TURNKEY LENDER CREDIT CARD DEFAULTS

Task-1

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 TurnKey Lender. The idea is to use this dataset to improve basic skills of data cleaning, data analysis and data visualization.

2) Business Objectives

The objective of this project is to identify factors that contribute to credit card defaults, which can help the credit card company to reduce losses, mitigate risks, and improve customer satisfaction. By gaining insights into the data and understanding the relationships between variables, the credit card company can make informed decisions to improve its business operations, marketing efforts, and customer targeting.

3) About Dataset

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients from April 2005 to September 2005.

Variables

ID: ID of each client.

AMT: Amount of given credit in dollars (includes individual and family/supplementary credit.

GENDER: Gender (1=male, 2=female).

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

MARITAL STATUS: Marital status (1=married, 2=single, 3=others).

AGE: Age in years.

REPAY_SEP: Repayment status in September, 2005 (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).

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

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

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

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

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

AMTBILL_SEP: Amount of bill statement in September, 2005.

AMTBILL_AUG: Amount of bill statement in August, 2005.

AMTBILL_JUL: Amount of bill statement in July, 2005.

AMTBILL_JUN: Amount of bill statement in June, 2005.

AMTBILL_MAY: Amount of bill statement in May, 2005.

AMTBILL_APR: Amount of bill statement in April, 2005.

PRE_SEP: Amount of previous payment in September, 2005.

PRE_AUG: Amount of previous payment in August, 2005.

PRE_JUL: Amount of previous payment in July, 2005.

PRE_JUN: Amount of previous payment in June, 2005.

PRE_MAY: Amount of previous payment in May, 2005.

PRE_APR: Amount of previous payment in April, 2005.

DEF_AMT: Default payment (1=yes, 0=no).

4) importing all important package....

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
#load data into pandas dataframe..
df = pd.read_csv("Dataset.csv")
df.head()
```

Out[2]:

	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY_JUL	REPAY_JUN	...	AM
0	1	20000.0	2	2.0	1	24	2	2	0	0	...	
1	2	120000.0	2	2.0	2	26	0	2	0	0	...	
2	3	90000.0	2	2.0	2	34	0	0	0	0	...	
3	4	50000.0	2	2.0	1	37	0	0	0	0	...	
4	5	50000.0	1	2.0	1	57	0	0	0	0	...	

5 rows × 25 columns

5) Data Exploration

In [3]:
```
#information of dataset..
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   AMT             30000 non-null  float64
 2   GENDER          30000 non-null  int64  
 3   EDUCATION       29986 non-null  float64
 4   MARITAL STATUS  30000 non-null  int64  
 5   AGE             30000 non-null  int64  
 6   REPAY_SEP       30000 non-null  int64  
 7   REPAY_AUG       30000 non-null  int64  
 8   REPAY_JUL       30000 non-null  int64  
 9   REPAY_JUN       30000 non-null  int64  
 10  REPAY_MAY       30000 non-null  int64  
 11  REPAY_APR       30000 non-null  int64  
 12  AMTBILL_SEP     30000 non-null  float64
 13  AMTBILL_AUG     30000 non-null  float64
 14  AMTBILL_JUL     30000 non-null  float64
 15  AMTBILL_JUN     30000 non-null  float64
 16  AMTBILL_MAY     30000 non-null  float64
 17  AMTBILL_APR     30000 non-null  float64
 18  PRE_SEP         30000 non-null  float64
 19  PRE_AUG         30000 non-null  float64
 20  PRE_JUL         30000 non-null  float64
 21  PRE_JUN         30000 non-null  float64
 22  PRE_MAY         30000 non-null  float64
 23  PRE_APR         30000 non-null  float64
 24  DEF_AMT         30000 non-null  int64  
dtypes: float64(14), int64(11)
memory usage: 5.7 MB
```

There were missing values in the education column data that need to be further processed.

In [4]:
```
#calculate these statistics
df.describe()
```

Out[4]:

	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REP
count	30000.000000	30000.000000	30000.000000	29986.000000	30000.000000	30000.000000	30000.000000	3000
mean	15000.500000	167484.322667	1.603733	1.852298	1.555567	35.485500	0.356767	(
std	8660.398374	129747.661567	0.489129	0.781622	0.518833	9.217904	0.760594	(
min	1.000000	10000.000000	1.000000	1.000000	1.000000	21.000000	0.000000	(
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	0.000000	(
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	(
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	(
max	30000.000000	1000000.000000	2.000000	5.000000	3.000000	79.000000	8.000000	ξ

8 rows × 25 columns

In [5]:
```
# Bill Statement description
df[['AMTBILL_SEP','AMTBILL_AUG','AMTBILL_JUL','AMTBILL_JUN','AMTBILL_MAY','AMTBILL_APR']
```

Out[5]:

	AMTBILL_SEP	AMTBILL_AUG	AMTBILL_JUL	AMTBILL_JUN	AMTBILL_MAY	AMTBILL_APR

count	30000.000000	30000.000000	3.000000e+04	30000.000000	30000.000000	30000.000000
mean	51245.526767	49194.349967	4.703570e+04	43287.066733	40327.549367	38907.530300
std	73610.241847	71162.228069	6.932805e+04	64309.431029	60785.288413	59511.193927
min	-9802.000000	-69777.000000	-6.150600e+04	-81334.000000	-81334.000000	-339603.000000
25%	3565.000000	2990.750000	2.684000e+03	2337.000000	1769.500000	1261.000000
50%	22385.500000	21200.000000	2.008900e+04	19052.000000	18104.500000	17075.500000
75%	67091.000000	64006.250000	6.016475e+04	54519.000000	50190.500000	49205.250000
max	964511.000000	983931.000000	1.664089e+06	891586.000000	927171.000000	961664.000000

Their is nagative values in AMTBILL_MONTH column have nagative values. Can Negative values be interpreted as credit? Need to investigate furthur

In [6]:

```
# check the no of unique values in every column for better data understanding
df.nunique()
```

Out[6]:

```
ID                30000
AMT                  81
GENDER                2
EDUCATION             5
MARITAL STATUS        3
AGE                  56
REPAY_SEP             9
REPAY_AUG             9
REPAY_JUL             9
REPAY_JUN             9
REPAY_MAY             8
REPAY_APR             8
AMTBILL_SEP       22617
AMTBILL_AUG       22232
AMTBILL_JUL       21906
AMTBILL_JUN       21408
AMTBILL_MAY       20878
AMTBILL_APR       20454
PRE_SEP            7943
PRE_AUG            7899
PRE_JUL            7518
PRE_JUN            6937
PRE_MAY            6897
PRE_APR            6939
DEF_AMT               2
dtype: int64
```

In [7]:

```
# Looking at all the unique values in features

for i in df.columns:
    print("Unique values in",'',i)
    unique_vals = df[i].unique()
    print(unique_vals)
    print(50*'*')
```

```
Unique values in  ID
[    1     2     3 ... 29998 29999 30000]
**************************************************
Unique values in  AMT
[  20000.  120000.   90000.   50000.  500000.  100000.  140000.  200000.
  260000.  630000.   70000.  250000.  320000.  360000.  180000.  130000.
  450000.   60000.  230000.  160000.  280000.   10000.   40000.  210000.
  150000.  380000.  310000.  400000.   80000.  290000.  340000.  300000.
```

```
   30000.  240000.  470000.  480000.  350000.  330000.  110000.  420000.
  170000.  370000.  270000.  220000.  190000.  510000.  460000.  440000.
  410000.  490000.  390000.  580000.  600000.  620000.  610000.  700000.
  670000.  680000.  430000.  550000.  540000. 1000000.  530000.  710000.
  560000.  520000.  750000.  640000.   16000.  570000.  590000.  660000.
  720000.  327680.  740000.  800000.  760000.  690000.  650000.  780000.
  730000.]
**************************************************
Unique values in  GENDER
[2 1]
**************************************************
Unique values in  EDUCATION
[ 2.  1.  3.  5.  4. nan]
**************************************************
Unique values in  MARITAL STATUS
[1 2 3]
**************************************************
Unique values in  AGE
[24 26 34 37 57 29 23 28 35 51 41 30 49 39 40 27 47 33 32 54 58 22 25 31
 46 42 43 45 56 44 53 38 63 36 52 48 55 60 50 75 61 73 59 21 67 66 62 70
 72 64 65 71 69 68 79 74]
**************************************************
Unique values in  REPAY_SEP
[2 0 1 3 4 8 7 5 6]
**************************************************
Unique values in  REPAY_AUG
[2 0 3 5 7 4 1 6 8]
**************************************************
Unique values in  REPAY_JUL
[0 2 3 4 6 7 1 5 8]
**************************************************
Unique values in  REPAY_JUN
[0 2 3 4 5 7 6 1 8]
**************************************************
Unique values in  REPAY_MAY
[0 2 3 5 4 7 8 6]
**************************************************
Unique values in  REPAY_APR
[0 2 3 6 4 7 8 5]
**************************************************
Unique values in  AMTBILL_SEP
[ 3913.  2682. 29239. ...  1683.   645. 47929.]
**************************************************
Unique values in  AMTBILL_AUG
[ 3102.  1725. 14027. ...  3356. 78379. 48905.]
**************************************************
Unique values in  AMTBILL_JUL
[  689.  2682. 13559. ...  2758. 76304. 49764.]
**************************************************
Unique values in  AMTBILL_JUN
[    0.  3272. 14331. ... 20878. 52774. 36535.]
**************************************************
Unique values in  AMTBILL_MAY
[    0.  3455. 14948. ... 31237.  5190. 32428.]
**************************************************
Unique values in  AMTBILL_APR
[    0.  3261. 15549. ... 19357. 48944. 15313.]
**************************************************
Unique values in  PRE_SEP
[    0.  1518.  2000. ... 10029.  9054. 85900.]
**************************************************
Unique values in  PRE_AUG
[   689.   1000.   1500. ...   2977. 111784.   3526.]
**************************************************
Unique values in  PRE_JUL
[     0.   1000.   1200. ... 349395.   8907.  25128.]
```

```
*************************************************
Unique values in  PRE_JUN
[    0.  1000.  1100. ...  2556. 10115.  8049.]
*************************************************
Unique values in  PRE_MAY
[    0.  1000.  1069. ...  8040.  3319. 52964.]
*************************************************
Unique values in  PRE_APR
[     0.   2000.   5000. ...  70052. 220076.  16080.]
*************************************************
Unique values in  DEF_AMT
[1 0]
*************************************************
```

There were no extra values present in the data, all values in every column were as per the data set description

6) Data Cleaning

In [8]:

```
#  checking the sum of null values in every column...
totalnull_val = df.isnull().sum()
totalnull_val
```

Out[8]:

```
ID                 0
AMT                0
GENDER             0
EDUCATION         14
MARITAL STATUS     0
AGE                0
REPAY_SEP          0
REPAY_AUG          0
REPAY_JUL          0
REPAY_JUN          0
REPAY_MAY          0
REPAY_APR          0
AMTBILL_SEP        0
AMTBILL_AUG        0
AMTBILL_JUL        0
AMTBILL_JUN        0
AMTBILL_MAY        0
AMTBILL_APR        0
PRE_SEP            0
PRE_AUG            0
PRE_JUL            0
PRE_JUN            0
PRE_MAY            0
PRE_APR            0
DEF_AMT            0
dtype: int64
```

In [9]:

```
# calculating the percentage of null values for every individual column….
percentnull_val = (totalnull_val/df.shape[0])*100
percentnull_val
```

Out[9]:

```
ID                0.000000
AMT               0.000000
GENDER            0.000000
EDUCATION         0.046667
MARITAL STATUS    0.000000
AGE               0.000000
REPAY_SEP         0.000000
REPAY_AUG         0.000000
REPAY_JUL         0.000000
REPAY_JUN         0.000000
```

```
REPAY_MAY        0.000000
REPAY_APR        0.000000
AMTBILL_SEP      0.000000
AMTBILL_AUG      0.000000
AMTBILL_JUL      0.000000
AMTBILL_JUN      0.000000
AMTBILL_MAY      0.000000
AMTBILL_APR      0.000000
PRE_SEP          0.000000
PRE_AUG          0.000000
PRE_JUL          0.000000
PRE_JUN          0.000000
PRE_MAY          0.000000
PRE_APR          0.000000
DEF_AMT          0.000000
dtype: float64
```

We can observe that EDUCATION have 0.047% null values Hence our decision of either drop the Null values or imputing them. As the percent null values were very small So we can go ahead and drop null values.

In [10]:
```
df.shape # With null values
```

Out[10]:
```
(30000, 25)
```

In [11]:
```
# droping null values
df = df.dropna()
```

In [12]:
```
df.shape #After removing null values
```

Out[12]:
```
(29986, 25)
```

Dealing with duplicate values ....

In [13]:
```
df.duplicated().sum()
```

Out[13]:
```
0
```

there are no duplicate values in the column

In [14]:
```
#information of dataset..
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29986 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   ID               29986 non-null   int64  
 1   AMT              29986 non-null   float64
 2   GENDER           29986 non-null   int64  
 3   EDUCATION        29986 non-null   float64
 4   MARITAL STATUS   29986 non-null   int64  
 5   AGE              29986 non-null   int64  
 6   REPAY_SEP        29986 non-null   int64  
 7   REPAY_AUG        29986 non-null   int64  
 8   REPAY_JUL        29986 non-null   int64  
 9   REPAY_JUN        29986 non-null   int64  
 10  REPAY_MAY        29986 non-null   int64  
 11  REPAY_APR        29986 non-null   int64  
```

```
 12  AMTBILL_SEP     29986 non-null  float64
 13  AMTBILL_AUG     29986 non-null  float64
 14  AMTBILL_JUL     29986 non-null  float64
 15  AMTBILL_JUN     29986 non-null  float64
 16  AMTBILL_MAY     29986 non-null  float64
 17  AMTBILL_APR     29986 non-null  float64
 18  PRE_SEP         29986 non-null  float64
 19  PRE_AUG         29986 non-null  float64
 20  PRE_JUL         29986 non-null  float64
 21  PRE_JUN         29986 non-null  float64
 22  PRE_MAY         29986 non-null  float64
 23  PRE_APR         29986 non-null  float64
 24  DEF_AMT         29986 non-null  int64
dtypes: float64(14), int64(11)
memory usage: 5.9 MB
```

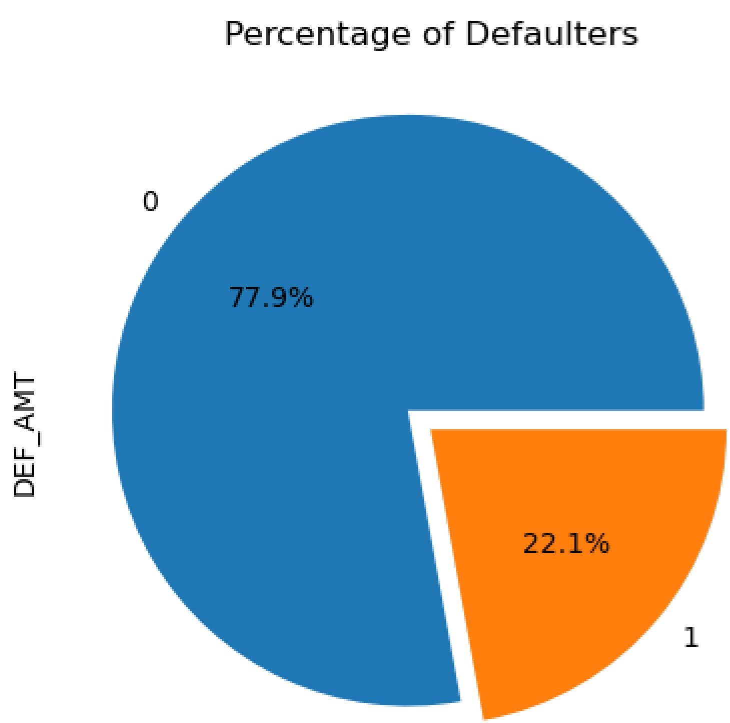
7) Exploratory Data Analysis (EDA)

How many defaulters

In [15]:
```
perc_default = df.DEF_AMT.sum() / len(df.DEF_AMT)
print(f'The percentage of defaulters in the data is {perc_default*100} %')
df['DEF_AMT'].value_counts().plot(kind='pie',explode=[0.1,0],autopct="%1.1f%%")
plt.title('Percentage of Defaulters')
plt.plot()
```

```
The percentage of defaulters in the data is 22.13032748616021 %
```

Out[15]:
```
[]
```

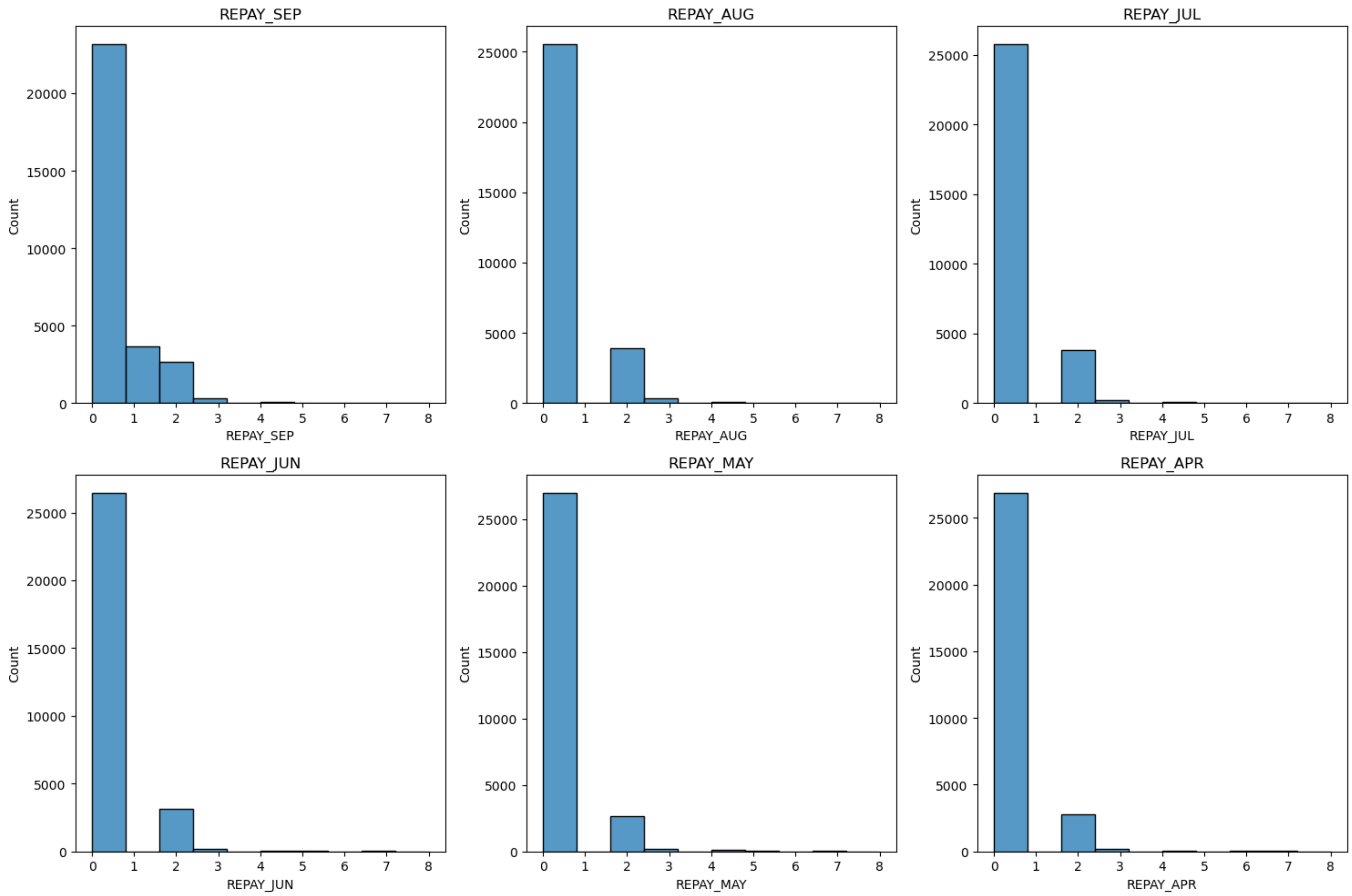


7.1) Repayment status

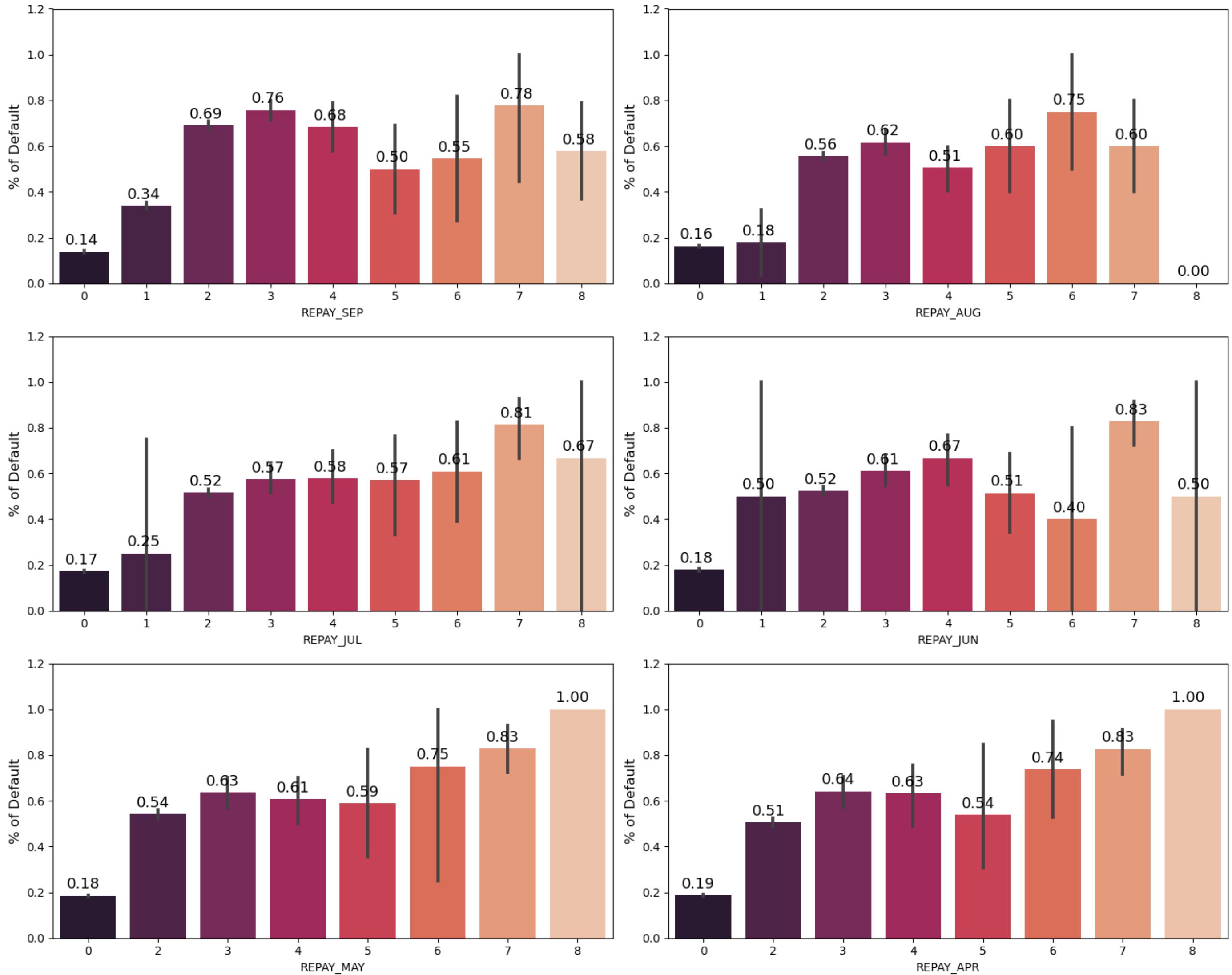
In [16]:
```
def draw_histograms(df, variables, n_rows, n_cols, n_bins):
    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(15, 10))
    for i, var_name in enumerate(variables):
```

```
        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 [17]:
```
late = df[['REPAY_SEP','REPAY_AUG','REPAY_JUL','REPAY_JUN','REPAY_MAY','REPAY_APR']]
draw_histograms(late, late.columns, 2, 3, 10)
```



In [18]:
```
# Set the size of the plot
plt.figure(figsize=(15, 12))


for i,col in enumerate(late):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "DEF_AMT", data = df, palette = 'rocket')
    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

# Adjust subplot layout
plt.tight_layout()
plt.show()
```



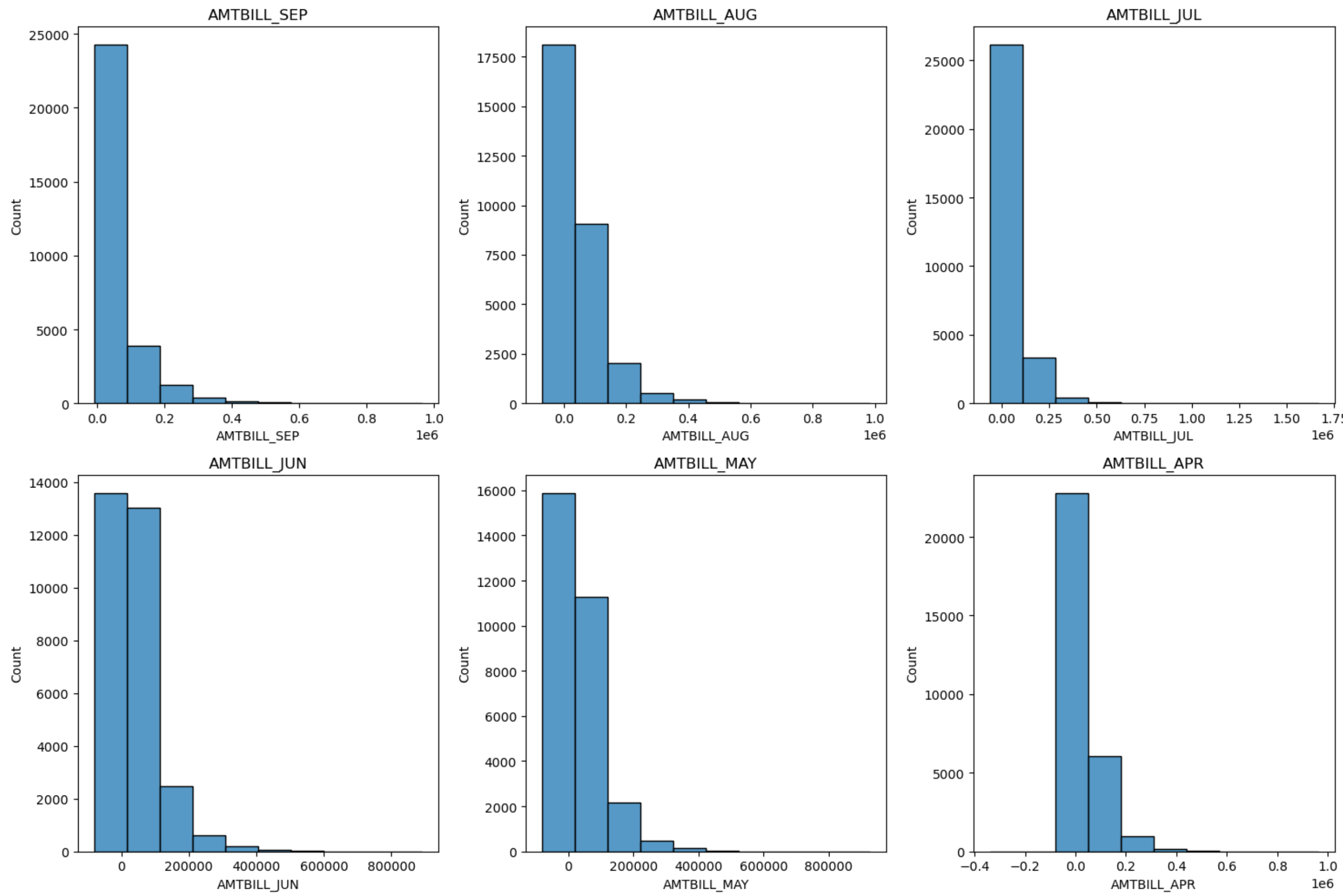
-> Most customers are duly paying their credit card bills. And it's pretty clear that their likelihood of default are much lower than the rest.

-> Credit card holders who consistently delay their payments for >= 2 months are significantly more likely to face defaults.

7.2) Amount of bill statement

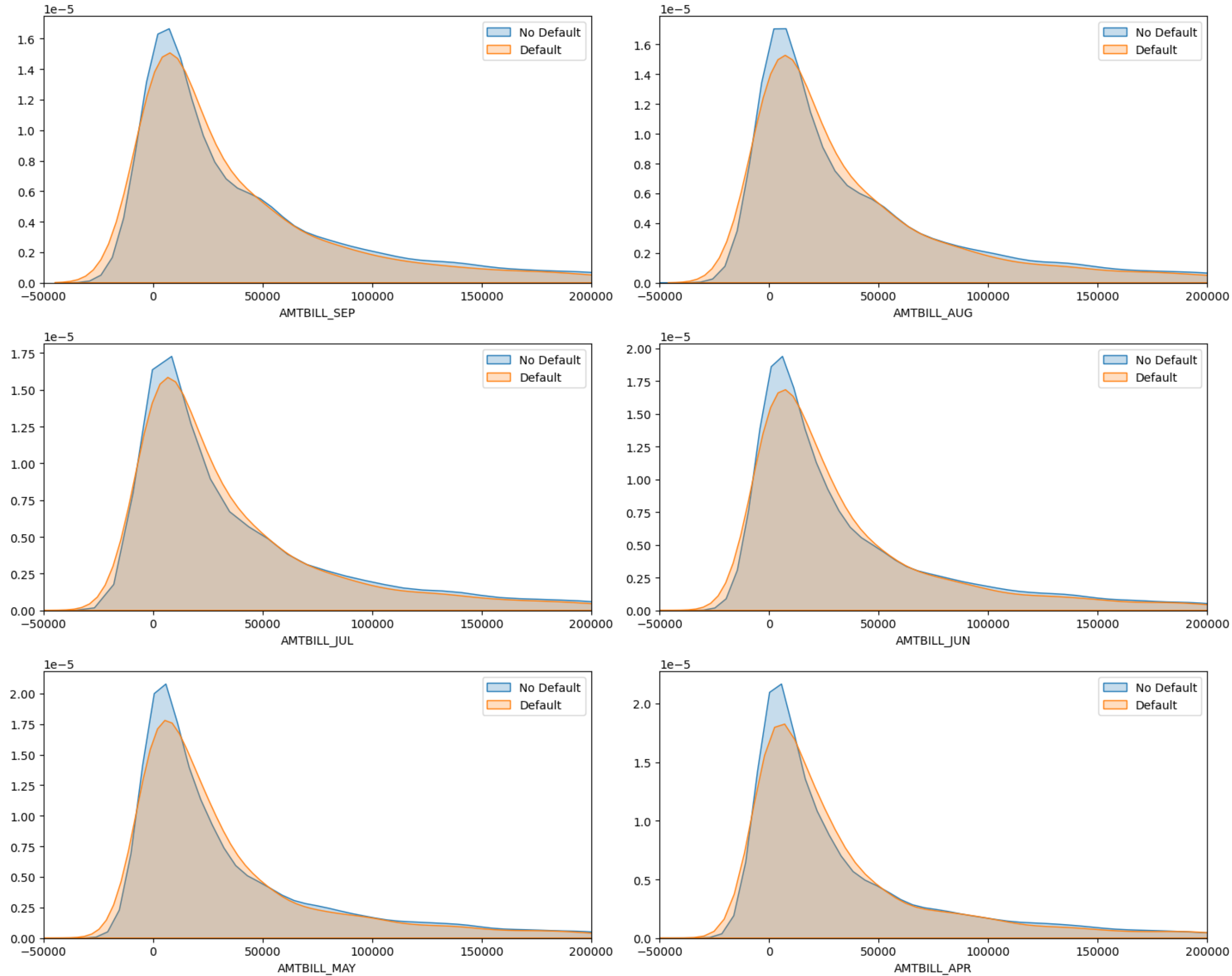
In [19]:

```
late1 = df[['AMTBILL_SEP','AMTBILL_AUG','AMTBILL_JUL','AMTBILL_JUN','AMTBILL_MAY','AMTBI
draw_histograms(late1, late1.columns, 2, 3, 10)
```



In [20]:

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

for i,col in enumerate(late1):
    plt.subplot(3,2,i + 1)
    sns.kdeplot(df.loc[(df['DEF_AMT'] == 0), col], label = 'No Default', fill = True)
    sns.kdeplot(df.loc[(df['DEF_AMT'] == 1), col], label = 'Default', fill = True)
    plt.xlim(-50000,200000)
    plt.ylabel('')
    plt.legend()
    plt.tight_layout()

plt.show()
```



Making bins will give us more clear analysis of this section

In [21]:
```
df['AMTBILL_SEP_bin'] = df['AMTBILL_SEP'].copy()
df['AMTBILL_AUG_bin'] = df['AMTBILL_AUG'].copy()
df['AMTBILL_JUL_bin'] = df['AMTBILL_JUL'].copy()
df['AMTBILL_JUN_bin'] = df['AMTBILL_JUN'].copy()
df['AMTBILL_MAY_bin'] = df['AMTBILL_MAY'].copy()
df['AMTBILL_APR_bin'] = df['AMTBILL_APR'].copy()
```

In [22]:
```
late2 = ['AMTBILL_SEP_bin','AMTBILL_AUG_bin','AMTBILL_JUL_bin','AMTBILL_JUN_bin','AMTBIL
for i, col in enumerate (late2):
    df[col] = pd.cut(df[late2[i]],[-350000,-1,0,25000, 75000, 200000, 2000000])
```

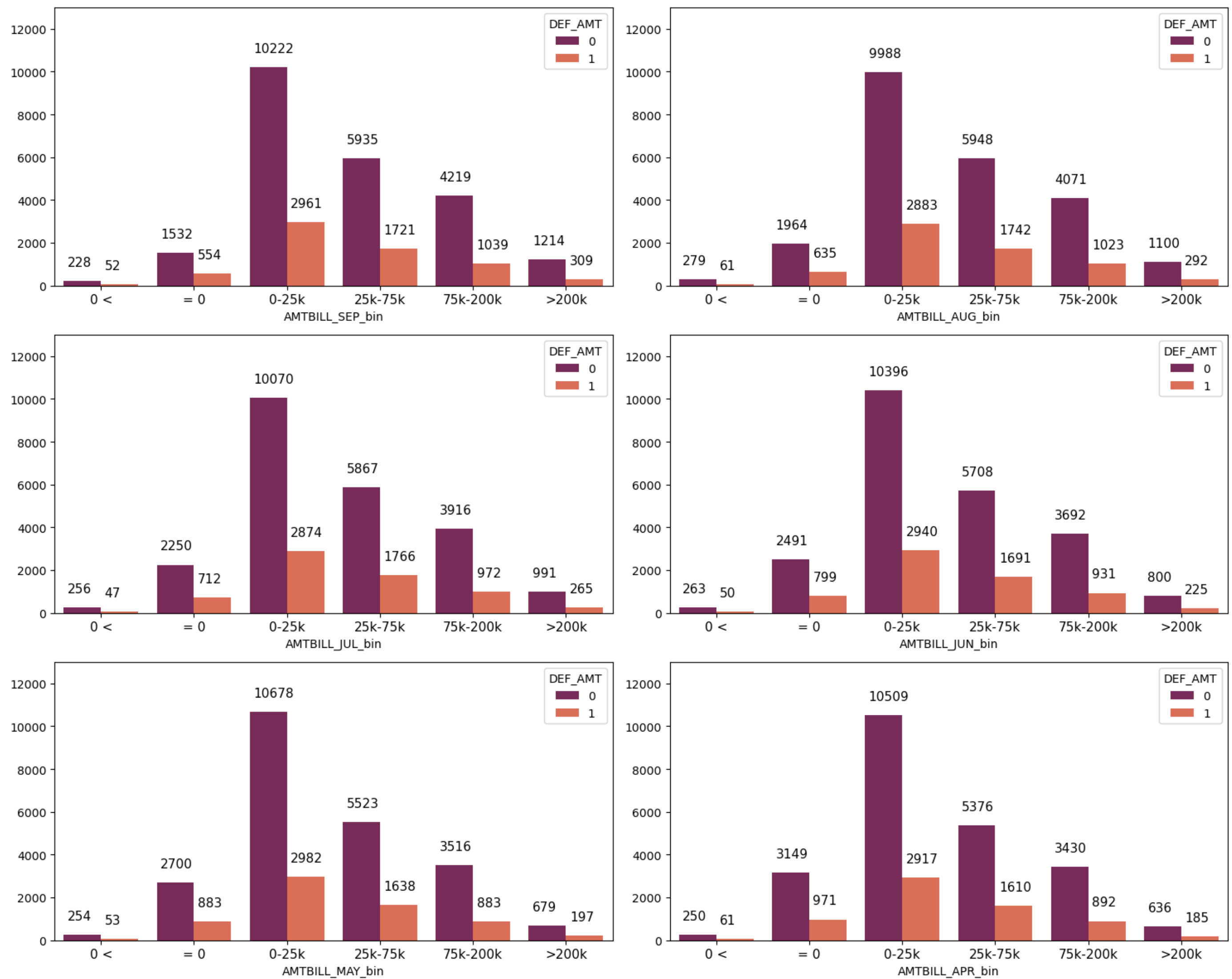
In [23]:
```
df.head()
```

Out[23]:

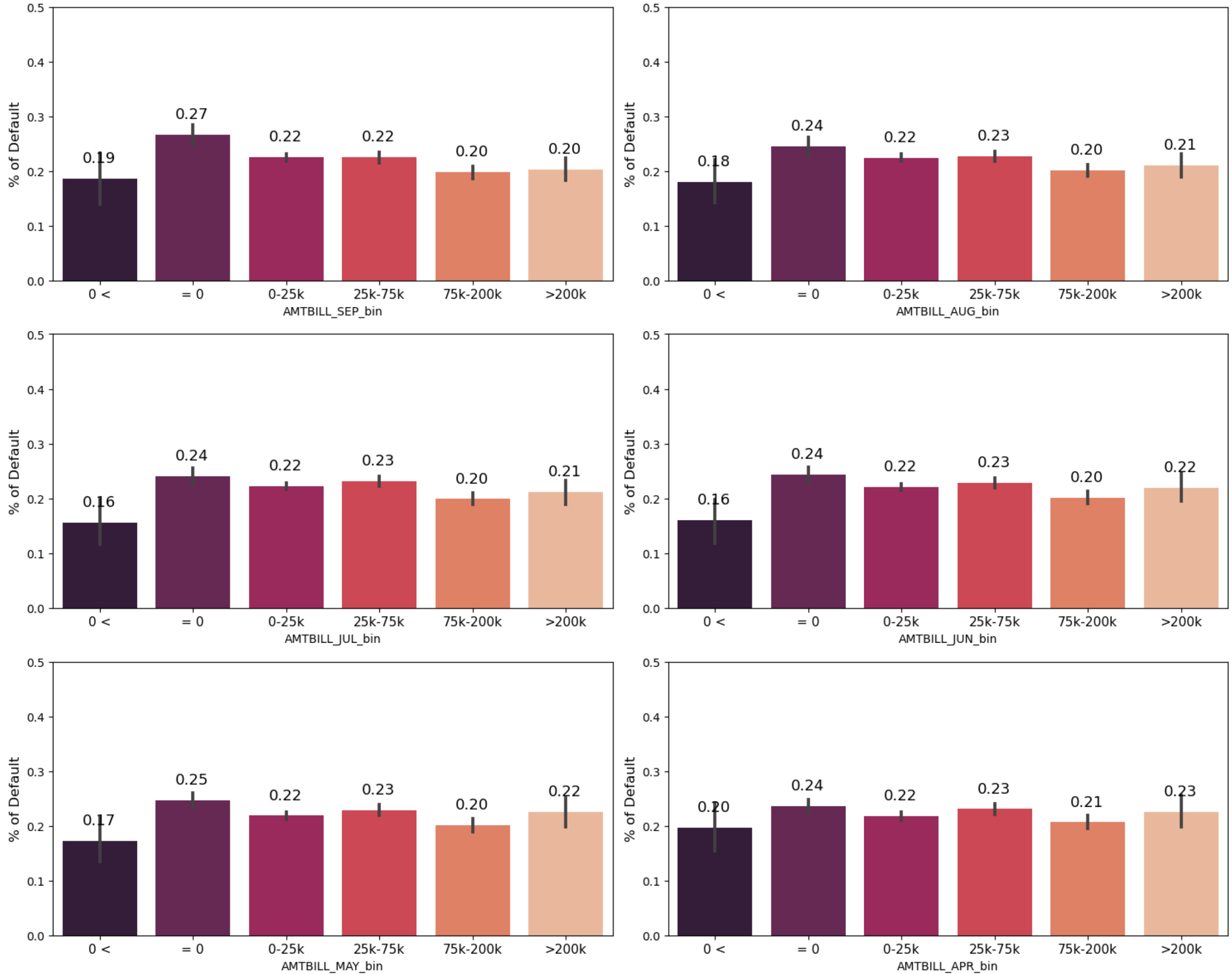
	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY_JUL	REPAY_JUN	...	PR
0	1	20000.0	2	2.0	1	24	2	2	0	0	...	
1	2	120000.0	2	2.0	2	26	0	2	0	0	...	
2	3	90000.0	2	2.0	2	34	0	0	0	0	...	
3	4	50000.0	2	2.0	1	37	0	0	0	0	...	
4	5	50000.0	1	2.0	1	57	0	0	0	0	...	

5 rows × 31 columns

In [24]:
```
plt.figure(figsize=(15, 12))
for i,col in enumerate(late2):
    plt.subplot(3,2,i + 1)
    ax = sns.countplot(data = df, x = col, hue="DEF_AMT", 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 [25]:
```
plt.figure(figsize=(15,12))

for i,col in enumerate(late2):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "DEF_AMT", data = df, palette = 'rocket')
    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.

In [26]:
```
df.head()
```

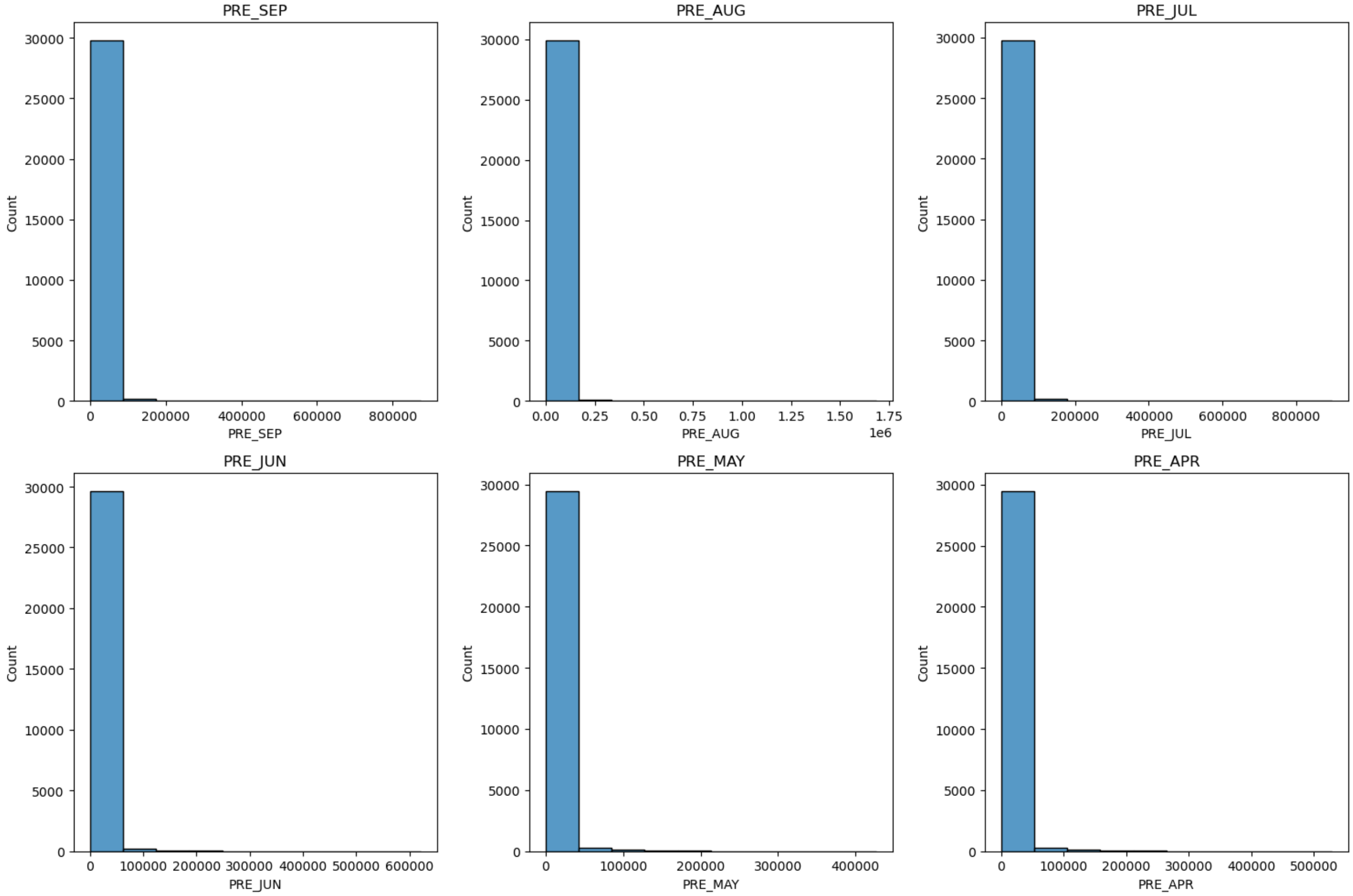
Out[26]:

	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY_JUL	REPAY_JUN	...	PR
0	1	20000.0	2	2.0	1	24	2	2	0	0	...	
1	2	120000.0	2	2.0	2	26	0	2	0	0	...	
2	3	90000.0	2	2.0	2	34	0	0	0	0	...	
3	4	50000.0	2	2.0	1	37	0	0	0	0	...	
4	5	50000.0	1	2.0	1	57	0	0	0	0	...	

5 rows × 31 columns

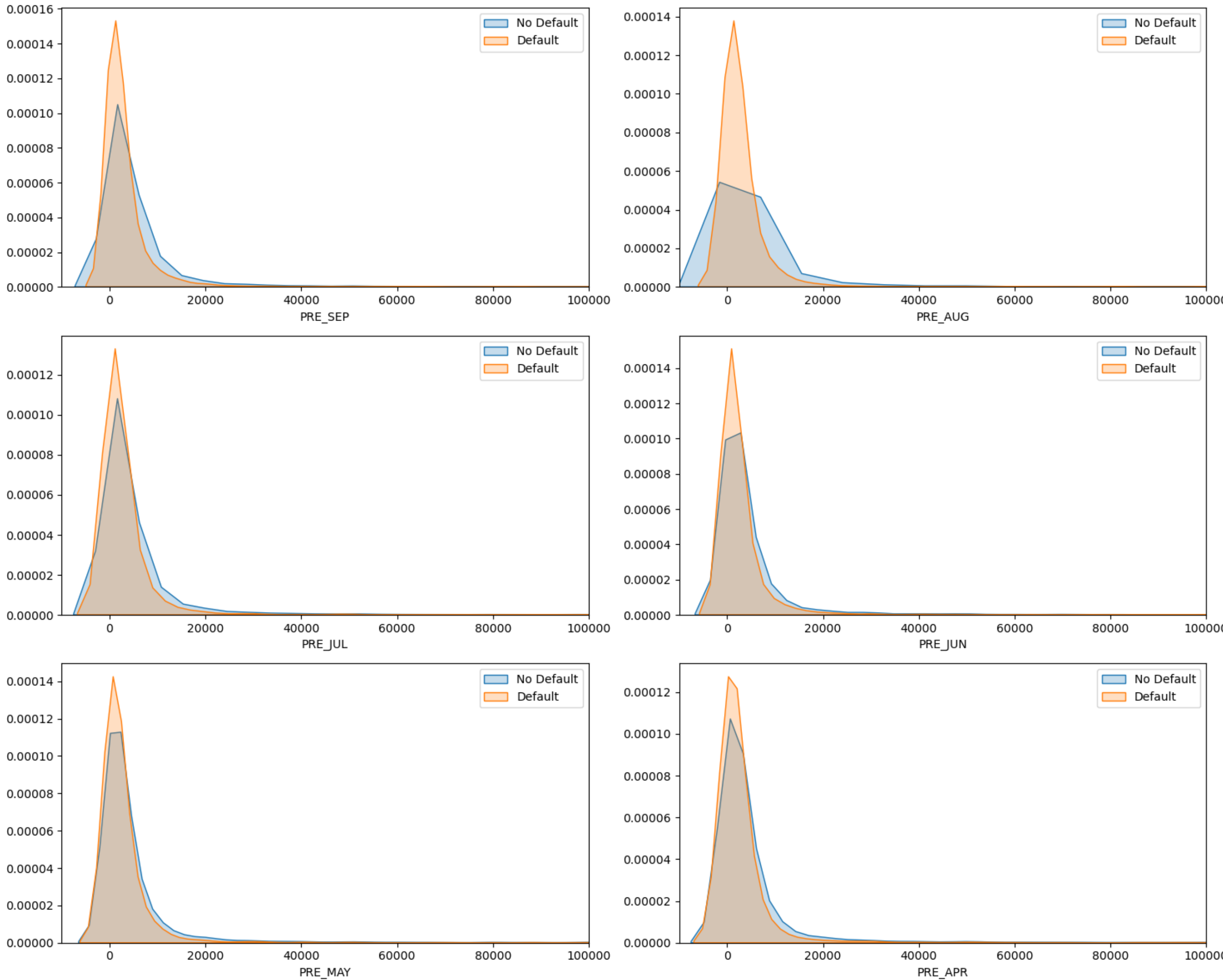
7.3) Amount of previous payment

In [27]:
```
late3 = df[['PRE_SEP','PRE_AUG','PRE_JUL','PRE_JUN','PRE_MAY','PRE_APR']]
draw_histograms(late3, late3.columns, 2, 3, 10)
```



In [28]:

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

for i,col in enumerate(late3):
    plt.subplot(3,2,i + 1)
    sns.kdeplot(df.loc[(df['DEF_AMT'] == 0), col], label = 'No Default', fill = True)
    sns.kdeplot(df.loc[(df['DEF_AMT'] == 1), col], label = 'Default', fill = True)
    plt.xlim(-10000,100000)
    plt.ylabel('')
    plt.legend()
    plt.tight_layout()

plt.show()
```



Let us Create bins for previous payments

In [29]:
```
df['PRE_SEP_bin'] = df['PRE_SEP'].copy()
df['PRE_AUG_bin'] = df['PRE_AUG'].copy()
df['PRE_JUL_bin'] = df['PRE_JUL'].copy()
df['PRE_JUN_bin'] = df['PRE_JUN'].copy()
df['PRE_MAY_bin'] = df['PRE_MAY'].copy()
df['PRE_APR_bin'] = df['PRE_APR'].copy()
```

In [30]:
```
late4 = ['PRE_SEP_bin','PRE_AUG_bin','PRE_JUL_bin','PRE_JUN_bin','PRE_MAY_bin','PRE_APR_
for i, col in enumerate (late4):
    df[col] = pd.cut(df[late4[i]],[-350000,-1,0,25000, 75000, 200000, 2000000])
```

In [31]:
```
df.head()
```

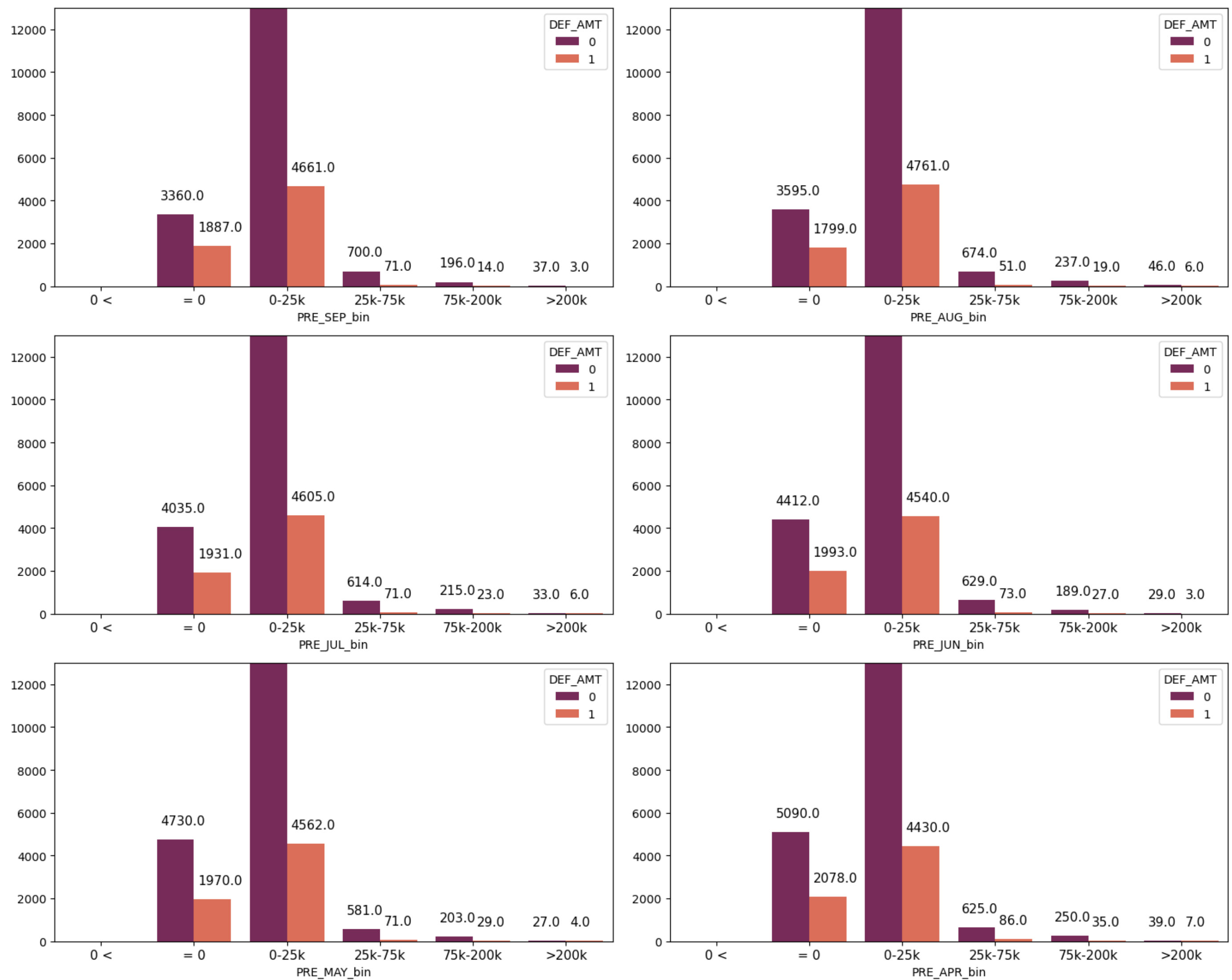
Out[31]:

	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY_JUL	REPAY_JUN	...	AM
0	1	20000.0	2	2.0	1	24	2	2	0	0	...	
1	2	120000.0	2	2.0	2	26	0	2	0	0	...	
2	3	90000.0	2	2.0	2	34	0	0	0	0	...	
3	4	50000.0	2	2.0	1	37	0	0	0	0	...	
4	5	50000.0	1	2.0	1	57	0	0	0	0	...	

5 rows × 37 columns

In [32]:

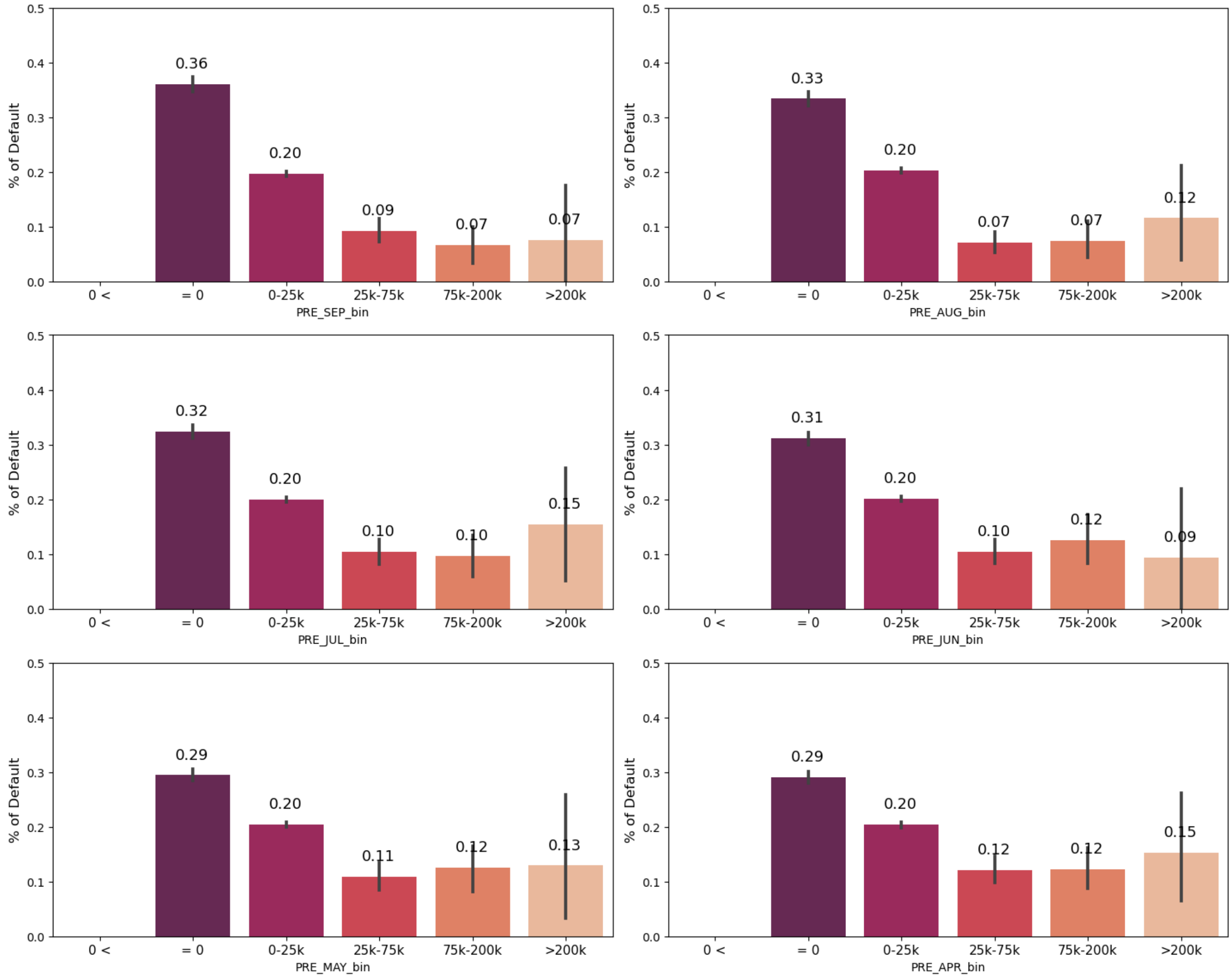
```
plt.figure(figsize=(15, 12))
for i,col in enumerate(late4):
    plt.subplot(3,2,i + 1)
    ax = sns.countplot(data = df, x = col, hue="DEF_AMT", palette = 'rocket')
    plt.ylim(0,13000)
    plt.ylabel('')
    plt.xticks([0,1,2,3,4,5],['0 <', '= 0', '0-25k', '25k-75k', '75k-200k', '>200k'], fo
    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 [33]:

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

for i,col in enumerate(late4):
    plt.subplot(3,2,i + 1)
    ax = sns.barplot(x = col, y = "DEF_AMT", data = df, palette = 'rocket')
    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()
```



-> 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.

7.4) Categorical Columns (GENDER, EDUCATION, MARITAL STATUS)

GENDER: Gender (1=male, 2=female).

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

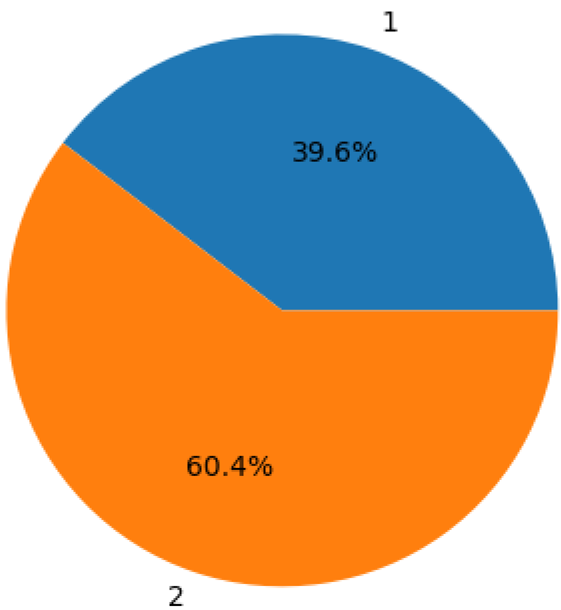
MARITAL STATUS: Marital status (1=married, 2=single, 3=others).

DEF_AMT: Default payment (1=yes, 0=no).

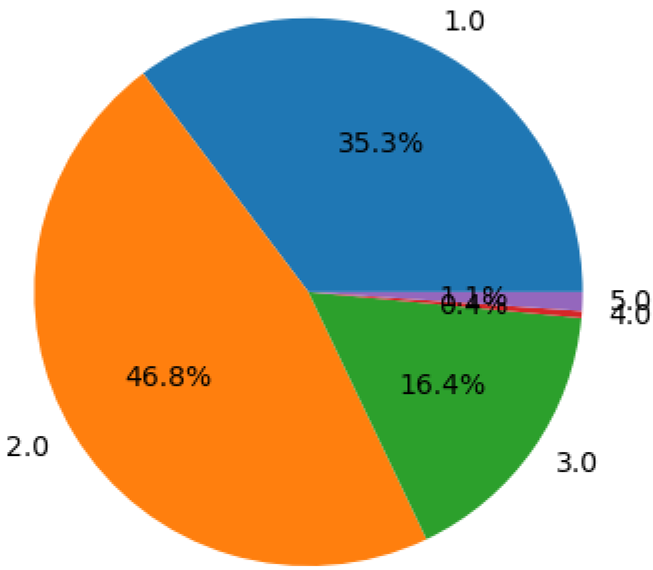
In [34]:
```
def show_value_counts(col):
    print(col)
    value_counts = df[col].value_counts()
    percentage = value_counts / len(df) * 100
    result_df = pd.DataFrame({'Value': value_counts.index, 'Count': value_counts, 'Perce
    result_df = result_df.sort_values(by='Value')
    print(result_df)
    print('---------------------------')
    generate_pie_plot(result_df)
```

```python
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('GENDER')
show_value_counts('EDUCATION')
show_value_counts('MARITAL STATUS')
```

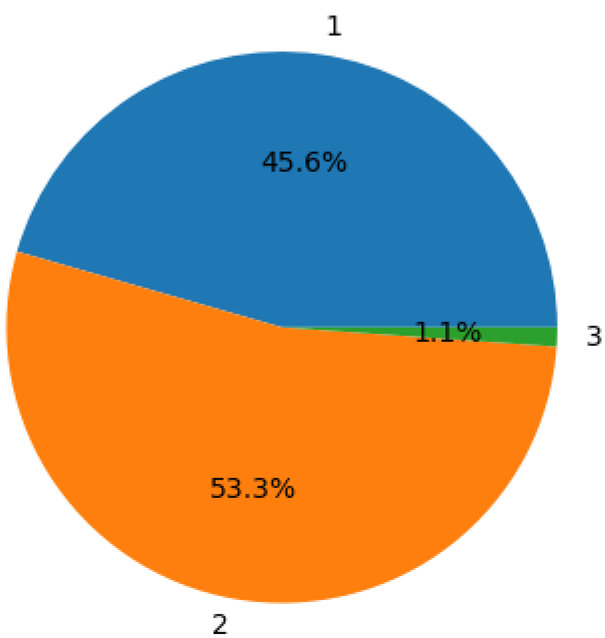
```
GENDER
   Value  Count  Percentage
1      1  11880   39.618489
2      2  18106   60.381511
--------------------------
```




```
EDUCATION
     Value  Count  Percentage
1.0    1.0  10585   35.299807
2.0    2.0  14030   46.788501
3.0    3.0   4917   16.397652
4.0    4.0    123    0.410191
5.0    5.0    331    1.103848
--------------------------
```



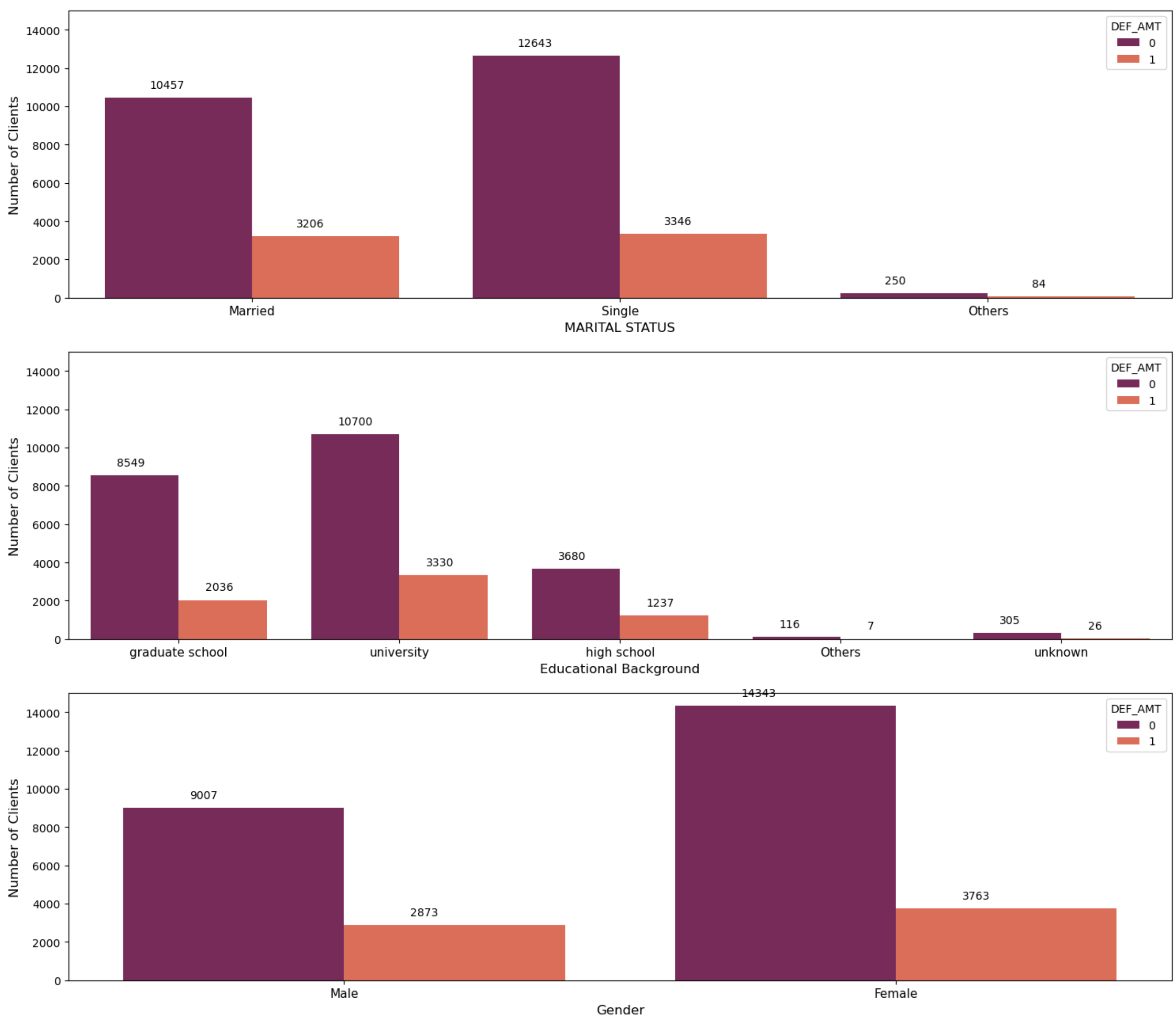

```
MARITAL STATUS
   Value  Count  Percentage
1      1  13663   45.564597
2      2  15989   53.321550
3      3    334    1.113853
--------------------------
```



-> There are significantly more women than men -> Most of Credit Card holders have university and graduate level education.

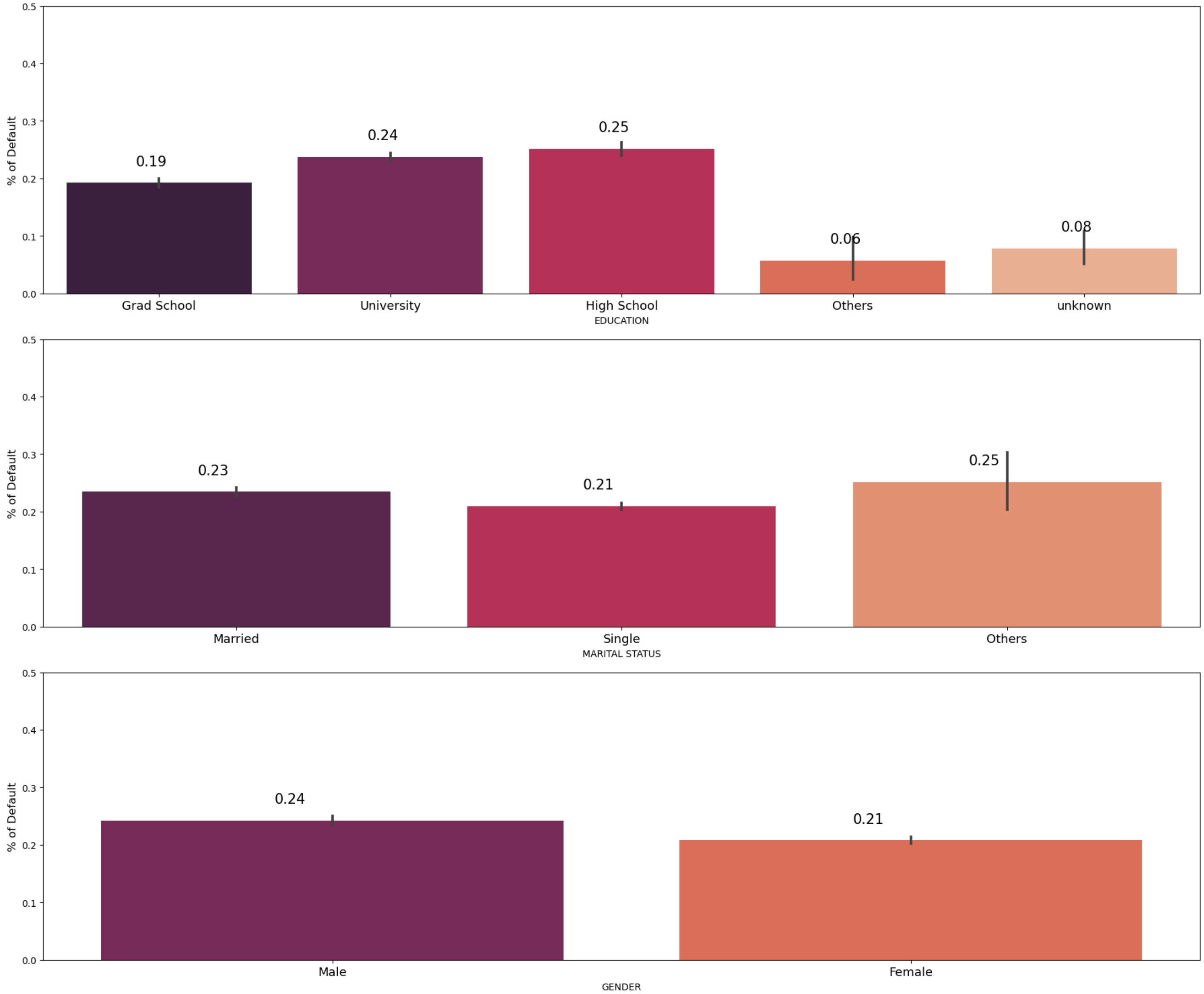
In [35]:
```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(15, 13))



# Count plot for MARITAL STATUS
ax1 = sns.countplot(data=df, x='MARITAL STATUS', hue='DEF_AMT', palette='rocket', ax=axe
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_AMT', 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, 3,4])
ax2.set_xticklabels(['graduate school', 'university', 'high school', 'Others','unknown']
for p in ax2.patches:
    ax2.annotate(int(p.get_height()), (p.get_x() + 0.12, p.get_height() + 500))


# Count plot for GENDER
ax3 = sns.countplot(data=df, x='GENDER', hue='DEF_AMT', 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()
```



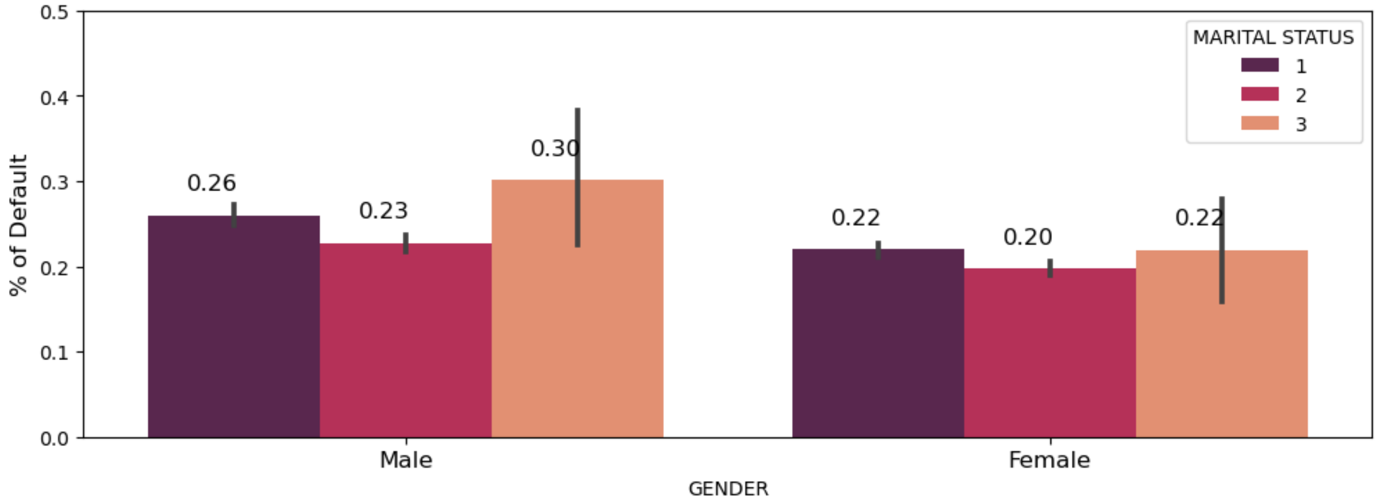
In [36]:

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

# Bar plot for EDUCATION
ax1 = sns.barplot(x="EDUCATION", y="DEF_AMT", data=df, palette='rocket', ax=axes[0])
ax1.set_ylabel("% of Default", fontsize=12)
ax1.set_ylim(0, 0.5)
ax1.set_xticks([0, 1, 2, 3,4])
ax1.set_xticklabels(['Grad School', 'University', 'High School', 'Others','unknown'], fo
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="MARITAL STATUS", y="DEF_AMT", data=df, palette='rocket', ax=axes[1]
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="GENDER", y="DEF_AMT", data=df, palette='rocket', ax=axes[2])
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()
```



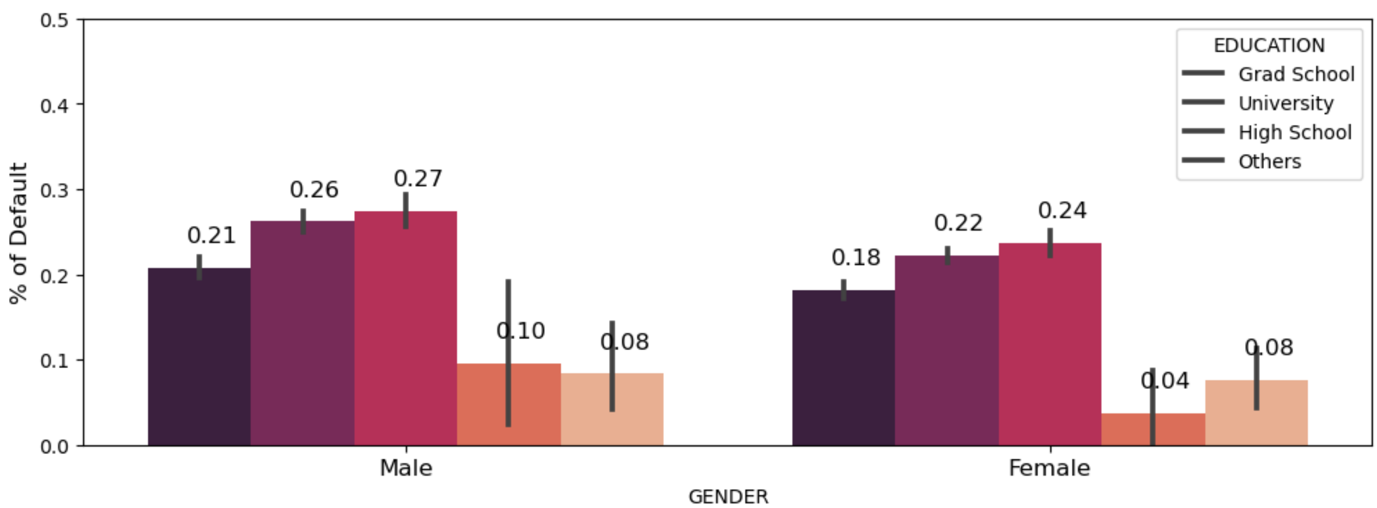
-> The likelihood of being a defaulter decreases as your education level increases.
-> Married and other marital statuses (possibly including divorced) have an approximately 0.24 probability of being defaulters, whereas single individuals have a lower likelihood at 0.21.
-> Despite a smaller number of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

In [37]:
```
plt.figure(figsize=(12,4))

hue_label_mapping = {0: 'married',1: 'single',2: 'others'}

ax = sns.barplot(x = "GENDER", y = "DEF_AMT",data = df, palette = 'rocket',hue='MARITAL

plt.ylabel("% of Default", fontsize= 12)
plt.ylim(0,0.5)
plt.xticks([0,1],['Male', 'Female'], fontsize = 12)

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

plt.show()
```



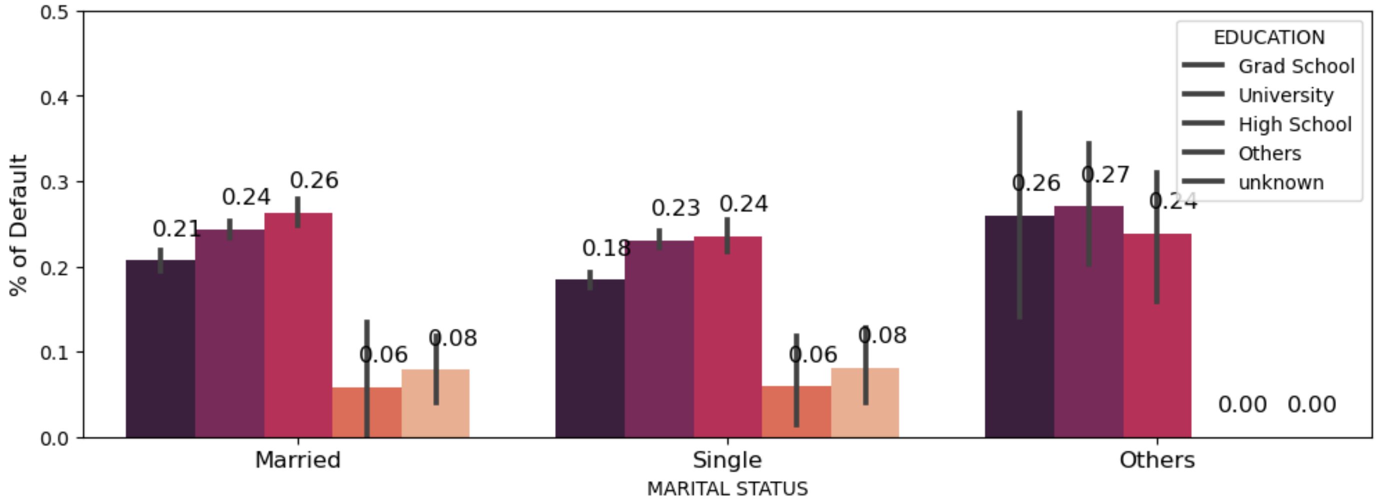
In [38]:

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

ax = sns.barplot(x = "GENDER", y = "DEF_AMT", hue = "EDUCATION", data = df, palette = 'r

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 [39]:

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

ax = sns.barplot(x = "MARITAL STATUS", y = "DEF_AMT", hue = "EDUCATION", data = df, pale

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','unknown'], title = 'ED

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

plt.show()
```

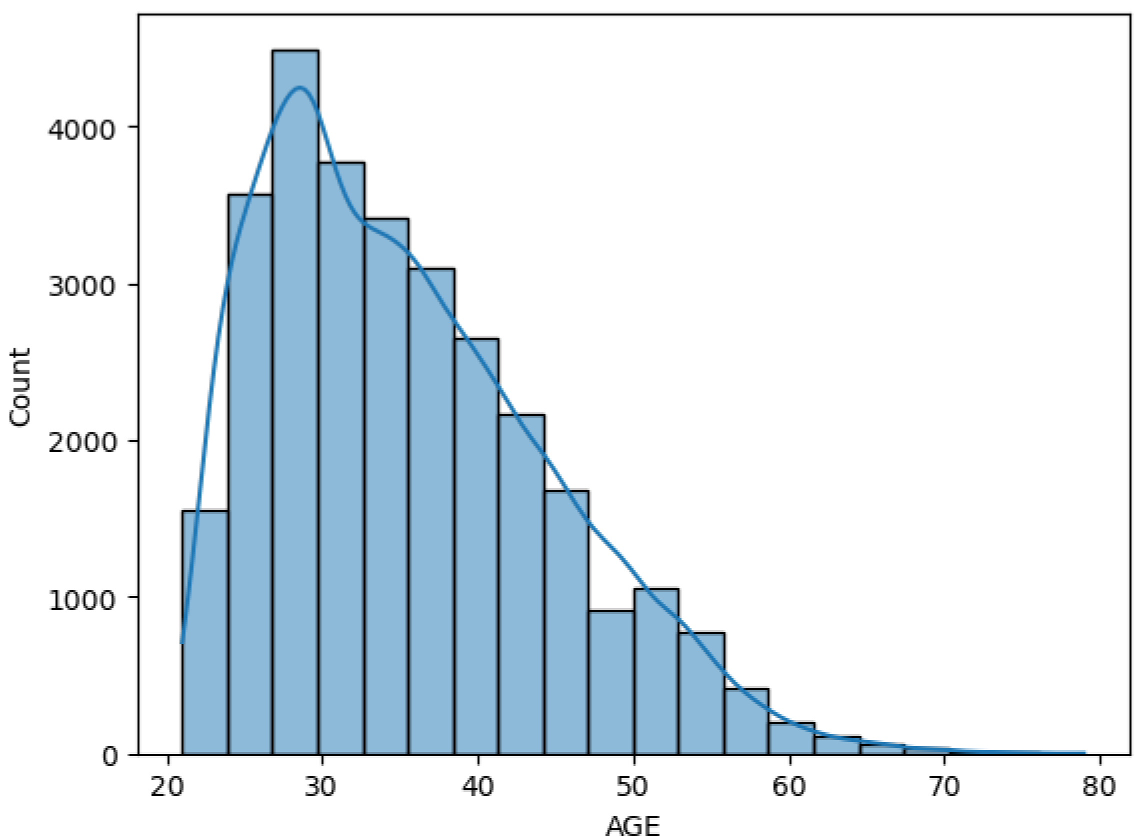


-> Being male, married, and having a high school education seems to increase the likelihood of being a defaulter.
-> People who are marked as "Others" in their marital status (likely indicating divorced individuals) have a notable probability of around 0.29 for facing defaults, which is a relatively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

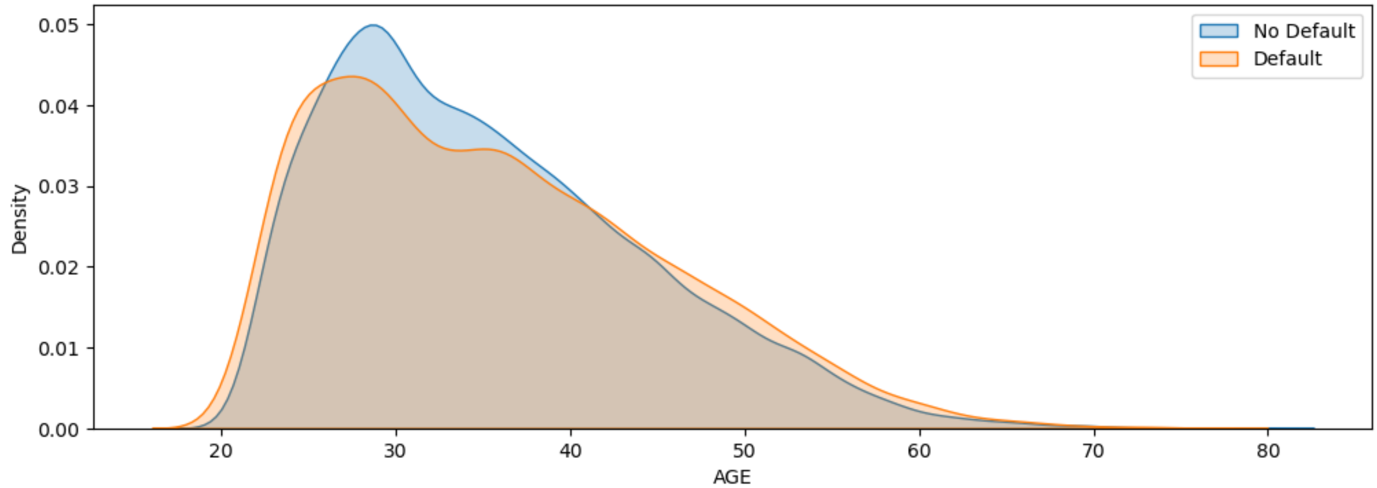
7.5) AGE COLUMN

In [40]:
```
sns.histplot(df['AGE'],bins=20,kde=True)
```

Out[40]:
```
<AxesSubplot:xlabel='AGE', ylabel='Count'>
```



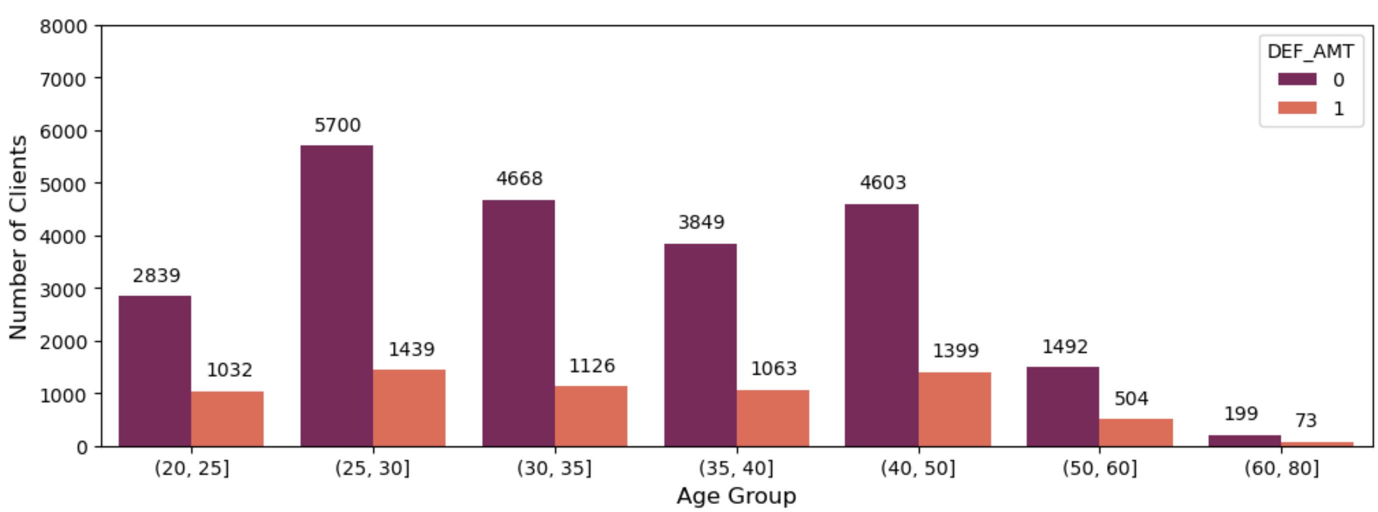
In [41]:
```
plt.figure(figsize=(12,4))

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



In [42]:
```python
df['AgeBin'] = pd.cut(df['AGE'],[20, 25, 30, 35, 40, 50, 60, 80])
print(df['AgeBin'].value_counts())
```

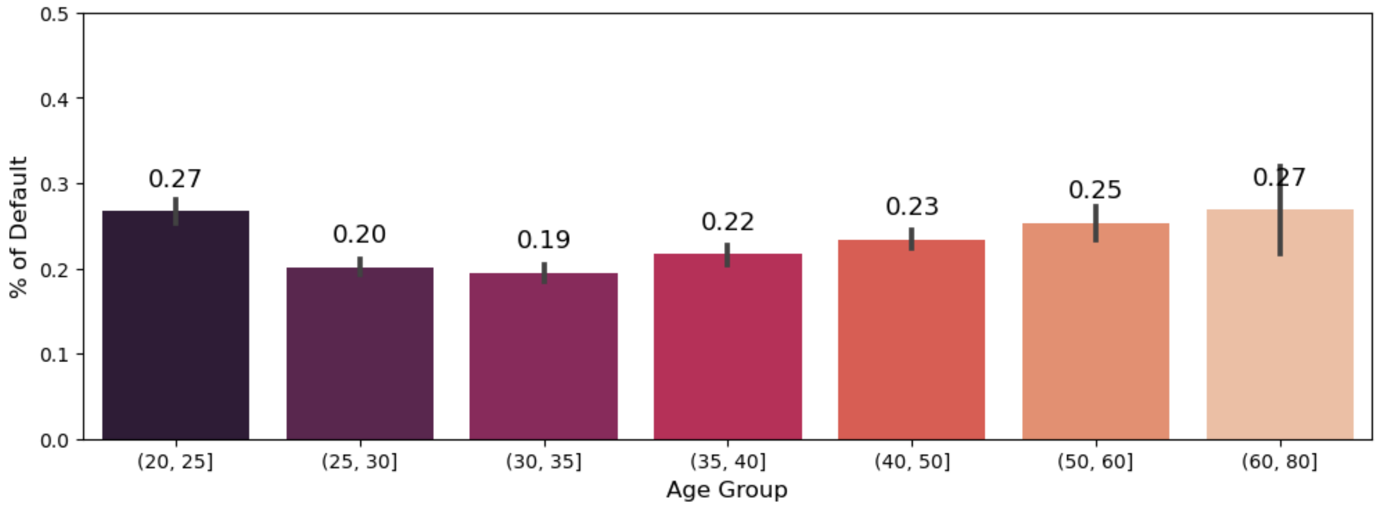
```
(25, 30]     7139
(40, 50]     6002
(30, 35]     5794
(35, 40]     4912
(20, 25]     3871
(50, 60]     1996
(60, 80]      272
Name: AgeBin, dtype: int64
```

In [43]:
```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_AMT", 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 [44]:
```python
plt.figure(figsize=(12,4))

ax = sns.barplot(x = "AgeBin", y = "DEF_AMT", data = df, palette = 'rocket', order = Age
```

```
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()
```

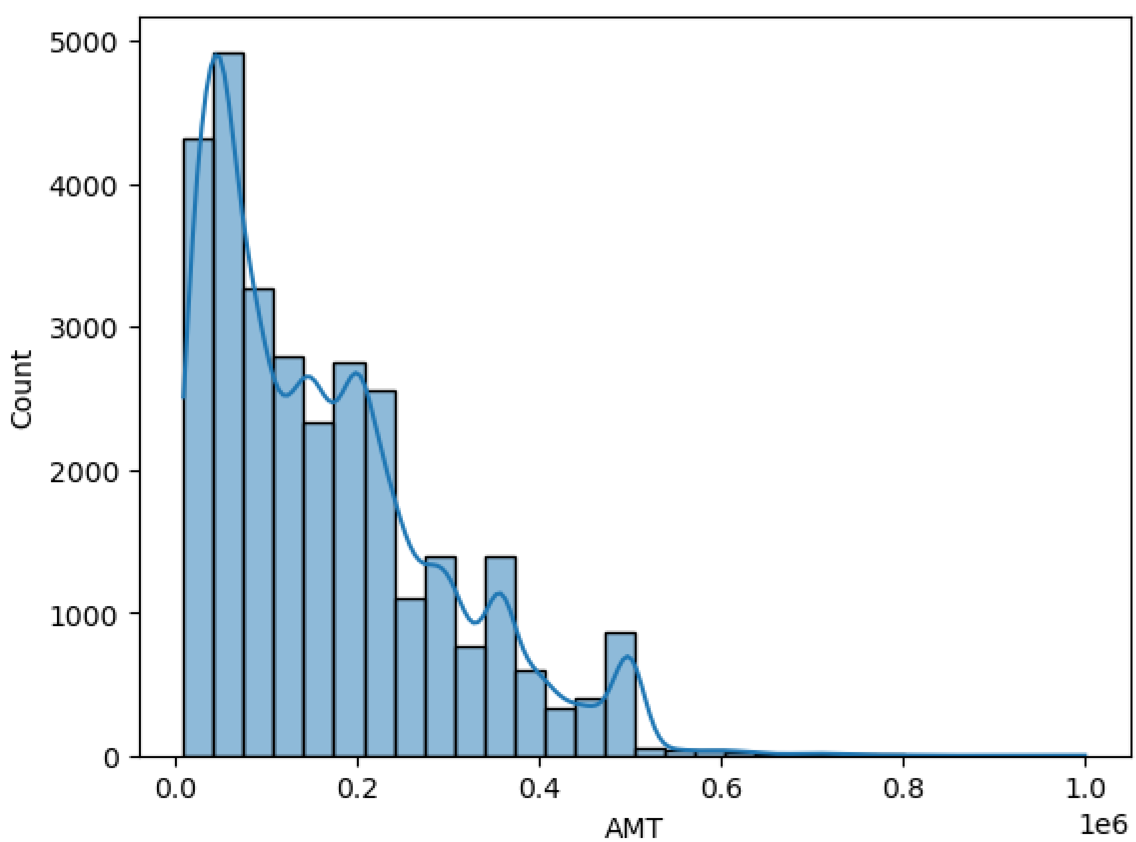


-> Individuals aged between 20 and 25, as well as those above 50, are more prone to default on their credit card payments. In contrast, individuals within other age ranges show lower tendencies for default.

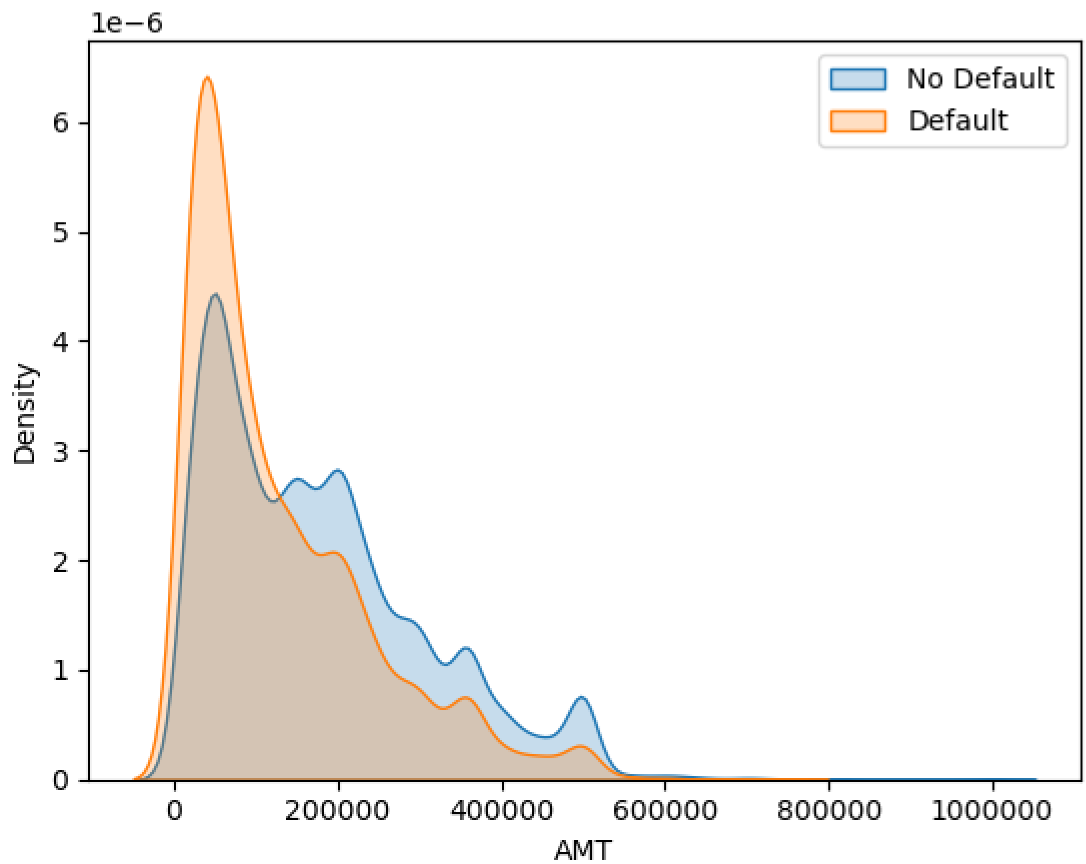
7.6)Amount of given credit in dollars(AMT)

In [45]:
```
sns.histplot(df['AMT'],bins=30,kde=True)
```

Out[45]:
```
<AxesSubplot:xlabel='AMT', ylabel='Count'>
```



In [46]:
```
sns.kdeplot(df.loc[(df['DEF_AMT'] == 0), 'AMT'], label = 'No Default', fill = True)
sns.kdeplot(df.loc[(df['DEF_AMT'] == 1), 'AMT'], label = 'Default', fill = True)
plt.ticklabel_format(style='plain', axis='x')
plt.legend()
plt.show()
```



In [47]:
```
df['AMT'].describe()
```

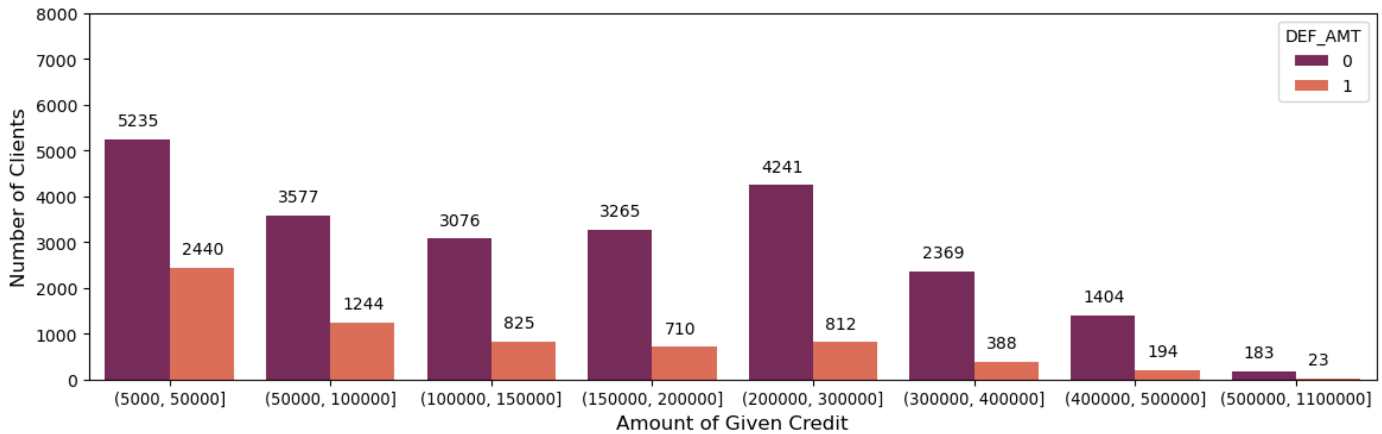
Out[47]:
```
count      29986.000000
mean      167461.137864
std       129760.982745
min        10000.000000
25%        50000.000000
50%       140000.000000
75%       240000.000000
max      1000000.000000
Name: AMT, dtype: float64
```

In [48]:
```
df['AMT_bin'] = pd.cut(df['AMT'],[5000, 50000, 100000, 150000, 200000, 300000, 400000, 5
print(df['AMT_bin'].value_counts())
```

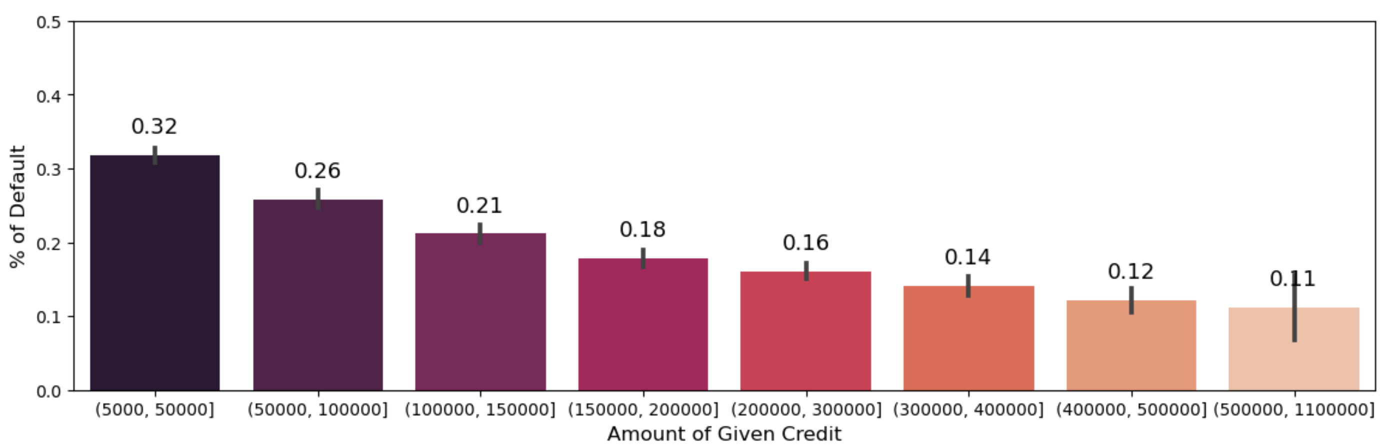
```
(5000, 50000]        7675
(200000, 300000]     5053
(50000, 100000]      4821
(150000, 200000]     3975
(100000, 150000]     3901
(300000, 400000]     2757
(400000, 500000]     1598
(500000, 1100000]     206
Name: AMT_bin, dtype: int64
```

In [49]:
```
plt.figure(figsize=(14,4))

df['AMT_bin'] = df['AMT_bin'].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 = 'AMT_bin', hue="DEF_AMT", 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 [50]:
```
plt.figure(figsize=(14,4))

ax = sns.barplot(x = "AMT_bin", y = "DEF_AMT", data = df, palette = 'rocket', order = Li

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.

In [51]:
```
df.head()
```

Out[51]:

ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY_JUL	REPAY_JUN	...	AM

0	1	20000.0	2	2.0	1	24	2	2	0	0	...
1	2	120000.0	2	2.0	2	26	0	2	0	0	...
2	3	90000.0	2	2.0	2	34	0	0	0	0	...
3	4	50000.0	2	2.0	1	37	0	0	0	0	...
4	5	50000.0	1	2.0	1	57	0	0	0	0	...

5 rows × 39 columns

In [52]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29986 entries, 0 to 29999
Data columns (total 39 columns):
 #   Column          Non-Null Count  Dtype   
---  ------          --------------  -----   
 0   ID              29986 non-null  int64   
 1   AMT             29986 non-null  float64 
 2   GENDER          29986 non-null  int64   
 3   EDUCATION       29986 non-null  float64 
 4   MARITAL STATUS  29986 non-null  int64   
 5   AGE             29986 non-null  int64   
 6   REPAY_SEP       29986 non-null  int64   
 7   REPAY_AUG       29986 non-null  int64   
 8   REPAY_JUL       29986 non-null  int64   
 9   REPAY_JUN       29986 non-null  int64   
 10  REPAY_MAY       29986 non-null  int64   
 11  REPAY_APR       29986 non-null  int64   
 12  AMTBILL_SEP     29986 non-null  float64 
 13  AMTBILL_AUG     29986 non-null  float64 
 14  AMTBILL_JUL     29986 non-null  float64 
 15  AMTBILL_JUN     29986 non-null  float64 
 16  AMTBILL_MAY     29986 non-null  float64 
 17  AMTBILL_APR     29986 non-null  float64 
 18  PRE_SEP         29986 non-null  float64 
 19  PRE_AUG         29986 non-null  float64 
 20  PRE_JUL         29986 non-null  float64 
 21  PRE_JUN         29986 non-null  float64 
 22  PRE_MAY         29986 non-null  float64 
 23  PRE_APR         29986 non-null  float64 
 24  DEF_AMT         29986 non-null  int64   
 25  AMTBILL_SEP_bin 29986 non-null  category
 26  AMTBILL_AUG_bin 29986 non-null  category
 27  AMTBILL_JUL_bin 29986 non-null  category
 28  AMTBILL_JUN_bin 29986 non-null  category
 29  AMTBILL_MAY_bin 29986 non-null  category
 30  AMTBILL_APR_bin 29986 non-null  category
 31  PRE_SEP_bin     29986 non-null  category
 32  PRE_AUG_bin     29986 non-null  category
 33  PRE_JUL_bin     29986 non-null  category
 34  PRE_JUN_bin     29986 non-null  category
 35  PRE_MAY_bin     29986 non-null  category
 36  PRE_APR_bin     29986 non-null  category
 37  AgeBin          29986 non-null  object  
 38  AMT_bin         29986 non-null  object  
dtypes: category(12), float64(14), int64(11), object(2)
memory usage: 7.8+ MB
```

In [53]:
```
df = df.astype({"AMT":"float64"})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29986 entries, 0 to 29999
Data columns (total 39 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               29986 non-null  int64
 1   AMT              29986 non-null  float64
 2   GENDER           29986 non-null  int64
 3   EDUCATION        29986 non-null  float64
 4   MARITAL STATUS   29986 non-null  int64
 5   AGE              29986 non-null  int64
 6   REPAY_SEP        29986 non-null  int64
 7   REPAY_AUG        29986 non-null  int64
 8   REPAY_JUL        29986 non-null  int64
 9   REPAY_JUN        29986 non-null  int64
 10  REPAY_MAY        29986 non-null  int64
 11  REPAY_APR        29986 non-null  int64
 12  AMTBILL_SEP      29986 non-null  float64
 13  AMTBILL_AUG      29986 non-null  float64
 14  AMTBILL_JUL      29986 non-null  float64
 15  AMTBILL_JUN      29986 non-null  float64
 16  AMTBILL_MAY      29986 non-null  float64
 17  AMTBILL_APR      29986 non-null  float64
 18  PRE_SEP          29986 non-null  float64
 19  PRE_AUG          29986 non-null  float64
 20  PRE_JUL          29986 non-null  float64
 21  PRE_JUN          29986 non-null  float64
 22  PRE_MAY          29986 non-null  float64
 23  PRE_APR          29986 non-null  float64
 24  DEF_AMT          29986 non-null  int64
 25  AMTBILL_SEP_bin  29986 non-null  category
 26  AMTBILL_AUG_bin  29986 non-null  category
 27  AMTBILL_JUL_bin  29986 non-null  category
 28  AMTBILL_JUN_bin  29986 non-null  category
 29  AMTBILL_MAY_bin  29986 non-null  category
 30  AMTBILL_APR_bin  29986 non-null  category
 31  PRE_SEP_bin      29986 non-null  category
 32  PRE_AUG_bin      29986 non-null  category
 33  PRE_JUL_bin      29986 non-null  category
 34  PRE_JUN_bin      29986 non-null  category
 35  PRE_MAY_bin      29986 non-null  category
 36  PRE_APR_bin      29986 non-null  category
 37  AgeBin           29986 non-null  object
 38  AMT_bin          29986 non-null  object
dtypes: category(12), float64(14), int64(11), object(2)
memory usage: 7.8+ MB
```

In [54]:
```
# Group by and calculate mean for each category
mean_by_GENDER = df.groupby('GENDER')['AMT'].mean()
mean_by_education = df.groupby('EDUCATION')['AMT'].mean()
mean_by_MARITAL_STATUS = df.groupby('MARITAL STATUS')['AMT'].mean()
mean_by_age_bin = df.groupby('AgeBin')['AMT'].mean()


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

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

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

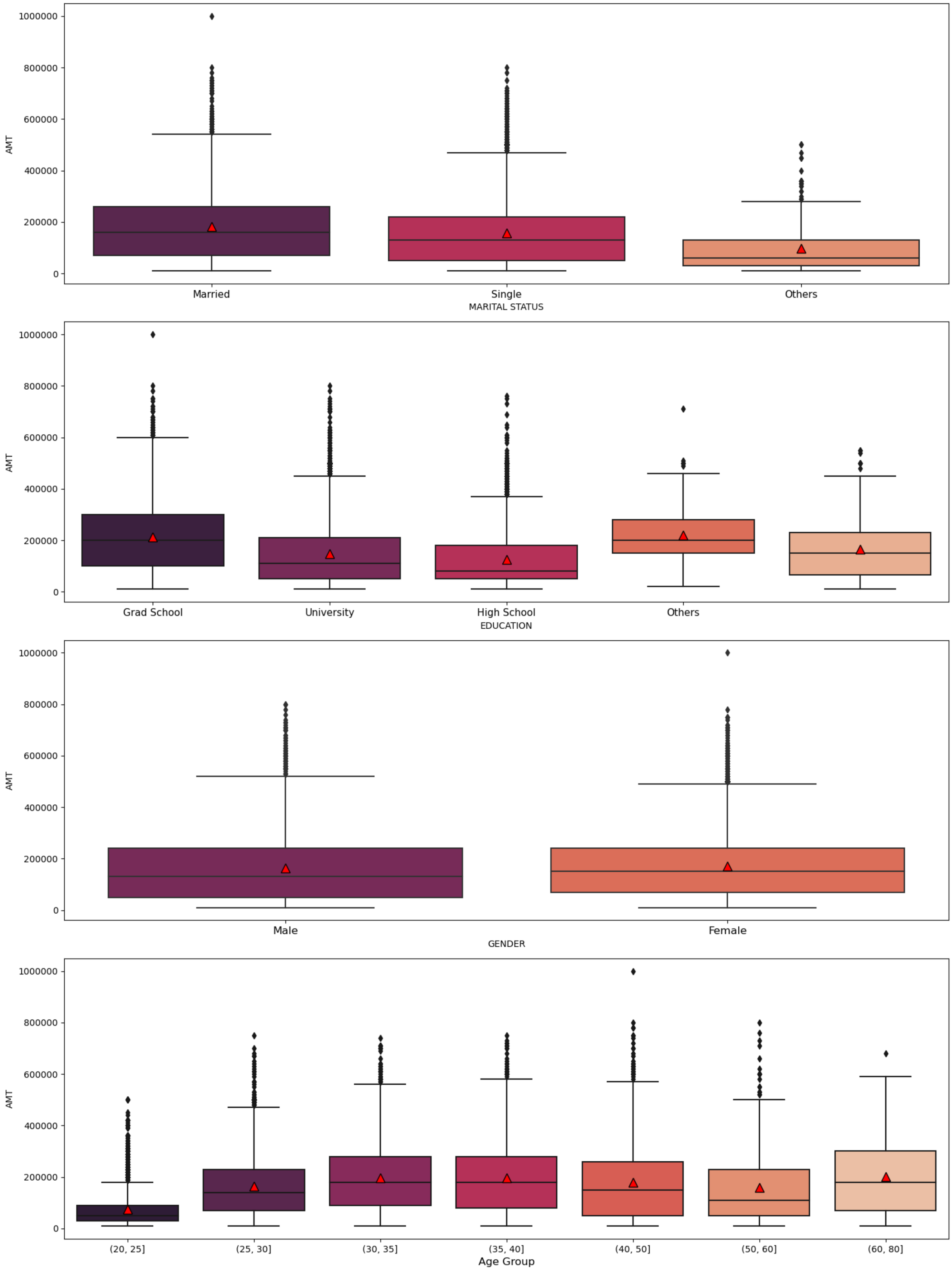
```
print(mean_by_MARITAL_STATUS)

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

```
Mean LIMIT_BAL by SEX:
GENDER
1    163486.841751
2    170068.816967
Name: AMT, dtype: float64
----------------------

Mean LIMIT_BAL by EDUCATION:
EDUCATION
1.0    212956.069910
2.0    147062.437634
3.0    126550.270490
4.0    220894.308943
5.0    165093.655589
Name: AMT, dtype: float64
----------------------

Mean LIMIT_BAL by MARRIAGE:
MARITAL STATUS
1    182201.712655
2    156319.824880
3     97814.371257
Name: AMT, dtype: float64
----------------------

Mean LIMIT_BAL by AGE_BIN:
AgeBin
(20, 25]     73763.885301
(25, 30]    164290.516879
(30, 35]    197675.181222
(35, 40]    196795.602606
(40, 50]    179670.056648
(50, 60]    159253.507014
(60, 80]    201617.647059
Name: AMT, dtype: float64
```

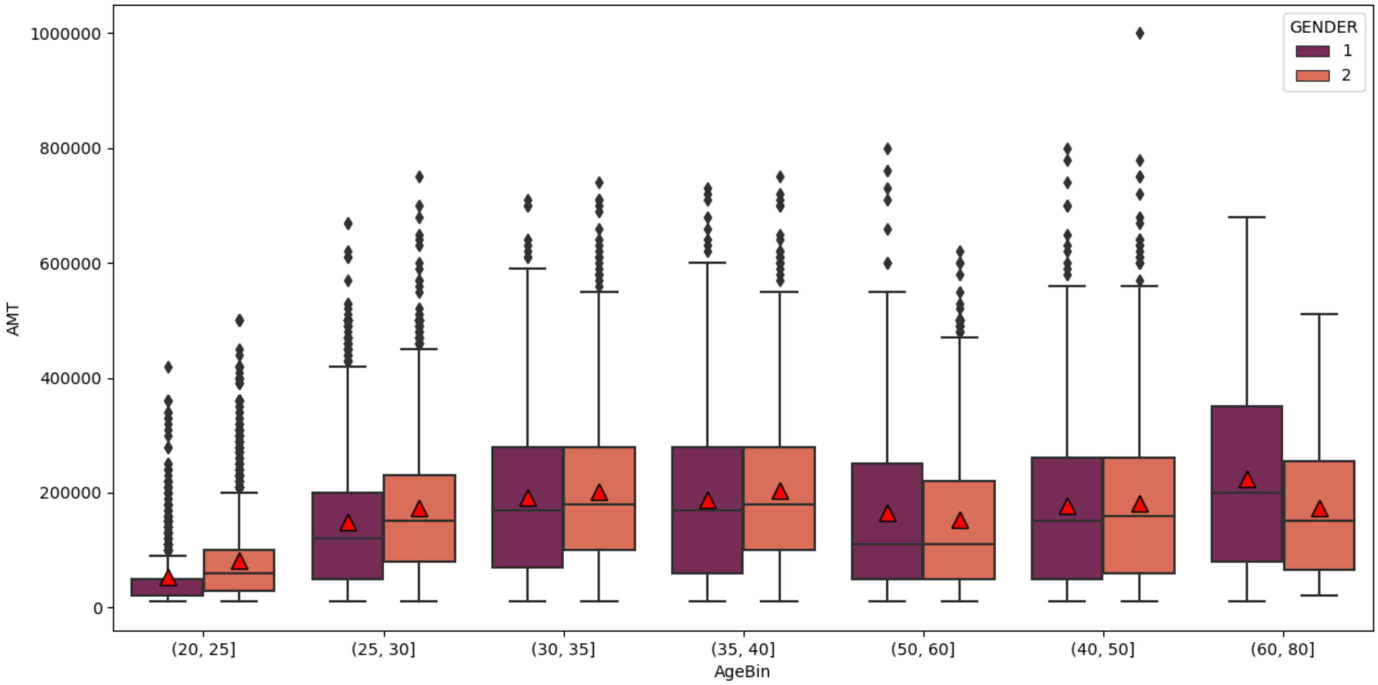
In [55]:

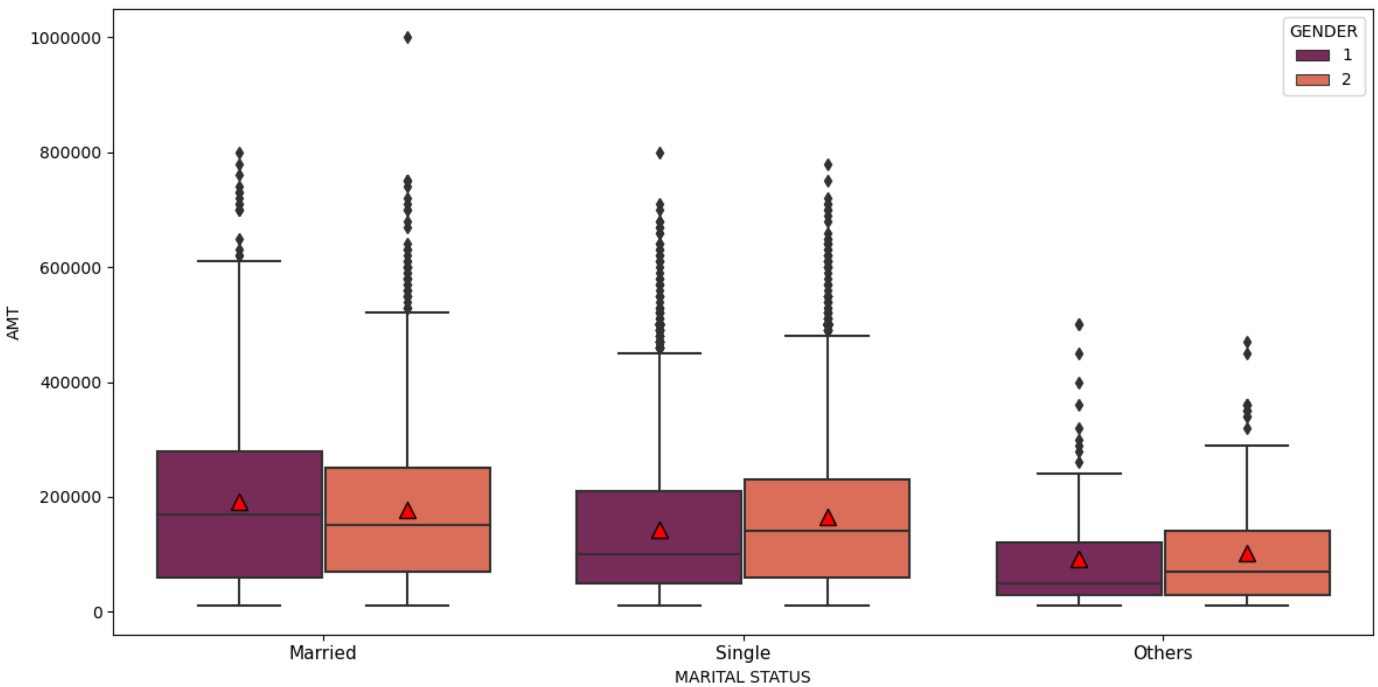
```
plt.figure(figsize=(15, 20))

plt.subplot(4, 1, 1)
sns.boxplot(x="MARITAL STATUS", y="AMT", 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="AMT", 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="GENDER", y="AMT", 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="AMT", data=df, palette='rocket', showmeans=True, order=AgeBin
```

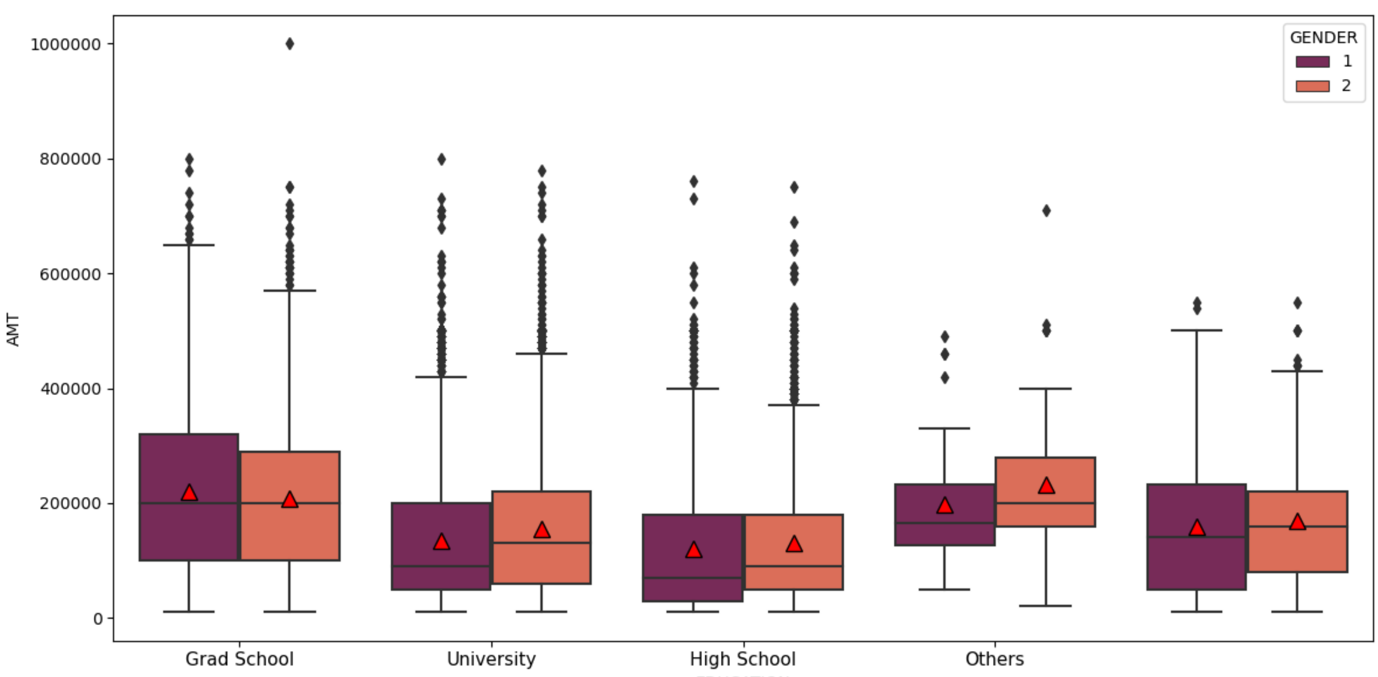
```
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')
plt.xlabel("Age Group", fontsize=12)

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



In [56]:
```
plt.figure(figsize=(12, 18))

# Subplot for AgeBin
plt.subplot(3, 1, 1)
sns.boxplot(x="AgeBin", y="AMT", hue='GENDER', data=df, palette='rocket', showmeans=True
            meanprops={"markerfacecolor": "red", "markeredgecolor": "black", "markersize
plt.ticklabel_format(style='plain', axis='y')


# Subplot for Marriage
plt.subplot(3, 1, 2)
sns.boxplot(x="MARITAL STATUS", y="AMT", hue='GENDER', data=df, palette='rocket', showme
            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="AMT", hue='GENDER', data=df, palette='rocket', showmeans=T
            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()
```


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


GENDER
1
2
AMT
1000000
800000
600000
400000
200000
0
Married
Single
Others
MARITAL STATUS


GENDER
1
2
AMT
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.

Correlation Analysis

In [57]:

```
cor= df.corr()
cor
```

Out[57]:

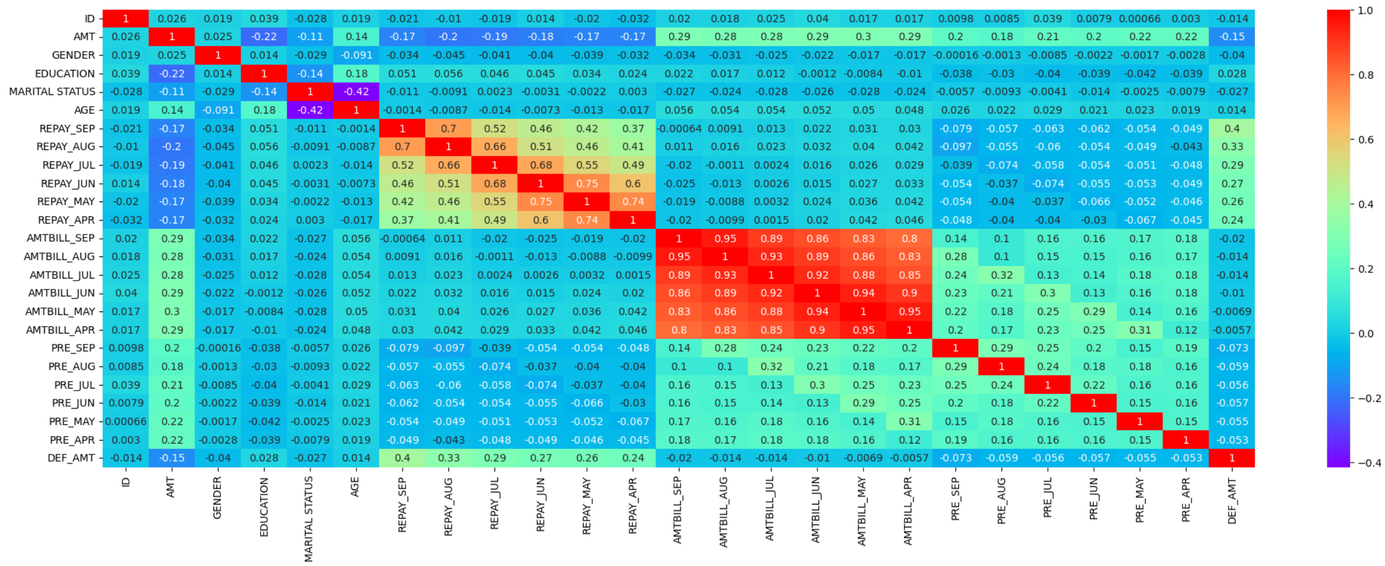
	ID	AMT	GENDER	EDUCATION	MARITAL STATUS	AGE	REPAY_SEP	REPAY_AUG	REPAY
ID	1.000000	0.026278	0.018574	0.039421	-0.027754	0.018653	-0.020837	-0.009959	-0.01
AMT	0.026278	1.000000	0.024810	-0.220942	-0.111286	0.144665	-0.170783	-0.197098	-0.19
GENDER	0.018574	0.024810	1.000000	0.014420	-0.029404	-0.090925	-0.034493	-0.044831	-0.04
EDUCATION	0.039421	-0.220942	0.014420	1.000000	-0.137063	0.175504	0.051008	0.056460	0.04
MARITAL STATUS	-0.027754	-0.111286	-0.029404	-0.137063	1.000000	-0.415527	-0.011134	-0.009081	0.00
AGE	0.018653	0.144665	-0.090925	0.175504	-0.415527	1.000000	-0.001417	-0.008708	-0.01
REPAY_SEP	-0.020837	-0.170783	-0.034493	0.051008	-0.011134	-0.001417	1.000000	0.698434	0.51
REPAY_AUG	-0.009959	-0.197098	-0.044831	0.056460	-0.009081	-0.008708	0.698434	1.000000	0.66
REPAY_JUL	-0.018831	-0.191322	-0.041375	0.046302	0.002269	-0.014154	0.517071	0.663588	1.00
REPAY_JUN	0.013718	-0.180655	-0.039880	0.044972	-0.003139	-0.007342	0.460322	0.512821	0.67
REPAY_MAY	-0.020306	-0.170015	-0.038596	0.034264	-0.002171	-0.013227	0.424488	0.462687	0.55
REPAY_APR	-0.032421	-0.167790	-0.032124	0.023727	0.002993	-0.016897	0.373814	0.407052	0.49
AMTBILL_SEP	0.019503	0.285867	-0.033844	0.022264	-0.026675	0.056372	-0.000637	0.011469	-0.02
AMTBILL_AUG	0.018034	0.278717	-0.031335	0.017459	-0.024284	0.054461	0.009097	0.015789	-0.00
AMTBILL_JUL	0.024517	0.283538	-0.024786	0.011786	-0.027840	0.053975	0.013245	0.022848	0.00
AMTBILL_JUN	0.040292	0.294282	-0.022130	-0.001168	-0.026324	0.051669	0.022017	0.032312	0.01
AMTBILL_MAY	0.016694	0.295840	-0.017128	-0.008407	-0.028119	0.049561	0.030705	0.040299	0.02
AMTBILL_APR	0.016916	0.290948	-0.016764	-0.009957	-0.024030	0.047663	0.030321	0.042253	0.02
PRE_SEP	0.009779	0.195217	-0.000165	-0.038407	-0.005690	0.026214	-0.079168	-0.097479	-0.03
PRE_AUG	0.008515	0.178268	-0.001342	-0.030070	-0.009313	0.021633	-0.057245	-0.054796	-0.07
PRE_JUL	0.039157	0.210065	-0.008500	-0.040475	-0.004094	0.029233	-0.062601	-0.059835	-0.05
PRE_JUN	0.007863	0.203238	-0.002225	-0.038790	-0.013730	0.021450	-0.061877	-0.053999	-0.05
PRE_MAY	0.000657	0.217253	-0.001724	-0.041964	-0.002477	0.022863	-0.053784	-0.048744	-0.05
PRE_APR	0.003008	0.219666	-0.002790	-0.038932	-0.007876	0.019479	-0.048727	-0.043124	-0.04
DEF_AMT	-0.013895	-0.153455	-0.040063	0.028075	-0.026670	0.013984	0.396029	0.327028	0.28

25 rows × 25 columns

In [58]:

```
#plotting the correlations on the heatmap
```

```python
plt.figure(figsize=(25,8))
sns.heatmap(cor, cmap="rainbow", annot=True)
plt.show()
```



-> 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.

8) Analysis Summary

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%

Task -2

1)Hypothesis Testing

Based on the columns provided in your dataset, it seems that you want to analyze factors related to credit card defaults (represented by the "DEF_AMT" column) among different clients. Since we are interested in modeling the number of credit card defaults (which is a binary outcome - either a client defaults or does not default), a binomial distribution would be more appropriate for this analysis.

Based on chart experiments, define three hypothetical statements from the dataset. In the next three questions, perform hypothesis testing to obtain final conclusion about the statements through codes and statistical testings.

Creating a class to calculate mean, median, variance, P value and all other metrics required for the calculation of hypothesis testing.

1. Men not defaulting are more than or equal to 40 years of AGE.
2. Customers defaulting have limit balance less than 100000.
3. Customers defaulting have total last bill amount of 50000.

In all of the hypothesis tests in this notebook, we will use a significance level of α = 0.05

1.1 Hypothetical Statement - 1

Men not defaulting are more than or equal to 40 years of AGE.

State Your research hypothesis as a null hypothesis and alternate hypothesis.

Null Hypothesis: N = 40

Alternate Hypothesis : N < 40

Test Type: Left Tailed Test

In [59]:
```
import math
from scipy.stats import *
from scipy import stats
```

In [60]:
```
# Creating Parameter Class
class findz:
  def proportion(self,sample,hyp,size):
    return (sample - hyp)/math.sqrt(hyp*(1-hyp)/size)
  def mean(self,hyp,sample,size,std):
    return (sample - hyp)*math.sqrt(size)/std
  def varience(self,hyp,sample,size):
    return (size-1)*sample/hyp
```

```python
variance = lambda x : sum([(i - np.mean(x))**2 for i in x])/(len(x)-1)
zcdf = lambda x: norm(0,1).cdf(x)
# Creating a function for getting P value
def p_value(z,tailed,t,hypothesis_number,df,col):
  if t!="true":
    z=zcdf(z)
    if tailed=='l':
      return z
    elif tailed == 'r':
      return 1-z
    elif tailed == 'd':
      if z>0.5:
        return 2*(1-z)
      else:
        return 2*z
    else:
      return np.nan
  else:
    z,p_value=stats.ttest_1samp(df[col],hypothesis_number)
    return p_value


# Conclusion about the P - Value
def conclusion(p):
  significance_level = 0.05
  if p>significance_level:
    return f"Failed to reject the Null Hypothesis for p = {p}."
  else:
    return f"Null Hypothesis rejected Successfully for p = {p}"

# Initializing the class
findz = findz()
```

In [61]:
```python
# Perform Statistical Test to obtain P-Value

# SEX:
# 1 = male; 2 = female

# DP_NEXT_MONTH:
# 0 = non-default; 1 = default

hypo_1 = df[(df['GENDER']==1) & (df["DEF_AMT"]==0)]

# Getting the required parameter values for hypothesis testing
hypothesis_number = 40
sample_mean = hypo_1["AGE"].mean()
size = len(hypo_1)
std=(variance(hypo_1["AGE"]))**0.5
```

In [62]:
```python
# Getting Z value
z = findz.mean(hypothesis_number,sample_mean,size,std)

# Getting P - Value
p = p_value(z=z,tailed='l',t="false",hypothesis_number=hypothesis_number,df=hypo_1,col="

# Getting Conclusion
print(conclusion(p))
```

```
Null Hypothesis rejected Successfully for p = 2.1473285127242563e-290
```

Men not defaulting are more than or equal to 40 years of AGE is a true statement.

1.2 Hypothetical Statement - 2

Customers defaulting have limit balance less than 100000

State Your research hypothesis as a null hypothesis and alternate hypothesis.

Null Hypothesis: N = 100000

Alternate Hypothesis : N > 100000

Test Type: Right Tailed Test

In [63]:
```
# Perform Statistical Test to obtain P-Value

# DP_NEXT_MONTH:
# 0 = non-default; 1 = default
hypo_2=df[(df["DEF_AMT"]==1)]

# Getting the required parameter values for hypothesis testing
hypothesis_number = 100000
sample_mean = hypo_2["AMT"].mean()
size = len(hypo_2)
std=(variance(hypo_2["AMT"]))**0.5
```

In [64]:
```
# Getting Z value
z = findz.mean(hypothesis_number,sample_mean,size,std)

# Getting P - Value
p = p_value(z=z,tailed='r',t="true",hypothesis_number=hypothesis_number,df=hypo_2,col="A

# Getting Conclusion
print(conclusion(p))
```

```
Null Hypothesis rejected Successfully for p = 4.4753017364632867e-97
```

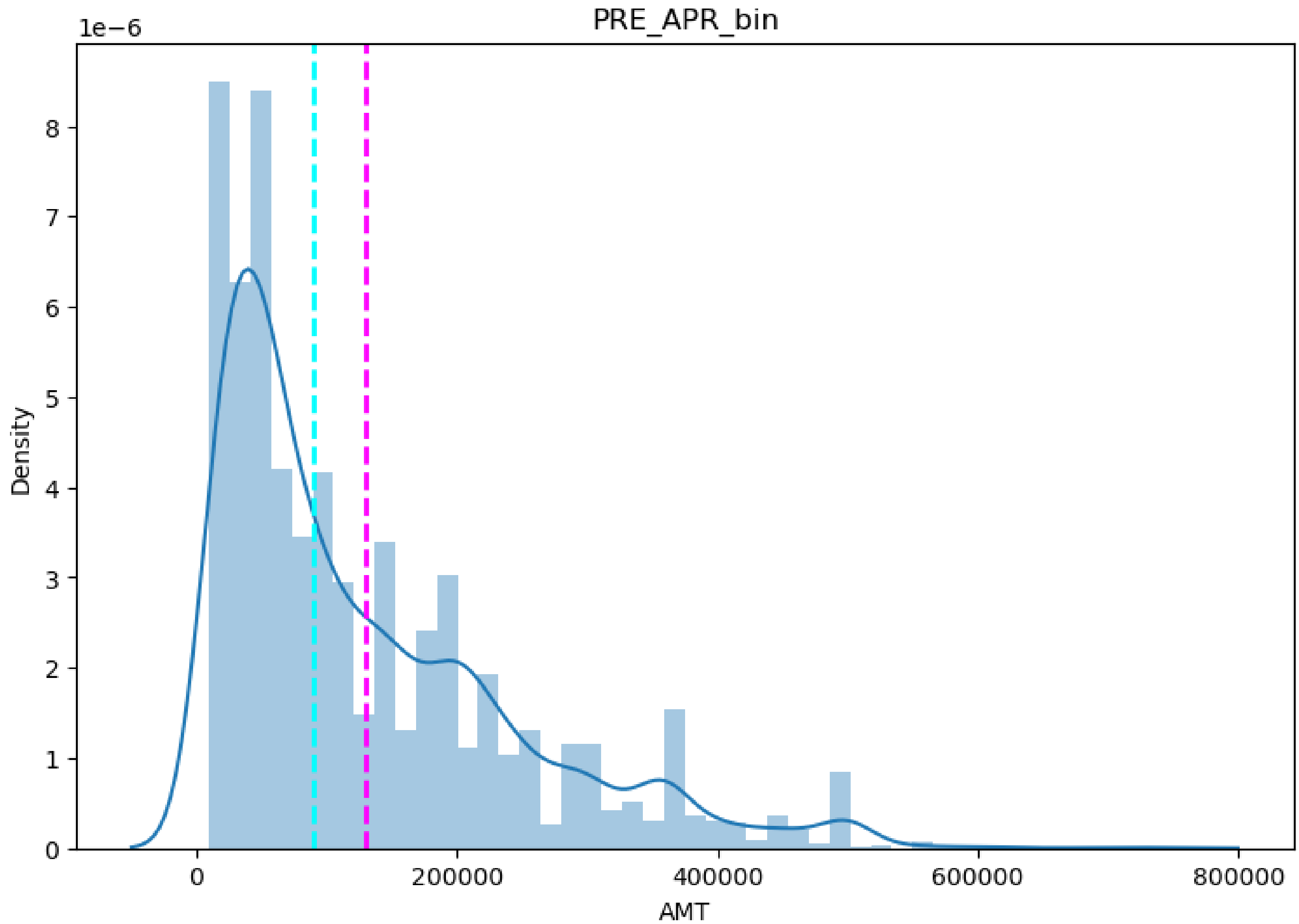
Which statistical test have you done to obtain P-Value?

I used T-Test as the statistical testing to get the P-Value, and the result showed that the null hypothesis was wrong and that customers who defaulted had a limit balance of less than 100,000.

Why did you choose the specific statistical test?

In [65]:
```
# Visualizing code of hist plot for required columns to know the data distibution

fig=plt.figure(figsize=(9,6))
ax=fig.gca()
feature= (hypo_2["AMT"])
sns.distplot(hypo_2["AMT"])
ax.axvline(feature.mean(),color='magenta', linestyle='dashed', linewidth=2)
ax.axvline(feature.median(),color='cyan', linestyle='dashed', linewidth=2)
ax.set_title(col)
plt.show()
```

```
C:\Users\SAI RAM\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please a
dapt your code to use either `displot` (a figure-level function with similar flexibilit
y) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [66]:
```
mean_median_difference=hypo_2["AMT"].mean()- hypo_2["AMT"].median()
print("Mean Median Difference is :-",mean_median_difference)
```

```
Mean Median Difference is :- 40109.65641952984
```

The graph above demonstrates that the median is greater than the mean above 10,000. As a result, the distribution is positively skewed. Z-Test cannot be used with skewed data.

For small studies, non-parametric tests are most useful. In large studies, the use of non-parametric tests may answer the wrong question, causing readers confusion. Even with heavily skewed data, t-tests and the confidence intervals that go along with them should be used in studies with large sample sizes.

Therefore, the T-test can yield better results for skewed data. So, I used the t-test.

Customers defaulting have limit balance less than 100000 is true statement

1.3 Hypothetical Statement - 3

Customers defaulting have total last bill amount of 50000.

State Your research hypothesis as a null hypothesis and alternate hypothesis.

Null Hypothesis: N = 50000

Alternate Hypothesis : N != 50000

Test Type: Two Tailed test

In [67]:
```
# Perform Statistical Test to obtain P-Value

# DP_NEXT_MONTH:
# 0 = non-default; 1 = default
hypo_3=df[(df["DEF_AMT"]==1)]

# Getting the required parameter values for hypothesis testing
hypothesis_number = 50000
sample_mean = hypo_3["AMTBILL_SEP"].mean()
size = len(hypo_3)
std=(variance(hypo_3["AMTBILL_SEP"]))**0.5
```

In [68]:
```
# Getting Z value
z = findz.mean(hypothesis_number,sample_mean,size,std)

# Getting P - Value
p = p_value(z=z,tailed='d',t="true",hypothesis_number=hypothesis_number,df=hypo_3,col="A

# Getting Conclusion
print(conclusion(p))
```

```
Failed to reject the Null Hypothesis for p = 0.10066286129402914.
```

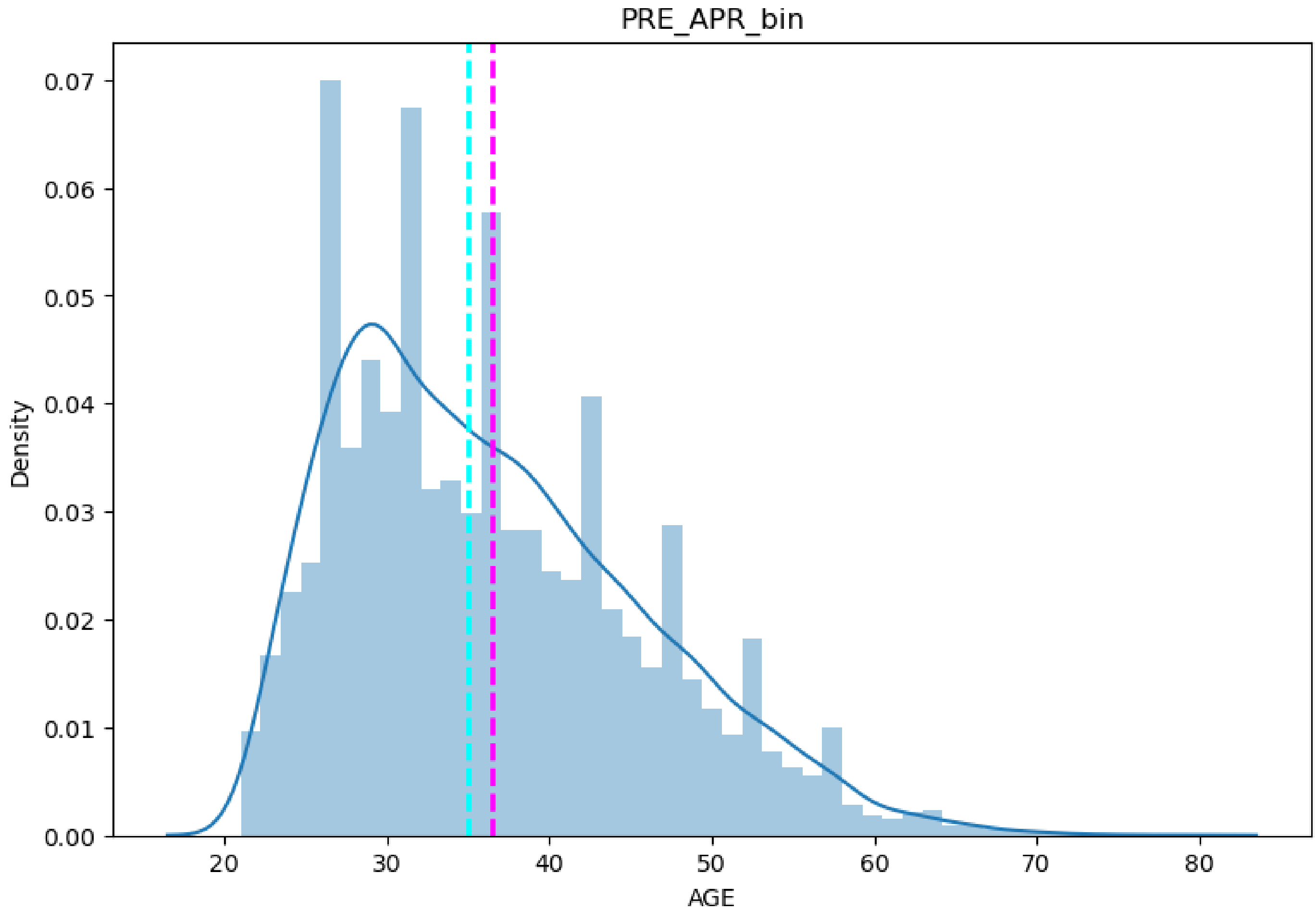
Which statistical test have you done to obtain P-Value?

I used Z-Test as the statistical testing to get the P-Value, and the results showed that the null hypothesis could not be rejected, and male customers who didn't default were over 40 years old.

Why did you choose the specific statistical test?

In [69]:
```
# Visualizing code of hist plot for required columns to know the data distibution

fig=plt.figure(figsize=(9,6))
ax=fig.gca()
feature= (hypo_1["AGE"])
sns.distplot(hypo_1["AGE"])
ax.axvline(feature.mean(),color='magenta', linestyle='dashed', linewidth=2)
ax.axvline(feature.median(),color='cyan', linestyle='dashed', linewidth=2)
ax.set_title(col)
plt.show()
```

```
C:\Users\SAI RAM\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please a
dapt your code to use either `displot` (a figure-level function with similar flexibilit
y) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [70]:
```
mean_median_difference=hypo_3["AMTBILL_SEP"].median()- hypo_3["AMTBILL_SEP"].mean()
print("Mean Median Difference is :-",mean_median_difference)
```

```
Mean Median Difference is :- -28327.95735382761
```

The graph above demonstrates that the median is greater than the mean above 10,000. As a result, the distribution is positively skewed Z-Test cannot be used with skewed data.

For small studies, nonparametric tests are most useful. In large studies, the use of non-parametric tests may answer the wrong question, causing readers confusion. Even with heavily skewed data, t-tests and the confidence intervals that go along with them should be used in studies with large sample sizes.

Therefore, the T-test can yield better results for skewed data. So, I used the t-test

Customers defaulting have total last bill amount of 50000 from the above statement so we consider this statement is false

Hypothetical Statement - 1.1------Men not defaulting are more than or equal to 40 years of AGE.

Hypothetical Statement - 1.2------Customers defaulting have limit balance less than 100000. are true

Let's assume that you have analyzed data and found that both of these statements are true based on your analysis. Here are some actionable recommendations:

Recommendation 1: Targeted Marketing for Men Aged 40 and Above

1) Given that men aged 40 and above have a lower default rate, consider implementing targeted marketing strategies for this demographic.

2) Create advertising campaigns or promotions that specifically appeal to the financial needs and preferences of this group.

3) offer products or services that are tailored to the financial goals and situations of older male customers.

Recommendation 2: Risk Assessment for Customers with Low Limit Balances

1) Identify customers with limit balances less than 100,000, as this group appears to have a higher likelihood of defaulting based on your analysis.

2) Implement a more rigorous risk assessment process for these customers. This may include stricter credit approval criteria, lower credit limits, or requiring additional collateral.

3) Provide financial education and resources to help these customers manage their finances effectively to reduce the risk of default.

4) Consider offering incentives for responsible credit card use to encourage better financial behavior among this group.

It's important to note that these recommendations are based on the hypothetical statements you provided, assuming they are supported by your data analysis. When implementing such recommendations, it's crucial to continually monitor the results and adapt your strategies based on ongoing data analysis to ensure their effectiveness. Additionally, considering ethical and legal implications in your marketing and risk assessment strategies is essential.

THE END