to the data
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Use the best estimator from GridSearchCV
best_logreg_model = grid_search.best_estimator_

# Make predictions
y_pred = best_logreg_model.predict(X_test_scaled)

# Evaluating the model
accuracy_lr = accuracy_score(y_test, y_pred)
conf_matrix_lr = confusion_matrix(y_test, y_pred)
classification_rep_lr = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_lr}")
print('--------------------')
print("Confusion Matrix:")
print(conf_matrix_lr)
print('--------------------')
print("Classification Report:")
print(classification_rep_lr)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:425: Fi
tFailedWarning:
20 fits failed out of a total of 80.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='ra
ise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\DELL\anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\DELL\anaconda3\lib\site-packages\sklearn\base.py", line 1151, in wrappe
r
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\DELL\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", li
```

```
ne 1168, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\DELL\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", li
ne 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:976: UserWa
rning: One or more of the test scores are non-finite: [0.81952381        nan 0.81947619
0.81842857 0.81947619        nan
 0.81966667 0.81947619 0.81971429        nan 0.81985714 0.81980952
 0.81980952        nan 0.81985714 0.81980952]
  warnings.warn(
Best Parameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Best Score: 0.8198571428571428
Accuracy: 0.814
-------------------
Confusion Matrix:
[[6687  322]
 [1352  639]]
-------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      7009
           1       0.66      0.32      0.43      1991

    accuracy                           0.81      9000
   macro avg       0.75      0.64      0.66      9000
weighted avg       0.79      0.81      0.79      9000
```

RandomForest Classifier

In [161…

```
# Create a Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # You can adjust t

# Fit the model to the scaled training data
rf_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test_scaled)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred)
conf_matrix_rf = confusion_matrix(y_test, y_pred)
classification_rep_rf = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_rf}")
print('---------------------')
print("Confusion Matrix:")
print(conf_matrix_rf)
print('---------------------')
print("Classification Report:")
print(classification_rep_rf)
```

```
Accuracy: 0.8137777777777778
---------------------
Confusion Matrix:
[[6600  409]
 [1267  724]]
---------------------
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      7009
           1       0.64      0.36      0.46      1991

    accuracy                           0.81      9000
   macro avg       0.74      0.65      0.68      9000
weighted avg       0.79      0.81      0.79      9000
```

Hyperparameter Tuning

In [162…

```
# Create a Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the GridSearchCV object
grid_rf = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the model to the scaled training data
grid_rf.fit(X_train_scaled, y_train)

# Get the best parameters and the best estimator from the grid search
best_params = grid_rf.best_params_
best_rf_model = grid_rf.best_estimator_

# Make predictions on the test set
y_pred = best_rf_model.predict(X_test_scaled)

print("Best Parameters:", best_params)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred)
conf_matrix_rf = confusion_matrix(y_test, y_pred)
classification_rep_rf = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_rf}")
print('---------------------')
print("Confusion Matrix:")
print(conf_matrix_rf)
print('---------------------')
print("Classification Report:")
print(classification_rep_rf)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_est
imators': 100}
Accuracy: 0.8164444444444444
---------------------
Confusion Matrix:
[[6665  344]
 [1308  683]]
---------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7009
           1       0.67      0.34      0.45      1991
```

```
    accuracy                           0.82      9000
   macro avg       0.75      0.65      0.67      9000
weighted avg       0.80      0.82      0.79      9000
```

Selecting Best Model

Decision Tree Model has given the highest accuracy score by tuning. Let us create the model using the best parameters

In [165…

```
classifier = DecisionTreeClassifier(criterion='gini', max_depth=3, max_leaf_nodes=10,ran
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

accuracy_tree = accuracy_score(y_test, y_pred)
conf_matrix_tree = confusion_matrix(y_test, y_pred)
classification_rep_tree = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_tree}")
print('-------------------')
print("Confusion Matrix:")
print(conf_matrix_tree)
print('-------------------')
print("Classification Report:")
print(classification_rep_tree)
```

```
Accuracy: 0.8177777777777778
-------------------
Confusion Matrix:
[[6649  360]
 [1280  711]]
-------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7009
           1       0.66      0.36      0.46      1991

    accuracy                           0.82      9000
   macro avg       0.75      0.65      0.68      9000
weighted avg       0.80      0.82      0.80      9000
```

Thus, DecisionTree Classifier Model is created with accuracy of nearly equal to 82 percent

Creating Pickle file

In [68]:

```
import pickle
```

In [69]:

```
pickle.dump(classifier, open('model_1.pkl', 'wb'))
```

In [70]:

```
with open('model_2.pkl', 'wb') as model_file:
    pickle.dump(classifier, model_file, protocol=3)
```