nmjnv6atm

December 18, 2024

1 Classification of Credit Card Default Risk: A Machine Learning Approach

1.1 Name: Ajo Babu A

1.2 Organization: Entri Elavate

1.2.1 Date : 18/12/2024

1.3 Overview of Problem Statement

In the banking and financial sector, credit risk is a major concern, as it directly impacts the profitability and sustainability of financial institutions. Banks issue credit cards to clients based on their financial standing and repayment capacity, but there is always a risk that clients may fail to pay their dues. This failure to pay (credit default) can result in significant losses for the bank.

The primary challenge is to accurately predict whether a credit card client will default on their payment in the next month based on their demographic details, financial history, and repayment behavior. Early identification of clients at high risk of default can help financial institutions take proactive measures to mitigate losses, such as adjusting credit limits, offering restructuring plans, or rejecting high-risk applications.

1.4 Objectives

Predict Credit Default: Build a machine learning model that accurately predicts whether a client will default on their credit card payment in the next month, using demographic, financial, and repayment history data.

Identify Key Factors: Analyze and identify the most significant features (e.g., repayment status, credit limit, bill amounts) that influence the likelihood of credit default.

1.4.1 Data Description

Source : From UCI ML Repository, link: https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

Features: 'ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'default payment next month'

In data set, Default payment has values as 1 and o. where 1 = Yes, 0 = No

Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

Marital status (1 = married; 2 = single; 3 = others).

pay 0 to 6 describes the status of repayment -2 = no payment; -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.

[175]:
```python
# Importing Libraries

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

from sklearn.preprocessing import PowerTransformer


import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

1.4.2 Data Collection

[19]:
```python
# Loading the data
data = pd.read_csv('default of credit card clients.csv')
```

[20]:
```python
# Dataframe
df = pd.DataFrame(data)
df
```

[20]:
```
          ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  \
0          1      20000    2          2         1   24      2      2     -1
1          2     120000    2          2         2   26     -1      2      0
2          3      90000    2          2         2   34      0      0      0
3          4      50000    2          2         1   37      0      0      0
4          5      50000    1          2         1   57     -1      0     -1
…      …         …  …       …         …  …      …     …      …
29995  29996     220000    1          3         1   39      0      0      0
29996  29997     150000    1          3         2   43     -1     -1     -1
29997  29998      30000    1          2         2   37      4      3      2
29998  29999      80000    1          3         1   41      1     -1      0
29999  30000      50000    1          2         1   46      0      0      0

       PAY_4  …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  \
```

```
0           -1  …         0          0          0          0        689
1            0  …      3272       3455       3261          0       1000
2            0  …     14331      14948      15549       1518       1500
3            0  …     28314      28959      29547       2000       2019
4            0  …     20940      19146      19131       2000      36681
…          …  …         …        …        …        …        …
29995        0  …     88004      31237      15980       8500      20000
29996       -1  …      8979       5190          0       1837       3526
29997       -1  …     20878      20582      19357          0          0
29998        0  …     52774      11855      48944      85900       3409
29999        0  …     36535      32428      15313       2078       1800

       PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  default payment next month
0             0         0         0         0                           1
1          1000      1000         0      2000                           1
2          1000      1000      1000      5000                           0
3          1200      1100      1069      1000                           0
4         10000      9000       689       679                           0
…           …       …       …       …                         …
29995      5003      3047      5000      1000                           0
29996      8998       129         0         0                           0
29997     22000      4200      2000      3100                           1
29998      1178      1926     52964      1804                           1
29999      1430      1000      1000      1000                           1

[30000 rows x 25 columns]
```

[21]:
```
df.head(10)
```

[21]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1      20000    2          2         1   24      2      2     -1     -1
1   2     120000    2          2         2   26     -1      2      0      0
2   3      90000    2          2         2   34      0      0      0      0
3   4      50000    2          2         1   37      0      0      0      0
4   5      50000    1          2         1   57     -1      0     -1      0
5   6      50000    1          1         2   37      0      0      0      0
6   7     500000    1          1         2   29      0      0      0      0
7   8     100000    2          2         2   23      0     -1     -1      0
8   9     140000    2          3         1   28      0      0      2      0
9  10      20000    1          3         2   35     -2     -2     -2     -2

   …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  …          0          0          0         0       689         0
1  …       3272       3455       3261         0      1000      1000
2  …      14331      14948      15549      1518      1500      1000
3  …      28314      28959      29547      2000      2019      1200
4  …      20940      19146      19131      2000     36681     10000
```

```
5  …      19394      19619      20024      2500      1815       657
6  …     542653     483003     473944     55000     40000     38000
7  …        221       -159        567       380       601         0
8  …      12211      11793       3719      3329         0       432
9  …          0      13007      13912         0         0         0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default payment next month
0         0         0         0                           1
1      1000         0      2000                           1
2      1000      1000      5000                           0
3      1100      1069      1000                           0
4      9000       689       679                           0
5      1000      1000       800                           0
6     20239     13750     13770                           0
7       581      1687      1542                           0
8      1000      1000      1000                           0
9     13007      1122         0                           0

[10 rows x 25 columns]
```

[22]:

```python
# Information
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  int64
 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  int64
 13  BILL_AMT2                   30000 non-null  int64
 14  BILL_AMT3                   30000 non-null  int64
 15  BILL_AMT4                   30000 non-null  int64
 16  BILL_AMT5                   30000 non-null  int64
 17  BILL_AMT6                   30000 non-null  int64
 18  PAY_AMT1                    30000 non-null  int64
```

```
 19  PAY_AMT2                    30000 non-null  int64
 20  PAY_AMT3                    30000 non-null  int64
 21  PAY_AMT4                    30000 non-null  int64
 22  PAY_AMT5                    30000 non-null  int64
 23  PAY_AMT6                    30000 non-null  int64
 24  default payment next month  30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

[23]:
```python
df.shape
```

[23]:
```
(30000, 25)
```

[24]:
```python
df.describe()
```

[24]:
```
                 ID        LIMIT_BAL           SEX     EDUCATION      MARRIAGE  \
count  30000.000000     30000.000000  30000.000000  30000.000000  30000.000000
mean   15000.500000    167484.322667      1.603733      1.853133      1.551867
std     8660.398374    129747.661567      0.489129      0.790349      0.521970
min        1.000000     10000.000000      1.000000      0.000000      0.000000
25%     7500.750000     50000.000000      1.000000      1.000000      1.000000
50%    15000.500000    140000.000000      2.000000      2.000000      2.000000
75%    22500.250000    240000.000000      2.000000      2.000000      2.000000
max    30000.000000   1000000.000000      2.000000      6.000000      3.000000

                AGE         PAY_0         PAY_2         PAY_3         PAY_4  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean      35.485500     -0.016700     -0.133767     -0.166200     -0.220667
std        9.217904      1.123802      1.197186      1.196868      1.169139
min       21.000000     -2.000000     -2.000000     -2.000000     -2.000000
25%       28.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%       34.000000      0.000000      0.000000      0.000000      0.000000
75%       41.000000      0.000000      0.000000      0.000000      0.000000
max       79.000000      8.000000      8.000000      8.000000      8.000000

       …      BILL_AMT4      BILL_AMT5      BILL_AMT6       PAY_AMT1  \
count  …   30000.000000   30000.000000   30000.000000   30000.000000
mean   …   43262.948967   40311.400967   38871.760400    5663.580500
std    …   64332.856134   60797.155770   59554.107537   16563.280354
min    … -170000.000000  -81334.000000 -339603.000000       0.000000
25%    …    2326.750000    1763.000000    1256.000000    1000.000000
50%    …   19052.000000   18104.500000   17071.000000    2100.000000
75%    …   54506.000000   50190.500000   49198.250000    5006.000000
max    …  891586.000000  927171.000000  961664.000000  873552.000000

           PAY_AMT2      PAY_AMT3       PAY_AMT4       PAY_AMT5  \
count  3.000000e+04   30000.00000   30000.000000   30000.000000
```

```
mean   5.921163e+03    5225.68150    4826.076867    4799.387633
std    2.304087e+04   17606.96147   15666.159744   15278.305679
min    0.000000e+00       0.00000       0.000000       0.000000
25%    8.330000e+02     390.00000     296.000000     252.500000
50%    2.009000e+03    1800.00000    1500.000000    1500.000000
75%    5.000000e+03    4505.00000    4013.250000    4031.500000
max    1.684259e+06  896040.00000  621000.000000  426529.000000

            PAY_AMT6  default payment next month
count   30000.000000                30000.000000
mean     5215.502567                    0.221200
std     17777.465775                    0.415062
min         0.000000                    0.000000
25%       117.750000                    0.000000
50%      1500.000000                    0.000000
75%      4000.000000                    0.000000
max    528666.000000                    1.000000

[8 rows x 25 columns]
```

[25]:
```
df.columns
```

[25]:
```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default payment next month'],
      dtype='object')
```

[26]:
```
df.dtypes
```

[26]:
```
ID                            int64
LIMIT_BAL                     int64
SEX                           int64
EDUCATION                     int64
MARRIAGE                      int64
AGE                           int64
PAY_0                         int64
PAY_2                         int64
PAY_3                         int64
PAY_4                         int64
PAY_5                         int64
PAY_6                         int64
BILL_AMT1                     int64
BILL_AMT2                     int64
BILL_AMT3                     int64
BILL_AMT4                     int64
```

```
BILL_AMT5                     int64
BILL_AMT6                     int64
PAY_AMT1                      int64
PAY_AMT2                      int64
PAY_AMT3                      int64
PAY_AMT4                      int64
PAY_AMT5                      int64
PAY_AMT6                      int64
default payment next month    int64
dtype: object
```

[27]:
```
print(df['default payment next month'].dtype)
```

```
int64
```

1.4.3 Data Cleaning and Preprocessing

[29]:
```
# Finding Duplicates
df.duplicated()
```

[29]:
```
0        False
1        False
2        False
3        False
4        False
         …
29995    False
29996    False
29997    False
29998    False
29999    False
Length: 30000, dtype: bool
```

[30]:
```
df.duplicated().sum()
```

[30]:
```
0
```

No duplicate values found in the Dataset

[32]:
```
# Finding and Handling null values
df.isnull().sum()
```

[32]:
```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
```

```
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default payment next month    0
dtype: int64
```

No Null values found in the dataset

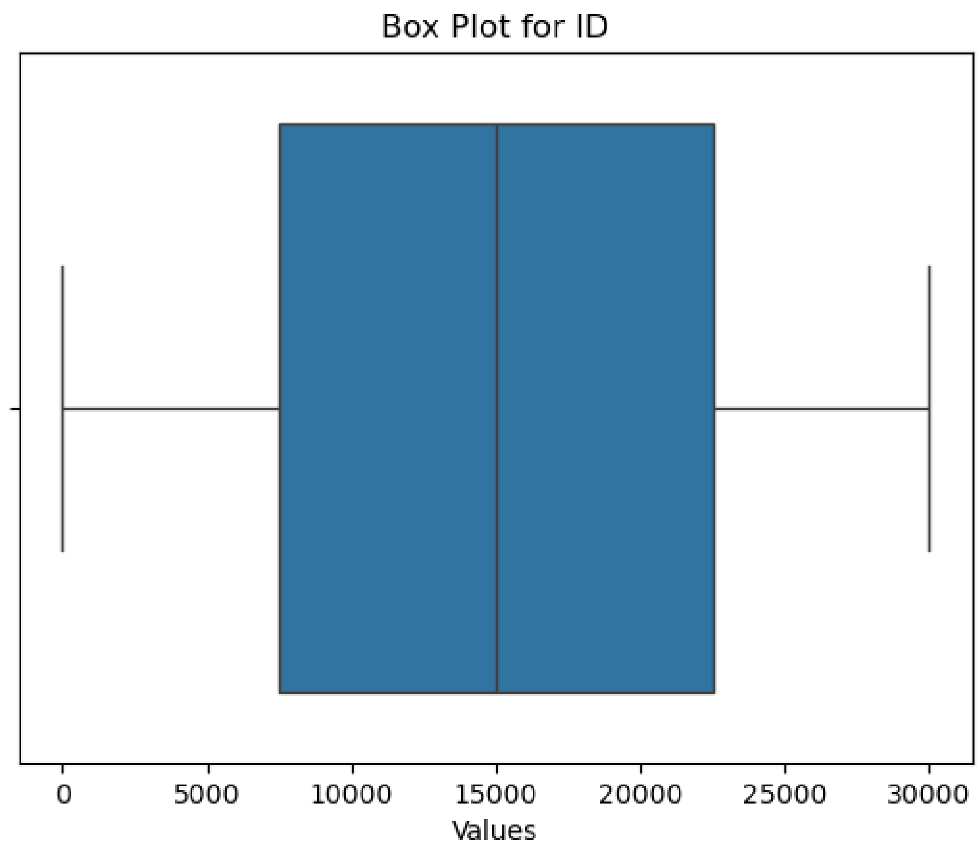
[34]:
```
df.describe()
```

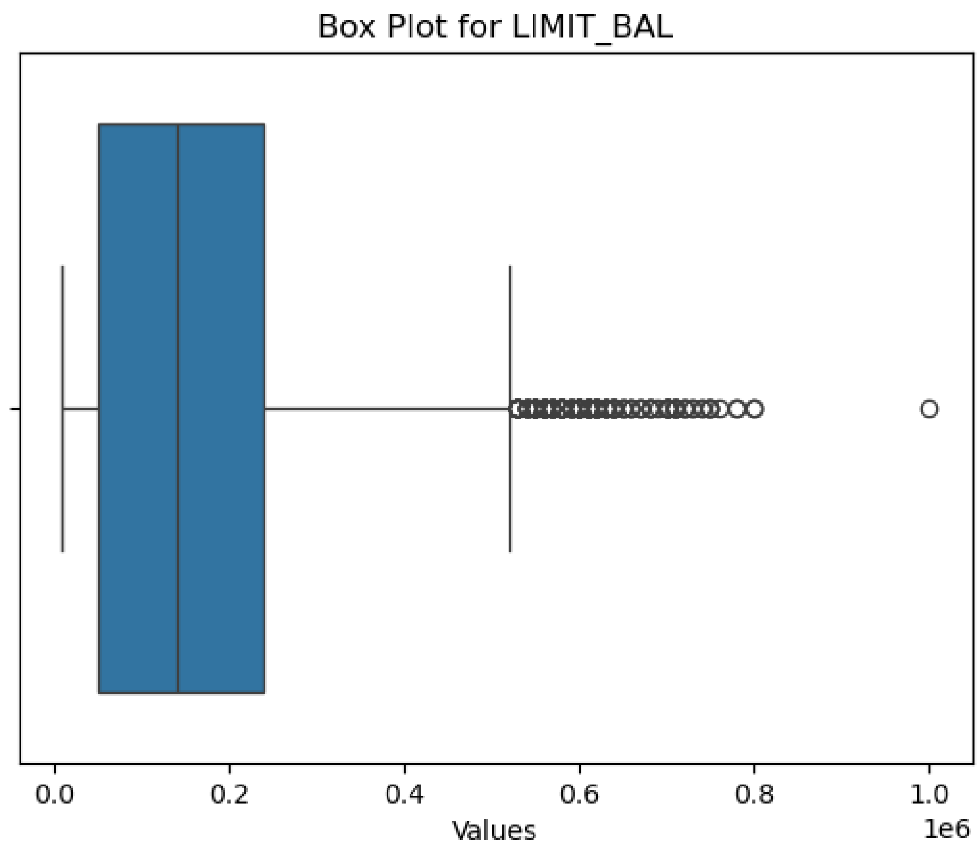
[34]:
```
                 ID        LIMIT_BAL           SEX     EDUCATION      MARRIAGE  \
count  30000.000000     30000.000000  30000.000000  30000.000000  30000.000000
mean   15000.500000    167484.322667      1.603733      1.853133      1.551867
std     8660.398374    129747.661567      0.489129      0.790349      0.521970
min        1.000000     10000.000000      1.000000      0.000000      0.000000
25%     7500.750000     50000.000000      1.000000      1.000000      1.000000
50%    15000.500000    140000.000000      2.000000      2.000000      2.000000
75%    22500.250000    240000.000000      2.000000      2.000000      2.000000
max    30000.000000   1000000.000000      2.000000      6.000000      3.000000

                AGE         PAY_0         PAY_2         PAY_3         PAY_4  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean      35.485500     -0.016700     -0.133767     -0.166200     -0.220667
std        9.217904      1.123802      1.197186      1.196868      1.169139
min       21.000000     -2.000000     -2.000000     -2.000000     -2.000000
25%       28.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%       34.000000      0.000000      0.000000      0.000000      0.000000
75%       41.000000      0.000000      0.000000      0.000000      0.000000
max       79.000000      8.000000      8.000000      8.000000      8.000000

       …     BILL_AMT4     BILL_AMT5     BILL_AMT6      PAY_AMT1  \
count  …  30000.000000  30000.000000  30000.000000  30000.000000
```

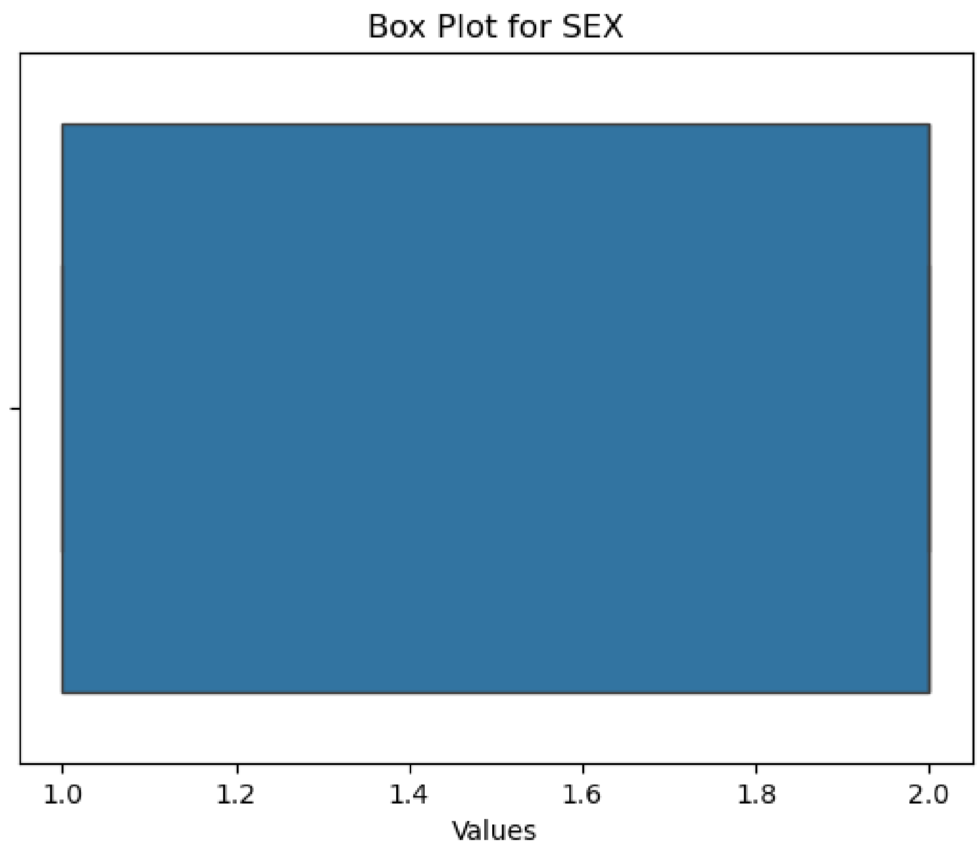
```
mean   …   43262.948967   40311.400967   38871.760400    5663.580500
std    …   64332.856134   60797.155770   59554.107537   16563.280354
min    … -170000.000000  -81334.000000 -339603.000000       0.000000
25%    …    2326.750000    1763.000000    1256.000000    1000.000000
50%    …   19052.000000   18104.500000   17071.000000    2100.000000
75%    …   54506.000000   50190.500000   49198.250000    5006.000000
max    …  891586.000000  927171.000000  961664.000000  873552.000000

           PAY_AMT2      PAY_AMT3       PAY_AMT4       PAY_AMT5  \
count  3.000000e+04   30000.00000   30000.000000   30000.000000
mean   5.921163e+03    5225.68150    4826.076867    4799.387633
std    2.304087e+04   17606.96147   15666.159744   15278.305679
min    0.000000e+00       0.00000       0.000000       0.000000
25%    8.330000e+02     390.00000     296.000000     252.500000
50%    2.009000e+03    1800.00000    1500.000000    1500.000000
75%    5.000000e+03    4505.00000    4013.250000    4031.500000
max    1.684259e+06  896040.00000  621000.000000  426529.000000

            PAY_AMT6  default payment next month
count   30000.000000                30000.000000
mean     5215.502567                    0.221200
std     17777.465775                    0.415062
min         0.000000                    0.000000
25%       117.750000                    0.000000
50%      1500.000000                    0.000000
75%      4000.000000                    0.000000
max    528666.000000                    1.000000

[8 rows x 25 columns]
```

[35]:
```
# OUtliers Detection and visualisation with boxplot

numerical_columns = df.select_dtypes(include=['number'])
```

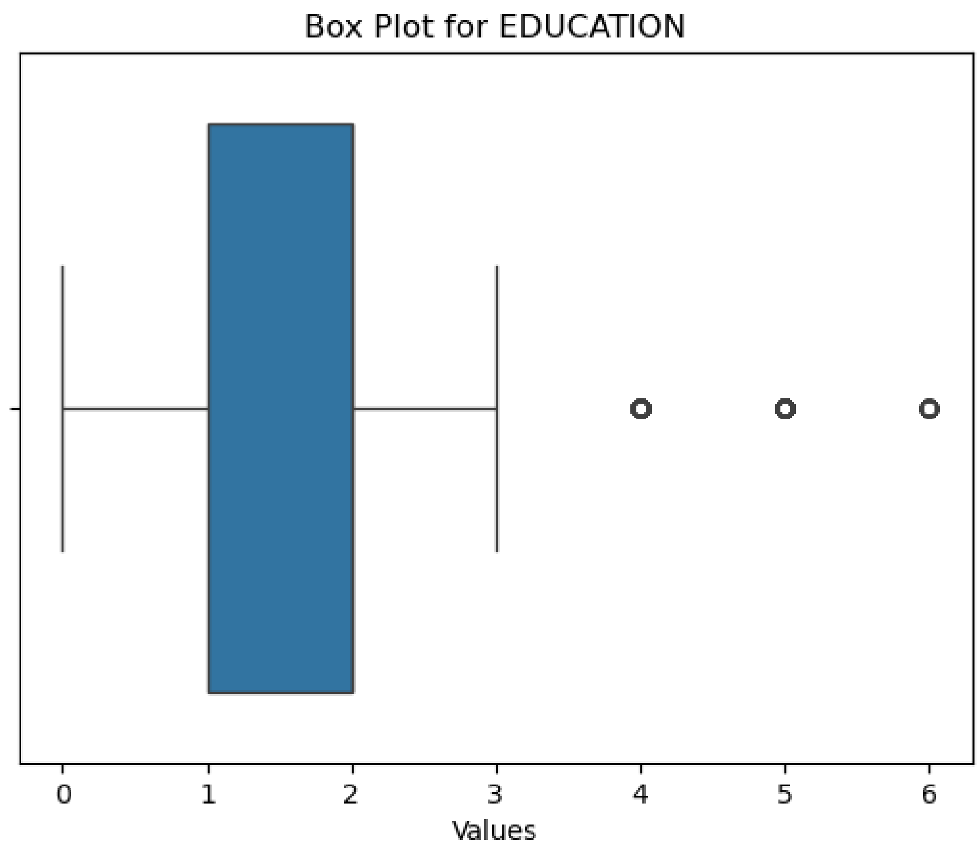
[36]:
```
# Forloop for Boxplot
for column in numerical_columns:
    plt.figure()
    sns.boxplot(data=df, x=column)
    plt.title(f"Box Plot for {column}")
    plt.xlabel("Values")
    plt.show()
```

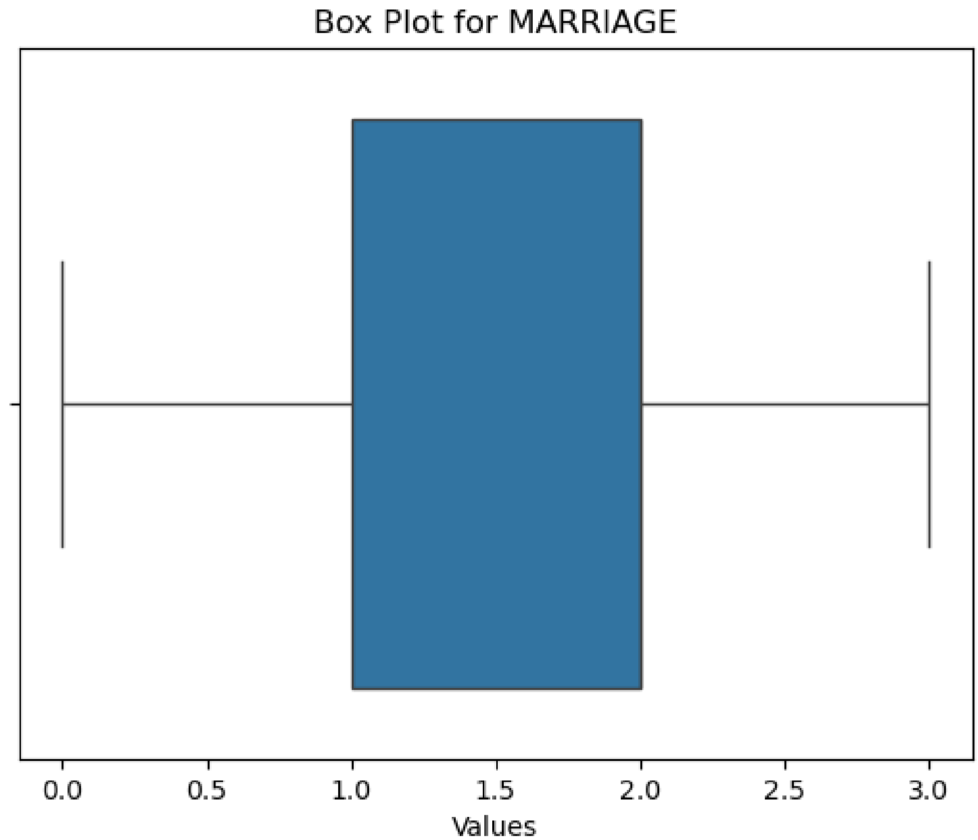

Box Plot for ID
0
5000
10000
15000
20000
25000
30000
Values

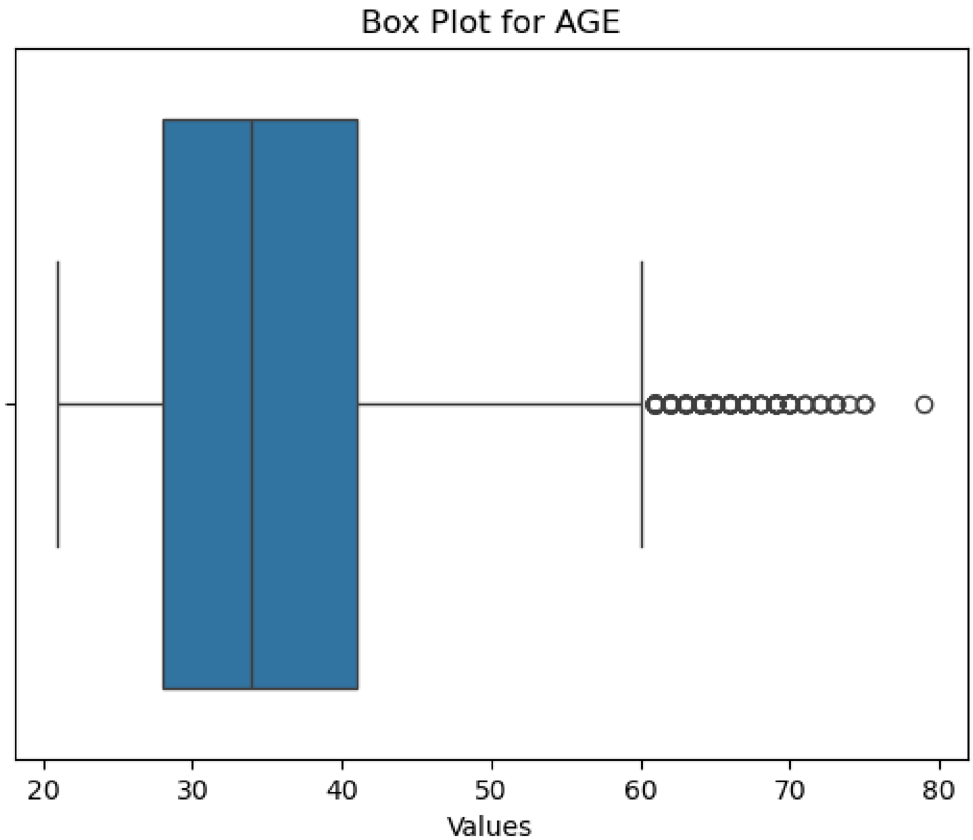

Box Plot for LIMIT_BAL
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

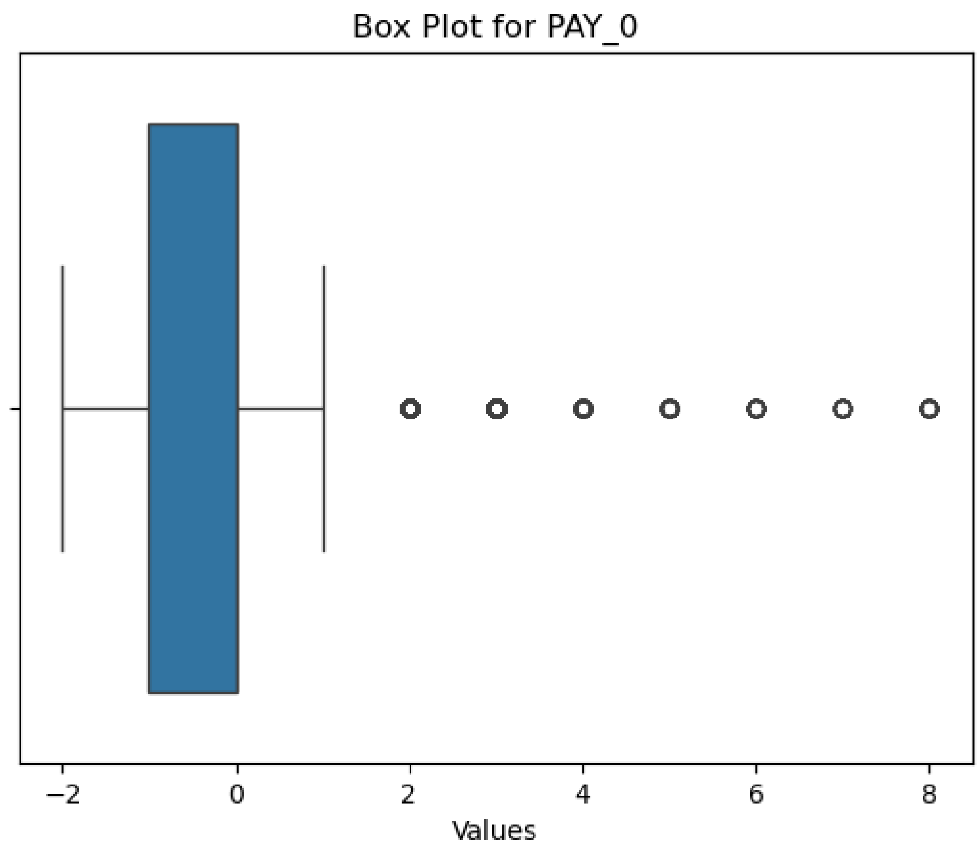


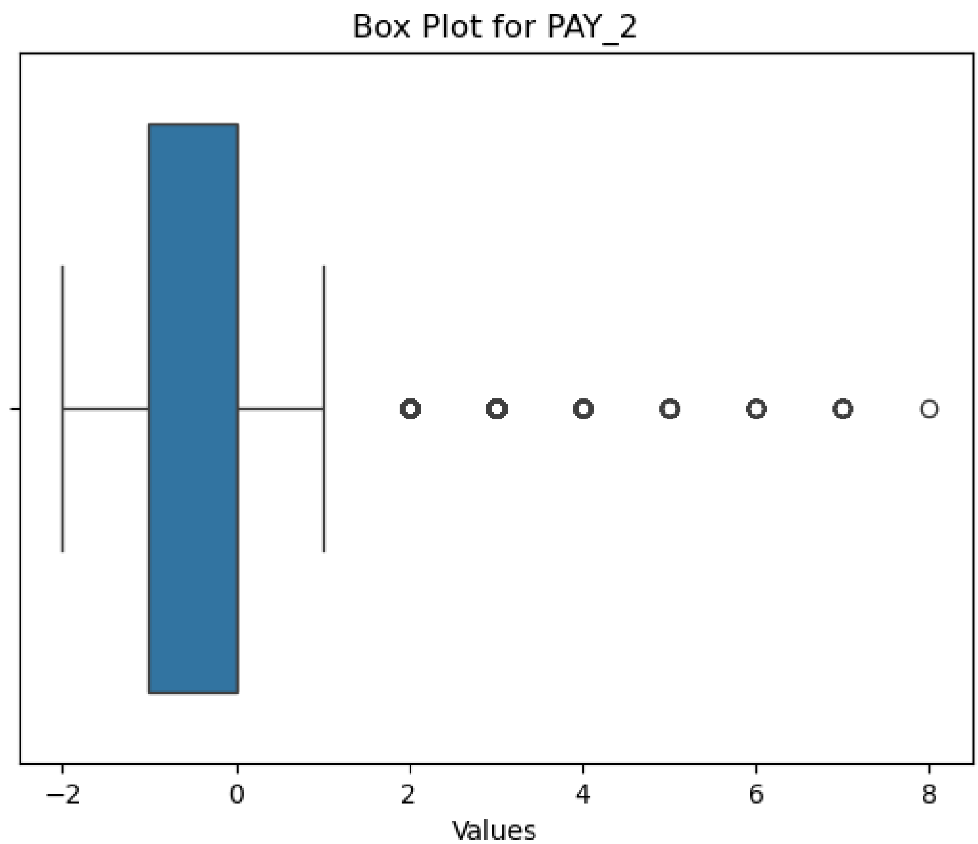
Box Plot for SEX
1.0
1.2
1.4
1.6
1.8
2.0
Values

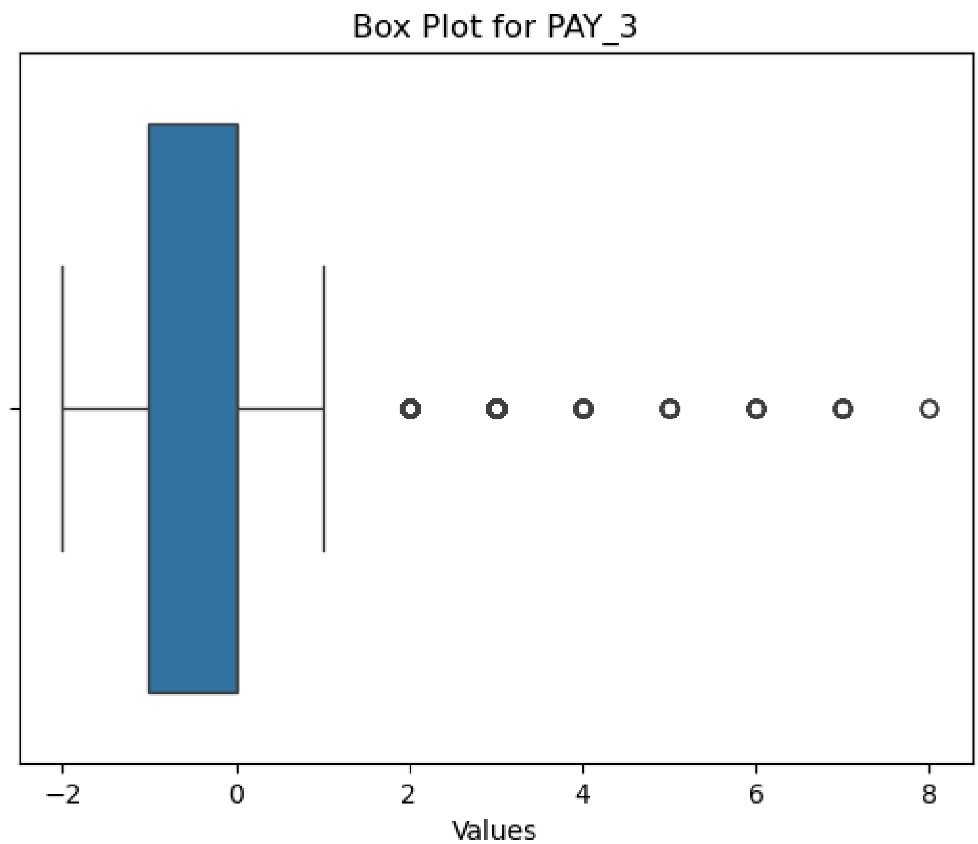

Box Plot for EDUCATION
0
1
2
3
4
5
6
Values

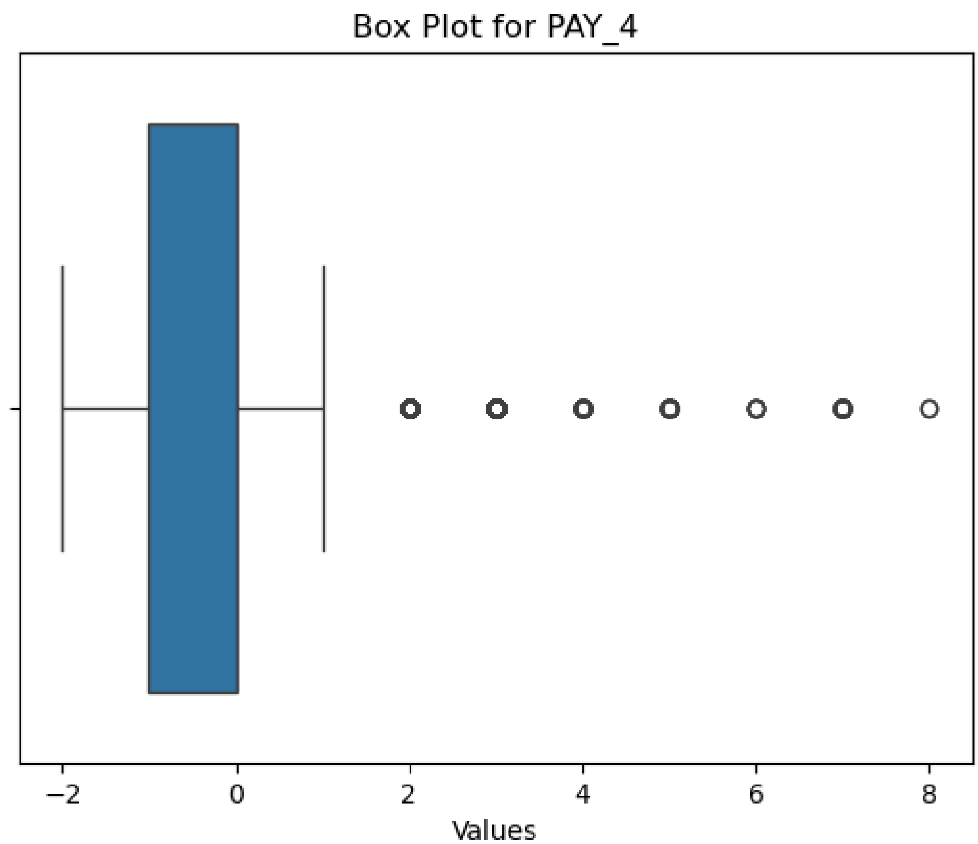

Box Plot for MARRIAGE
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Values

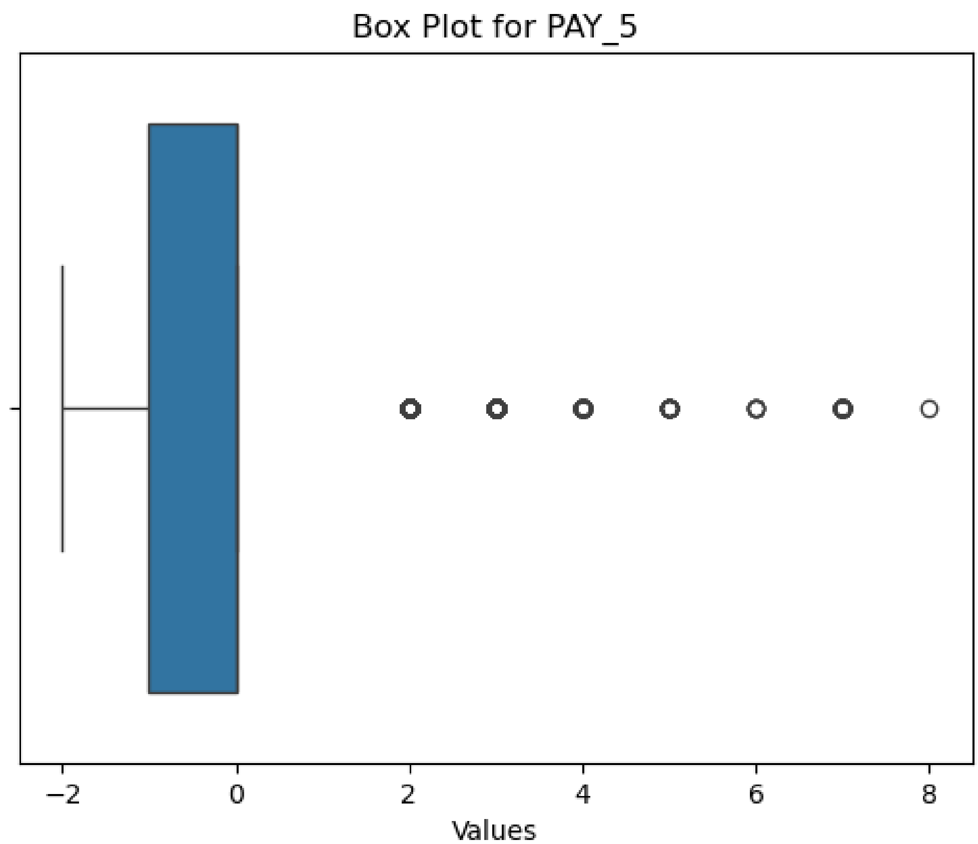

Box Plot for AGE
20
30
40
50
60
70
80
Values

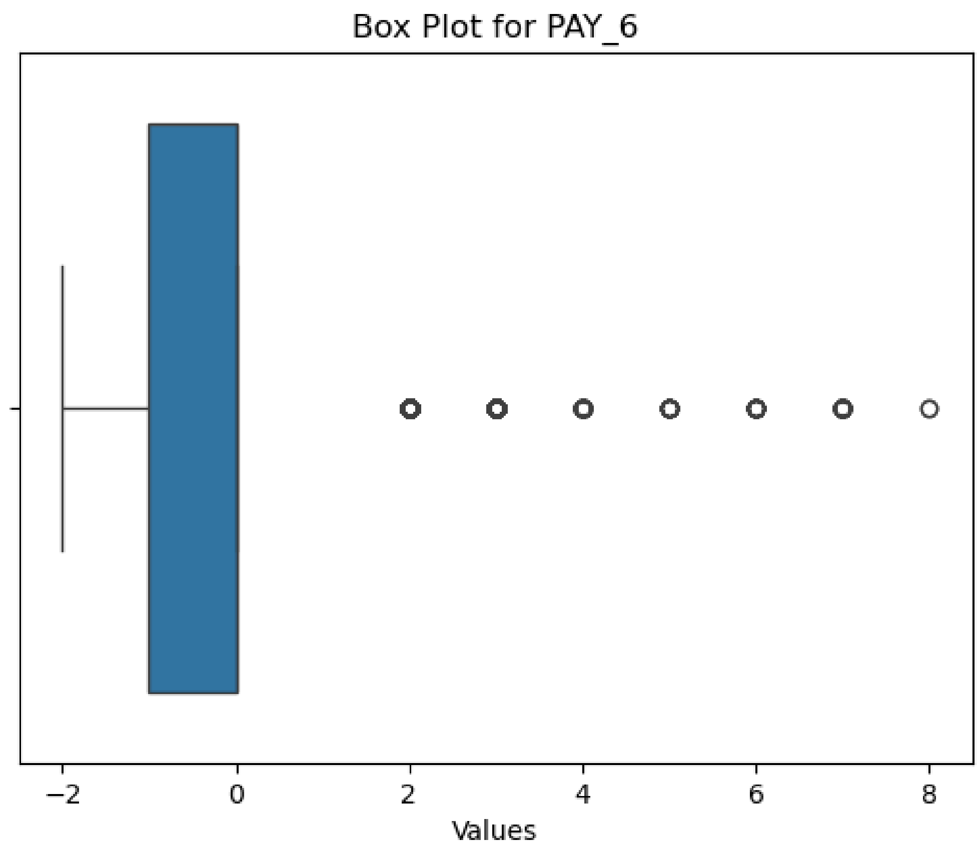

Box Plot for PAY_0
−2
0
2
4
6
8
Values

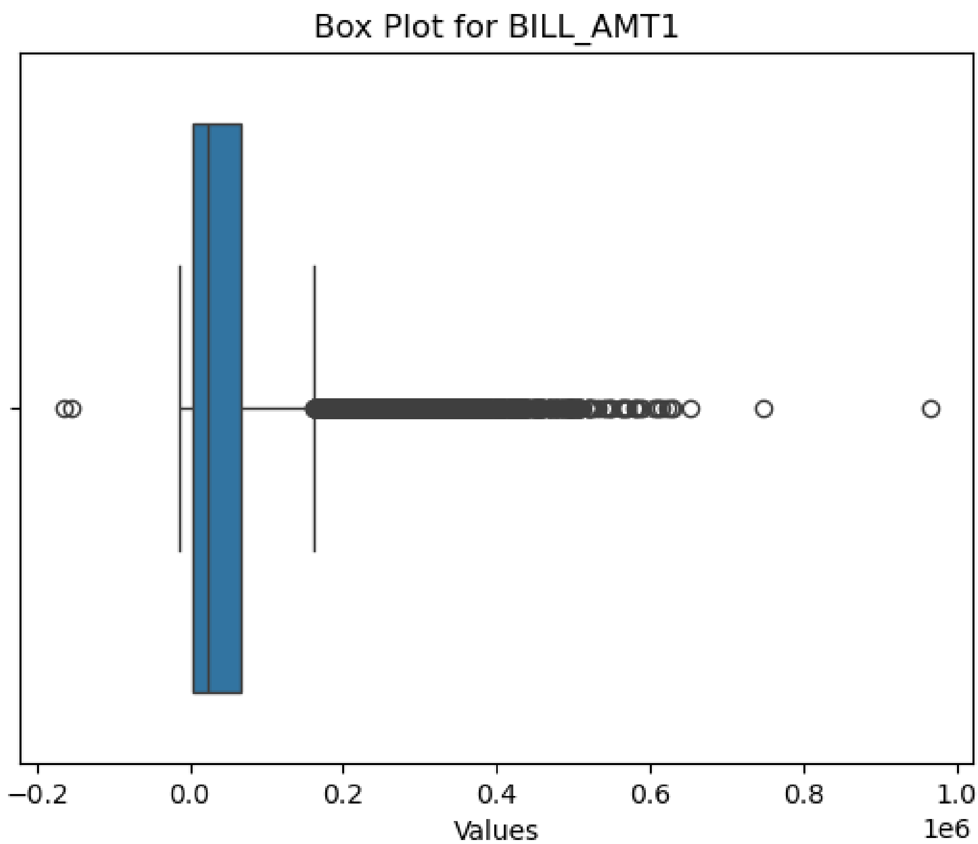

Box Plot for PAY_2
−2
0
2
4
6
8
Values

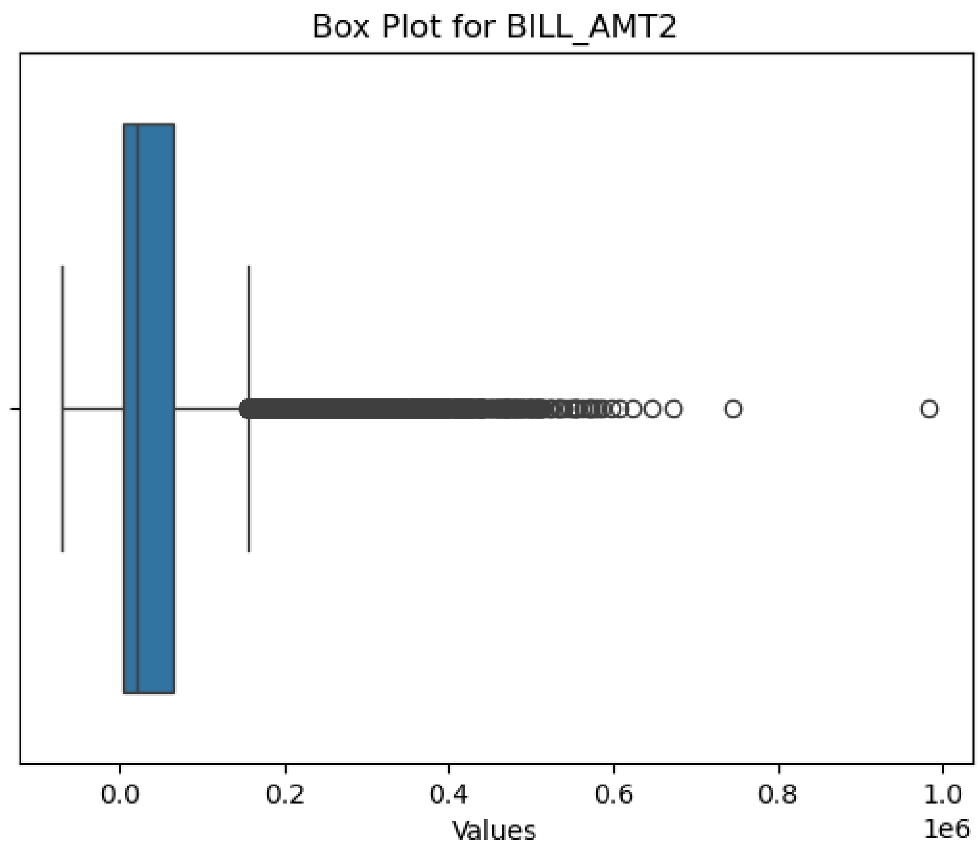

Box Plot for PAY_3
−2
0
2
4
6
8
Values

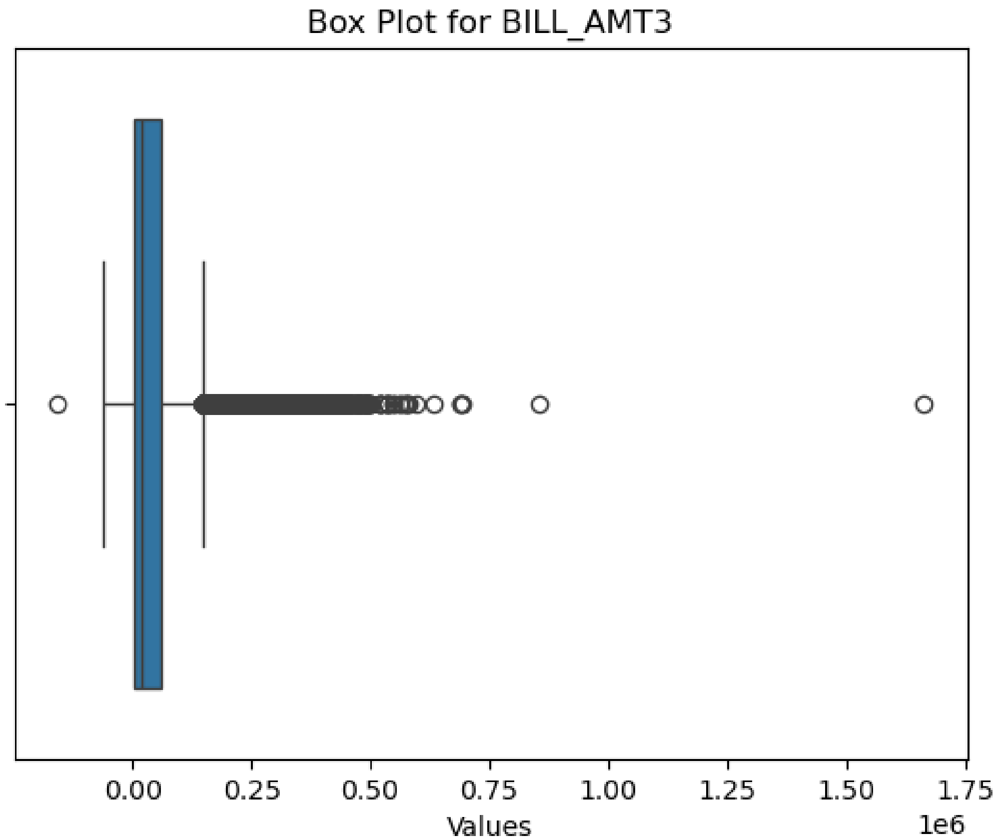

Box Plot for PAY_4
−2
0
2
4
6
8
Values


Box Plot for PAY_5
−2
0
2
4
6
8
Values

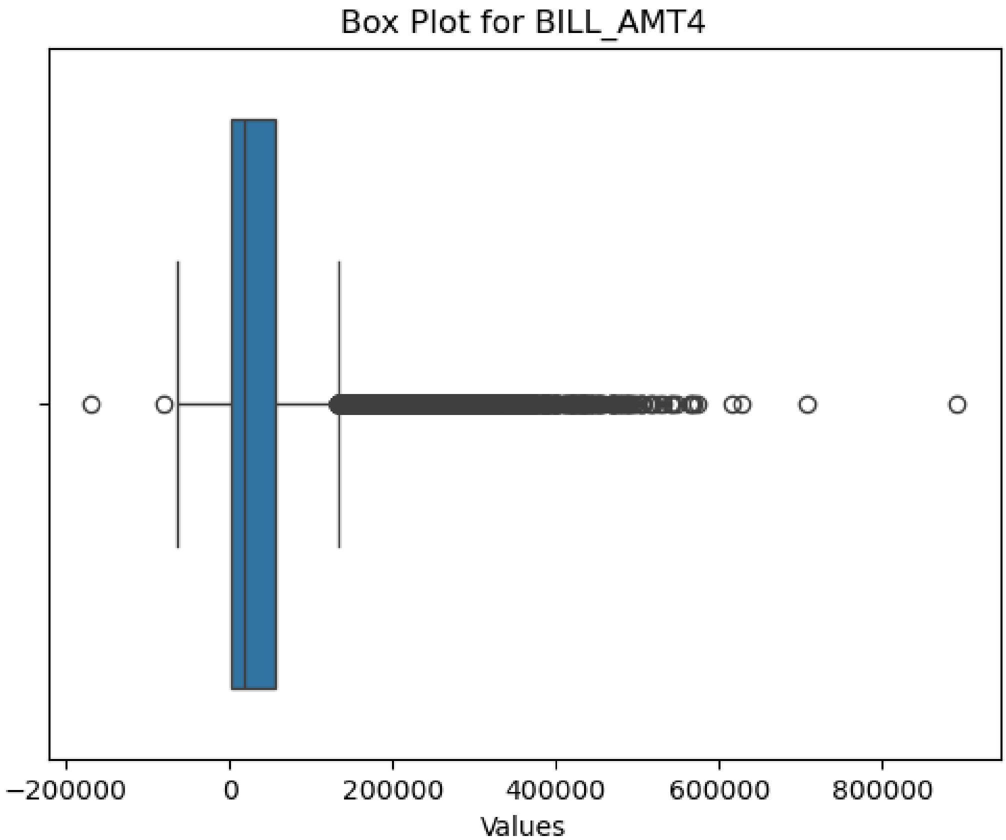

Box Plot for PAY_6
−2
0
2
4
6
8
Values


Box Plot for BILL_AMT1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

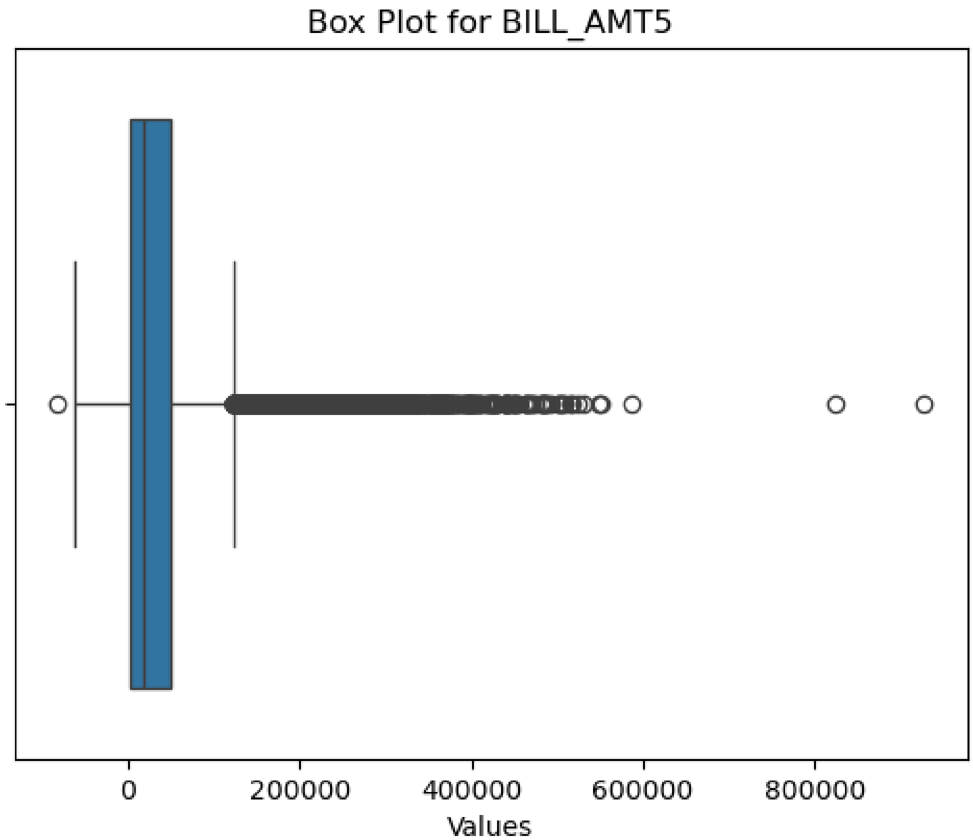

Box Plot for BILL_AMT2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

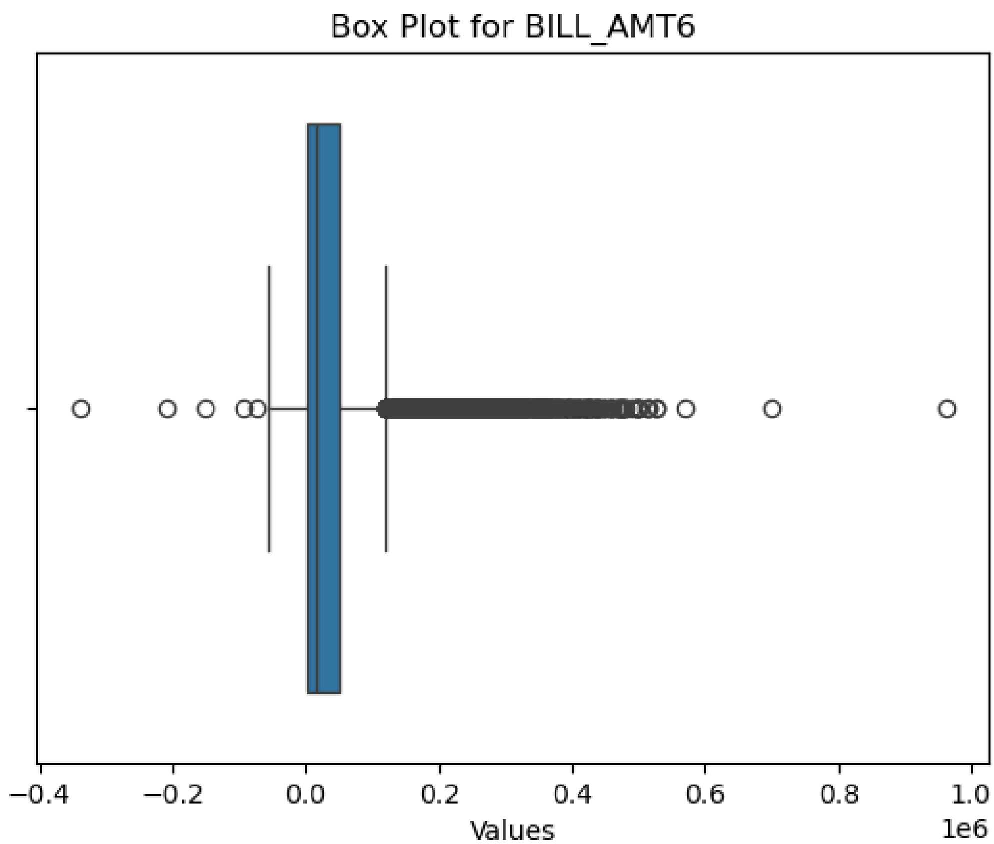


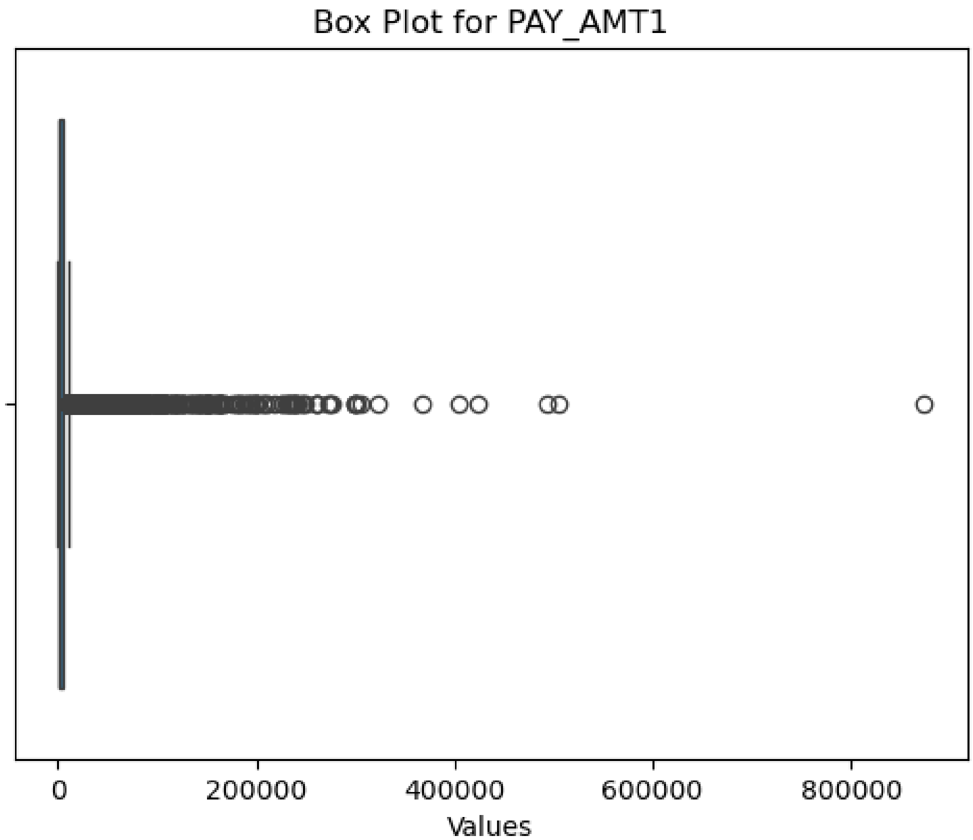
Box Plot for BILL_AMT3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Values
1e6

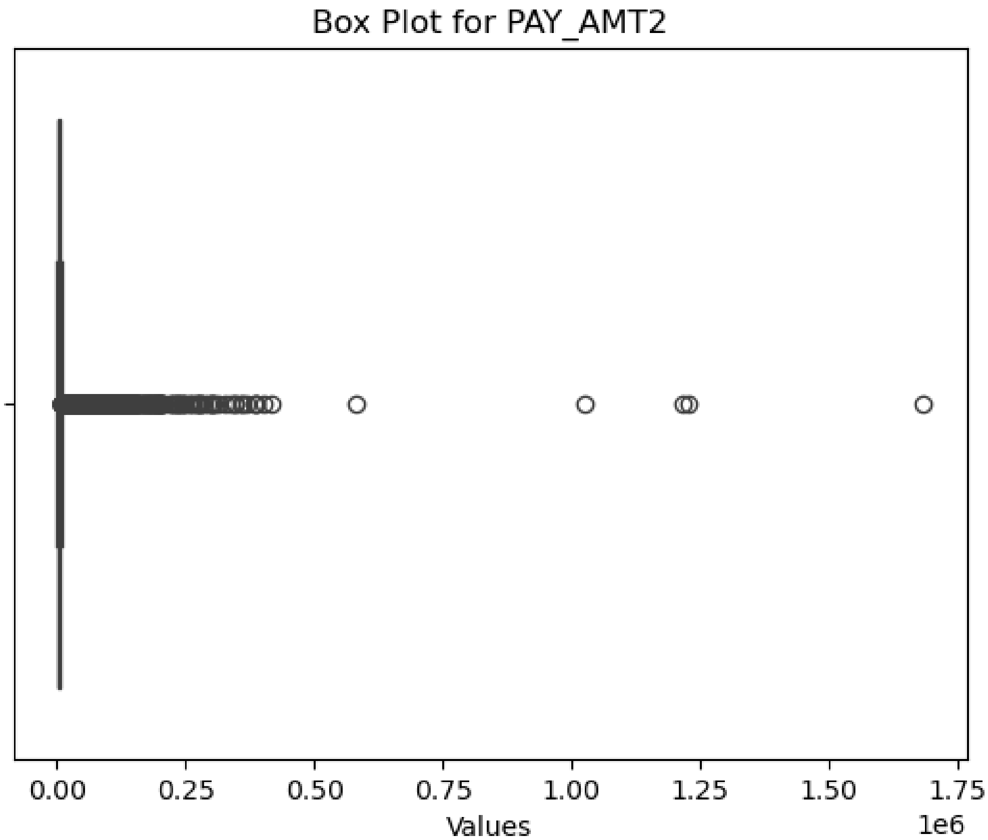


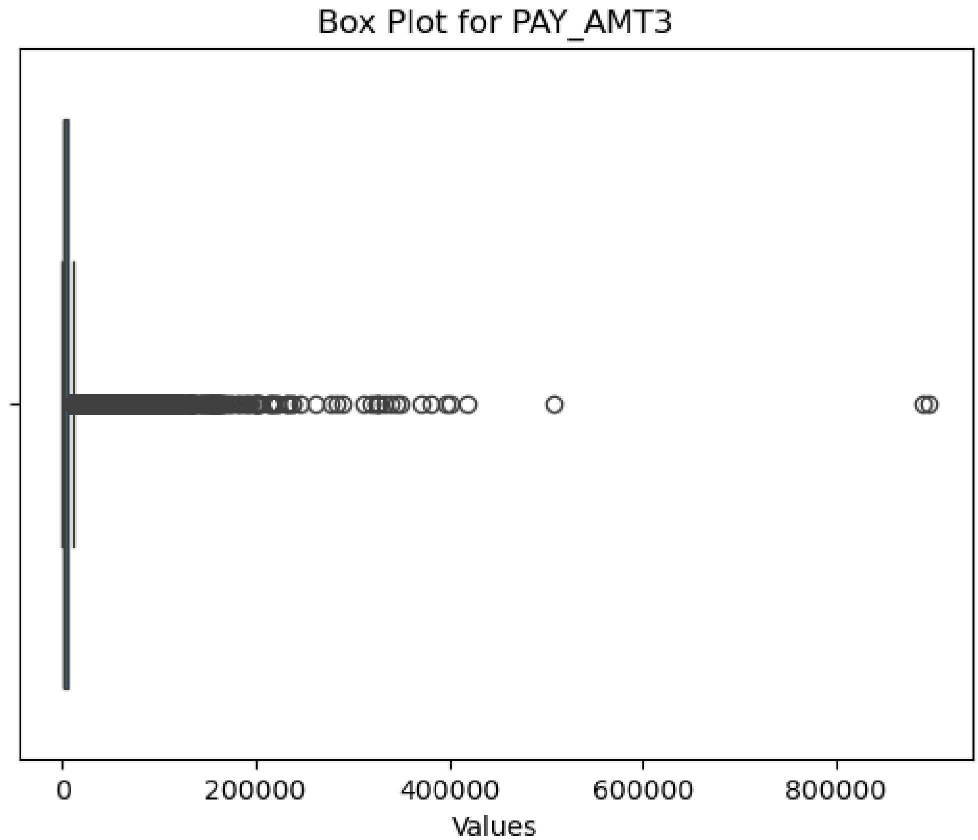
Box Plot for BILL_AMT4
−200000
0
200000
400000
600000
800000
Values

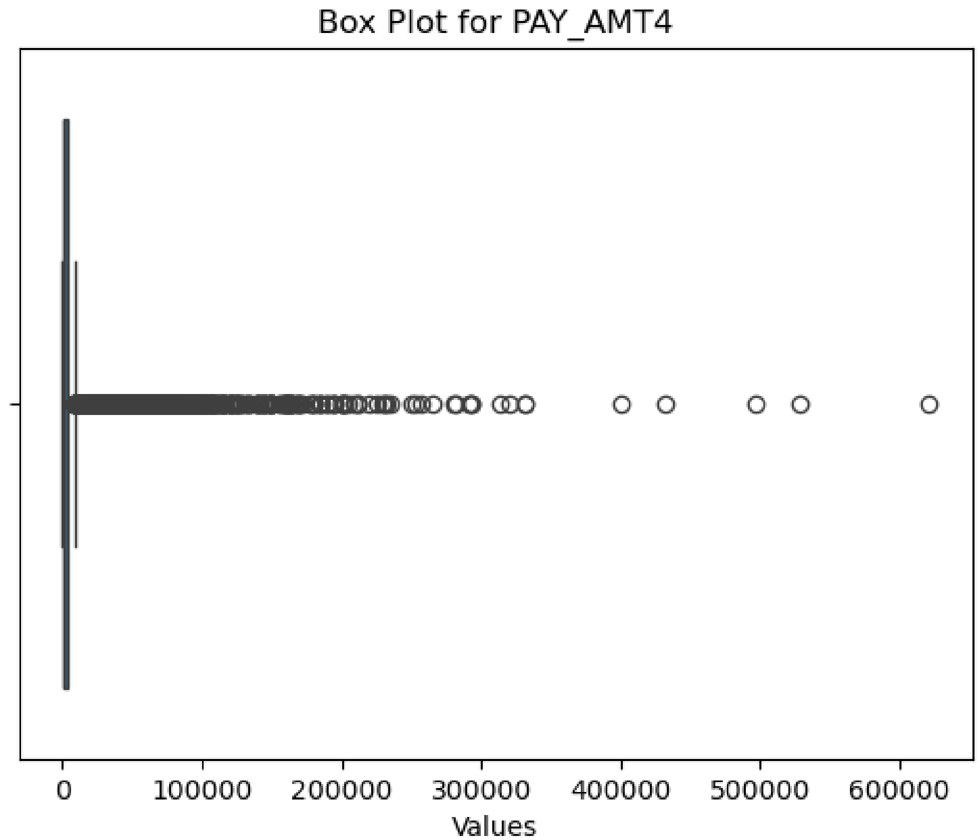

Box Plot for BILL_AMT5
0
200000
400000
600000
800000
Values

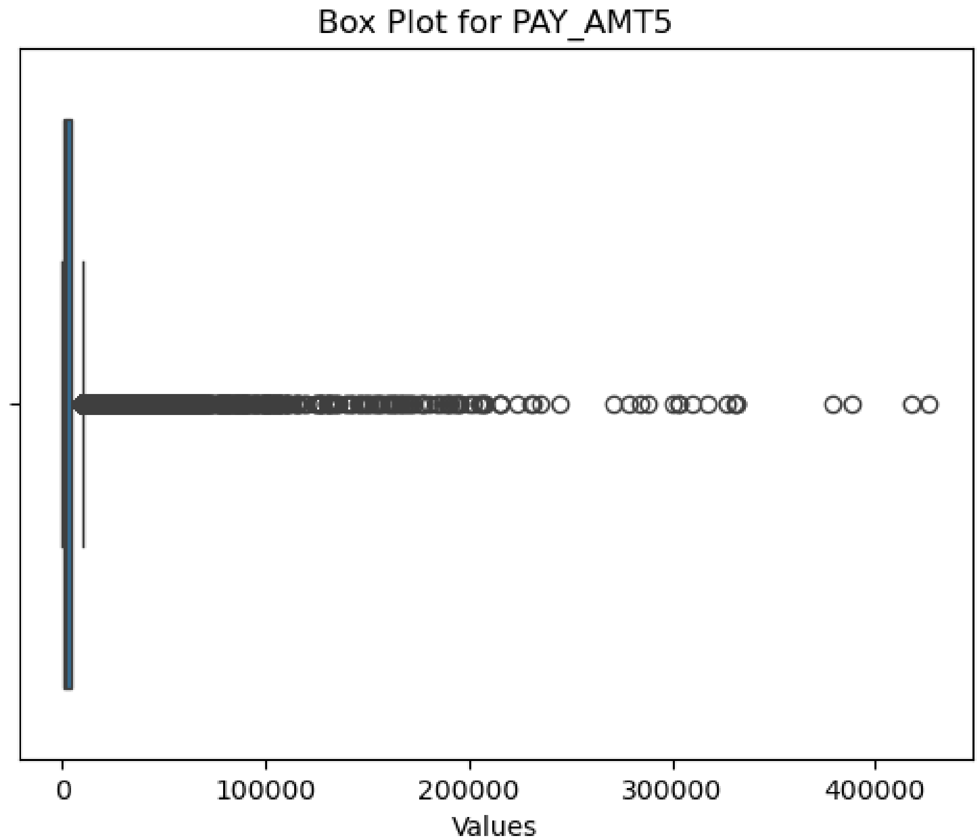

Box Plot for BILL_AMT6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

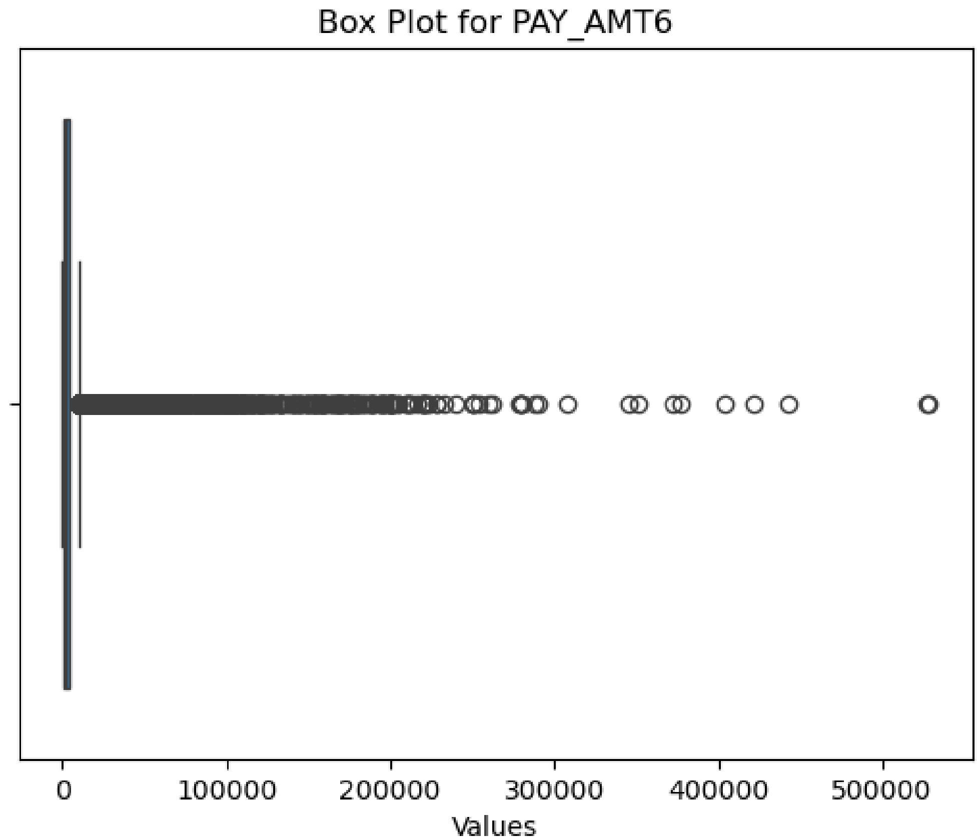

Box Plot for PAY_AMT1
0
200000
400000
600000
800000
Values

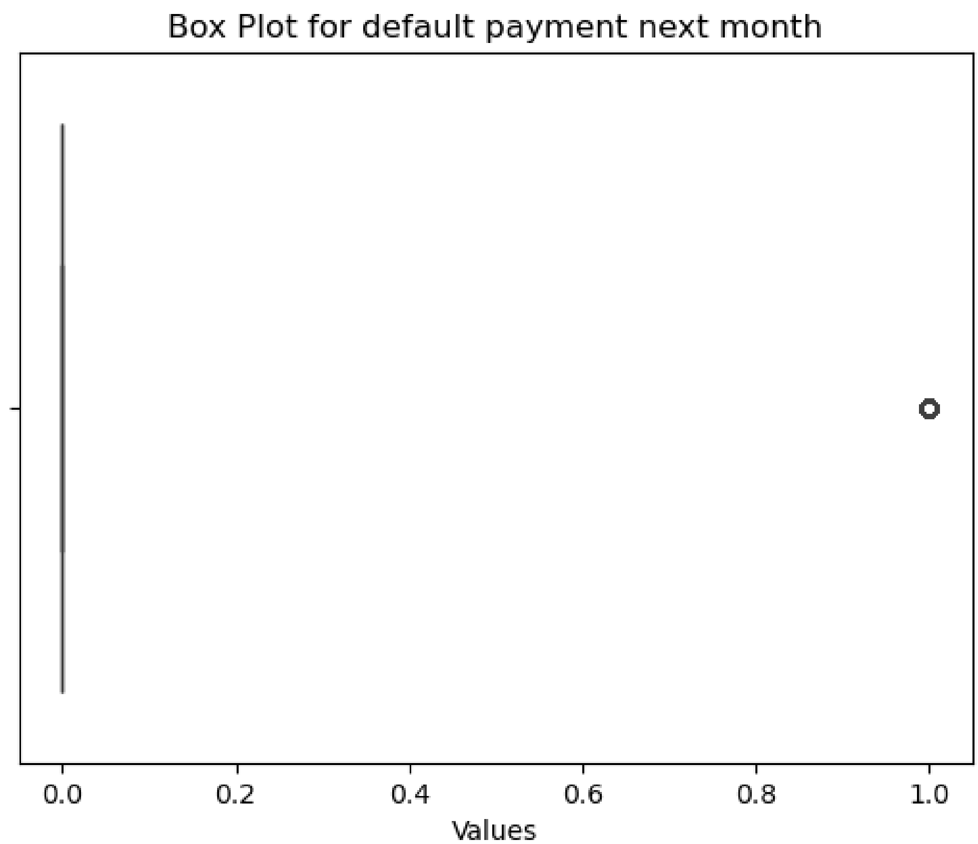

Box Plot for PAY_AMT2
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Values
1e6


Box Plot for PAY_AMT3
0
200000
400000
600000
800000
Values


Box Plot for PAY_AMT4
0
100000
200000
300000
400000
500000
600000
Values


Box Plot for PAY_AMT5
0
100000
200000
300000
400000
Values


Box Plot for PAY_AMT6
0
100000
200000
300000
400000
500000
Values



Found out that the Dataset has Outliers

[38]:
```
df['default payment next month'].unique()
```

[38]:
```
array([1, 0], dtype=int64)
```

[39]:
```
print(df['default payment next month'].value_counts())
```

```
default payment next month
0    23364
1     6636
Name: count, dtype: int64
```

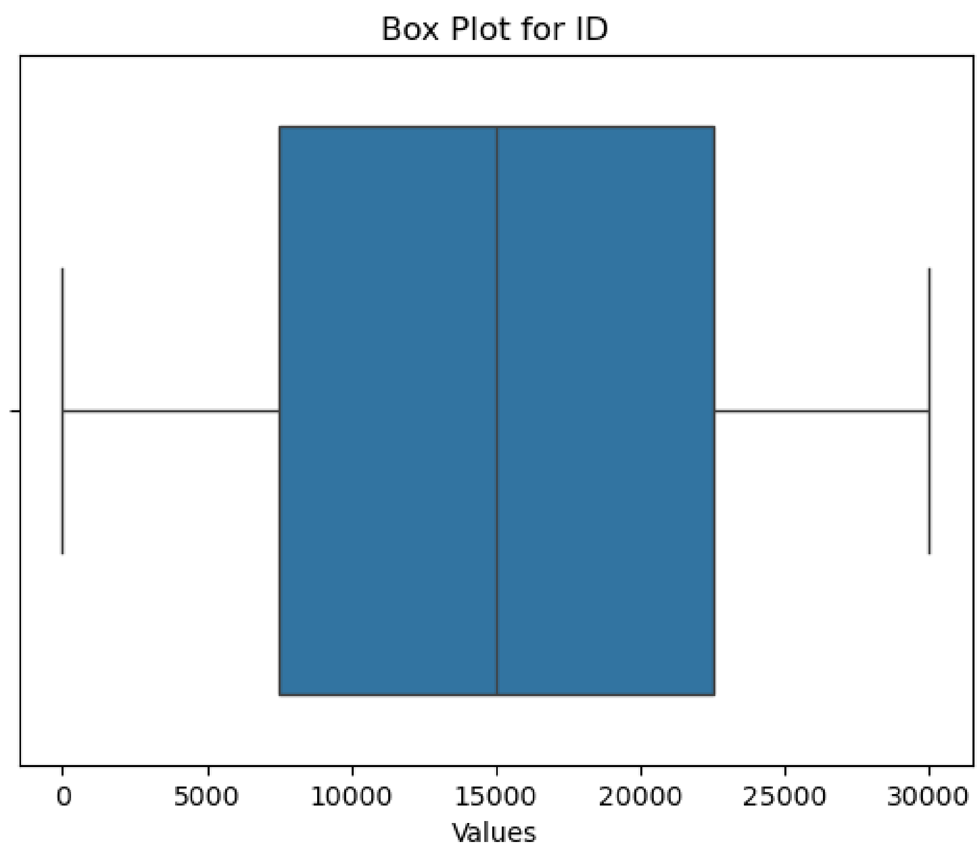
The Target feature has only two values. 1(yes) and 0(No). Since this is a classification machine learning model, outliers in target variable doesn't exist.

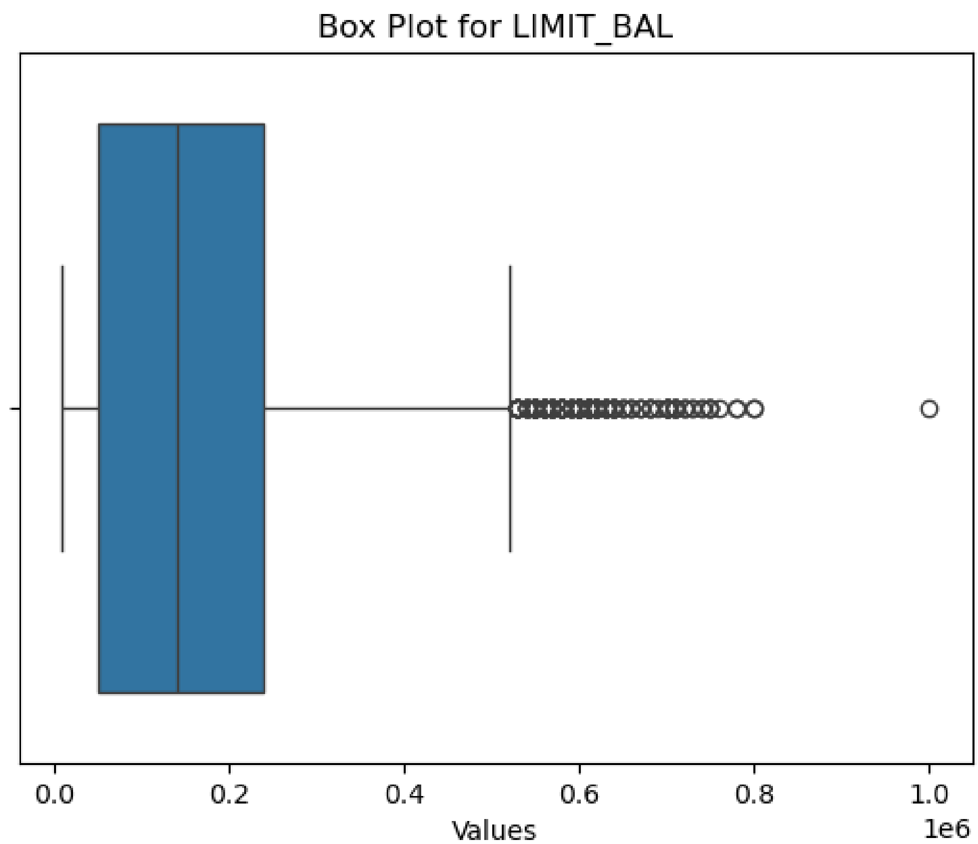
[41]:
```
# Feature to fix Outliers
```

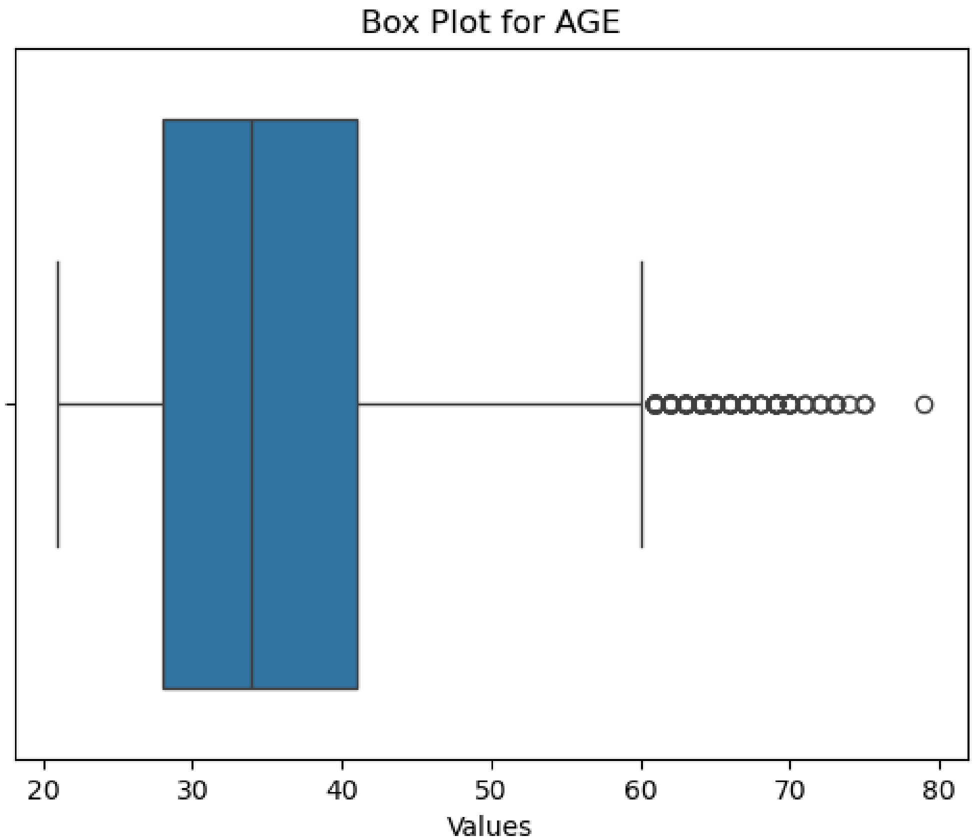
In the dataset, features like PAY_0 to PAY_6 are ordinal values (-2 to 9) which represent the status of repayment. All the values are in the pre determined range. So there are no true outliers in such columns. In the dataset Education, Sex , Marriage are also ordinal values having categorical behaviour.

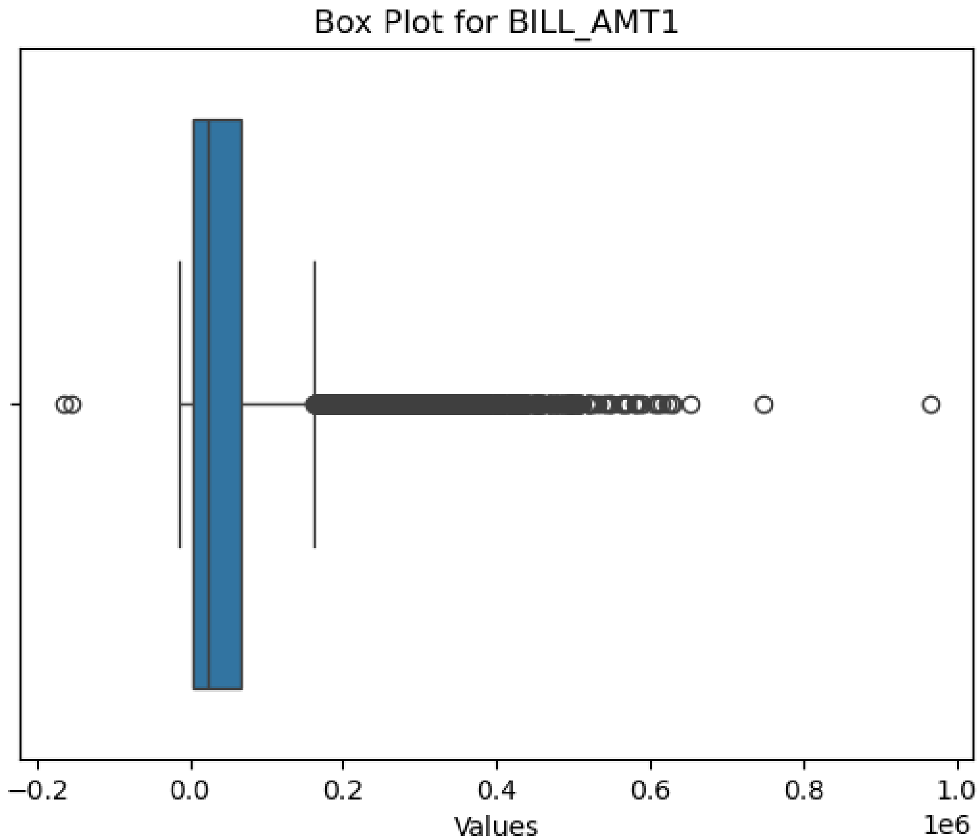
[43]:
```
# These are the features having continues numerical values. (ordinal value
 ↪having categotical behaviour doesnt have true outliers)
outlier_fix_columns = [
    'ID', 'LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3',
    'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
    'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6'
]
```

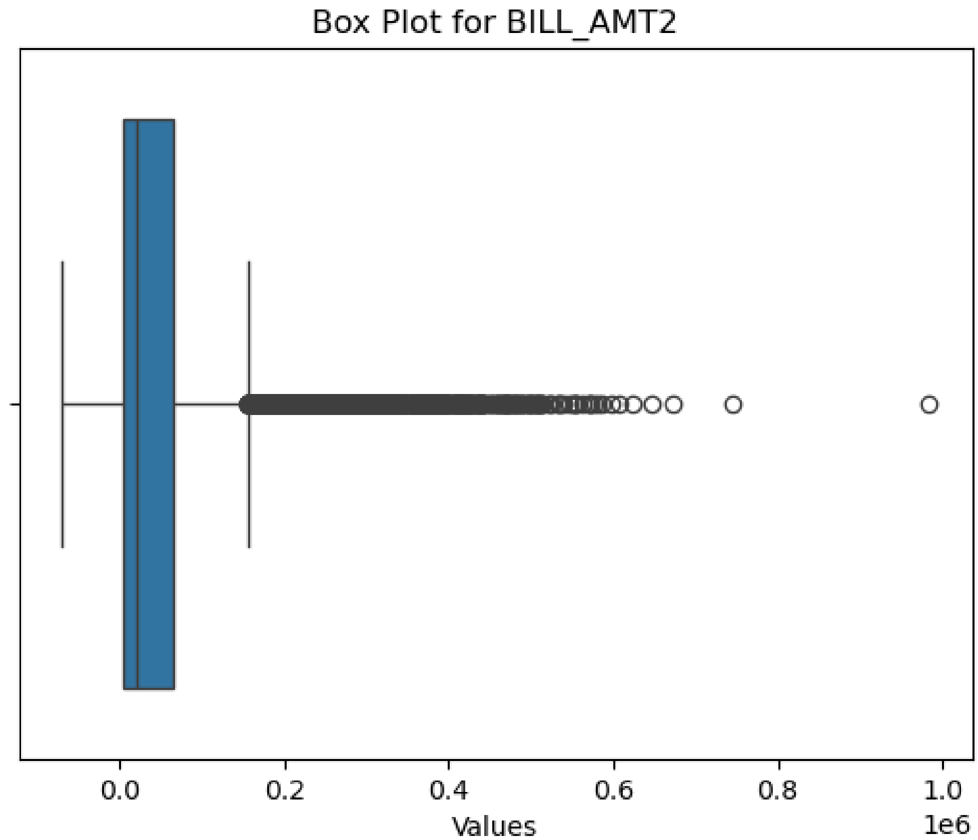
[44]:
```
# Visualising outliers using boxplot
for column in outlier_fix_columns:
    plt.figure()
    sns.boxplot(data=df, x=column)
    plt.title(f"Box Plot for {column}")
    plt.xlabel("Values")
    plt.show()
```



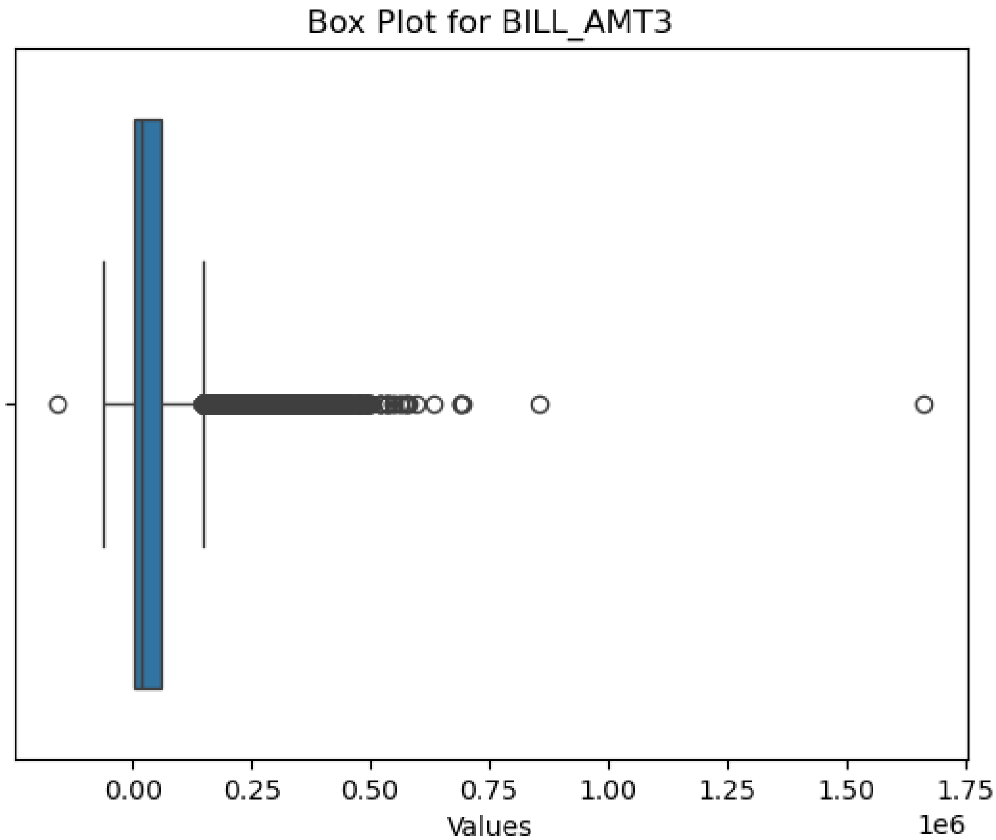

Box Plot for LIMIT_BAL
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

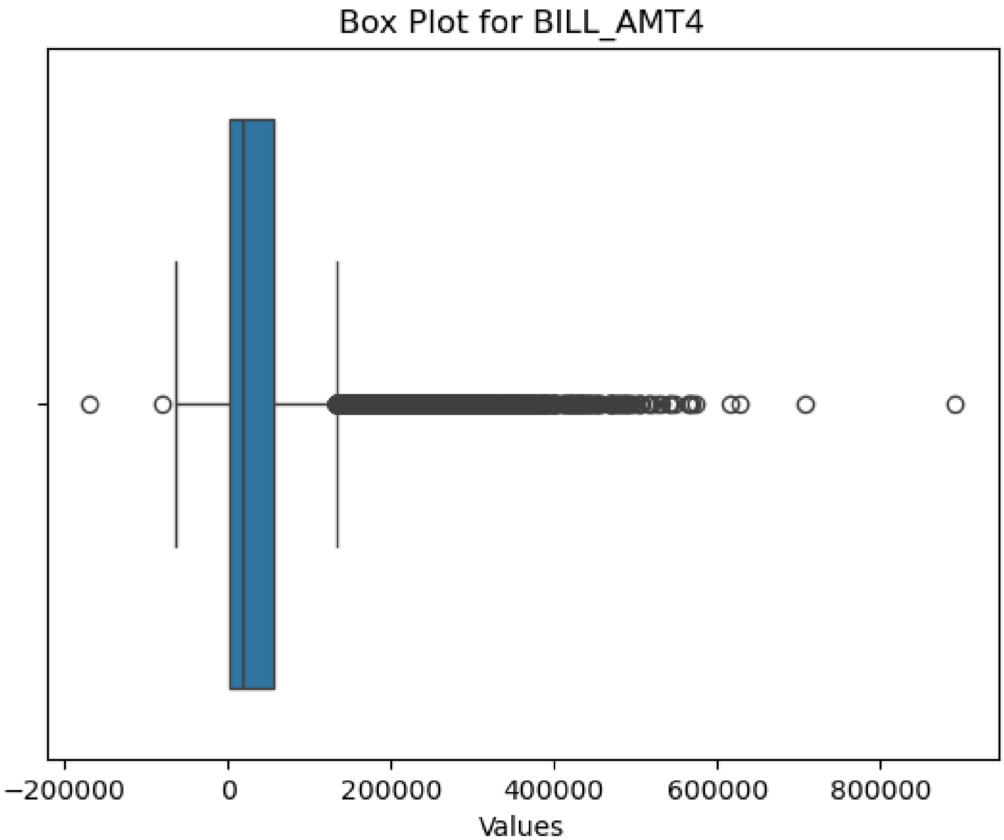

Box Plot for AGE
20
30
40
50
60
70
80
Values

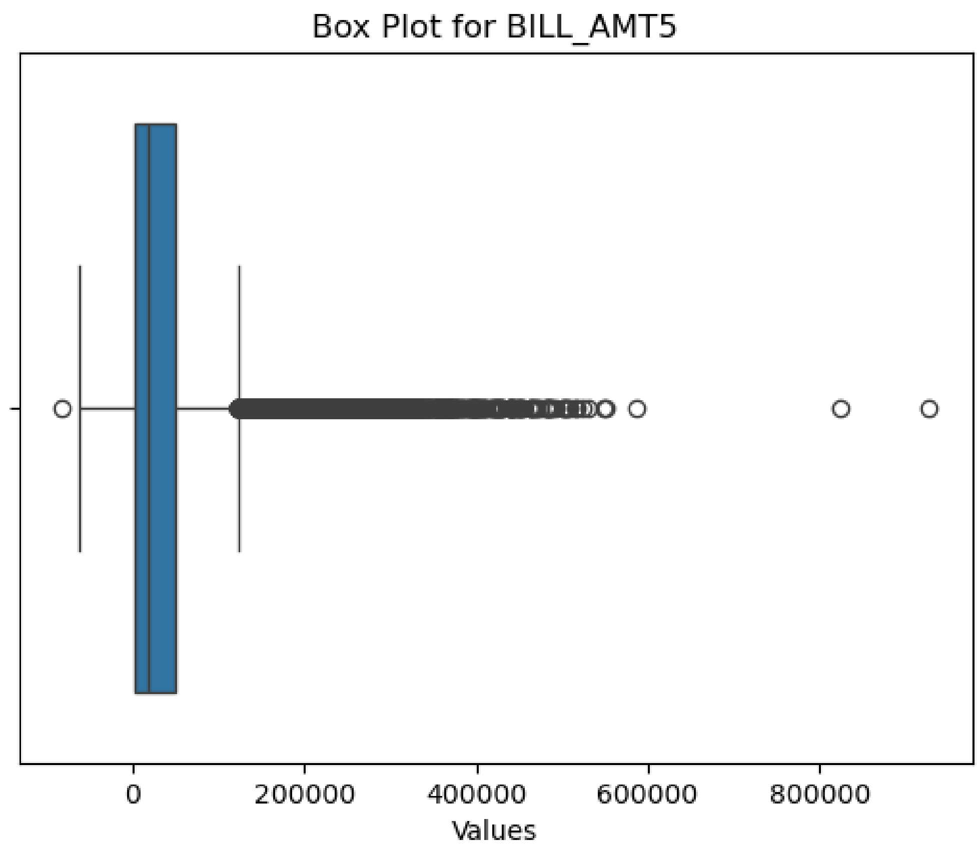

Box Plot for BILL_AMT1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

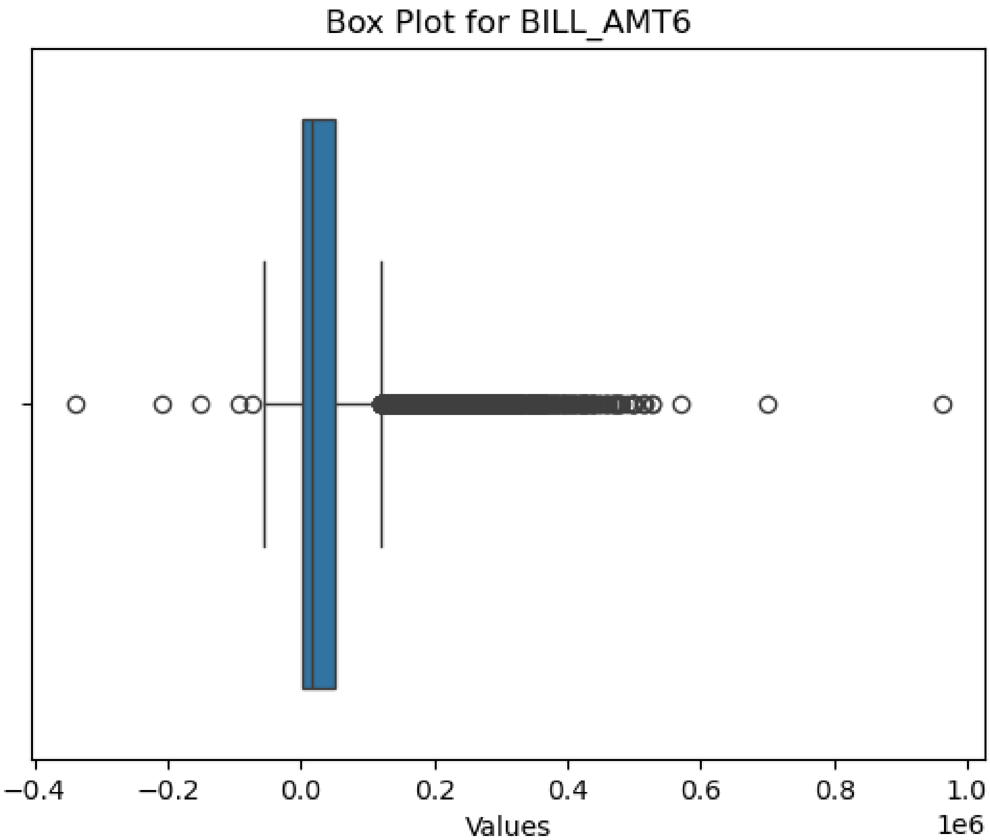


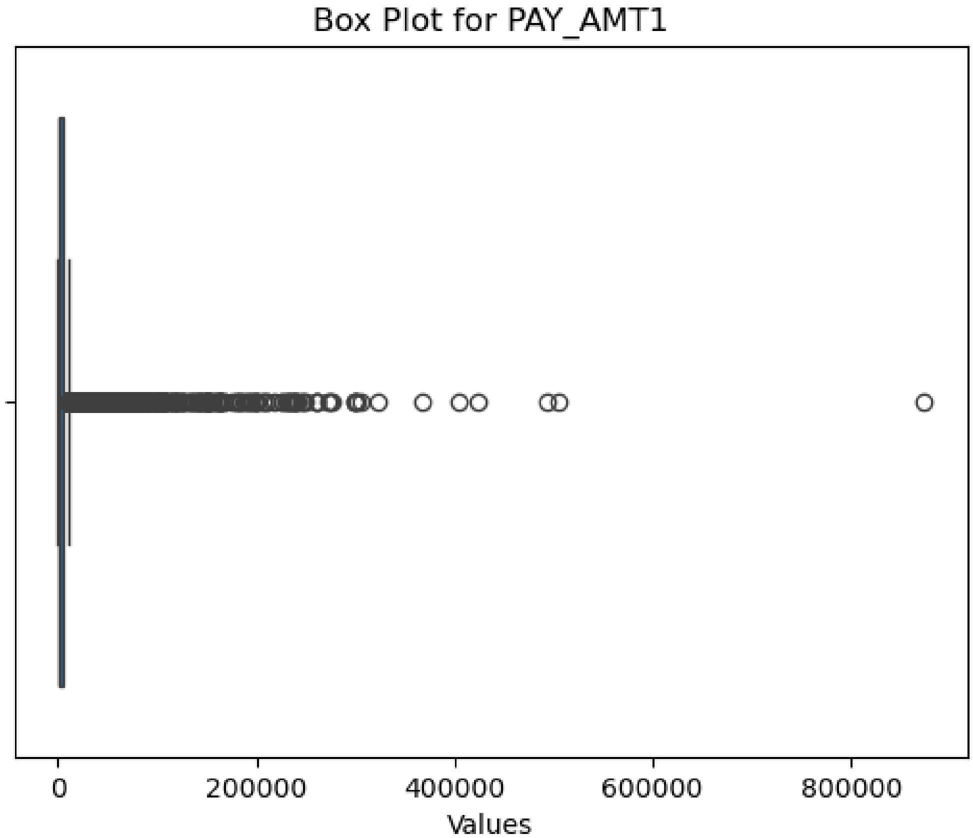
Box Plot for BILL_AMT2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

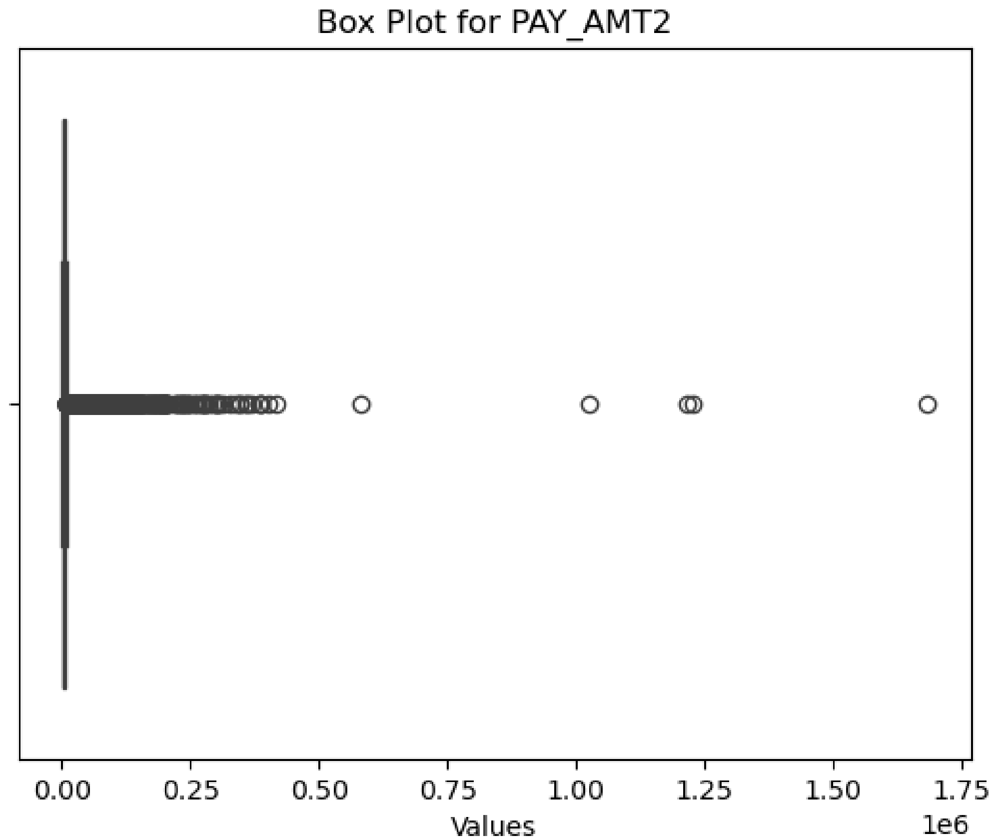

Box Plot for BILL_AMT3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Values
1e6

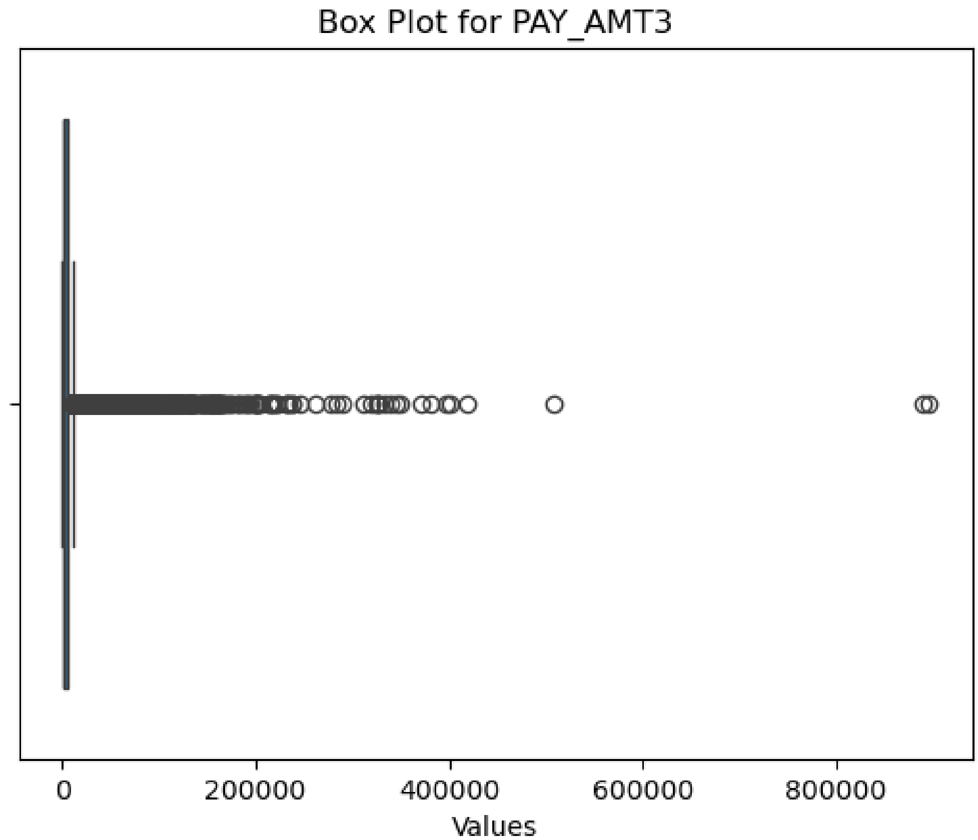

Box Plot for BILL_AMT4
−200000
0
200000
400000
600000
800000
Values

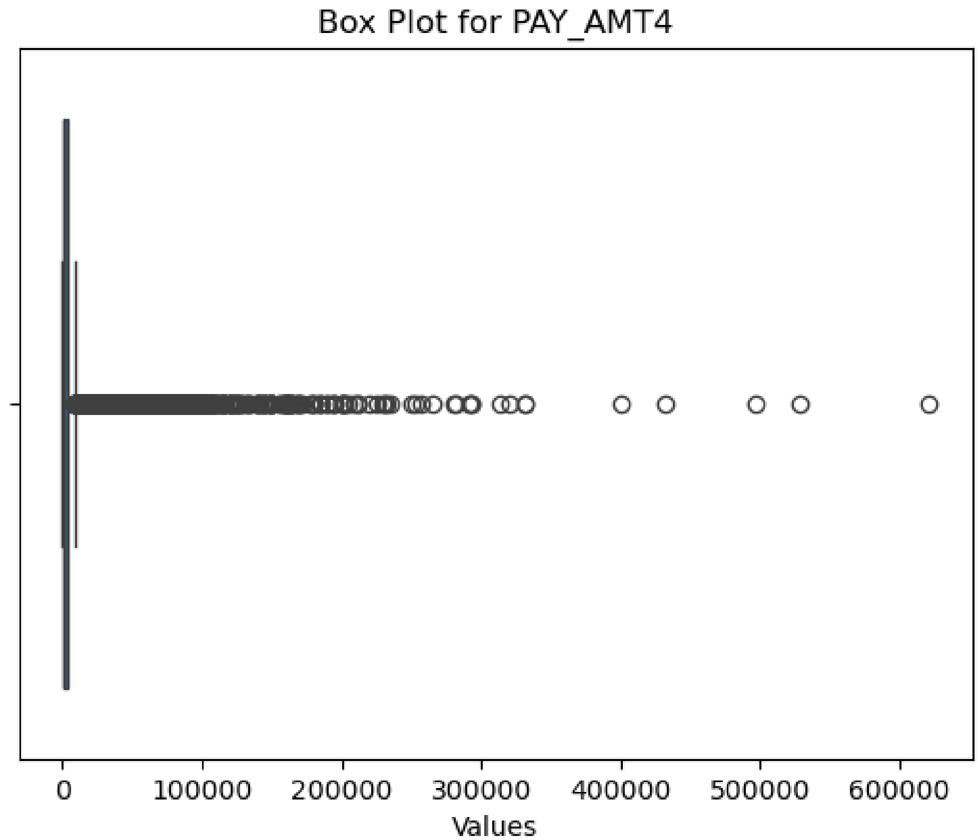

Box Plot for BILL_AMT5
0
200000
400000
600000
800000
Values

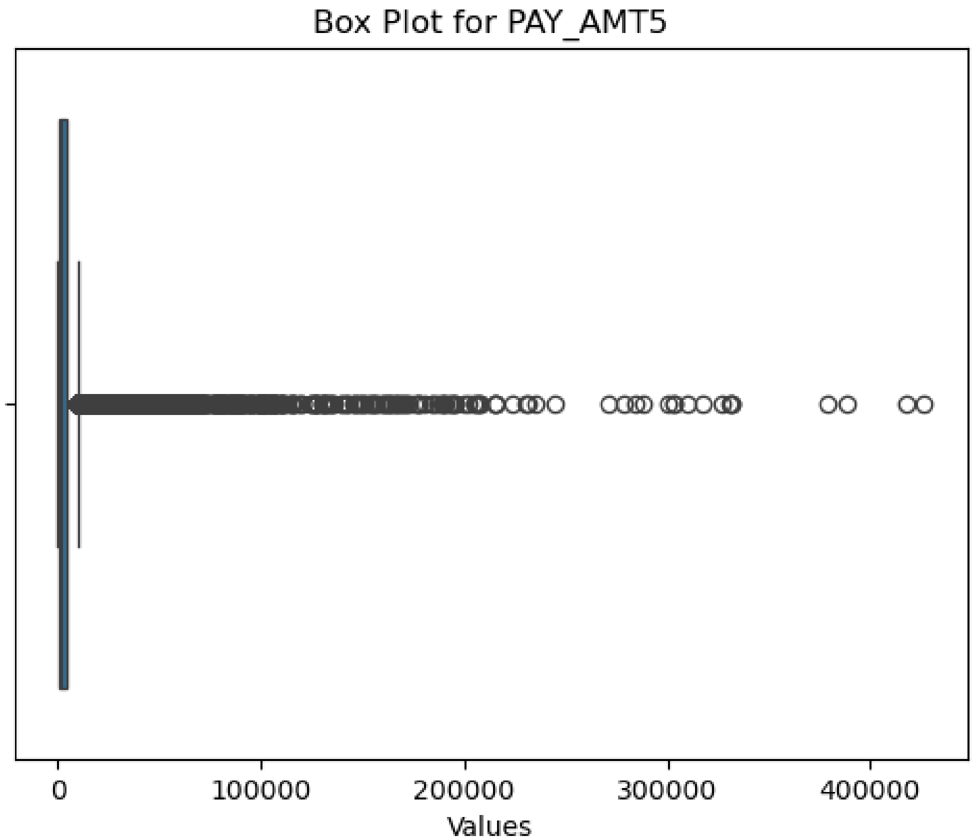

Box Plot for BILL_AMT6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Values

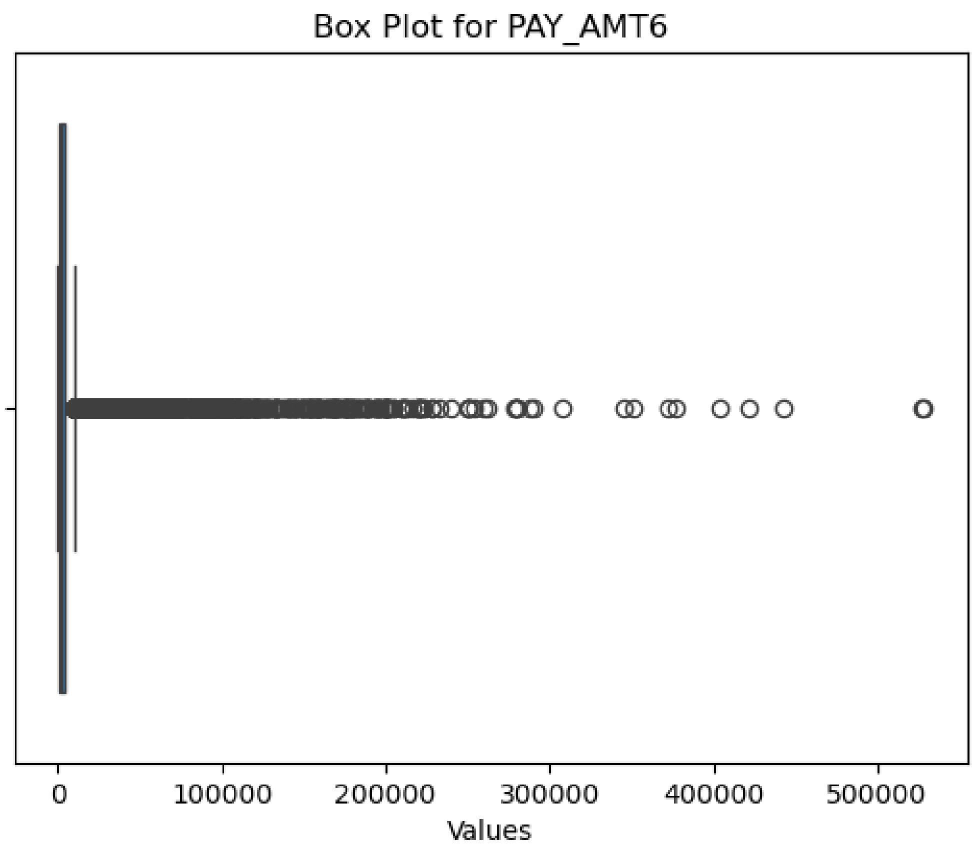

Box Plot for PAY_AMT1
0
200000
400000
600000
800000
Values


Box Plot for PAY_AMT2
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Values
1e6


Box Plot for PAY_AMT3
0
200000
400000
600000
800000
Values


Box Plot for PAY_AMT4
0
100000
200000
300000
400000
500000
600000
Values

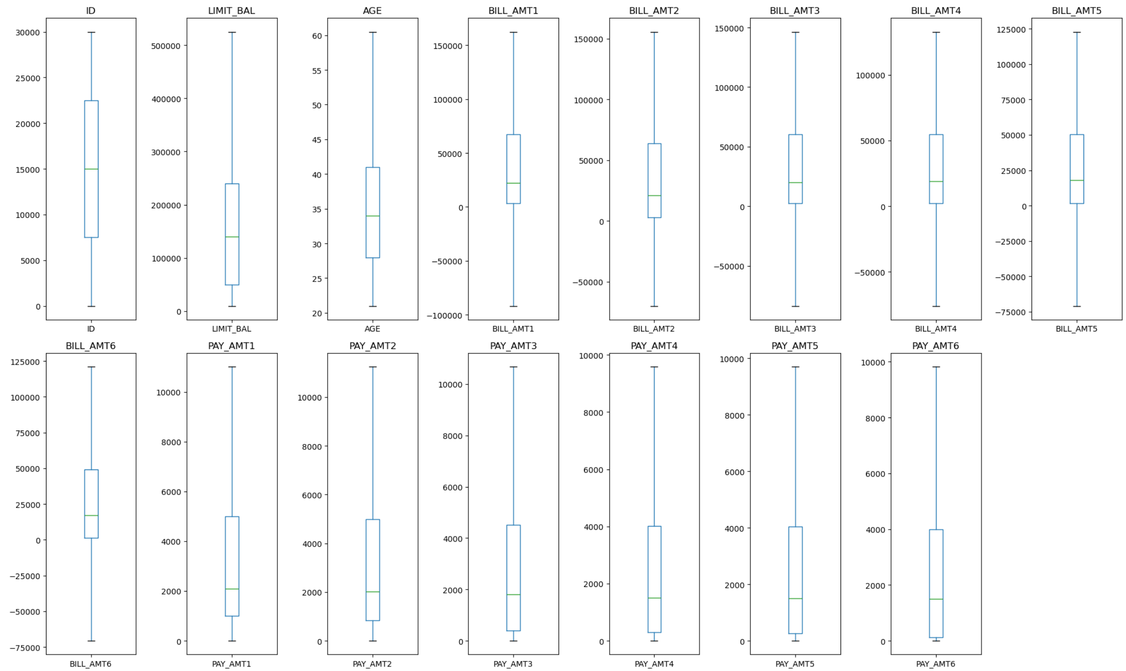

Box Plot for PAY_AMT5
0
100000
200000
300000
400000
Values



[45]:
```
# Fix Outliers using the IQR method
for column in outlier_fix_columns:
    Q1 = df[column].quantile(0.25)  # First quartile (25th percentile)
    Q3 = df[column].quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1                   # Interquartile range
    lower_bound = Q1 - 1.5 * IQR    # Lower whisker
    upper_bound = Q3 + 1.5 * IQR    # Upper whisker

    # Capping outliers
    df[column] = np.where(df[column] < lower_bound, lower_bound, df[column])
    df[column] = np.where(df[column] > upper_bound, upper_bound, df[column])
```

[46]:
```
# Visualising outliers after fixing it using boxplot

plt.figure(figsize=(20, 12))
for i, column in enumerate(outlier_fix_columns):
    plt.subplot(2, len(outlier_fix_columns) // 2 + len(outlier_fix_columns) %
 ↪2, i + 1)
    df.boxplot(column=column, grid=False)
    plt.title(column)
```

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



Features that needed Outlier Fixing has been handled Using IQR method

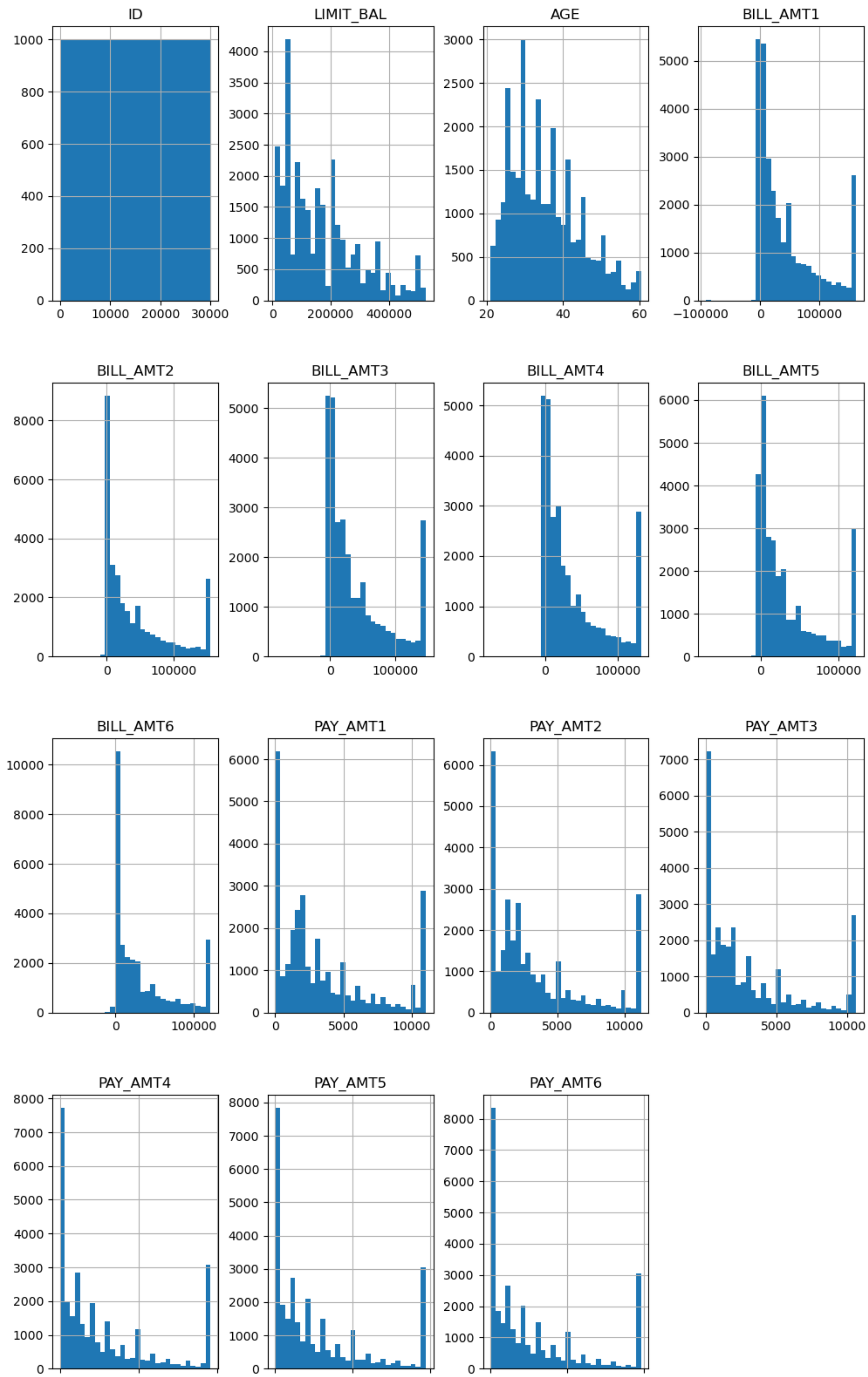
[48]:
```
# Checking skewness of the data
df.skew()
```

[48]:
```
ID             0.000000
LIMIT_BAL      0.904504
SEX           -0.424183
EDUCATION      0.970972
MARRIAGE      -0.018742
AGE            0.654467
PAY_0          0.731975
PAY_2          0.790565
PAY_3          0.840682
PAY_4          0.999629
PAY_5          1.008197
PAY_6          0.948029
BILL_AMT1      1.194178
BILL_AMT2      1.189649
BILL_AMT3      1.184730
BILL_AMT4      1.183997
```

```
BILL_AMT5                     1.184657
BILL_AMT6                     1.199718
PAY_AMT1                      1.032414
PAY_AMT2                      1.113399
PAY_AMT3                      1.200528
PAY_AMT4                      1.176348
PAY_AMT5                      1.183906
PAY_AMT6                      1.211015
default payment next month    1.343504
dtype: float64
```

[49]:

```
# Plot histograms before transformed features

df[outlier_fix_columns].hist(figsize=(12, 20), bins=30)
plt.show()
```

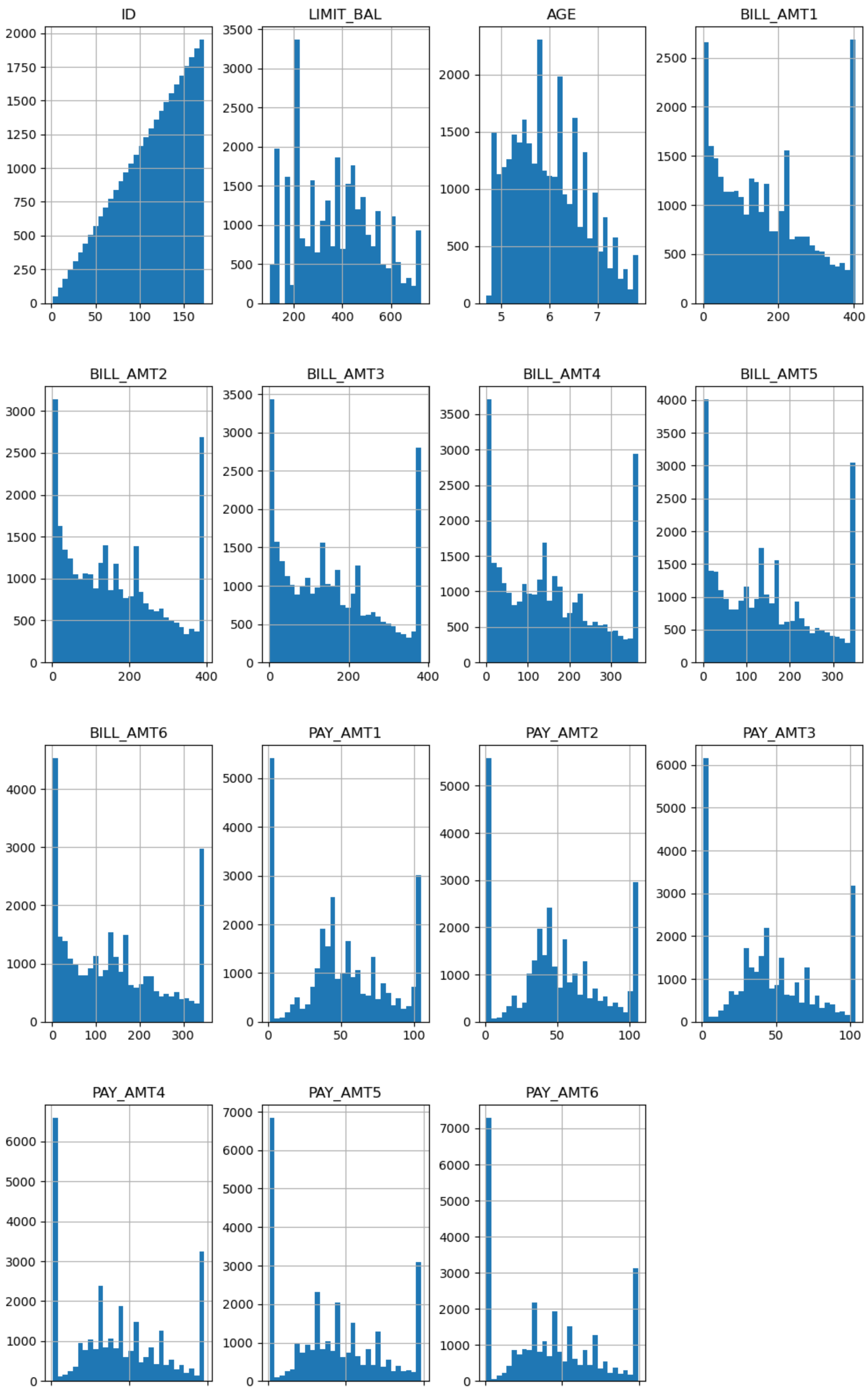

ID
LIMIT_BAL
AGE
BILL_AMT1
BILL_AMT2
BILL_AMT3
BILL_AMT4
BILL_AMT5
BILL_AMT6
PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
PAY_AMT6

[50]:
```python
# Applying square root transformation to fix skewness of needed features

df[outlier_fix_columns] = np.sqrt(np.abs(df[outlier_fix_columns]) + 1)
```

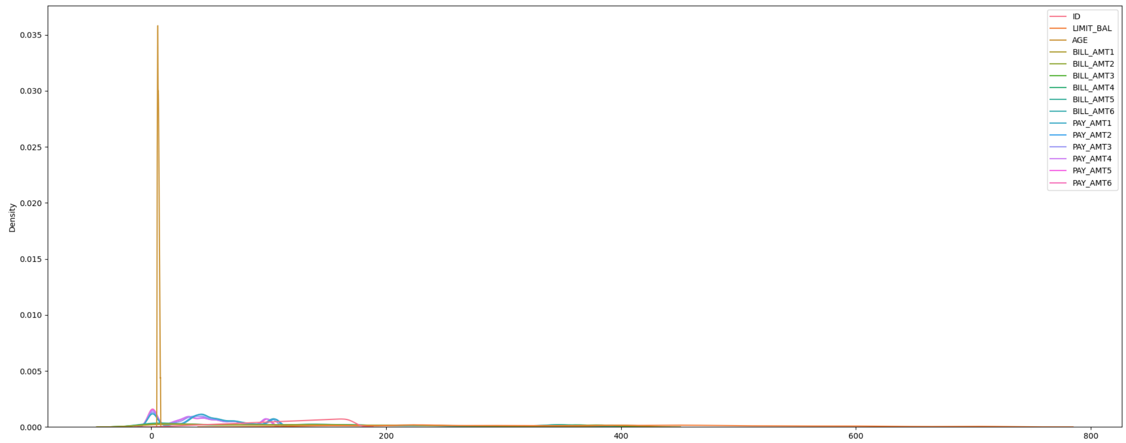
[78]:
```python
print(df[outlier_fix_columns].skew())
```

```
ID          -0.565347
LIMIT_BAL    0.247482
AGE          0.436065
BILL_AMT1    0.437944
BILL_AMT2    0.422110
BILL_AMT3    0.415432
BILL_AMT4    0.418846
BILL_AMT5    0.427006
BILL_AMT6    0.448018
PAY_AMT1     0.095435
PAY_AMT2     0.163805
PAY_AMT3     0.285515
PAY_AMT4     0.313511
PAY_AMT5     0.308738
PAY_AMT6     0.332968
dtype: float64
```

[80]:
```python
# Plot histograms After transformed features

df[outlier_fix_columns].hist(figsize=(12, 20), bins=30)
plt.show()
```

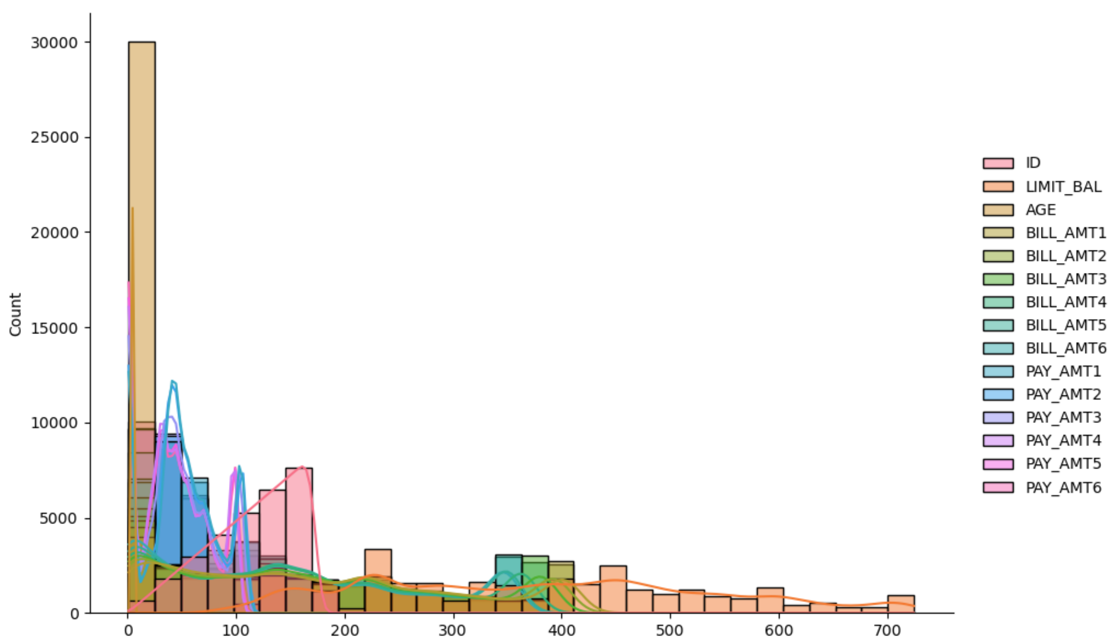

ID
LIMIT_BAL
AGE
BILL_AMT1
BILL_AMT2
BILL_AMT3
BILL_AMT4
BILL_AMT5
BILL_AMT6
PAY_AMT1
PAY_AMT2
PAY_AMT3
PAY_AMT4
PAY_AMT5
PAY_AMT6

[90]:
```
# Drawing KDE plot
plt.figure(figsize=(20,8))
sns.kdeplot(df[outlier_fix_columns])
plt.tight_layout()
plt.show()
```



[98]:
```
sns.displot(df[outlier_fix_columns], bins=30, kde=True, height=6, aspect=1.5)
↪# Create the distribution plot
```

[98]: <seaborn.axisgrid.FacetGrid at 0x1339cf34b30>



The skewness of needed features are handled, Feature[ID] have high skew but it can be removed since it is not an important Feature

1.5 Exploratory Data Analysis (EDA)

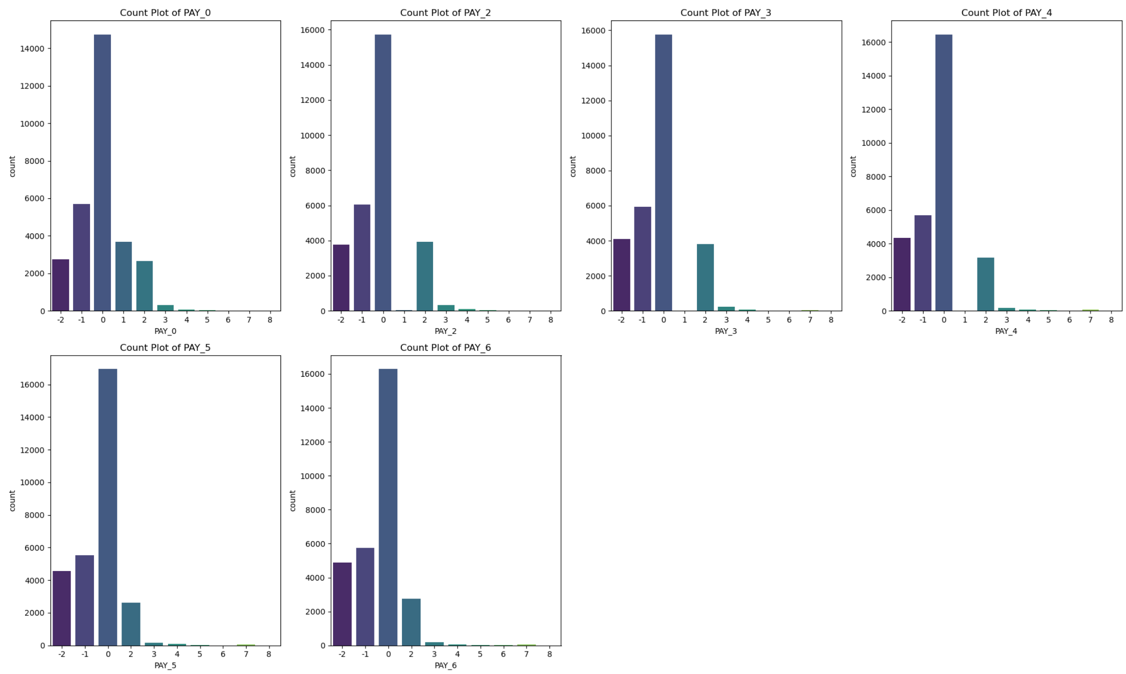
[102]:
```
# Removing Id column from dataset since it is not important
df = df.drop(columns=['ID'])
df.head()
```

[102]:
```
    LIMIT_BAL  SEX  EDUCATION  MARRIAGE       AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0  141.424892    2          2         1  5.000000      2      2     -1     -1
1  346.411605    2          2         2  5.196152     -1      2      0      0
2  300.001667    2          2         2  5.916080      0      0      0      0
3  223.609034    2          2         1  6.164414      0      0      0      0
4  223.609034    1          2         1  7.615773     -1      0     -1      0

   PAY_5  …   BILL_AMT4   BILL_AMT5   BILL_AMT6   PAY_AMT1    PAY_AMT2  \
0     -2  …    1.000000    1.000000    1.000000   1.000000   26.267851
1      0  …   57.210139   58.787754   57.113921   1.000000   31.638584
2      0  …  119.716331  122.266103  124.699639  38.974351   38.742741
3      0  …  168.270615  170.176379  171.895317  44.732538   44.944410
4      0  …  144.710055  138.372685  138.318473  44.732538  106.073088

     PAY_AMT3   PAY_AMT4   PAY_AMT5   PAY_AMT6  default payment next month
0    1.000000   1.000000   1.000000   1.000000                           1
1   31.638584  31.638584   1.000000  44.732538                           1
2   31.638584  31.638584  31.638584  70.717749                           0
3   34.655447  33.181320  32.710854  31.638584                           0
4  100.005000  94.873600  26.267851  26.076810                           0

[5 rows x 24 columns]
```

[112]:
```
continues_num_col =  [
    'LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3',
    'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2',
    'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6'
]
```

[110]:
```
# Count Plot for status of  repayment

# List of columns to plot
pay_columns = ['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']



# Set the figure size
```

```
plt.figure(figsize=(20, 12))

# Loop through each PAY column and create a count plot
for i, column in enumerate(pay_columns):
    plt.subplot(2, 4, i + 1)  # Create a grid of subplots
    sns.countplot(x=df[column], palette="viridis")
    plt.title(f"Count Plot of {column}")

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

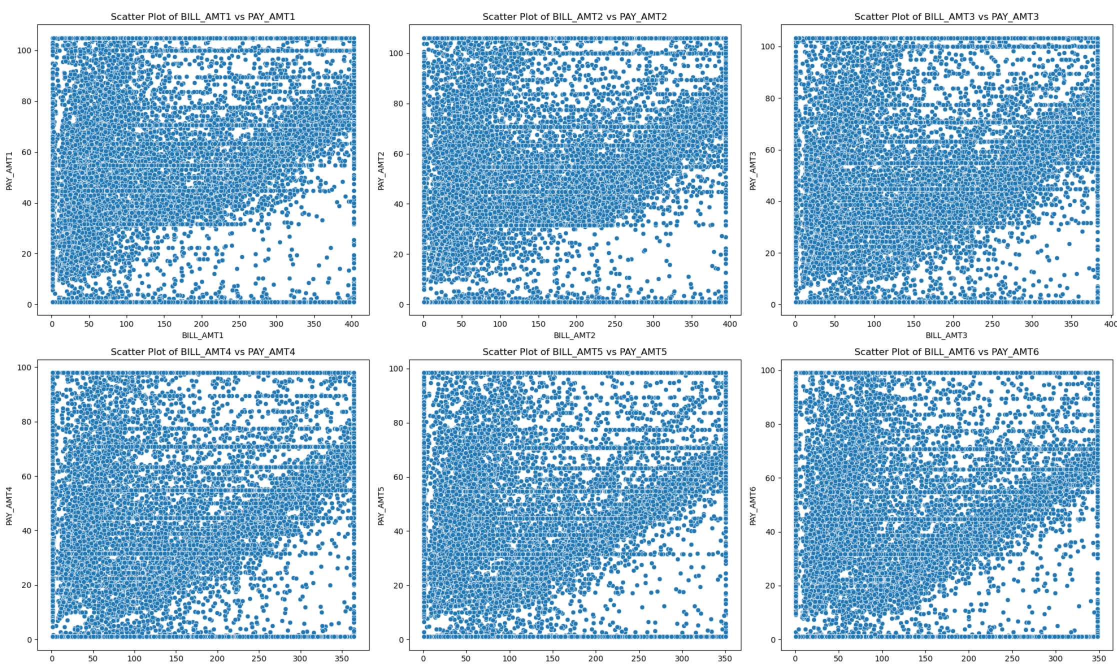


[150]:
```
'''
In Above Figure
-1 = Fully paid
1- 9 means delayed for 1-9 respectevely
-2 = no payment
0 = no due
'''
```

[150]: '\nIn Above Figure \n-1 = Fully paid\n1- 9 means delayed for 1-9 respectevely \n-2 = no payment \n0 = no due\n'

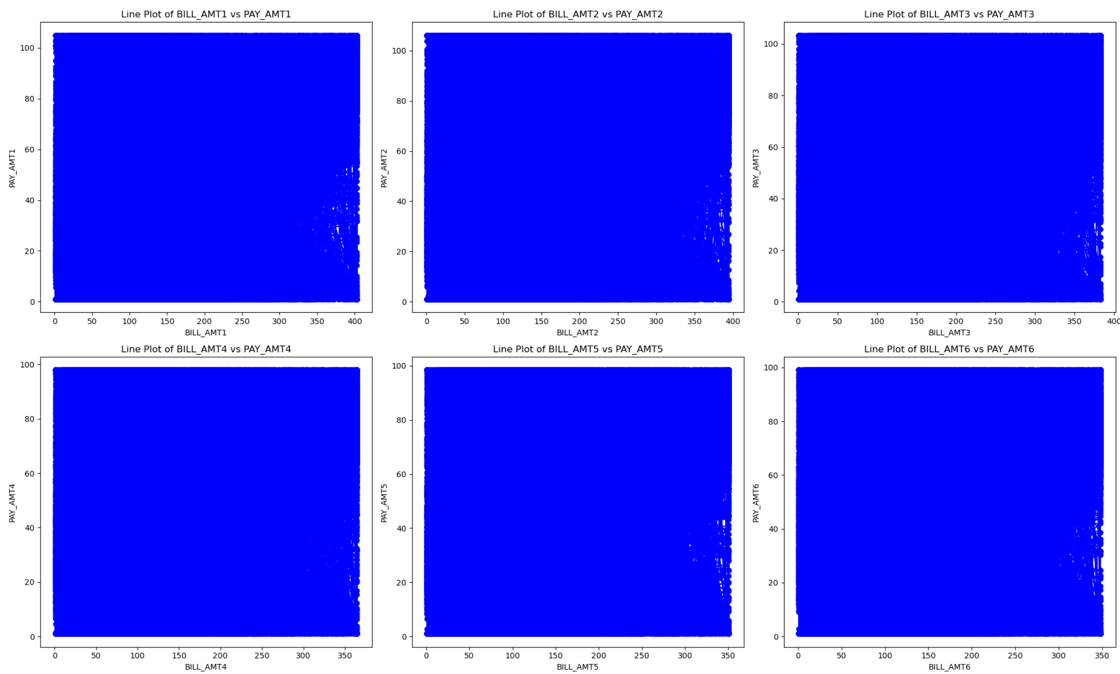
[114]:
```
# List of BILL and PAY columns to plot
bill_columns = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',␣
 ↪'BILL_AMT5', 'BILL_AMT6']
```

```
pay_columns = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',␣
↪'PAY_AMT6']

# Set the figure size
plt.figure(figsize=(20, 12))

# Loop through each pair of BILL_AMT and PAY_AMT columns and create scatter␣
↪plots
for i in range(len(bill_columns)):
    plt.subplot(2, 3, i + 1)  # Create a grid of subplots (2 rows and 3 columns)
    sns.scatterplot(x=df[bill_columns[i]], y=df[pay_columns[i]],␣
↪palette="viridis")
    plt.title(f"Scatter Plot of {bill_columns[i]} vs {pay_columns[i]}")
    plt.xlabel(bill_columns[i])
    plt.ylabel(pay_columns[i])

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



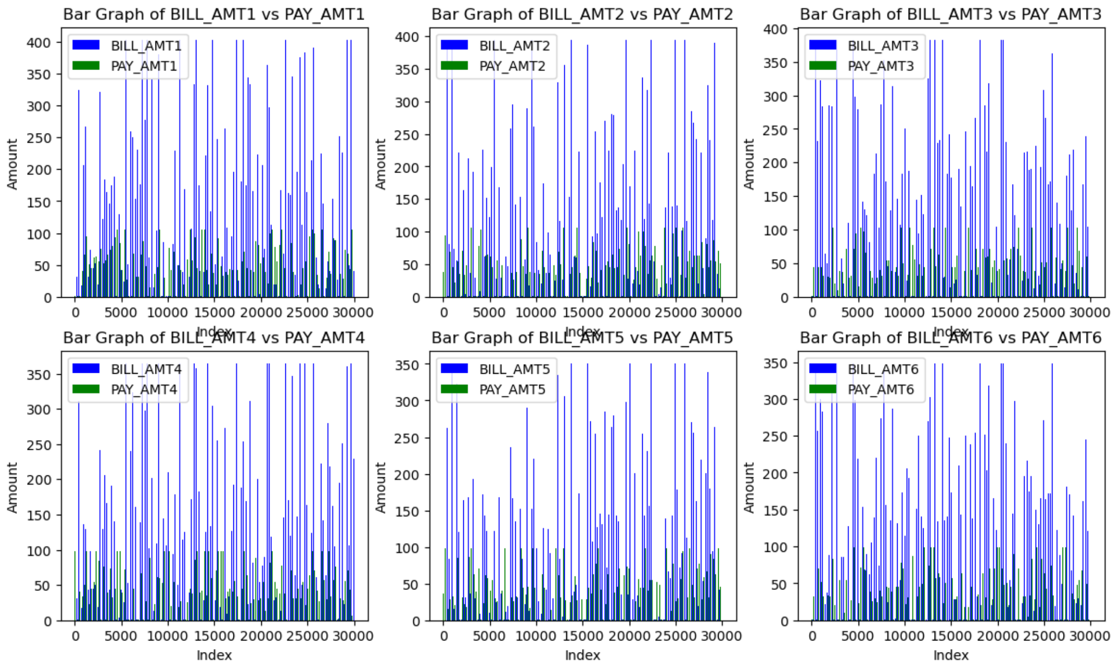
[118]:
```
# Set the figure size
plt.figure(figsize=(20, 12))

# Loop through each pair of BILL_AMT and PAY_AMT columns and create line plots
for i in range(len(bill_columns)):
    plt.subplot(2, 3, i + 1)  # Create a grid of subplots (2 rows and 3 columns)
```

```
    plt.plot(df[bill_columns[i]], df[pay_columns[i]], marker='o',␣
↪linestyle='-', color='b')
    plt.title(f"Line Plot of {bill_columns[i]} vs {pay_columns[i]}")
    plt.xlabel(bill_columns[i])
    plt.ylabel(pay_columns[i])

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

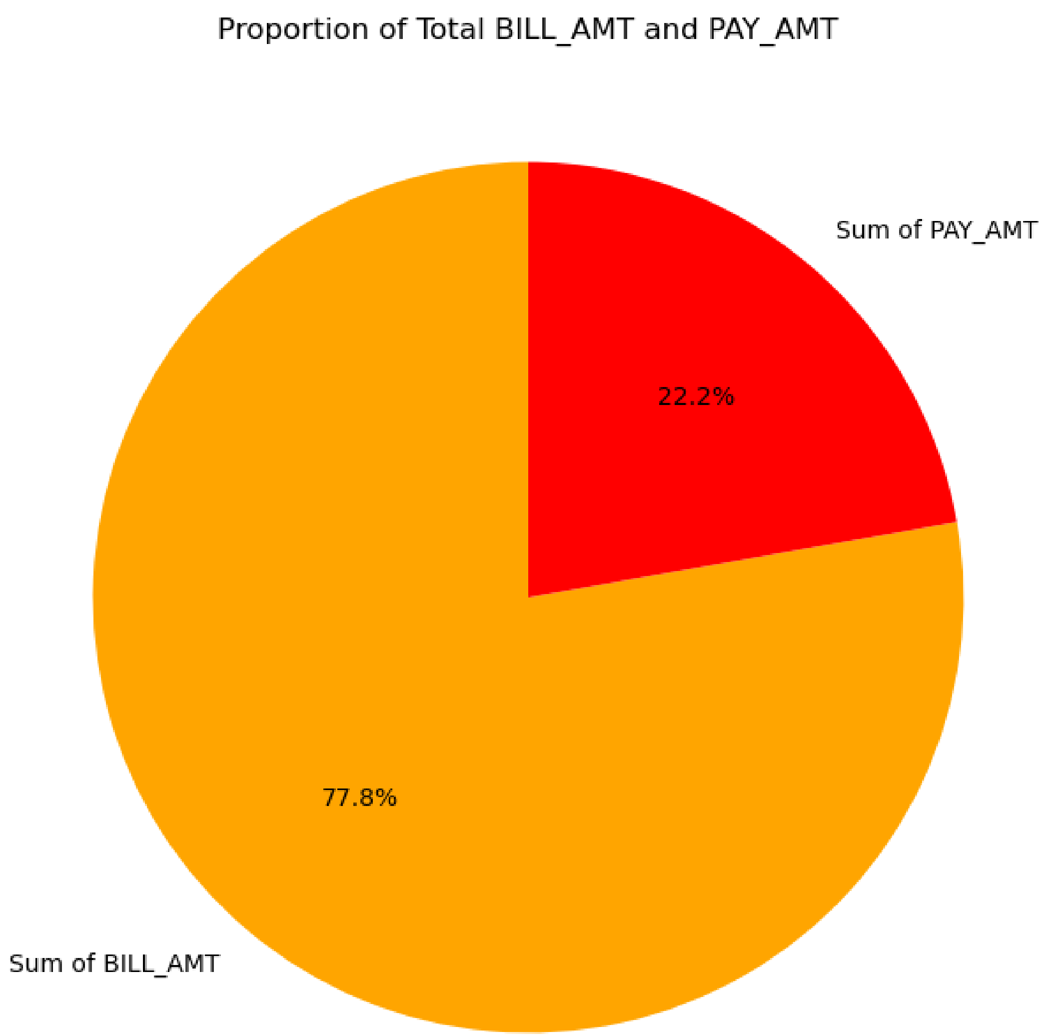


[120]:
```
# Set figure size
plt.figure(figsize=(14, 8))

# Loop through each BILL_AMT and PAY_AMT pair and create bar graphs
for i in range(len(bill_columns)):
    # Create a bar graph for each pair of columns
    plt.subplot(2, 3, i + 1)
    index = np.arange(len(df))
    plt.bar(index - 0.2, df[bill_columns[i]], width=0.4,␣
↪label=f'{bill_columns[i]}', color='blue')
    plt.bar(index + 0.2, df[pay_columns[i]], width=0.4,␣
↪label=f'{pay_columns[i]}', color='green')

    # Set titles and labels
    plt.title(f"Bar Graph of {bill_columns[i]} vs {pay_columns[i]}")
    plt.xlabel('Index')
```

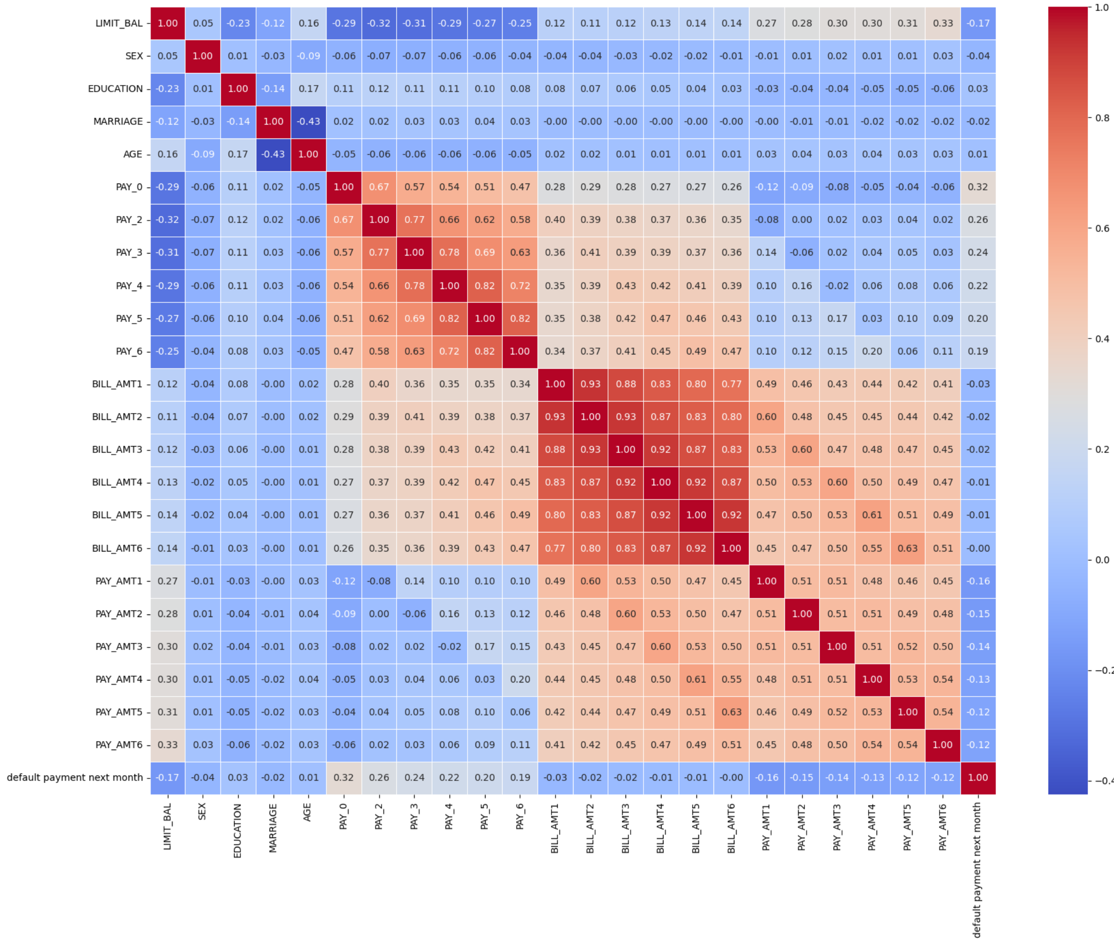
```
    plt.ylabel('Amount')
    plt.legend()
```



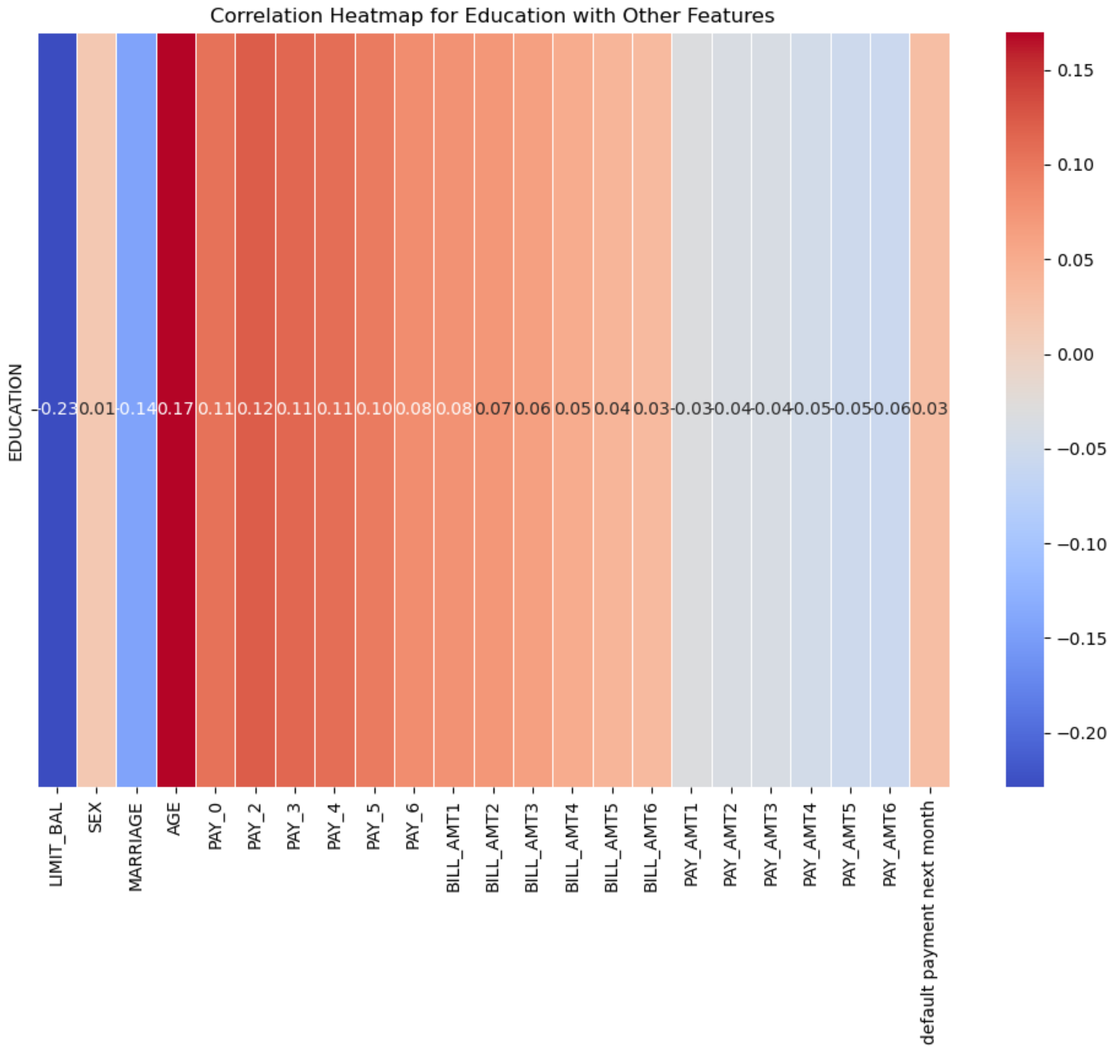
[124]:
```
# Calculate the sum of all BILL_AMT and PAY_AMT columns
bill_amt_sum = df[bill_columns].sum().sum()  # Sum of all BILL_AMT columns
pay_amt_sum = df[pay_columns].sum().sum()  # Sum of all PAY_AMT columns

# Create a pie chart to show the proportion of each sum
labels = ['Sum of BILL_AMT', 'Sum of PAY_AMT']
sizes = [bill_amt_sum, pay_amt_sum]
colors = ['Orange', 'red']

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)
plt.title("Proportion of Total BILL_AMT and PAY_AMT")
plt.show()
```



[132]:
```
# Heatmap to show correlation

plt.figure(figsize=(20, 15))  # Adjust the figure size as needed
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Display the heatmap
plt.show()
```



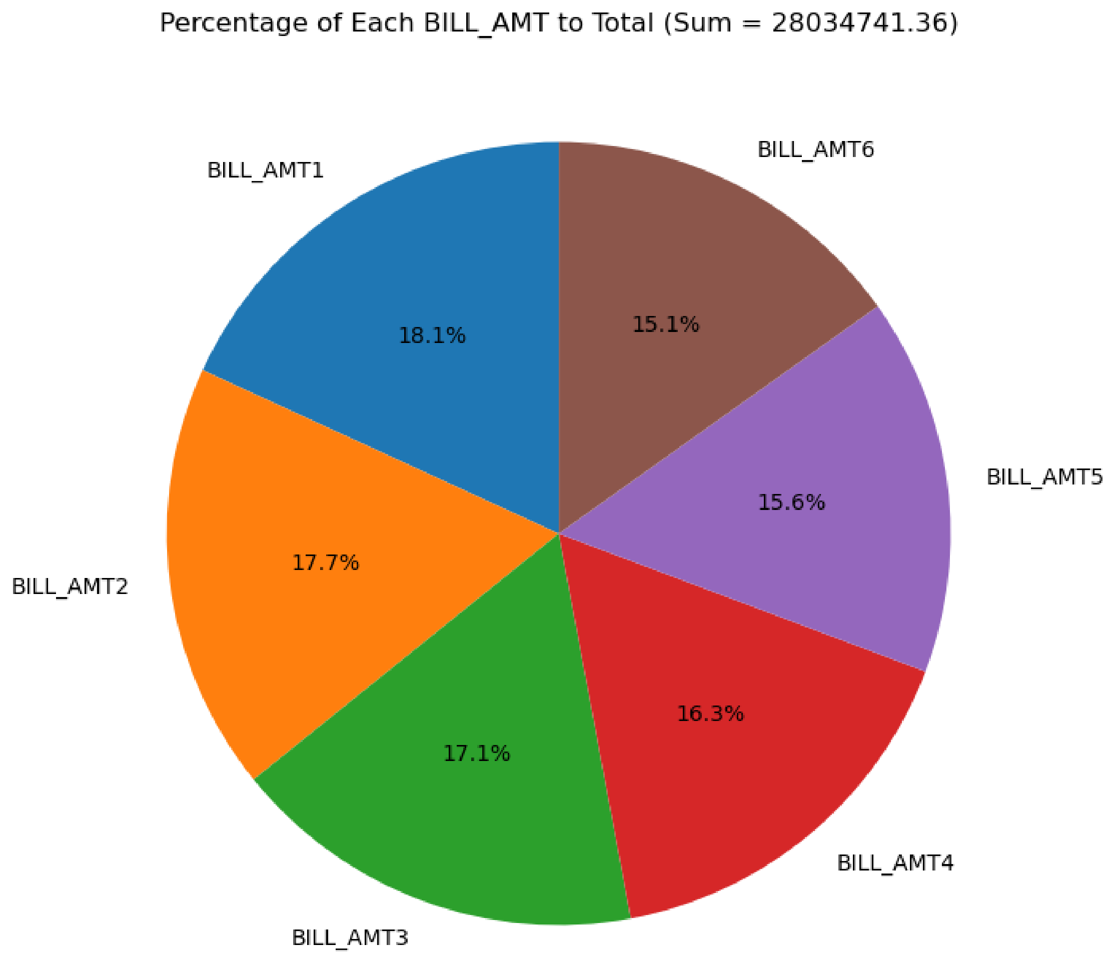
[144]:
```python
# Calculate correlation of 'education' column with all other numerical columns
correlation_education = df.corr()['EDUCATION'].drop('EDUCATION')  # Drop␣
↪'education' to avoid self-correlation

# Convert the correlation series to a DataFrame for heatmap plotting
correlation_education = correlation_education.to_frame()

# Plotting the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_education.T, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=0.5)
plt.title("Correlation Heatmap for Education with Other Features")
plt.show()
```



[146]:

```
# List of the BILL_AMT columns
bill_columns = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',␣
 ↪'BILL_AMT5', 'BILL_AMT6']

# Calculate the sum of each BILL_AMT column
bill_amt_sums = df[bill_columns].sum()

# Calculate the total sum of all BILL_AMT columns
total_bill_amt = bill_amt_sums.sum()

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(bill_amt_sums, labels=bill_amt_sums.index, autopct='%1.1f%%',␣
 ↪startangle=90)
```

```
plt.title(f"Percentage of Each BILL_AMT to Total (Sum = {total_bill_amt:.2f})")
plt.show()
```



[148]:
```
'''
Bill Amt 1 = September
Bill Amt2 = august
Bill Amt 3 = July
Bil Amt 4 = June
Bil Amt 5 = May
Bill Amt 6 = April
'''
```

[148]: '\nBill Amt 1 = September\nBill Amt2 = august\nBill Amt 3 = July\nBil Amt 4 = June\nBil Amt 5 = May\nBill Amt 6 = April\n'

1.5.1 Feature Selection

[167]:
```
#Assigning data as X and Y

X = df.drop(columns=['default payment next month'])
y = df['default payment next month']  # Target variable
```

[169]:
```
# Initialize the RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
rf.fit(X, y)
```

[169]:
```
RandomForestClassifier(random_state=42)
```

[171]:
```
# Get feature importances
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index=X.columns,
                                   columns=["importance"]).
 ↪sort_values("importance", ascending=False)

# Display the most important features
print(feature_importances)
```

```
           importance
PAY_0        0.101703
AGE          0.071053
LIMIT_BAL    0.064371
BILL_AMT1    0.059987
BILL_AMT2    0.052745
PAY_AMT1     0.051139
BILL_AMT3    0.049078
BILL_AMT4    0.048182
PAY_AMT2     0.048103
BILL_AMT6    0.047881
BILL_AMT5    0.047392
PAY_AMT6     0.046052
PAY_AMT3     0.045623
PAY_AMT4     0.043927
PAY_AMT5     0.042942
PAY_2        0.041006
PAY_3        0.024954
PAY_6        0.022091
PAY_4        0.022075
EDUCATION    0.021293
PAY_5        0.020481
MARRIAGE     0.014703
SEX          0.013221
```

[173]:
```python
# Select top 10 features
top_n_features = feature_importances.head(10).index
print(f"Top 10 features: {top_n_features}")
```

```
Top 10 features: Index(['PAY_0', 'AGE', 'LIMIT_BAL', 'BILL_AMT1', 'BILL_AMT2',
'PAY_AMT1',
       'BILL_AMT3', 'BILL_AMT4', 'PAY_AMT2', 'BILL_AMT6'],
      dtype='object')
```

1.5.2 Split Data into Training and Testing Sets:

[178]:
```python
X = df[['PAY_0', 'AGE', 'LIMIT_BAL', 'BILL_AMT1', 'BILL_AMT2', 'PAY_AMT1',
        'BILL_AMT3', 'BILL_AMT4', 'PAY_AMT2', 'BILL_AMT6']]

# Split the data into training and testing sets (80% training, 20% testing)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print(f"Training Features Shape: {X_train.shape}")
print(f"Testing Features Shape: {X_test.shape}")
print(f"Training Target Shape: {y_train.shape}")
print(f"Testing Target Shape: {y_test.shape}")
```

```
Training Features Shape: (24000, 10)
Testing Features Shape: (6000, 10)
Training Target Shape: (24000,)
Testing Target Shape: (6000,)
```

[ ]:
```python

```