```
In [1]:
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
```

```
In [54]:
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

```
In [128…
import sys
```

```
In [2]:
%load_ext pycodestyle_magic
```

```
In [3]:
%pycodestyle_on
```

```
In [61]:
%pycodestyle_off
```

```
In [85]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from pandas_profiling import ProfileReport
from pandas.plotting import scatter_matrix

%matplotlib inline
```

index

1.Load the data

```
In [57]:
input_path = "default of credit card clients.xls"
df = pd.read_excel(input_path, header=1)
df.info()
df.head()
df.shape
```

Out[57]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_A
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	

5 rows × 25 columns

Out[57]: (30000, 25)

2.Review the data

missing value & the basic description

```
In [22]:
df.isna().sum()
```

```
Out[22]:
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
```

There is no missing value in all the columns , so we don't need to deal with the missing value.

```
In [17]:
df.describe().T
```

Out[17]:

	count	mean	std	min	25%	50%	75%	max
ID	30000.0	15000.500000	8660.398374	1.0	7500.75	15000.5	22500.25	30000.0
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	1000000.0
SEX	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	2.0
EDUCATION	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	6.0
MARRIAGE	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	3.0
AGE	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	79.0
PAY_0	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	8.0
PAY_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	8.0
PAY_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	8.0
PAY_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	8.0
PAY_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	8.0
PAY_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	8.0
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	964511.0
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	983931.0
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	1664089.0
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	891586.0
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	927171.0
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	961664.0
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	873552.0
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	1684259.0
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	896040.0
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	621000.0
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	426529.0
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	528666.0
default payment next month	30000.0	0.221200	0.415062	0.0	0.00	0.0	0.00	1.0

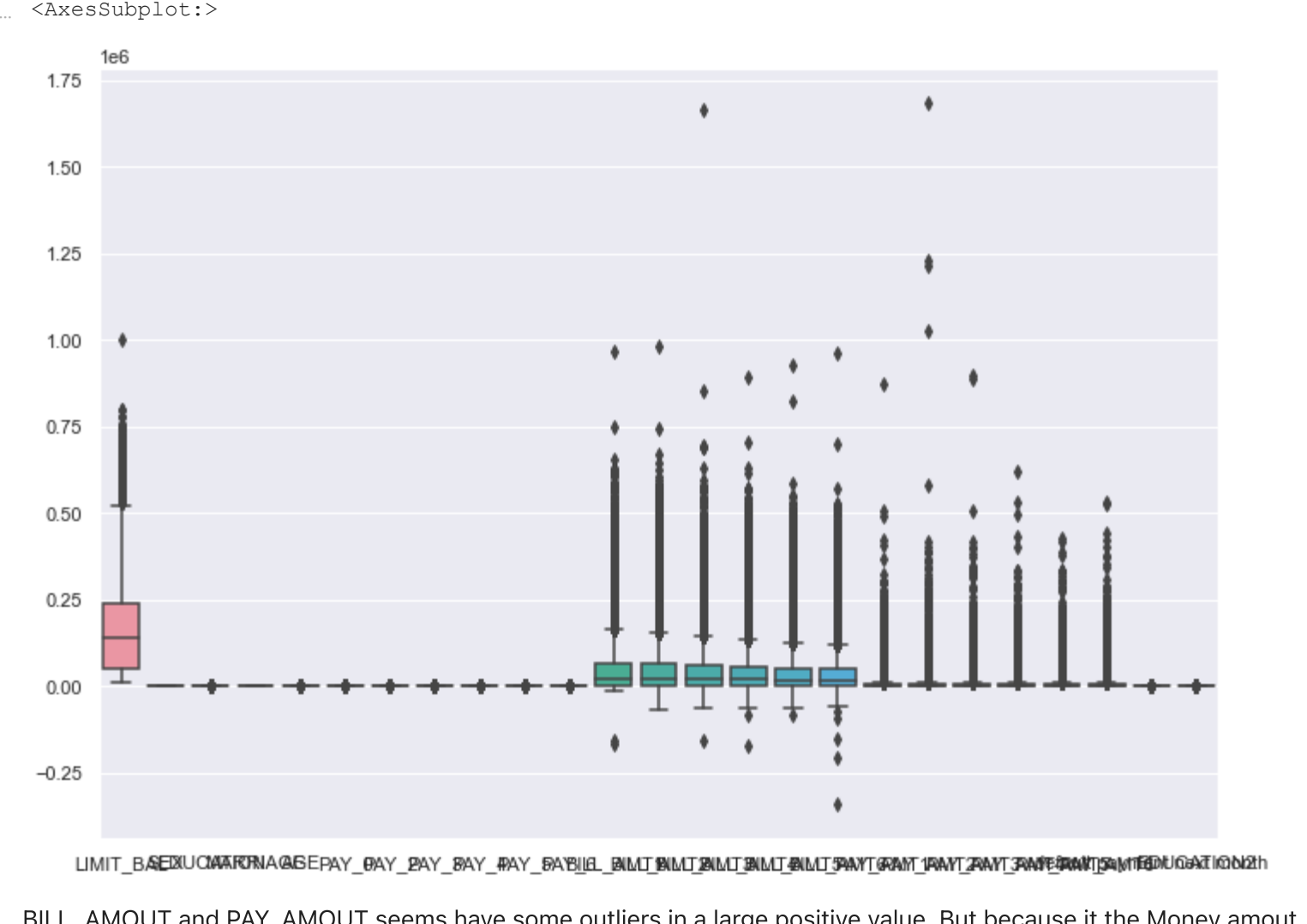
```
In [18]:
df.dtypes
```

```
Out[18]:
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
```

check the outliers

```
In [183…
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.boxplot(data=df.drop(columns=['ID']))
```

Out[183… <AxesSubplot:>



BILL_AMOUT and PAY_AMOUT seems have some outliers in a large positive value. But because it the Money amout, it is likely to be a long tail distribution. So I think it make sense, I don't deal with these "outliers".

data-profile

```
In [21]:
from pandas_profiling import ProfileReport
profile = ProfileReport(df, title="Report_ModelingTest")
profile.to_file("Report_ModelingTest.html")
```

from the html files, we have some feeling about the data.

1. The age variable has several high histograms.
2. The BILL_AMT1~6 contains some negative values
3. The default rate is 22% percent.
4. The pay_1 ~ pay_6 are highly correlated with bill_amt1 ~ bill_amt6
5. The bill_amt1 ~ bill_amt6 are highly correlated with pay_amt1 ~ pay_amt6

Next, we will check the above infomation and get a deeper understanding of the data.

Analysis the features

1.Age

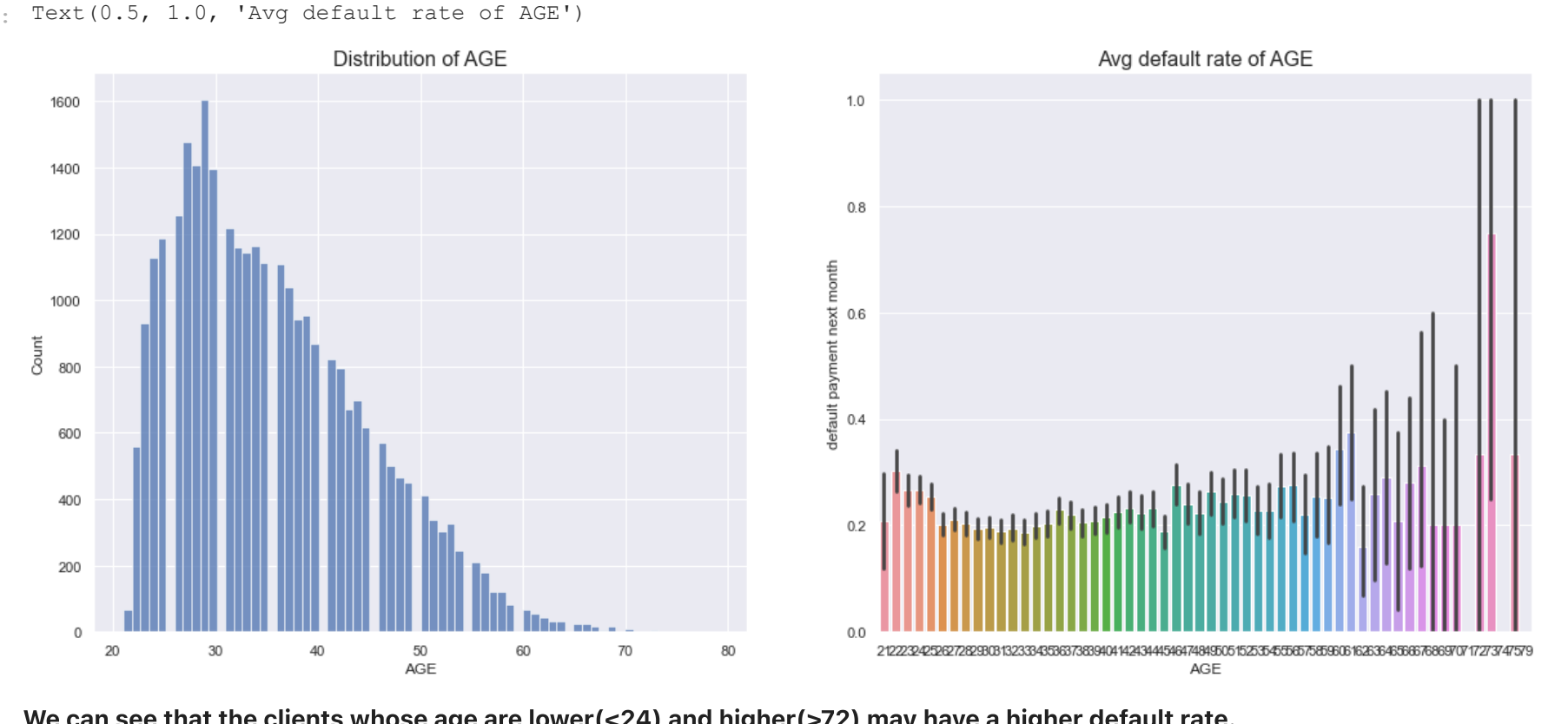
```
In [63]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
col = 'AGE'
sns.histplot(data=df, x=col,ax=ax[0])
ax[0].set_title(f'Distribution of {col}', fontsize=16)
sns.barplot(data=df,x=col,y='default payment next month',ax=ax[1])
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[63]: <AxesSubplot:xlabel='AGE', ylabel='Count'>

Out[63]: Text(0.5, 1.0, 'Distribution of AGE')

Out[63]: <AxesSubplot:xlabel='AGE', ylabel='default payment next month'>

Out[63]: Text(0.5, 1.0, 'Avg default rate of AGE')



We can see that the clients whose age are lower(<24) and higher(>72) may have a higher default rate.

2. SEX

```
In [64]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
data = df
col = 'SEX'
target = 'default payment next month'
count = data.groupby(col)[target].count()
pct = count/data.shape[0]
rate = data.groupby(col)[target].mean()

tmp = pd.concat([count,pct,rate],axis=1)
tmp.columns = ['count','pct','avg_default_rate']
tmp=tmp.reset_index().sort_values(by='avg_default_rate')
tmp

sns.barplot(data=tmp,x=col,y='pct',ax=ax[0])
sns.barplot(data=tmp,x=col,y='avg_default_rate',ax=ax[1])

ax[0].set_title(f'Distribution of {col}', fontsize=16)
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[64]:

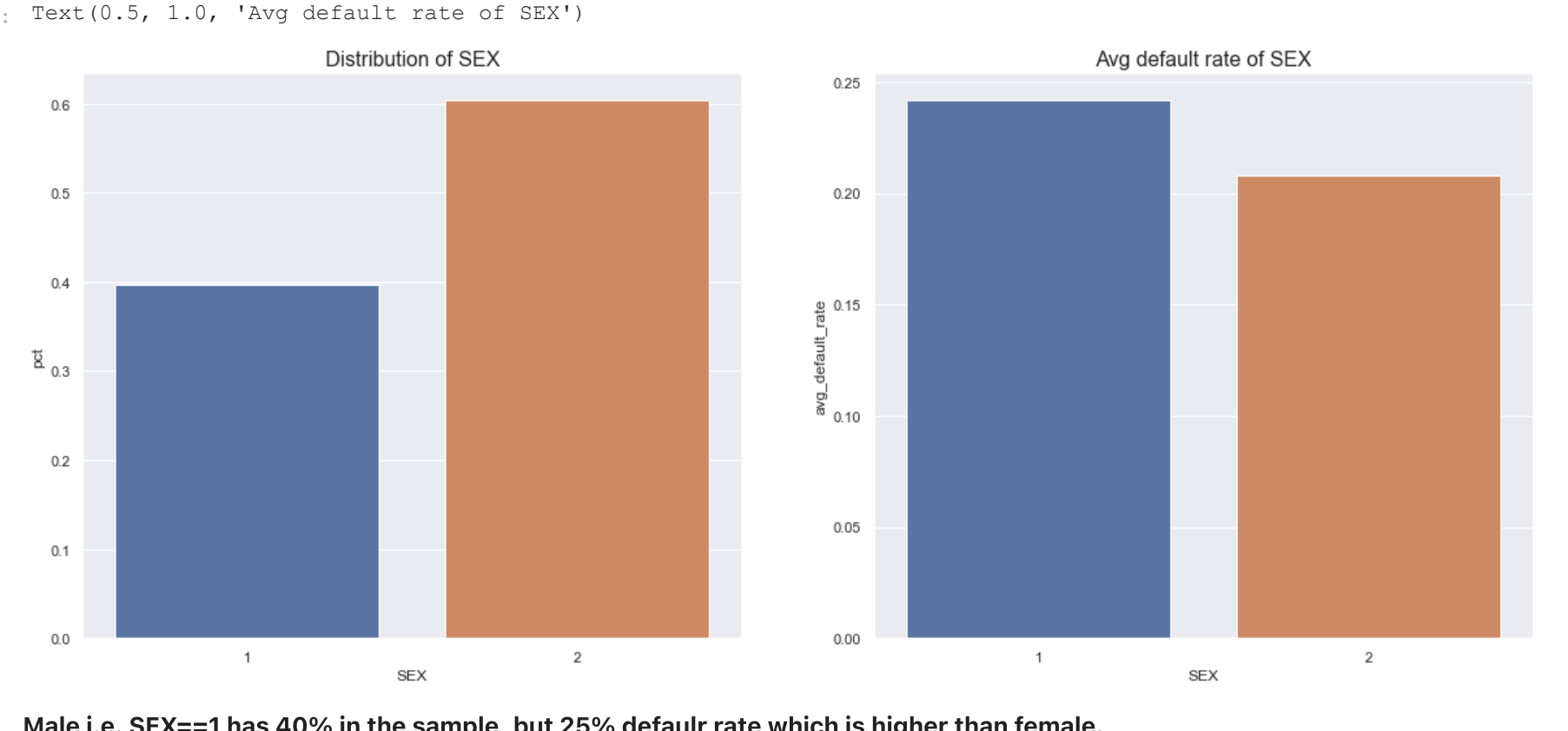
	SEX	count	pct	avg_default_rate
1	2	18112	0.603733	0.207763
0	1	11888	0.396267	0.241672

Out[64]: <AxesSubplot:xlabel='SEX', ylabel='pct'>

Out[64]: <AxesSubplot:xlabel='SEX', ylabel='avg_default_rate'>

Out[64]: Text(0.5, 1.0, 'Distribution of SEX')

Out[64]: Text(0.5, 1.0, 'Avg default rate of SEX')



Male i.e. SEX==1 has 40% in the sample, but 25% defaulr rate which is higher than female.

3. EDUCATION

```
In [73]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
data = df
col = 'EDUCATION'
target = 'default payment next month'

d = {0:'zero',1:'graduate school',2:'university',3:'high school',4:'others',5:'unknown1',6:'unknown2'}

count = data.groupby(col)[target].count()
pct = count/data.shape[0]
rate = data.groupby(col)[target].mean()

tmp = pd.concat([count,pct,rate],axis=1)
tmp.columns = ['count','pct','avg_default_rate']
tmp=tmp.reset_index()#.sort_values(by='avg_default_rate')
tmp[col] = tmp[col].apply(lambda x:d[x])
tmp

sns.barplot(data=tmp,x=col,y='pct',ax=ax[0])
sns.barplot(data=tmp,x=col,y='avg_default_rate',ax=ax[1])

ax[0].set_title(f'Distribution of {col}', fontsize=16)
ax[0].set
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[73]:

	EDUCATION	count	pct	avg_default_rate
0	zero	14	0.000467	0.000000
1	graduate school	10585	0.352833	0.192348
2	university	14030	0.467667	0.237349
3	high school	4917	0.163900	0.251576
4	others	123	0.004100	0.056911
5	unknown1	280	0.009333	0.064286
6	unknown2	51	0.001700	0.156863

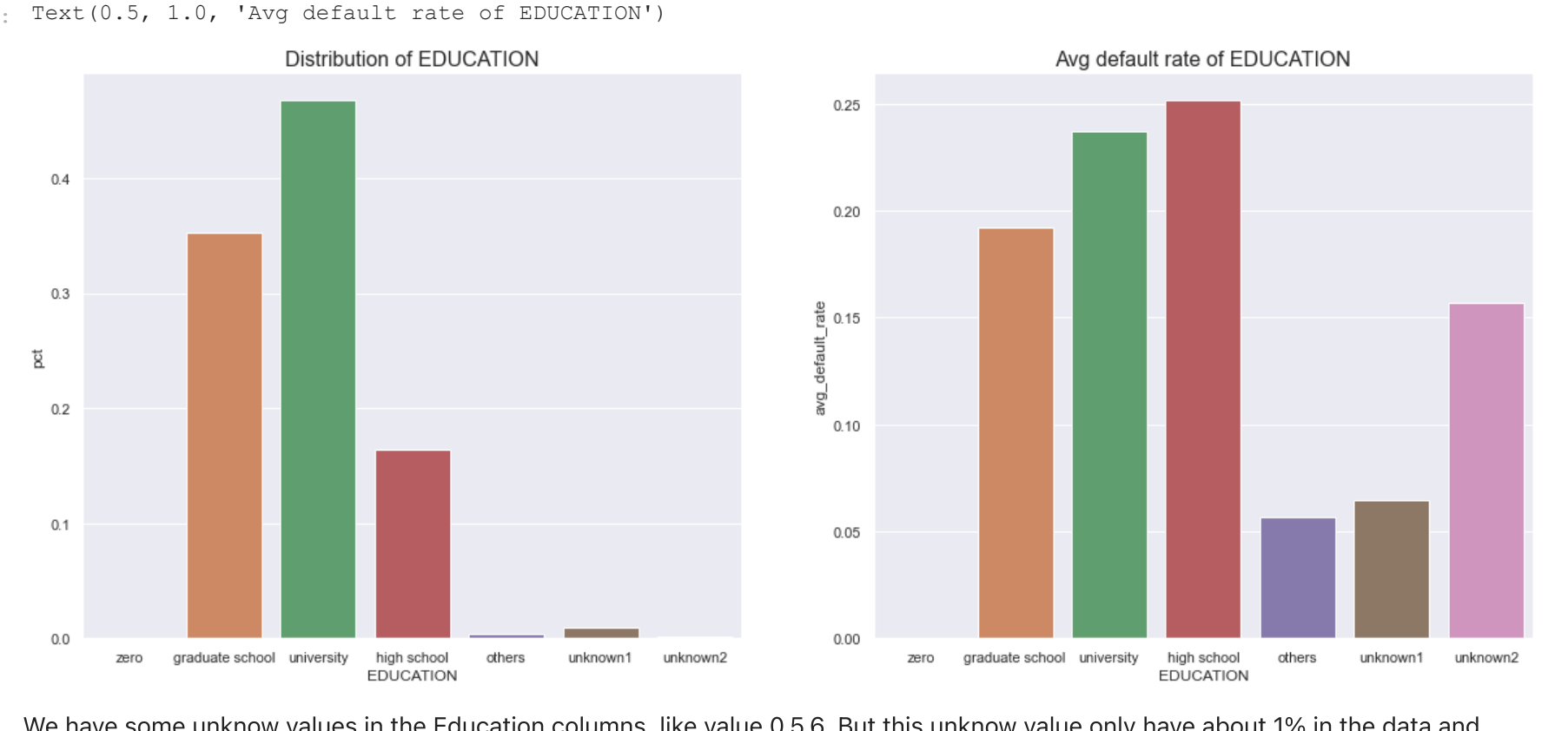
Out[73]: <AxesSubplot:xlabel='EDUCATION', ylabel='pct'>

Out[73]: <AxesSubplot:xlabel='EDUCATION', ylabel='avg_default_rate'>

Out[73]: Text(0.5, 1.0, 'Distribution of EDUCATION')

Out[73]: <bound method Artist.set of <AxesSubplot:title={'center':'Distribution of EDUCATION'}, xlabel='EDUCATION', ylabel='pct'>>

Out[73]: Text(0.5, 1.0, 'Avg default rate of EDUCATION')



We have some unknow values in the Education columns, like value 0,5,6. But this unknow value only have about 1% in the data and the default rate is quiet low. We can just ignore these value. It won't affect the result.

But from the knowing value 1~3, we can find out that the higher the education level, the lower risk.

If we want to get more meaningful variable, we should reorder this columns by using the average of the target like below,but in this case, I don't use the reorder.

```
In [80]:
data = df
col = 'EDUCATION'
target = 'default payment next month'
rate = data.groupby(col)[target].mean()
rate = rate.sort_values()
rate

reorder_edu = { k:v for k,v in zip(rate.index, range(len(rate)))}
reorder_edu
df['EDUCATION2'] = df['EDUCATION'].apply(lambda x: reorder_edu[x])
df[['EDUCATION','EDUCATION2']].sample(5)
```

```
Out[80]:
EDUCATION
0    0.000000
4    0.056911
5    0.064286
6    0.156863
1    0.192348
2    0.237349
3    0.251576
Name: default payment next month, dtype: float64
```

Out[80]: {0: 0, 4: 1, 5: 2, 6: 3, 1: 4, 2: 5, 3: 6}

Out[80]:

	EDUCATION	EDUCATION2
23719	3	6
25566	2	5
231	2	5
22459	2	5
24650	2	5

4. MARRIAGE

```
In [83]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
data = df
col = 'MARRIAGE'
target = 'default payment next month'

d = {0:'zero',1:'married',2:'single',3:'others'}

count = data.groupby(col)[target].count()
pct = count/data.shape[0]
rate = data.groupby(col)[target].mean()

tmp = pd.concat([count,pct,rate],axis=1)
tmp.columns = ['count','pct','avg_default_rate']
tmp=tmp.reset_index()#.sort_values(by='avg_default_rate')
tmp[col] = tmp[col].apply(lambda x:d[x])
tmp

sns.barplot(data=tmp,x=col,y='pct',ax=ax[0])
sns.barplot(data=tmp,x=col,y='avg_default_rate',ax=ax[1])

ax[0].set_title(f'Distribution of {col}', fontsize=16)
ax[0].set
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[83]:

	MARRIAGE	count	pct	avg_default_rate
0	zero	54	0.001800	0.092593
1	married	13659	0.455300	0.234717
2	single	15964	0.532133	0.209283
3	others	323	0.010767	0.260062

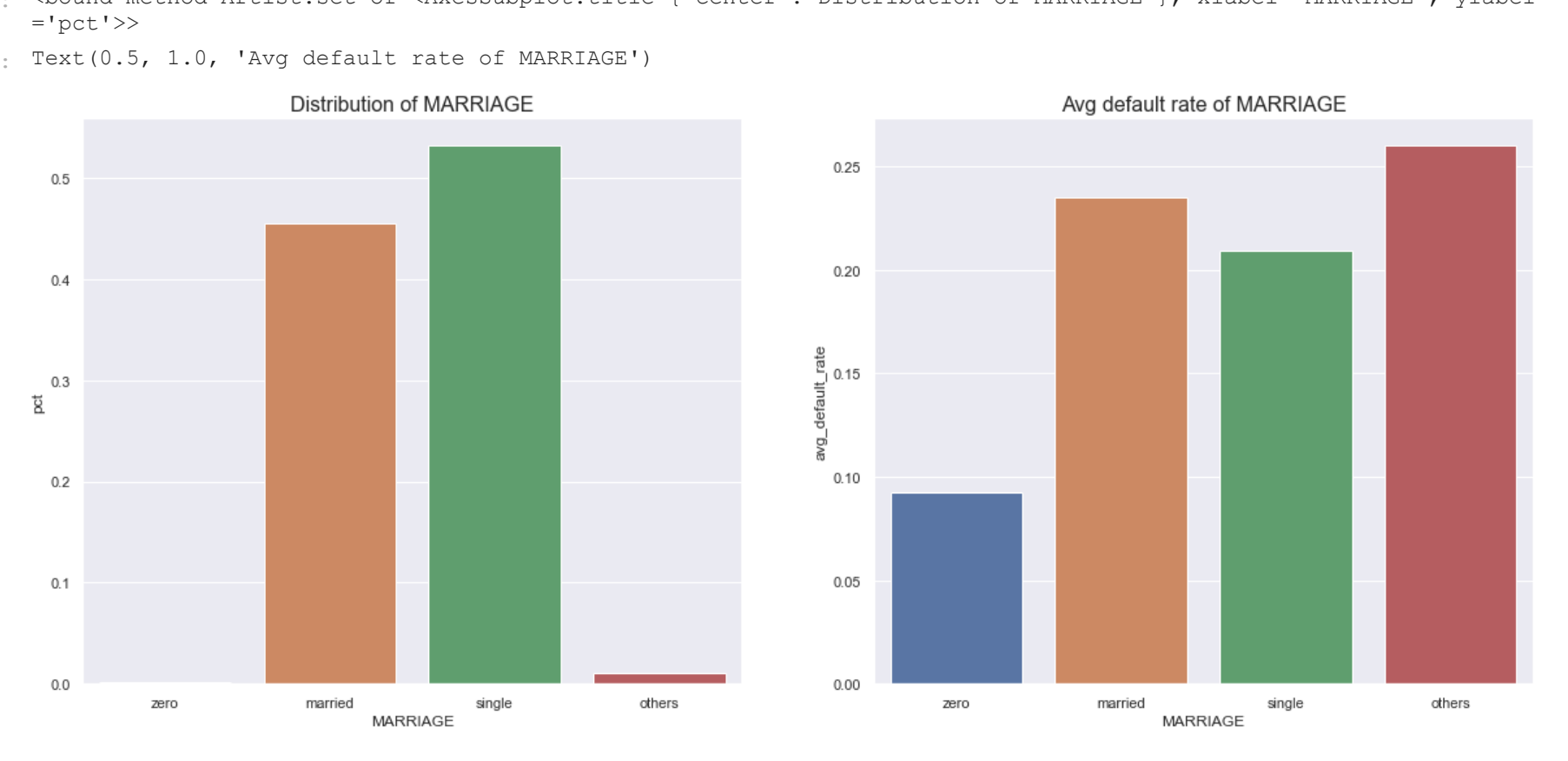
Out[83]: <AxesSubplot:xlabel='MARRIAGE', ylabel='pct'>

Out[83]: <AxesSubplot:xlabel='MARRIAGE', ylabel='avg_default_rate'>

Out[83]: Text(0.5, 1.0, 'Distribution of MARRIAGE')

Out[83]: <bound method Artist.set of <AxesSubplot:title={'center':'Distribution of MARRIAGE'}, xlabel='MARRIAGE', ylabel='pct'>>

Out[83]: Text(0.5, 1.0, 'Avg default rate of MARRIAGE')



"Others" marriage status has a higher risk than the other two. Married status has a higher risk than single.

We has some unkonwed value in the MARRIAGE, like the 0. But because the percentage of it is quite low, so I don't deal with the zeros.

5. LIMIT_BAL

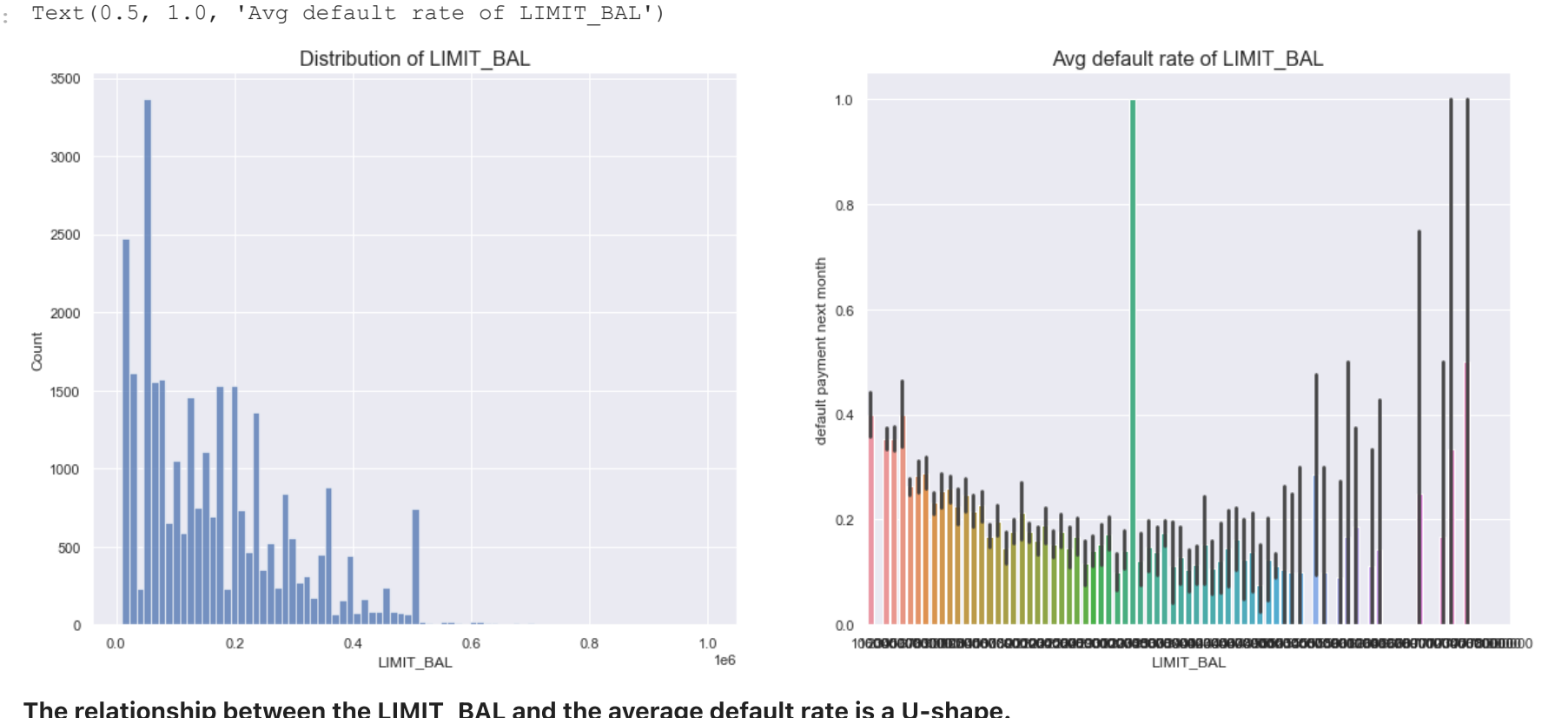
```
In [87]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
col = 'LIMIT_BAL'
sns.histplot(data=df, x=col,ax=ax[0])
ax[0].set_title(f'Distribution of {col}', fontsize=16)
sns.barplot(data=df,x=col,y='default payment next month',ax=ax[1])
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[87]: <AxesSubplot:xlabel='LIMIT_BAL', ylabel='Count'>

Out[87]: Text(0.5, 1.0, 'Distribution of LIMIT_BAL')

Out[87]: <AxesSubplot:xlabel='LIMIT_BAL', ylabel='default payment next month'>

Out[87]: Text(0.5, 1.0, 'Avg default rate of LIMIT_BAL')



The relationship between the LIMIT_BAL and the average default rate is a U-shape.

6. PAY_0

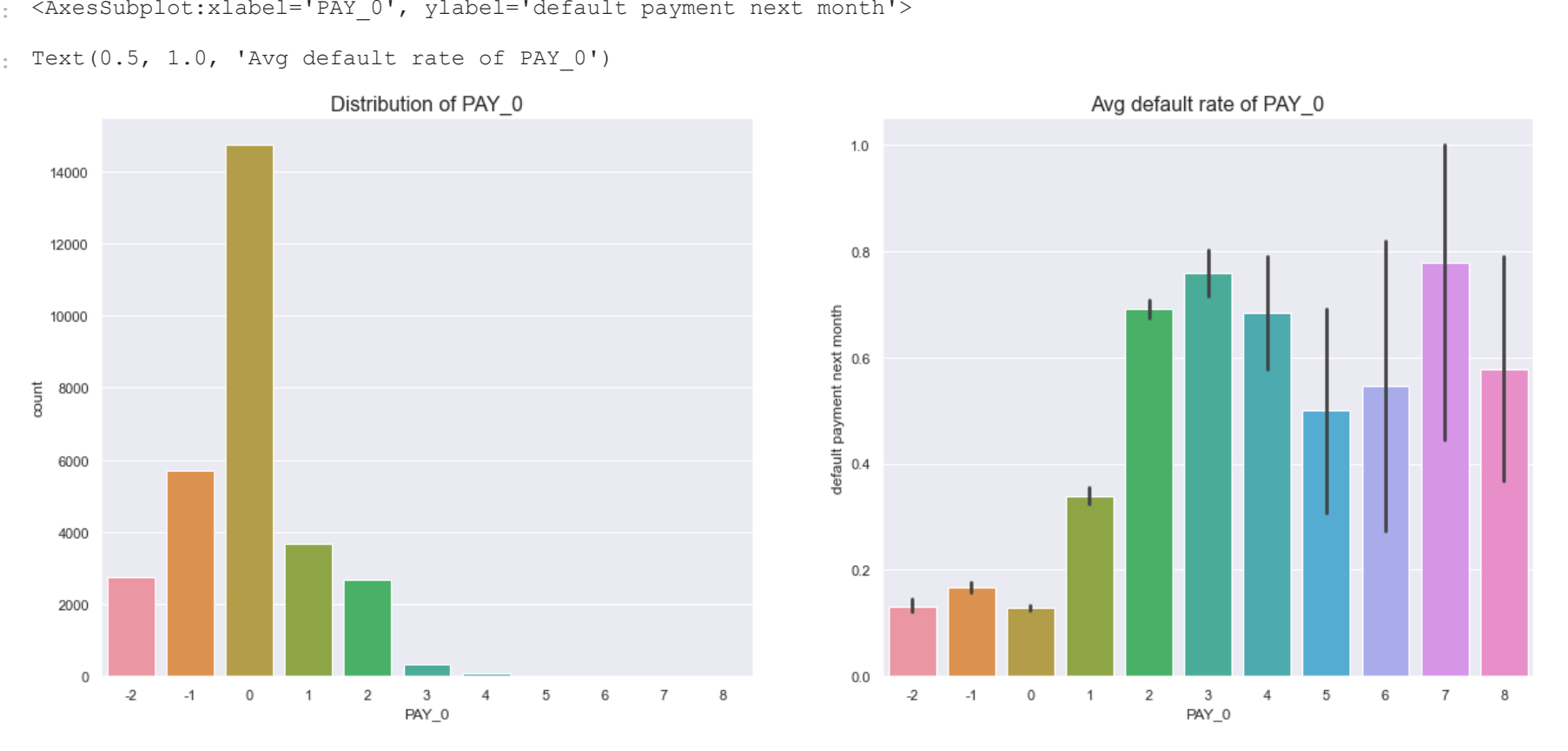
```
In [89]:
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
col = 'PAY_0'
sns.countplot(data=df, x=col,ax=ax[0])
ax[0].set_title(f'Distribution of {col}', fontsize=16)
sns.barplot(data=df,x=col,y='default payment next month',ax=ax[1])
ax[1].set_title(f'Avg default rate of {col}', fontsize=16)
```

Out[89]: <AxesSubplot:xlabel='PAY_0', ylabel='count'>

Out[89]: Text(0.5, 1.0, 'Distribution of PAY_0')

Out[89]: <AxesSubplot:xlabel='PAY_0', ylabel='default payment next month'>

Out[89]: Text(0.5, 1.0, 'Avg default rate of PAY_0')



In general, the higher the PAY_0, the Higher risk.

7. BILL_AMT1

```
In [140…
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
col = 'BILL_AMT1'
sns.histplot(data=df, x=col,ax=ax[0])
ax[0].set_title(f'Distribution of {col}', fontsize=16)


sns.distplot(df[df[target]==0][col],label='No_Default',ax=ax[1])
sns.distplot(df[df[target]==1][col],label='Default',ax=ax[1])
plt.legend()
ax[1].set_title(f'Distribution of {col} in different target', fontsize=16)
```

Out[140… <AxesSubplot:xlabel='BILL_AMT1', ylabel='Count'>

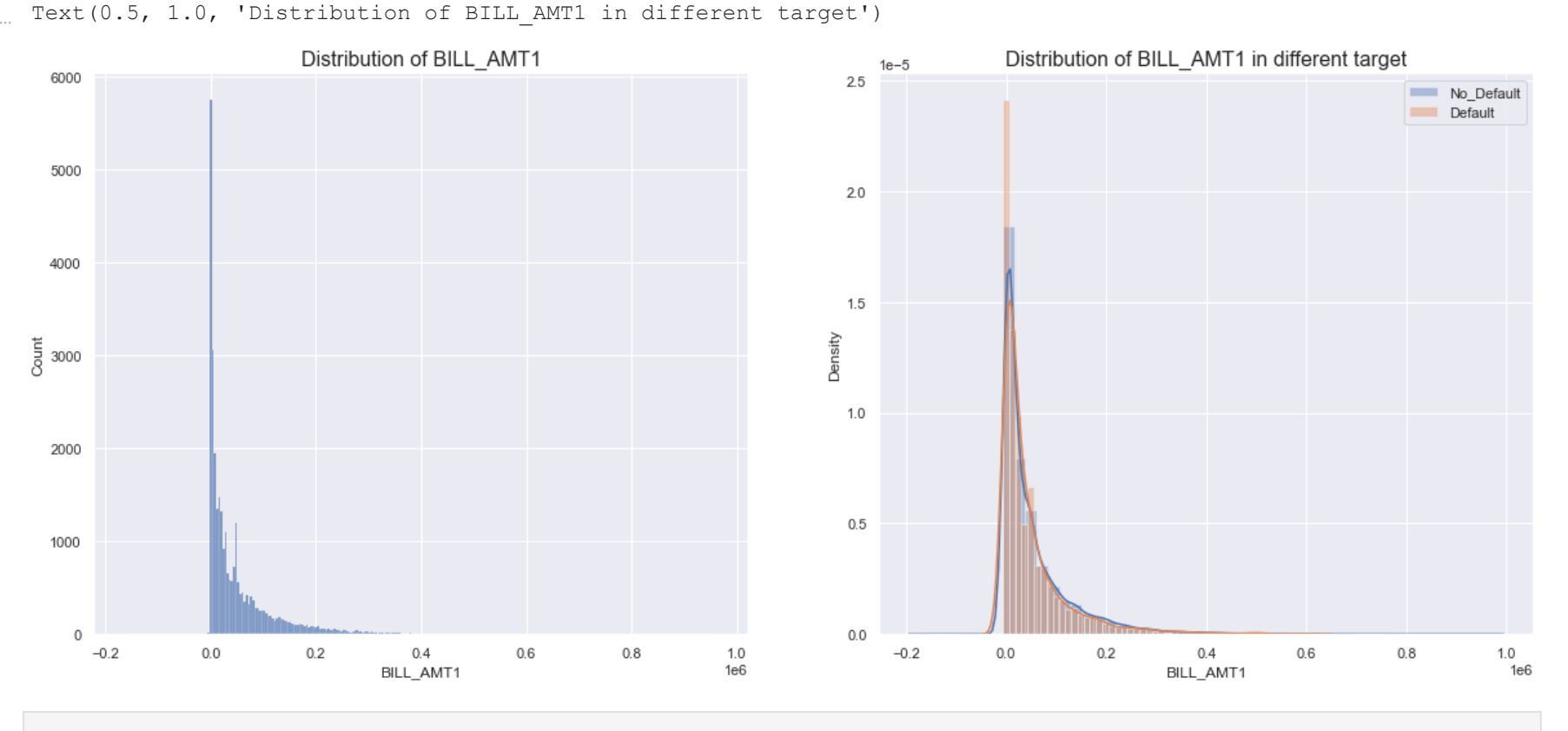
Out[140… Text(0.5, 1.0, 'Distribution of BILL_AMT1')

Out[140… <AxesSubplot:xlabel='BILL_AMT1', ylabel='Density'>

Out[140… <AxesSubplot:xlabel='BILL_AMT1', ylabel='Density'>

Out[140… <matplotlib.legend.Legend at 0x7fe640a07190>

Out[140… Text(0.5, 1.0, 'Distribution of BILL_AMT1 in different target')




```
In [148…
num_bins=20
q= [round(i*(1/num_bins),3) for i in range(num_bins)]
bins = pd.qcut(df[col],q)

data = df.copy()
data['bins'] = bins

count = data.groupby('bins')[target].count()
pct = count/data.shape[0]
rate = data.groupby('bins')[target].mean()

tmp = pd.concat([count,pct,rate],axis=1)
tmp.columns = ['count','pct','avg_default_rate']
tmp=tmp.reset_index()#.sort_values(by='avg_default_rate')
tmp
```

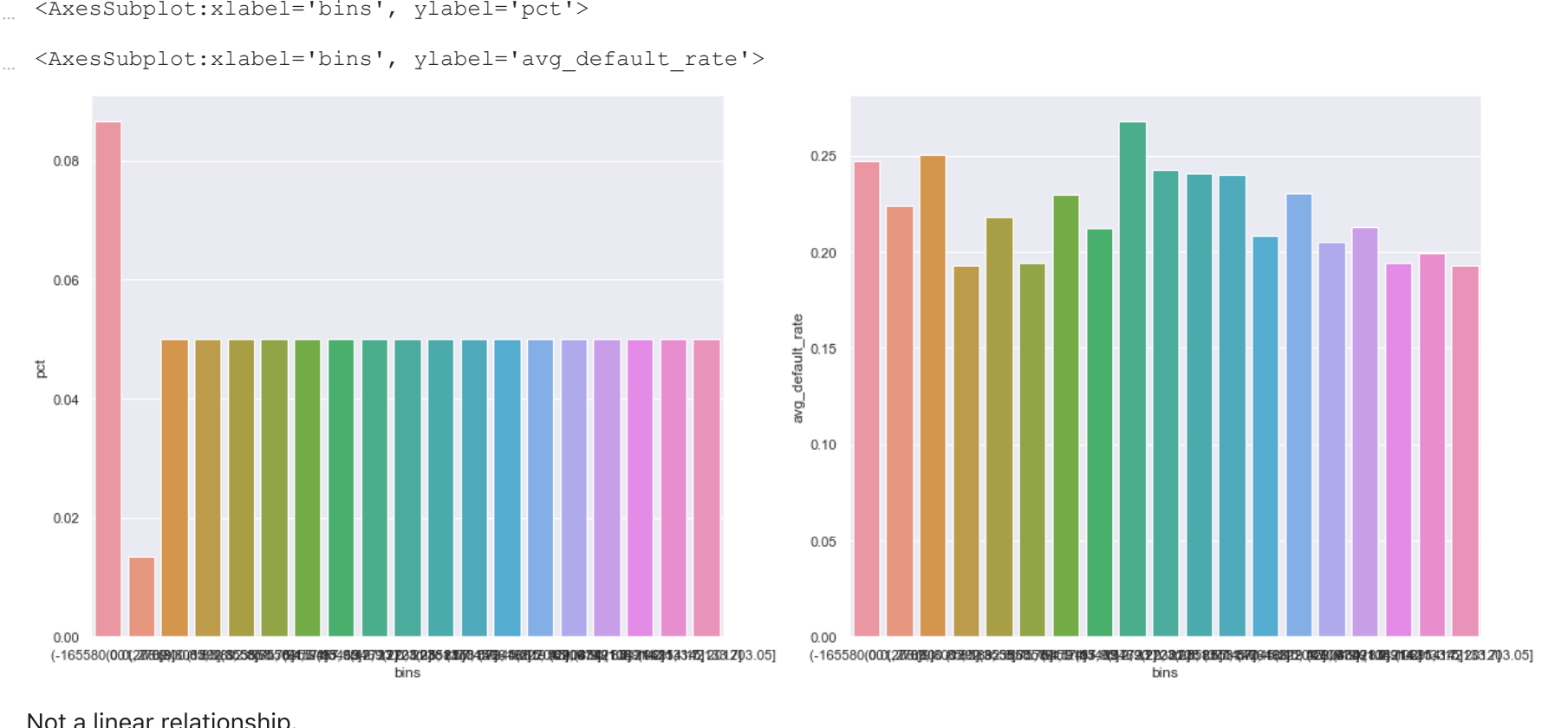
Out[148…

	bins	count	pct	avg_default_rate
0	(-165580.001, 0.0]	2598	0.086600	0.247498
1	(0.0, 278.9]	402	0.013400	0.223881
2	(278.9, 800.85]	1500	0.050000	0.250667
3	(800.85, 1892.8]	1500	0.050000	0.192667
4	(1892.8, 3558.75]	1500	0.050000	0.218000
5	(3558.75, 6050.4]	1500	0.050000	0.194000
6	(6050.4, 9157.95]	1500	0.050000	0.230000
7	(9157.95, 13469.2]	1500	0.050000	0.212000
8	(13469.2, 17723.0]	1501	0.050033	0.267821
9	(17723.0, 22381.5]	1499	0.049967	0.242829
10	(22381.5, 28581.7]	1500	0.050000	0.240667
11	(28581.7, 37045.2]	1500	0.050000	0.240000
12	(37045.2, 46619.15]	1500	0.050000	0.208667
13	(46619.15, 52204.9]	1500	0.050000	0.230667
14	(52204.9, 67091.0]	1500	0.050000	0.205333
15	(67091.0, 83421.2]	1500	0.050000	0.212667
16	(83421.2, 106914.15]	1500	0.050000	0.194000
17	(106914.15, 142133.7]	1500	0.050000	0.199333
18	(142133.7, 201203.05]	1500	0.050000	0.192667

```
In [156…
col = 'bins'
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
sns.barplot(data=tmp,x=col,y='pct',ax=ax[0])
sns.barplot(data=tmp,x=col,y='avg_default_rate',ax=ax[1])
```

Out[156… <AxesSubplot:xlabel='bins', ylabel='pct'>

Out[156… <AxesSubplot:xlabel='bins', ylabel='avg_default_rate'>



Not a linear relationship.

8. PAY_AMT1

```
In [178…
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
col = 'PAY_AMT1'
sns.histplot(data=df, x=col,ax=ax[0])
ax[0].set_xlim(0,75000)
ax[0].set_title(f'Distribution of {col}', fontsize=16)



sns.distplot(df[df[target]==0][col],label='No_Default',ax=ax[1])
sns.distplot(df[df[target]==1][col],label='Default',ax=ax[1])
plt.legend()
ax[1].set_xlim(0,75000)
ax[1].set_title(f'Distribution of {col} in different target', fontsize=16)
```

Out[178… <AxesSubplot:xlabel='PAY_AMT1', ylabel='Count'>

Out[178… (0.0, 75000.0)

Out[178… Text(0.5, 1.0, 'Distribution of PAY_AMT1')

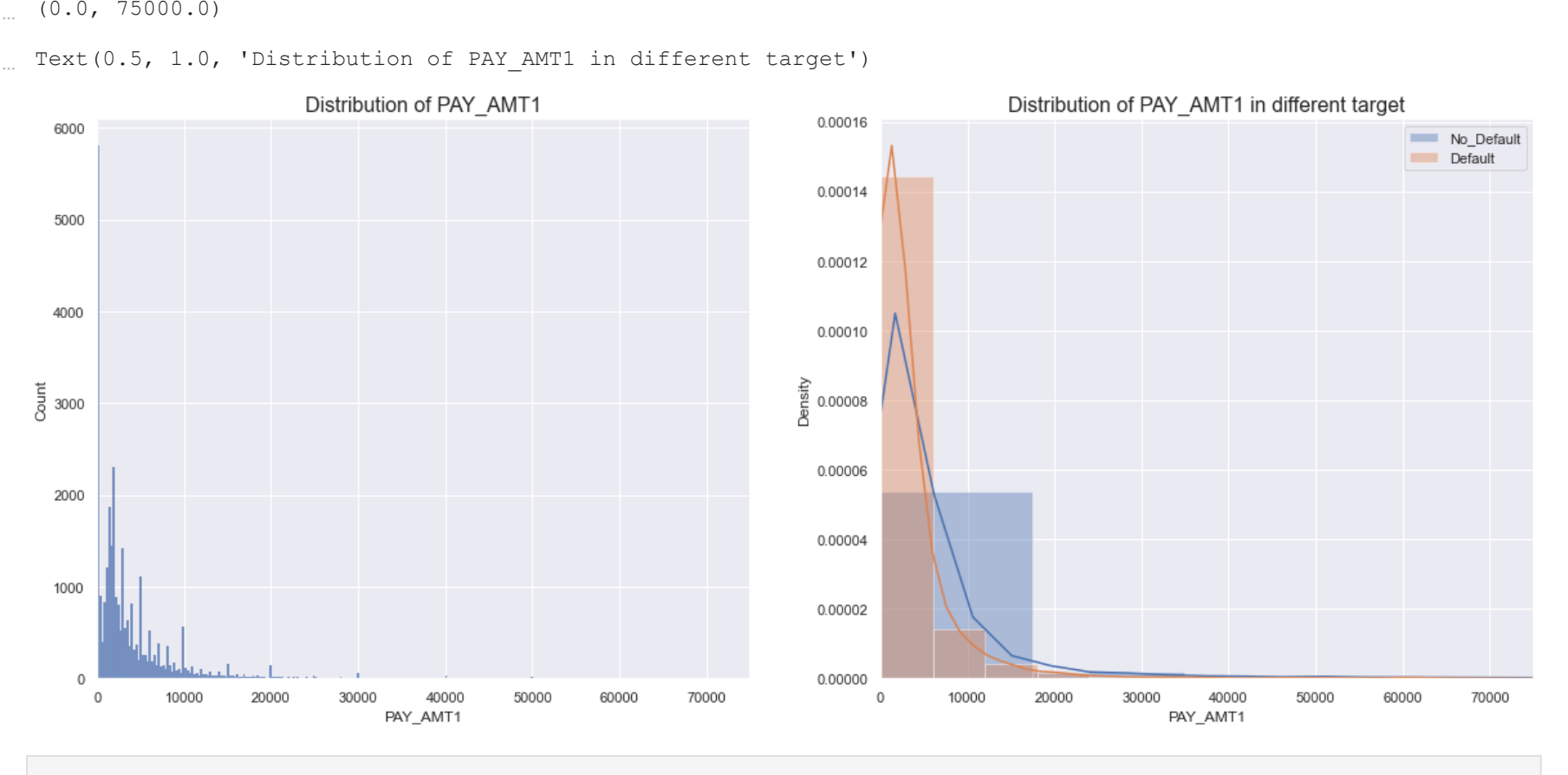
Out[178… <AxesSubplot:xlabel='PAY_AMT1', ylabel='Density'>

Out[178… <AxesSubplot:xlabel='PAY_AMT1', ylabel='Density'>

Out[178… <matplotlib.legend.Legend at 0x7fe5b3612280>

Out[178… (0.0, 75000.0)

Out[178… Text(0.5, 1.0, 'Distribution of PAY_AMT1 in different target')




```
In [181…
num_bins=20
q= [round(i*(1/num_bins),3) for i in range(num_bins)]
bins = pd.qcut(df[col],q,duplicates='drop')

data = df.copy()
data['bins'] = bins

count = data.groupby('bins')[target].count()
pct = count/data.shape[0]
rate = data.groupby('bins')[target].mean()

tmp = pd.concat([count,pct,rate],axis=1)
tmp.columns = ['count','pct','avg_default_rate']
tmp=tmp.reset_index()#.sort_values(by='avg_default_rate')
tmp
```

Out[181…

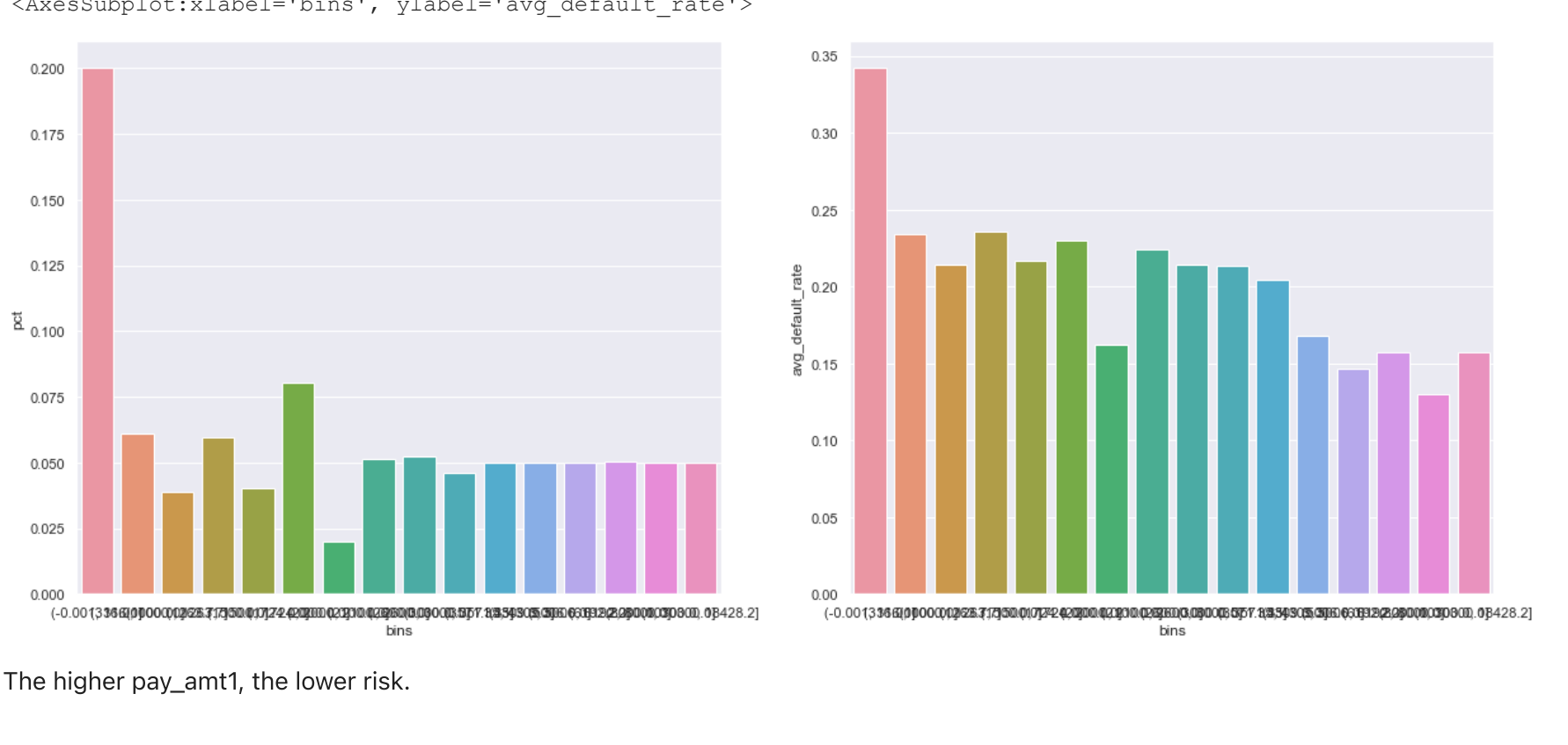
	bins	count	pct	avg_default_rate
0	(-0.001, 316.0]	6002	0.200067	0.342219
1	(316.0, 1000.0]	1837	0.061233	0.234077
2	(1000.0, 1263.7]	1161	0.038700	0.214470
3	(1263.7, 1500.0]	1793	0.059767	0.235360
4	(1500.0, 1724.0]	1210	0.040333	0.216529
5	(1724.0, 2000.0]	2412	0.080400	0.230100
6	(2000.0, 2100.0]	598	0.019933	0.162207
7	(2100.0, 2600.0]	1539	0.051300	0.224172
8	(2600.0, 3000.0]	1564	0.052133	0.214194
9	(3000.0, 3571.35]	1384	0.046133	0.213150
10	(3571.35, 4309.3]	1500	0.050000	0.204000
11	(4309.3, 5006.0]	1503	0.050100	0.167665
12	(5006.0, 6192.2]	1497	0.049900	0.146293
13	(6192.2, 8000.0]	1512	0.050400	0.157407
14	(8000.0, 10300.0]	1491	0.049700	0.130114
15	(10300.0, 18428.2]	1497	0.049900	0.157649

In [182…

```python
col = 'bins'
fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,8))
sns.barplot(data=tmp,x=col,y='pct',ax=ax[0])
sns.barplot(data=tmp,x=col,y='avg_default_rate',ax=ax[1])
```

Out[182… `<AxesSubplot:xlabel='bins', ylabel='pct'>`

Out[182… `<AxesSubplot:xlabel='bins', ylabel='avg_default_rate'>`



The higher pay_amt1, the lower risk.

3.Modeling

From the feature analysis above, If we want to keep the model have pretty good explainability, we will use Weight of Evidence to transform the features , then use Logistic Regression.

But WOE is time-consumeing, So here I will directly use some simple modeling methods.

Here I use LightGBM to train the model, just because of these reasons:

1. LightGBM has higher accuracy
2. LightGBM can deal with the catagorical data. Although we can label the catagorical features, but in this case, I want to speed up the training process.
3. LighGBM runs fast.

training the model

In [111…

```python
df.columns
```

Out[111…

```
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', 'EDUCATION2'],
      dtype='object')
```

In [112…

```python
from sklearn.model_selection import train_test_split

features = [
      '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',
         ]


cate_features = ['SEX','EDUCATION','MARRIAGE']

X = df[features]
y = df['default payment next month']

X.dtypes

# transfer some features into category feature
for c in cate_features:
    X[c] = X[c].astype('category')

X.dtypes
```

Out[112…

```
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
dtype: object
```

Out[112…

```
LIMIT_BAL       int64
SEX          category
EDUCATION    category
MARRIAGE     category
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
dtype: object
```

In [113…

```python
# split training and testing sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33)
X_train.shape, X_test.shape
```

Out[113… `((20100, 23), (9900, 23))`

In [114…

```python
import lightgbm as lgb
fit_params={"early_stopping_rounds":10,
            "eval_metric" : 'auc',
            "eval_set" : [(X_test,y_test)],
            'eval_names': ['valid'],
            'verbose': 100,
            'feature_name': 'auto', # that's actually the default
            'categorical_feature': 'auto' # that's actually the default
           }


#n_estimators is set to a "large value". The actual number of trees build will depend on early stopping and 10(
clf = lgb.LGBMClassifier(num_leaves= 15, max_depth=-1,
                         random_state=314,
                         silent=True,
                         metric='None',
                         n_jobs=4,
                         n_estimators=1000,
                         colsample_bytree=0.9,
                         subsample=0.9,
                         learning_rate=0.1)

clf.fit(X_train, y_train, **fit_params)
```

Out[114…

```
LGBMClassifier(colsample_bytree=0.9, metric='None', n_estimators=1000, n_jobs=4,
               num_leaves=15, random_state=314, subsample=0.9)
```

In [115…

```python
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)

y_train_proba = clf.predict_proba(X_train)
y_test_proba = clf.predict_proba(X_test)
```

In [116…

```python
from sklearn import metrics
report = metrics.classification_report(y_test,y_test_pred)
print(report)
train_auc = metrics.roc_auc_score(y_train,y_train_proba[:,1])
test_auc = metrics.roc_auc_score(y_test,y_test_proba[:,1])
train_auc,test_auc
```

Out[116… `(0.8201940958265221, 0.7811186918489301)`

In [117…

```python
from sklearn.metrics import roc_curve,auc
fpr_train, tpr_train,th_train = roc_curve(y_train,y_train_proba[:,1],pos_label=1)
fpr_test, tpr_test,th_test = roc_curve(y_test,y_test_proba[:,1],pos_label=1)

train_auc = auc(fpr_train,tpr_train)
test_auc = auc(fpr_test,tpr_test)

train_auc,test_auc

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(fpr_train,tpr_train,label = f'Train AUC: {train_auc:.3f}')
ax.plot(fpr_test,tpr_test,label = f'Test AUC: {test_auc:.3f}')

ax.plot(fpr_train, fpr_train, 'k--', label='Chance Curve')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.grid(True)
ax.legend(fontsize=12)
plt.show()
```

Out[117… `(0.8201940958265221, 0.7811186918489301)`

Out[117… `[<matplotlib.lines.Line2D at 0x7fe643ca23a0>]`

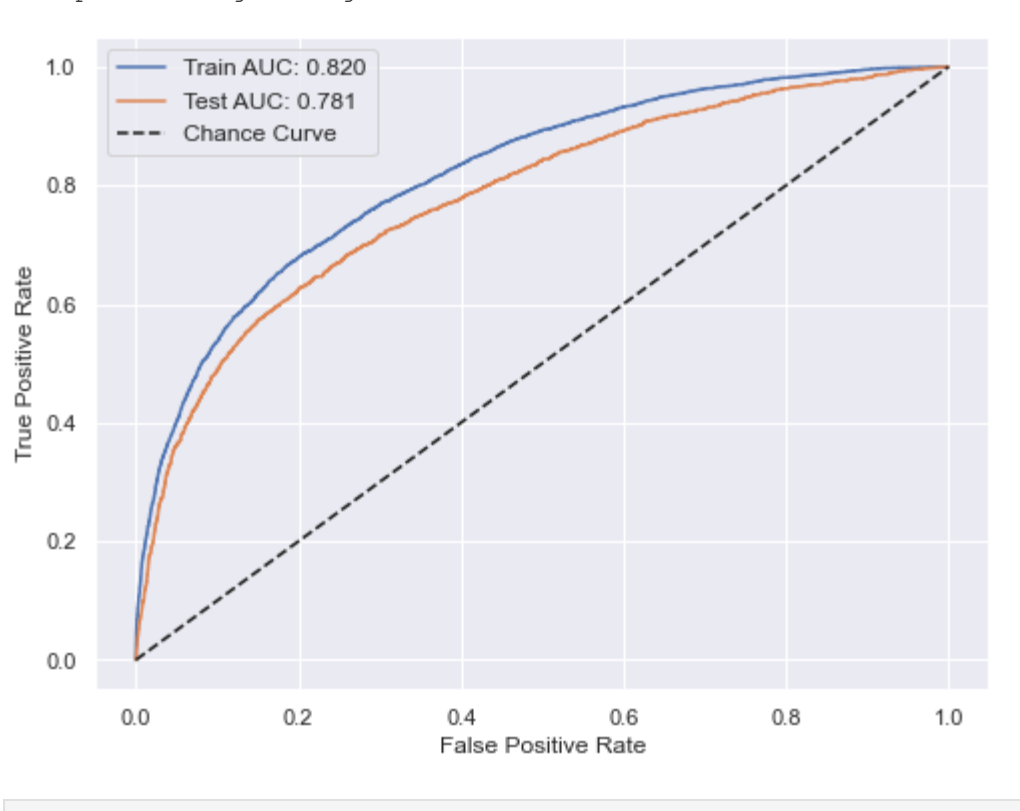
Out[117… `[<matplotlib.lines.Line2D at 0x7fe643ca26a0>]`

Out[117… `[<matplotlib.lines.Line2D at 0x7fe643ca2af0>]`

Out[117… `Text(0.5, 0, 'False Positive Rate')`

Out[117… `Text(0, 0.5, 'True Positive Rate')`

Out[117… `<matplotlib.legend.Legend at 0x7fe5d636ad90>`



In [118…

```python
def cal_ks(scores,labels,num_bins=10,EPS=1e-7):
    table = pd.DataFrame(zip(scores,labels),columns=['score','label'])
    table['score_rank_pct'] = table['score'].rank(na_option='bottom',pct=True)
    ## cut into bins
    q= [round(i*(1/num_bins),3) for i in range(num_bins)]
    q.insert(0,float('-inf'))
    q.append(float('inf'))
    table['rank_bins'] = pd.cut(table['score_rank_pct'],q)
    #
    res = table.groupby('rank_bins').agg({'score':['max','min','count'],'label':['sum']})
    res.columns = ['score_max','score_min','count_all','count_1']
    res['count_0'] = res['count_all']-res['count_1']
    res['margin_1_rate'] = round(res['count_1']/res['count_1'].sum()*100,2)
    res['margin_0_rate'] = round(res['count_0']/res['count_0'].sum()*100,2)
    res['cumcount_1'] = res['count_1'].cumsum()
    res['cumcount_0'] = res['count_0'].cumsum()
    res['cumcount_all'] = res['count_all'].cumsum()
    res['cumpct_all'] = round(res['cumcount_all']/res['count_all'].sum()*100,2)
    res['cumpct_1']=round(res['cumcount_1']/res['count_1'].sum()*100,2)
    res['cumpct_0']=round(res['cumcount_0']/res['count_0'].sum()*100,2)
    res['_ks'] = res['cumpct_0']-res['cumpct_1']
    res['_woe'] = np.log((res['margin_1_rate']+EPS)/(res['margin_0_rate']+EPS))
    res['_iv'] = (res['margin_1_rate']-res['margin_0_rate'])*res['_woe']

    return res['_ks'].max(),res
```

In [134…

```python
ks_train,ks_table_train = cal_ks(y_train_proba[:,1],y_train)
ks_test,ks_table_test = cal_ks(y_test_proba[:,1],y_test)
ks_train,ks_test
ks_table_train
ks_table_test
```

Out[134… `(47.79999999999999, 42.38999999999999)`

Out[134…

	score_max	score_min	count_all	count_1	count_0	margin_1_rate	margin_0_rate	cumcount_1	cumcount_0	cumcount_al
rank_bins										
(-inf, 0.0]	NaN	NaN	0	0	0	0.00	0.00	0	0	0
(0.0, 0.1]	0.063856	0.022775	2010	36	1974	0.81	12.61	36	1974	2010
(0.1, 0.2]	0.084303	0.063857	2010	91	1919	2.05	12.26	127	3893	4020
(0.2, 0.3]	0.103297	0.084312	2010	125	1885	2.81	12.04	252	5778	6030
(0.3, 0.4]	0.125756	0.103329	2010	199	1811	4.48	11.57	451	7589	8040
(0.4, 0.5]	0.149166	0.125761	2010	267	1743	6.01	11.13	718	9332	10050
(0.5, 0.6]	0.177263	0.149183	2010	329	1681	7.40	10.74	1047	11013	12060
(0.6, 0.7]	0.224241	0.177272	2009	410	1599	9.22	10.21	1457	12612	14069
(0.7, 0.8]	0.311883	0.224275	2011	577	1434	12.98	9.16	2034	14046	16080
(0.8, 0.9]	0.573131	0.311946	2010	908	1102	20.42	7.04	2942	15148	18090
(0.9, inf]	0.867862	0.573224	2010	1504	506	33.83	3.23	4446	15654	20100

Out[134…

	score_max	score_min	count_all	count_1	count_0	margin_1_rate	margin_0_rate	cumcount_1	cumcount_0	cumcount_al
rank_bins										
(-inf, 0.0]	NaN	NaN	0	0	0	0.00	0.00	0	0	0
(0.0, 0.1]	0.064031	0.024044	990	48	942	2.19	12.22	48	942	990
(0.1, 0.2]	0.085301	0.064032	990	60	930	2.74	12.06	108	1872	1980
(0.2, 0.3]	0.103942	0.085344	988	86	902	3.93	11.70	194	2774	2968
(0.3, 0.4]	0.125249	0.103971	992	119	873	5.43	11.32	313	3647	3960
(0.4, 0.5]	0.148817	0.125277	990	146	844	6.67	10.95	459	4491	4950
(0.5, 0.6]	0.177044	0.148817	990	146	844	6.67	10.95	605	5335	5940
(0.6, 0.7]	0.222332	0.177063	990	205	785	9.36	10.18	810	6120	6930
(0.7, 0.8]	0.309041	0.222332	990	263	727	12.01	9.43	1073	6847	7920
(0.8, 0.9]	0.576514	0.309158	990	426	564	19.45	7.32	1499	7411	8910
(0.9, inf]	0.845043	0.577258	990	691	299	31.55	3.88	2190	7710	9900

From the table above, we can see the best cut-off for predicting a client is default is when this client get a probability of default over 0.6, we can assume this client is highly default.

In [177…

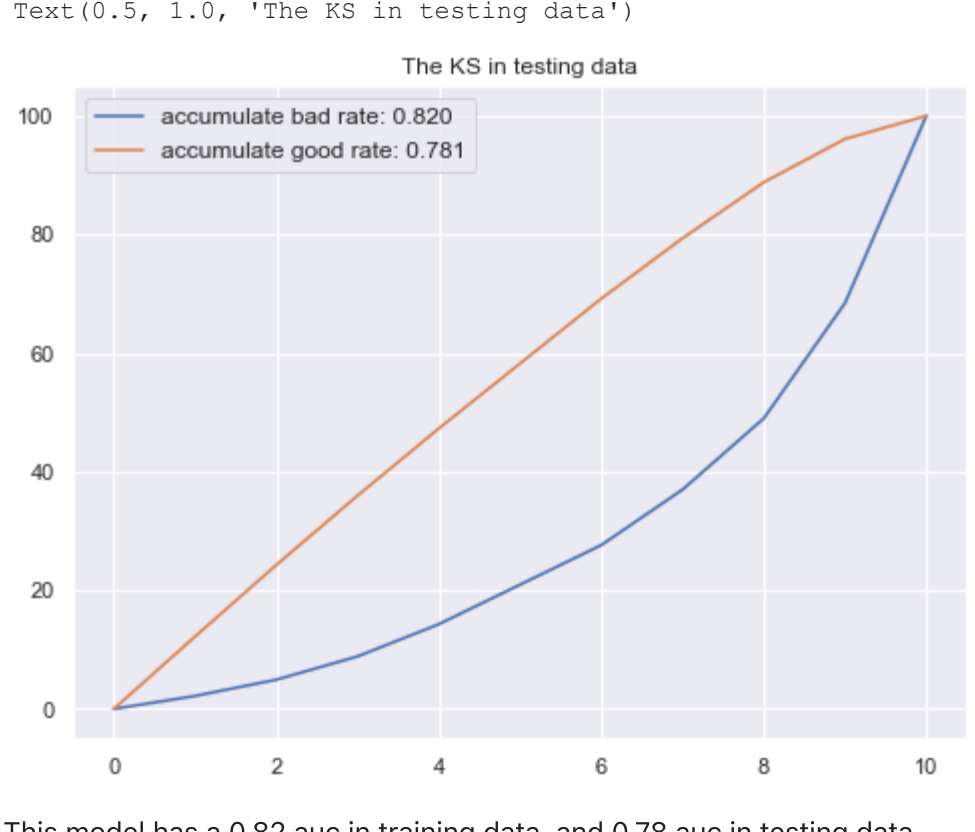
```python
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(ks_table_test['cumpct_1'],label = f'accumulate bad rate: {train_auc:.3f}')
ax.plot(ks_table_test['cumpct_0'],label = f'accumulate good rate: {test_auc:.3f}')
ax.grid(True)
ax.legend(fontsize=12)
ax.set_title("The KS in testing data")
plt.show()
```

Out[177… `[<matplotlib.lines.Line2D at 0x7fe6d7436940>]`

Out[177… `[<matplotlib.lines.Line2D at 0x7fe6d7436b50>]`

Out[177… `<matplotlib.legend.Legend at 0x7fe6d7419730>`

Out[177… `Text(0.5, 1.0, 'The KS in testing data')`



This model has a 0.82 auc in training data, and 0.78 auc in testing data.

For the KS-statistic , it is 48% in training data and 42 in testing data.

So we can consider this model has a good performance.

feature importance and explainability

In [136…

```python
from sklearn.inspection import permutation_importance
feature_importance = clf.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(X.columns)[sorted_idx])
plt.title("Feature Importance (impurity-based feature importances)")

result = permutation_importance(
    clf, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2
)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(
    result.importances[sorted_idx].T,
    vert=False,
    labels=np.array(X.columns)[sorted_idx],
)
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
```

Out[136… `<AxesSubplot:>`

Out[136… `<BarContainer object of 23 artists>`

Out[136…

```
([<matplotlib.axis.YTick at 0x7fe5e76619a0>,
  <matplotlib.axis.YTick at 0x7fe5e7661580>,
  <matplotlib.axis.YTick at 0x7fe5e7660400>,
  <matplotlib.axis.YTick at 0x7fe5a0301a90>,
  <matplotlib.axis.YTick at 0x7fe5a0301fa0>,
  <matplotlib.axis.YTick at 0x7fe5d6deb4f0>,
  <matplotlib.axis.YTick at 0x7fe5d6deba00>,
  <matplotlib.axis.YTick at 0x7fe5d6debf10>,
  <matplotlib.axis.YTick at 0x7fe5d6de8460>,
  <matplotlib.axis.YTick at 0x7fe5d6de8970>,
  <matplotlib.axis.YTick at 0x7fe5d6de8e80>,
  <matplotlib.axis.YTick at 0x7fe5d6de8580>,
  <matplotlib.axis.YTick at 0x7fe5d6deb610>,
  <matplotlib.axis.YTick at 0x7fe5e7687550>,
  <matplotlib.axis.YTick at 0x7fe5d6df5580>,
  <matplotlib.axis.YTick at 0x7fe5d6df5a90>,
  <matplotlib.axis.YTick at 0x7fe5d6df5fa0>,
  <matplotlib.axis.YTick at 0x7fe5d6dfc4f0>,
  <matplotlib.axis.YTick at 0x7fe5d6dfca00>,
  <matplotlib.axis.YTick at 0x7fe5d6dfcf10>,
  <matplotlib.axis.YTick at 0x7fe5d6e00460>,
  <matplotlib.axis.YTick at 0x7fe5d6dfc100>,
  <matplotlib.axis.YTick at 0x7fe5d6df5250>],
 [Text(0, 0.5, 'PAY_5'),
  Text(0, 1.5, 'MARRIAGE'),
  Text(0, 2.5, 'SEX'),
  Text(0, 3.5, 'EDUCATION'),
  Text(0, 4.5, 'BILL_AMT6'),
  Text(0, 5.5, 'PAY_3'),
  Text(0, 6.5, 'BILL_AMT5'),
  Text(0, 7.5, 'BILL_AMT4'),
  Text(0, 8.5, 'PAY_4'),
  Text(0, 9.5, 'PAY_2'),
  Text(0, 10.5, 'PAY_6'),
  Text(0, 11.5, 'BILL_AMT3'),
  Text(0, 12.5, 'BILL_AMT2'),
  Text(0, 13.5, 'AGE'),
  Text(0, 14.5, 'PAY_AMT6'),
  Text(0, 15.5, 'PAY_AMT5'),
  Text(0, 16.5, 'PAY_AMT4'),
  Text(0, 17.5, 'PAY_AMT3'),
  Text(0, 18.5, 'PAY_AMT2'),
  Text(0, 19.5, 'PAY_0'),
  Text(0, 20.5, 'PAY_AMT1'),
  Text(0, 21.5, 'LIMIT_BAL'),
  Text(0, 22.5, 'BILL_AMT1')])
```

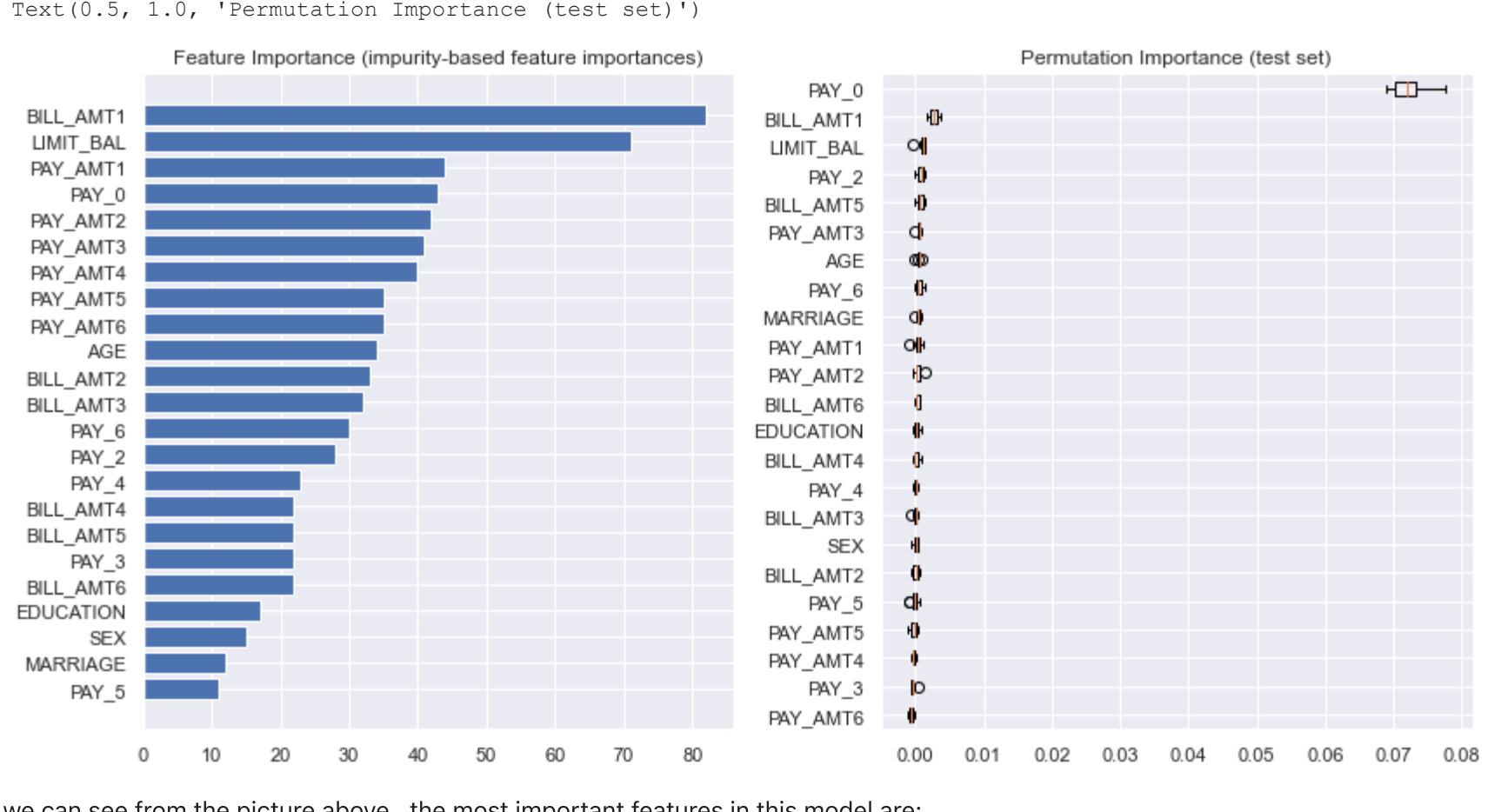
Out[136… `Text(0.5, 1.0, 'Feature Importance (impurity-based feature importances)')`

Out[136… `<AxesSubplot:>`

Out[136…

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe5e772c370>,
  <matplotlib.lines.Line2D at 0x7fe5e772c6d0>,
  <matplotlib.lines.Line2D at 0x7fe5e77383a0>,
  <matplotlib.lines.Line2D at 0x7fe5e7738700>,
  <matplotlib.lines.Line2D at 0x7fe5e7746b80>,
  <matplotlib.lines.Line2D at 0x7fe5e7746ee0>,
  <matplotlib.lines.Line2D at 0x7fe5e769c3a0>,
  <matplotlib.lines.Line2D at 0x7fe5e769c700>,
  <matplotlib.lines.Line2D at 0x7fe5e76a7b80>,
  <matplotlib.lines.Line2D at 0x7fe5e76a7ee0>,
  <matplotlib.lines.Line2D at 0x7fe5e76be3a0>,
  <matplotlib.lines.Line2D at 0x7fe5e76be700>,
  <matplotlib.lines.Line2D at 0x7fe5d6e20b80>,
  <matplotlib.lines.Line2D at 0x7fe5d6e20ee0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e383a0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e38700>,
  <matplotlib.lines.Line2D at 0x7fe5d6e46b80>,
  <matplotlib.lines.Line2D at 0x7fe5d6e46ee0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e5c3a0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e5c700>,
  <matplotlib.lines.Line2D at 0x7fe5d6d57b80>,
  <matplotlib.lines.Line2D at 0x7fe5d6d57ee0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d703a0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d70700>,
  <matplotlib.lines.Line2D at 0x7fe5d6d7bb80>,
  <matplotlib.lines.Line2D at 0x7fe5d6d7bee0>,
  <matplotlib.lines.Line2D at 0x7fe5b10ad3a0>,
  <matplotlib.lines.Line2D at 0x7fe5b10ad700>,
  <matplotlib.lines.Line2D at 0x7fe5b10b9b80>,
  <matplotlib.lines.Line2D at 0x7fe5b10b9ee0>,
  <matplotlib.lines.Line2D at 0x7fe5b10d03a0>,
  <matplotlib.lines.Line2D at 0x7fe5b10d0700>,
  <matplotlib.lines.Line2D at 0x7fe5b10ddb80>,
  <matplotlib.lines.Line2D at 0x7fe5b10ddee0>,
  <matplotlib.lines.Line2D at 0x7fe5e76d53a0>,
  <matplotlib.lines.Line2D at 0x7fe5e76d5700>,
  <matplotlib.lines.Line2D at 0x7fe5e76e0b80>,
  <matplotlib.lines.Line2D at 0x7fe5e76e0ee0>,
  <matplotlib.lines.Line2D at 0x7fe5e76f43a0>,
  <matplotlib.lines.Line2D at 0x7fe5e76f4700>,
  <matplotlib.lines.Line2D at 0x7fe5e7704b80>,
  <matplotlib.lines.Line2D at 0x7fe5e7704ee0>,
  <matplotlib.lines.Line2D at 0x7fe5e77583a0>,
  <matplotlib.lines.Line2D at 0x7fe5e7758700>,
  <matplotlib.lines.Line2D at 0x7fe5e7767b80>,
  <matplotlib.lines.Line2D at 0x7fe5e7767ee0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe5e772ca30>,
  <matplotlib.lines.Line2D at 0x7fe5e772cd90>,
  <matplotlib.lines.Line2D at 0x7fe5e7738a60>,
  <matplotlib.lines.Line2D at 0x7fe5e7738dc0>,
  <matplotlib.lines.Line2D at 0x7fe5e7690280>,
  <matplotlib.lines.Line2D at 0x7fe5e76905e0>,
  <matplotlib.lines.Line2D at 0x7fe5e769ca60>,
  <matplotlib.lines.Line2D at 0x7fe5e769cdc0>,
  <matplotlib.lines.Line2D at 0x7fe5e76b3280>,
  <matplotlib.lines.Line2D at 0x7fe5e76b35e0>,
  <matplotlib.lines.Line2D at 0x7fe5e76bea60>,
  <matplotlib.lines.Line2D at 0x7fe5e76bedc0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e2c280>,
  <matplotlib.lines.Line2D at 0x7fe5d6e2c5e0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e38a60>,
  <matplotlib.lines.Line2D at 0x7fe5d6e38dc0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e4f280>,
  <matplotlib.lines.Line2D at 0x7fe5d6e4f5e0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e5ca60>,
  <matplotlib.lines.Line2D at 0x7fe5d6e5cdc0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d64280>,
  <matplotlib.lines.Line2D at 0x7fe5d6d645e0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d70a60>,
  <matplotlib.lines.Line2D at 0x7fe5d6d70dc0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d89280>,
  <matplotlib.lines.Line2D at 0x7fe5d6d895e0>,
  <matplotlib.lines.Line2D at 0x7fe5b10ada60>,
  <matplotlib.lines.Line2D at 0x7fe5b10addc0>,
  <matplotlib.lines.Line2D at 0x7fe5b10c3280>,
  <matplotlib.lines.Line2D at 0x7fe5b10c35e0>,
  <matplotlib.lines.Line2D at 0x7fe5b10d0a60>,
  <matplotlib.lines.Line2D at 0x7fe5b10d0dc0>,
  <matplotlib.lines.Line2D at 0x7fe5b10e7280>,
  <matplotlib.lines.Line2D at 0x7fe5b10e75e0>,
  <matplotlib.lines.Line2D at 0x7fe5e76d5a60>,
  <matplotlib.lines.Line2D at 0x7fe5e76d5dc0>,
  <matplotlib.lines.Line2D at 0x7fe5e76ec280>,
  <matplotlib.lines.Line2D at 0x7fe5e76ec5e0>,
  <matplotlib.lines.Line2D at 0x7fe5e76f4a60>,
  <matplotlib.lines.Line2D at 0x7fe5e76f4dc0>,
  <matplotlib.lines.Line2D at 0x7fe5e774f280>,
  <matplotlib.lines.Line2D at 0x7fe5e774f5e0>,
  <matplotlib.lines.Line2D at 0x7fe5e7758a60>,
  <matplotlib.lines.Line2D at 0x7fe5e7758dc0>,
  <matplotlib.lines.Line2D at 0x7fe5e7774280>,
  <matplotlib.lines.Line2D at 0x7fe5e77745e0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe5e771cfd0>,
  <matplotlib.lines.Line2D at 0x7fe5d6de8a90>,
  <matplotlib.lines.Line2D at 0x7fe5e7746820>,
  <matplotlib.lines.Line2D at 0x7fe5e769c040>,
  <matplotlib.lines.Line2D at 0x7fe5e76a7820>,
  <matplotlib.lines.Line2D at 0x7fe5e76be040>,
  <matplotlib.lines.Line2D at 0x7fe5d6e20820>,
  <matplotlib.lines.Line2D at 0x7fe5d6e38040>,
  <matplotlib.lines.Line2D at 0x7fe5d6e46820>,
  <matplotlib.lines.Line2D at 0x7fe5d6e5c040>,
  <matplotlib.lines.Line2D at 0x7fe5d6d57820>,
  <matplotlib.lines.Line2D at 0x7fe5d6d70040>,
  <matplotlib.lines.Line2D at 0x7fe5d6d7b820>,
  <matplotlib.lines.Line2D at 0x7fe5b10ad040>,
  <matplotlib.lines.Line2D at 0x7fe5b10b9820>,
  <matplotlib.lines.Line2D at 0x7fe5b10d0040>,
  <matplotlib.lines.Line2D at 0x7fe5b10dd820>,
  <matplotlib.lines.Line2D at 0x7fe5e76d5040>,
  <matplotlib.lines.Line2D at 0x7fe5e76e0820>,
  <matplotlib.lines.Line2D at 0x7fe5e76f4040>,
  <matplotlib.lines.Line2D at 0x7fe5e7704820>,
  <matplotlib.lines.Line2D at 0x7fe5e7758040>,
  <matplotlib.lines.Line2D at 0x7fe5e7767820>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe5e7738130>,
  <matplotlib.lines.Line2D at 0x7fe5e7746160>,
  <matplotlib.lines.Line2D at 0x7fe5e7690940>,
  <matplotlib.lines.Line2D at 0x7fe5e76a7160>,
  <matplotlib.lines.Line2D at 0x7fe5e76b3940>,
  <matplotlib.lines.Line2D at 0x7fe5d6e20160>,
  <matplotlib.lines.Line2D at 0x7fe5d6e2c940>,
  <matplotlib.lines.Line2D at 0x7fe5d6e46160>,
  <matplotlib.lines.Line2D at 0x7fe5d6e4f940>,
  <matplotlib.lines.Line2D at 0x7fe5d6d57160>,
  <matplotlib.lines.Line2D at 0x7fe5d6d64940>,
  <matplotlib.lines.Line2D at 0x7fe5d6d7b160>,
  <matplotlib.lines.Line2D at 0x7fe5d6d89940>,
  <matplotlib.lines.Line2D at 0x7fe5b10b9160>,
  <matplotlib.lines.Line2D at 0x7fe5b10c3940>,
  <matplotlib.lines.Line2D at 0x7fe5b10dd160>,
  <matplotlib.lines.Line2D at 0x7fe5b10e7940>,
  <matplotlib.lines.Line2D at 0x7fe5e76e0160>,
  <matplotlib.lines.Line2D at 0x7fe5e76ec940>,
  <matplotlib.lines.Line2D at 0x7fe5e7704160>,
  <matplotlib.lines.Line2D at 0x7fe5e774f940>,
  <matplotlib.lines.Line2D at 0x7fe5e7767160>,
  <matplotlib.lines.Line2D at 0x7fe5e7774940>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe5e766d3a0>,
  <matplotlib.lines.Line2D at 0x7fe5e77464c0>,
  <matplotlib.lines.Line2D at 0x7fe5e7690ca0>,
  <matplotlib.lines.Line2D at 0x7fe5e76a74c0>,
  <matplotlib.lines.Line2D at 0x7fe5e76b3ca0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e204c0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e2cca0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e464c0>,
  <matplotlib.lines.Line2D at 0x7fe5d6e4fca0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d574c0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d64ca0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d7b4c0>,
  <matplotlib.lines.Line2D at 0x7fe5d6d89ca0>,
  <matplotlib.lines.Line2D at 0x7fe5b10b94c0>,
  <matplotlib.lines.Line2D at 0x7fe5b10c3ca0>,
  <matplotlib.lines.Line2D at 0x7fe5b10dd4c0>,
  <matplotlib.lines.Line2D at 0x7fe5b10e7ca0>,
  <matplotlib.lines.Line2D at 0x7fe5e76e04c0>,
  <matplotlib.lines.Line2D at 0x7fe5e76ecca0>,
  <matplotlib.lines.Line2D at 0x7fe5e77044c0>,
  <matplotlib.lines.Line2D at 0x7fe5e774fca0>,
  <matplotlib.lines.Line2D at 0x7fe5e77674c0>,
  <matplotlib.lines.Line2D at 0x7fe5e7774ca0>],
 'means': []}
```

Out[136… `Text(0.5, 1.0, 'Permutation Importance (test set)')`



we can see from the picture above , the most important features in this model are:

BILL_AMT1, PAY_0,LIMIT_BAL

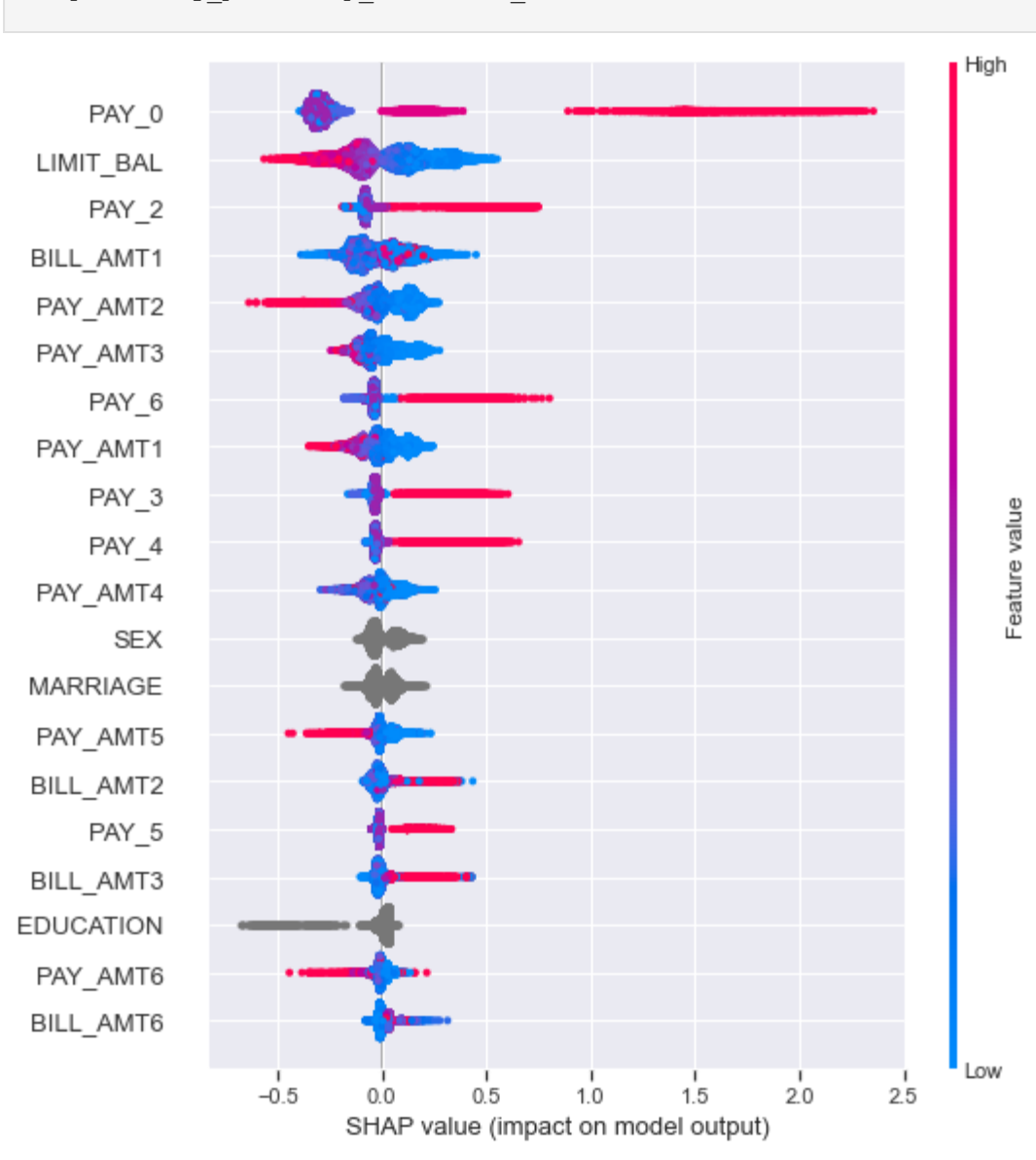
In [138…

```python
import shap
shap.initjs()
shap_values = shap.TreeExplainer(clf.booster_).shap_values(X_train)
len(shap_values),len(shap_values[0])
if len(shap_values) == 2:
    shap_values = shap_values[1]
```

Out[138… `(2, 20100)`

In [139…

```python
shap.summary_plot(shap_values, X_train)
```



From the picture above,we can get these informations:

1. PAY_0 is the most influence feature in this model. If a the payment delayed longer, the higher risk she/he will be default.
2. LIMIT_BAL has negative influence in model. It also make sense, because the limit we made was base on the risk level.
3. BILL_AMT1 has no clear linear relationship with the target.
4. PAY_AMT1 has a negative influence in predicting a client is likely to be default.

4.Conclusion

1. Data Quality

There is no missing value in all the columns , so we don't need to deal with the missing value.

BILL_AMOUT and PAY_AMOUT seems have some outliers in a large positive value. But because it the Money amount, it is likely to be a long tail distribution. So I think it make sense, I don't deal with these "outliers".

2. Features

2.1 AGE

We can see that the clients whose age are lower(<24) and higher(>72) may have a higher default rate.

2.2 SEX

Male i.e. SEX==1 has 40% in the sample, but 25% defaulr rate which is higher than female.

2.3 EDUCATION

We have some unknow values in the Education columns, like value 0,5,6. But this unknowed value only have about 1% in the data and the default rate is quite low. We can just ignore these value. It won't affect the result.

But from the knowing value 1~3, we can find out that the higher the education level, the lower risk.

If we want to get more meaningful variable, we should reorder this columns by using the average of the target like below, but in this case, I don't use the reorder.

2.4 MARRIAGE

"Others" marriage status has a higher risk than the other two. Married status has a higher risk than single.

We has some unkonw value in the MARRIAGE, like the 0. But because the percentage of it is quite low, so I don't deal with the zeros.

2.5 LIMIT_BAL

The relationship between the LIMIT_BAL and the average default rate is a U-shape.

2.6 PAY_0

In general, the higher the PAY_0, the Higher risk.

2.7 BILL_AMT

BILL_AMT don't have a linear relationship with the default.

2.8. PAY_AMT1

The higher pay_amt1, the lower risk.

3. Modeling

3.1 performance

I use LightGBM because it's friendly to categorical features. This model has a 0.82 auc in training data, and 0.78 auc in testing data. For the KS-statistic , it is 48% in training data and 42 in testing data. So we can consider this model has a good performance.

3.2 feature importance and explainability

we can see from the picture above , the most important features in this model are: BILL_AMT1, PAY_0,LIMIT_BAL

From the picture of snap values, we can get these informations:

1. PAY_0 is the most influence feature in this model. If a the payment delayed longer, the higher risk she/he will be default.
2. LIMIT_BAL has negative influence in model. It also make sense, because the limit we made was base on the risk level.
3. BILL_AMT1 has no clear linear relationship with the target.
4. PAY_AMT1 has a negative influence in predicting a client is likely to be default.

In [ ]: