分工

刘晓旭 数据预处理与特征工程

刘承杰 模型选择与训练

赵凝晖 数据收集

徐晨 模型应用

〇、步骤示例

1.数据收集（15’）

合并选取字段并给出选取每个字段的理由5’💔

数据盘点5‘💔

可视化5‘（进阶）💔

2.数据预处理（25‘）

数据标准化5‘💔

异常值处理5‘💔

缺失值处理5’💔

缺失值选用机器学习模型处理3‘（进阶）💔

异常值处理选用合适算法3‘（进阶）💔

使用除基本分外的方法进行数据预处理并给出理由4’（进阶）💔

3.特征工程（20’）

相关性，多重共线性。对于每一个特征的剔除，有详细的说明理由15‘💔

使用除基本分外的方法进行特征工程并给出理由5’💔

4.特征选择（0‘）

5.模型选择（15’）

至少选择一种模型进行拟合10‘💔

使用多于一种模型并对不同模型进行评估，或者使用多种模型加权的方式进行评估5’（进阶）💔

6.模型训练（0‘）

7.模型评估（15’）

准确率、精确率、召回率10’💔

F1，Cohen‘s Kappa等，选择合适的模型5‘（进阶）💔

8.模型应用（10‘）

提供完整的结果数据10’💔

一、数据收集15'

1.选择字段2.5'

初步选取以下数据：

信用等级 credit_info(3)

uid 用户uid，证件号码，数据类型varchar

credit_level 信用等级

客户星级 star_info(9)

uid 用户uid，证件号码，数据类型varchar

star_level 客户星级

基本信息 base_info(6)

uid 用户uid，证件号码，数据类型varchar

sex 性别，数据类型varchar

marrige 婚姻状况，数据类型varchar

education 教育程度，数据类型varchar

is_black 是否黑名单，数据类型varchar

is_contact 是否关联人，数据类型varchar

存款汇总信息(5)

uid 用户uid，证件号码，数据类型varchar

all_bal 总余额，数据类型Long

avg_mth 月日均余额，数据类型Long

avg_qur 季度日均余额，数据类型Long

avg_year 年日均余额，数据类型Long

sa_bal 活期余额，数据类型Long

td_bal 定期余额，数据类型Long

fin_bal 理财余额，数据类型Long

sa_crd_bal 卡活期余额，数据类型Long

td_crd_bal 卡内定期余额，数据类型Long

sa_td_bal 定活两便，数据类型Long

ntc_bal 通知存款，数据类型Long

td_3m_bal 定期3个月，数据类型Long

td_6m_bal 定期6个月，数据类型Long

td_1y_bal 定期1年，数据类型Long

td_2y_bal 定期2年，数据类型Long

td_3y_bal 定期3年，数据类型Long

td_5y_bal 定期5年，数据类型Long

oth_td_bal 定期其它余额，数据类型Long

cd_bal 大额存单余额，数据类型Long

存款账号信息(4)

uid 用户uid，证件号码，数据类型varchar

term 存款期限，数据类型varchar

deps_type 存款种类，数据类型varchar

is_secu_card 是否社保卡，数据类型varchar

acct_sts 账户状态，数据类型varchar

frz_sts 冻结状态，数据类型varchar

stp_sts 止付状态，数据类型varchar

acct_bal 原币余额，数据类型varchar

bal 余额，数据类型Long

avg_mth 月日均，数据类型Long

avg_qur 季度日均，数据类型Long

avg_year 年日均，数据类型Long

贷记卡开户(1)

uid 用户uid，证件号码，数据类型varchar

card_sts 卡片状态，数据类型varchar

is_withdrw 是否开通取现功能，数据类型varchar

is_transfer 是否开通转账功能，数据类型varchar

is_deposit 是否开通存款功能，数据类型varchar

is_purchase 是否开通消费功能，数据类型varchar

cred_limit 信用额度，数据类型Long

deposit 贷记卡存款，数据类型Long

over_draft 普通额度透支，数据类型Long

dlay_amt 逾期金额，数据类型Long

five_class 五级分类，数据类型Long

bankacct_bal 还款账号余额，数据类型Long

贷记卡分期付款(2)

uid 用户uid，证件号码，数据类型varchar

mp_type 分期类型，数据类型varchar

mp_status 分期付款状态，数据类型varchar

贷款汇总信息(8)

uid 用户uid，证件号码，数据类型varchar

all_bal 总余额，数据类型Long

bad_bal 不良余额，数据类型Long

due_intr 欠息余额，数据类型Long

norm_bal 正常余额，数据类型Long

delay_bal 逾期余额，数据类型Long

贷款账号信息(7)

uid 用户uid，证件号码，数据类型varchar

loan_amt 贷款金额，数据类型Long

loan_bal 贷款余额，数据类型Long

overdue_flag 逾期标志，数据类型varchar

owed_int_flag 欠息标志，数据类型varchar

owed_int_in 表内欠息金额，数据类型Long

owed_int_out 表外欠息金额，数据类型Long

delay_bal 逾期金额，数据类型Long

在星级评定中，使用**客户星级，基本信息，存款汇总信息，存款账号信息，贷记卡开户，贷记卡分期付款**六项，按照uid拥有的项目灵活分配。

在信用评定中，使用**信用等级，基本信息，贷记卡开户，贷记卡分期，贷款汇总信息，贷款账号信息**六项，按照uid拥有的项目灵活分配。

2.配置环境0’

起始步骤使用MySQL环境，因为它可以更好地支持表的关联。

部分环节使用mongo DB环境，因为它可以更好地适应流式数据。

其中，mongo配置方法如下：

```
1.安装mongoDB
2.设置系统环境变量
3.写入config文件
4.启动服务
```

3.合并相关数据2.5'

3.1删除无用字段

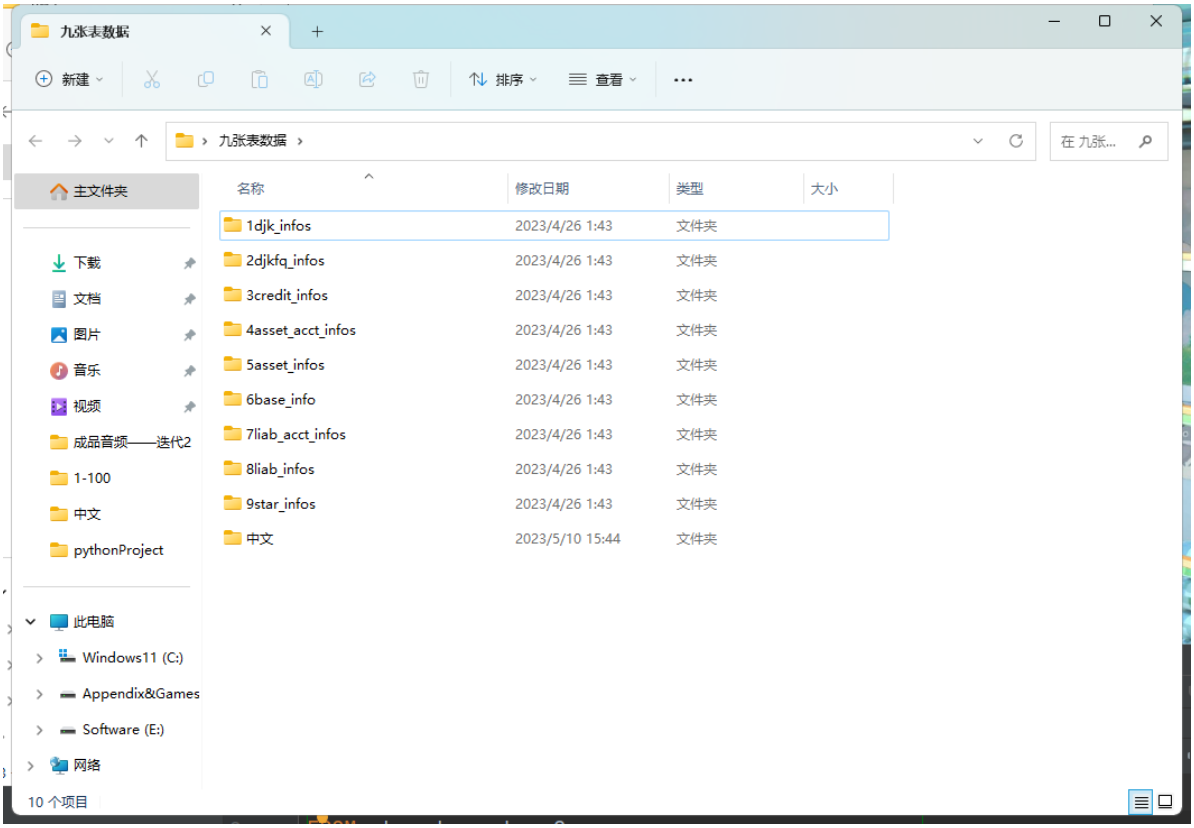
合并前，对选择字段步骤中一些实际上不利于模型训练的加以删除。新的字段选择如下：

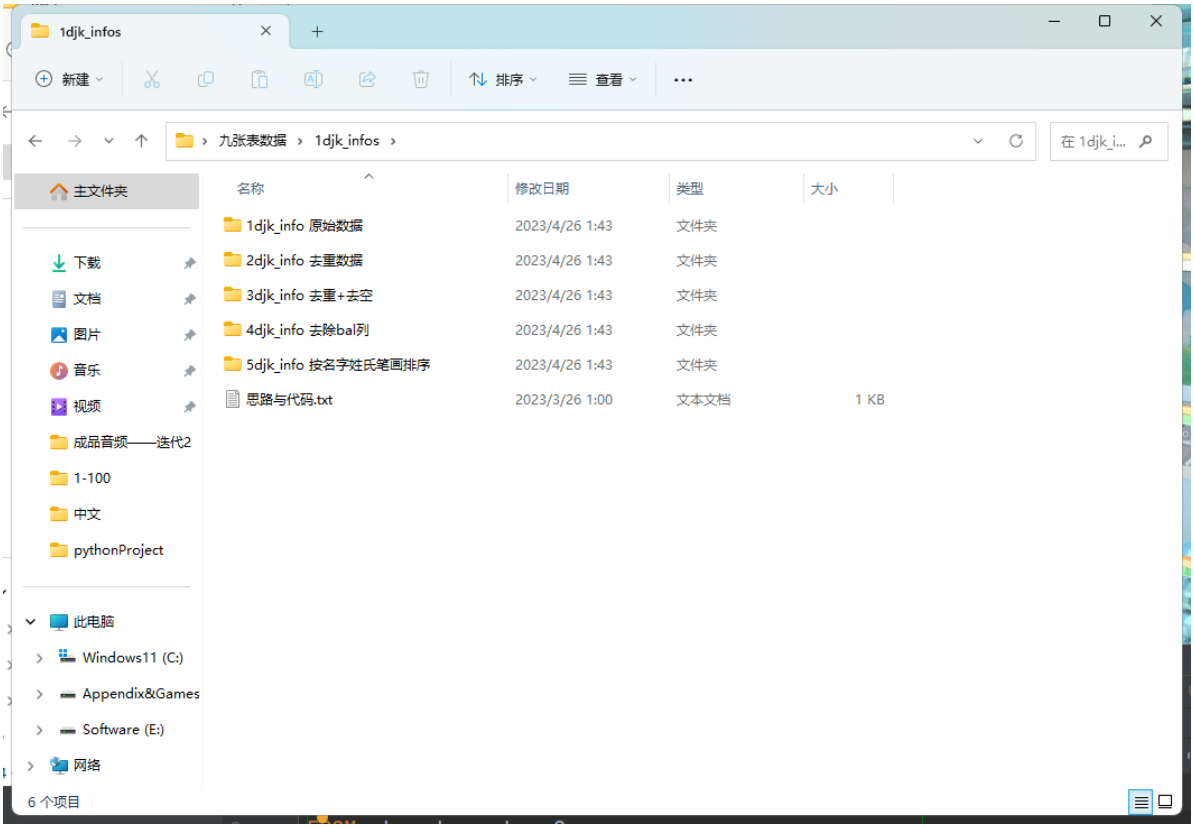
在星级评定中，使用**客户星级，基本信息，存款汇总信息，贷记卡开户**；在信用评定中，使用**信用等级，基本信息，贷记卡开户，贷款汇总信息。**

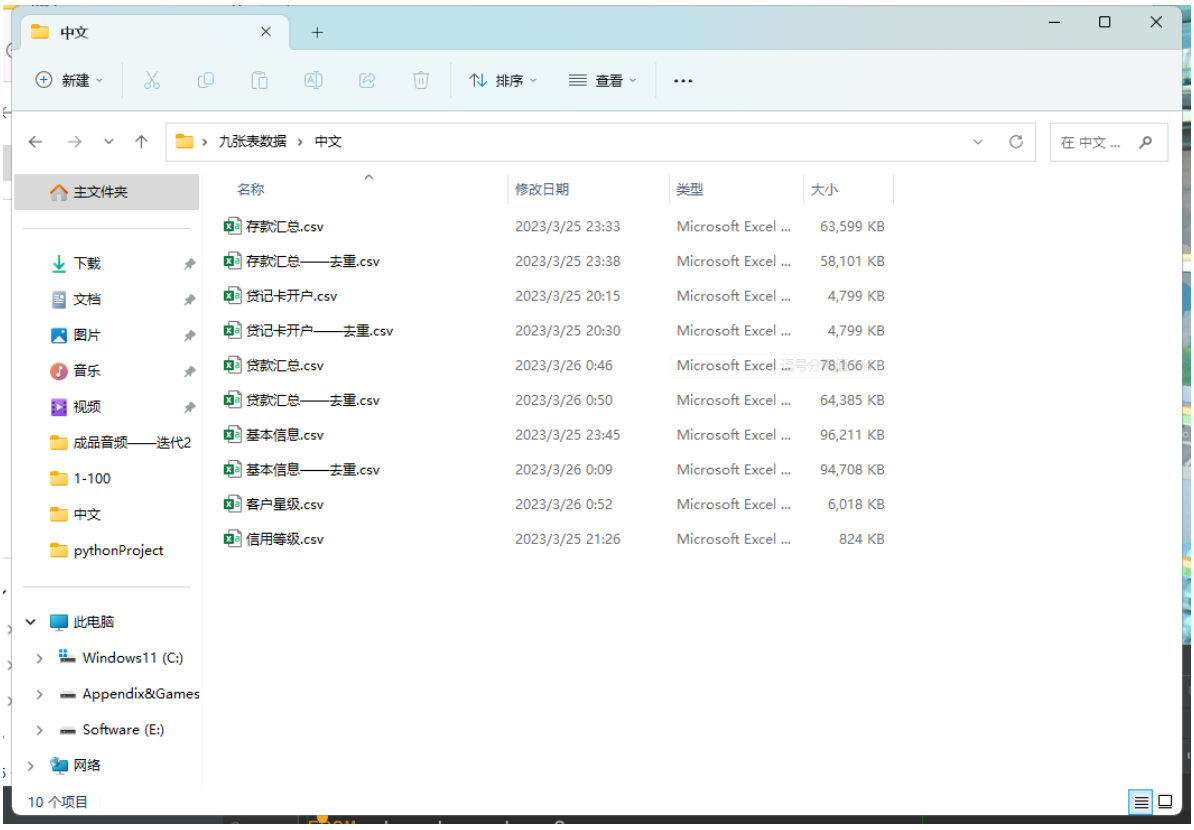
其中，存款帐号信息，贷款账号信息存在冗余，且贷款账号信息数据量过大；贷记卡分期所含数据过少（约400条），不具有判断性。

3.2分离清洗

将数据从hive中分离出来，并进行去重和去空操作。







3.3合并

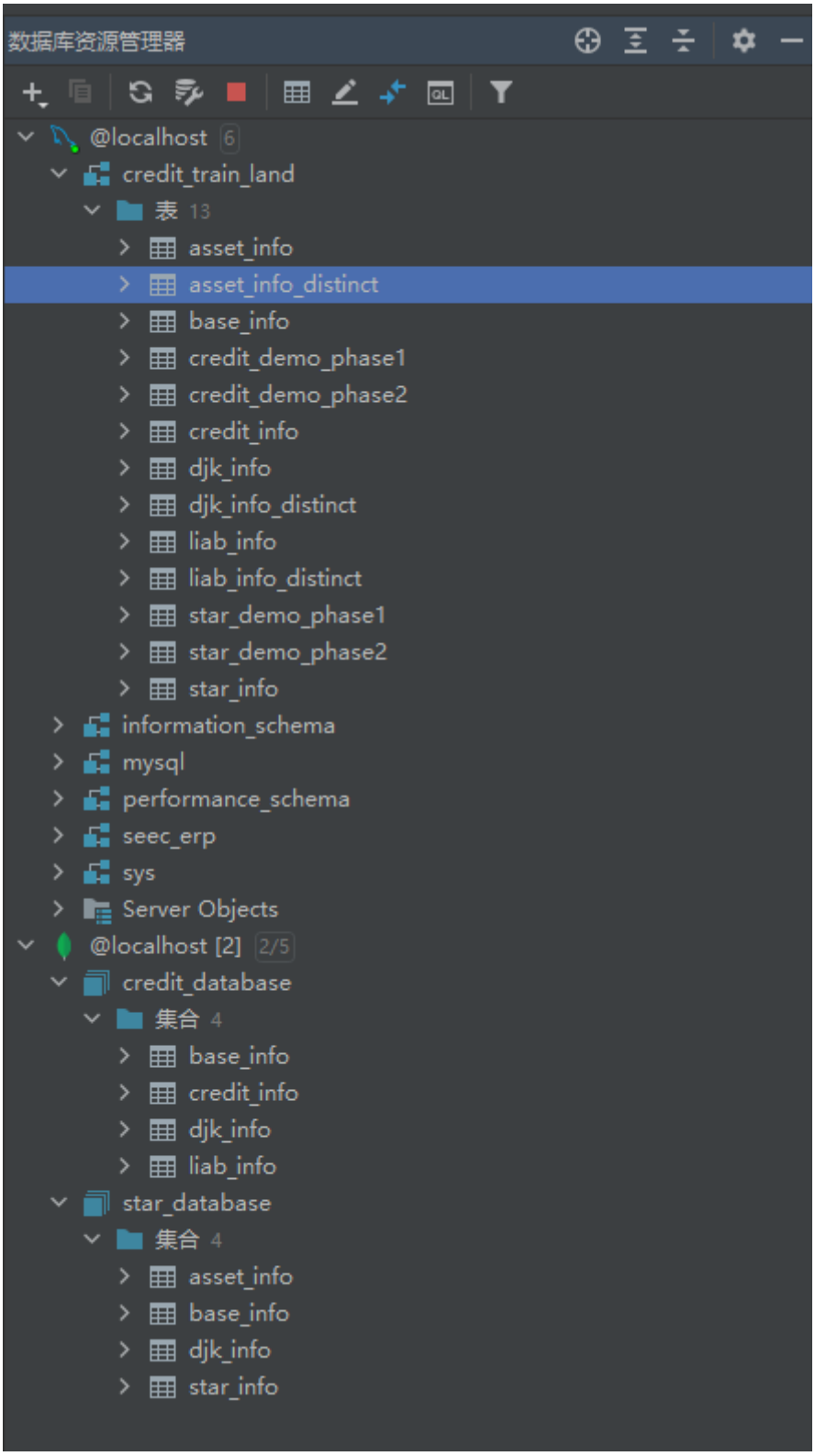
1.使用mysql/mongo将六张表写入Sheet/Collection。

```
import pymysql as mysql
import csv
import pandas as pd
from collections import namedtuple
```

```
def main():
    df = pd.read_csv('存款汇总--去重.csv')
    df = df.fillna(value='None')
    mydb = mysql.connect(
        host="localhost",
        user="root",
        password="root",
        port=3306,
        database="credit_train_land"
    )

    cursor = mydb.cursor()

    for index, row in df.iterrows():
        sql = "INSERT INTO asset_info (uid, all_bal, avg_mth, avg_qur, avg_year,
sa_bal, td_bal, fin_bal, sa_crd_bal, td_crd_bal, sa_td_bal, ntc_bal, td_3m_bal,
td_6m_bal, td_1y_bal, td_2y_bal, td_3y_bal, td_5y_bal, oth_td_bal, cd_bal) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s)"
        val = (row['uid'], row['all_bal'], row['avg_mth'], row['avg_qur'],
row['avg_year'], row['sa_bal'], row['td_bal'], row['fin_bal'], row['sa_crd_bal'],
row['td_crd_bal'], row['sa_td_bal'], row['ntc_bal'], row['td_3m_bal'],
row['td_6m_bal'], row['td_1y_bal'], row['td_2y_bal'], row['td_3y_bal'],
row['td_5y_bal'], row['oth_td_bal'], row['cd_bal'])
        cursor.execute(sql, val)

    mydb.commit()


if __name__ == '__main__':
    main()
```

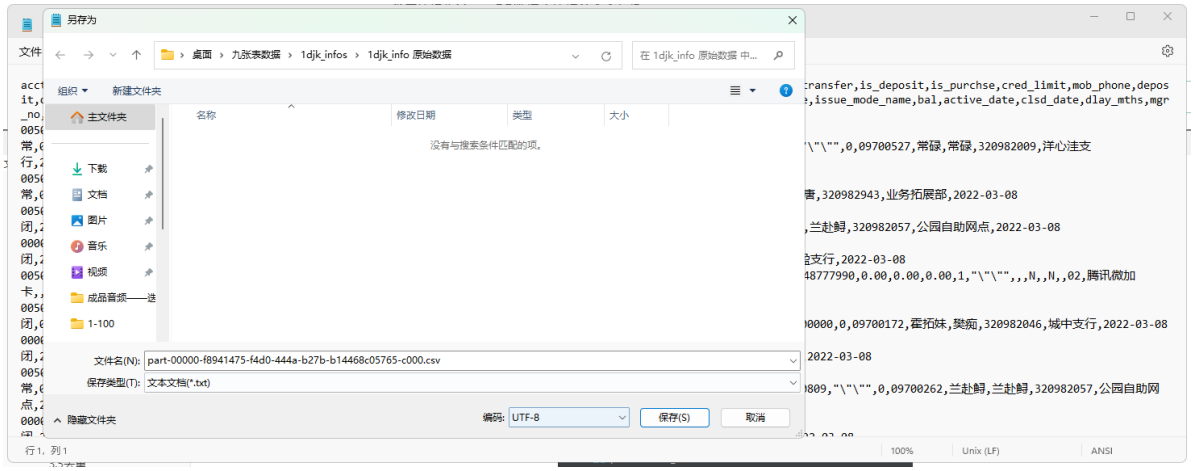
```
    for index, row in df.iterrows():
        sql = "INSERT INTO base_info (uid, sex, marriage, education, is_black,
is_contact) VALUES (%s, %s, %s, %s, %s, %s)"
        val = (row['uid'], row['sex'], row['marrige'], row['education'],
row['is_black'], row['is_contact'])
        cursor.execute(sql, val)
    for index, row in df.iterrows():
        sql = "INSERT INTO credit_info (uid, credit_level) VALUES (%s, %s)"
        val = (row['uid'], row['credit_level'])
        cursor.execute(sql, val)
df = df.fillna(value='None')
    for index, row in df.iterrows():
        sql = "INSERT INTO djk_info (uid, card_sts, is_withdrw, is_transfer,
is_deposit, is_purchase, cred_limit, deposit, over_draft, dlay_amt, five_class,
bankacct_bal) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
        val = (row['uid'], row['card_sts'], row['is_withdrw'],
row['is_transfer'], row['is_deposit'], row['is_purchse'], row['cred_limit'],
row['deposit'], row['over_draft'], row['dlay_amt'], row['five_class'],
row['bankacct_bal'])
        cursor.execute(sql, val)
df = df.fillna(value='None')
    for index, row in df.iterrows():
        sql = "INSERT INTO liab_info (uid, all_bal, bad_bal, due_intr, norm_bal,
delay_bal) VALUES (%s, %s, %s, %s, %s, %s)"
```

```
        val = (row['uid'], row['all_bal'], row['bad_bal'], row['due_intr'],
row['norm_bal'], row['delay_bal'])
        cursor.execute(sql, val)
    for index, row in df.iterrows():
        sql = "INSERT INTO star_info (uid, star_level) VALUES (%s, %s)"
        val = (row['uid'], row['star_level'])
        cursor.execute(sql, val)
df = df.fillna(value='None')
    for index, row in df.iterrows():
        sql = "INSERT INTO asset_info (uid, all_bal, avg_mth, avg_qur, avg_year,
sa_bal, td_bal, fin_bal, sa_crd_bal, td_crd_bal, sa_td_bal, ntc_bal, td_3m_bal,
td_6m_bal, td_1y_bal, td_2y_bal, td_3y_bal, td_5y_bal, oth_td_bal, cd_bal) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s)"
        val = (row['uid'], row['all_bal'], row['avg_mth'], row['avg_qur'],
row['avg_year'], row['sa_bal'], row['td_bal'], row['fin_bal'], row['sa_crd_bal'],
row['td_crd_bal'], row['sa_td_bal'], row['ntc_bal'], row['td_3m_bal'],
row['td_6m_bal'], row['td_1y_bal'], row['td_2y_bal'], row['td_3y_bal'],
row['td_5y_bal'], row['oth_td_bal'], row['cd_bal'])
        cursor.execute(sql, val)
```



2.记录：gbk问题

在使用python脚本将文件写入之前，需要对从hive中抽取的csv进行转换，转换至UTF8格式。（显示GBK错误，但其实是ANSI格式）



3.记录：permission denied问题

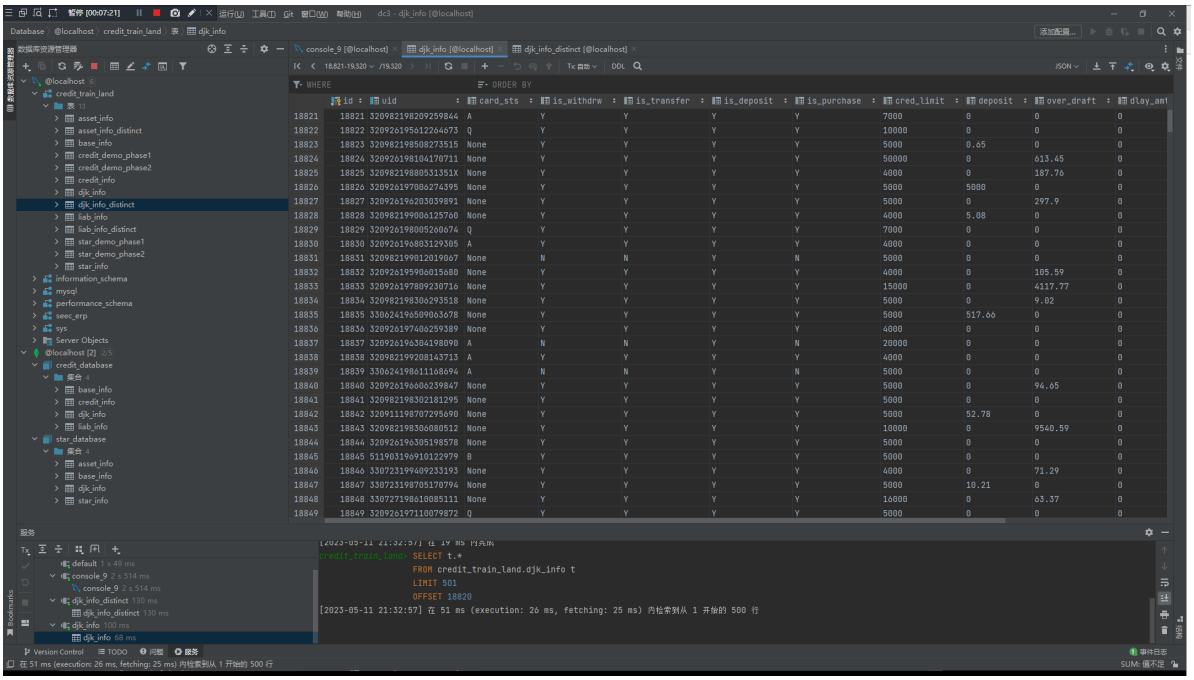
这个很简单，在写入的同时要避免读取查看数据，避免死锁。

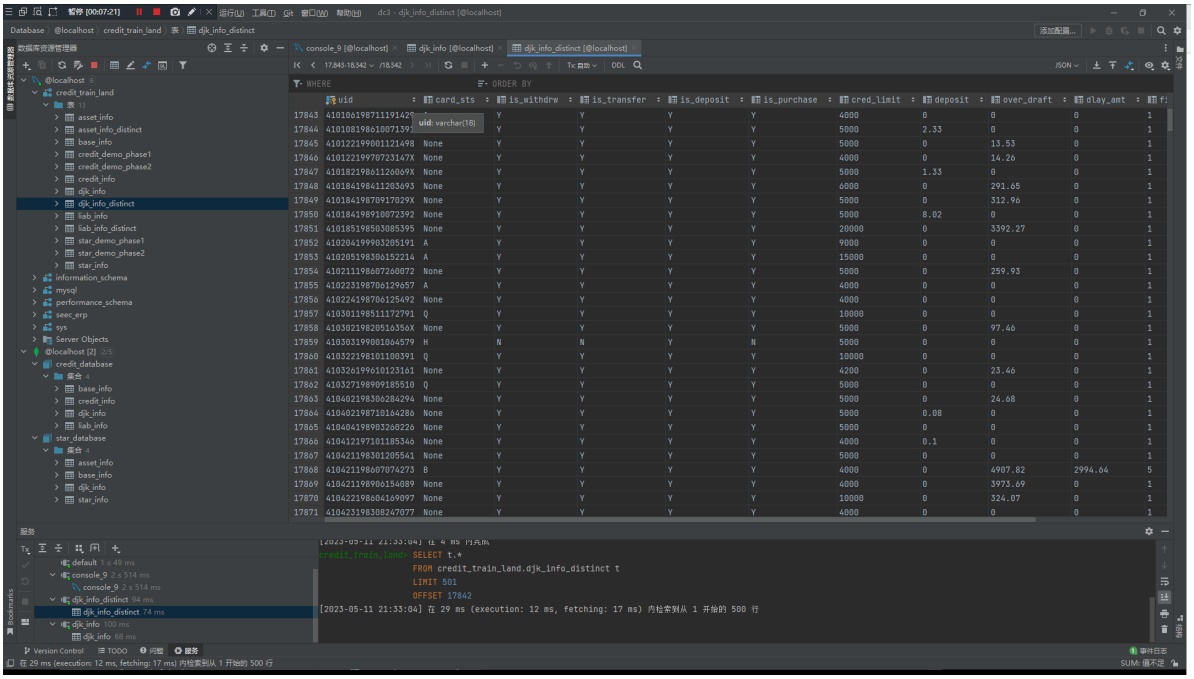
4.记录：非系统数据库问题

mongoDB数据库不能在admin，config等系统数据库进行聚合操作，需要转移到其它名称的非系统数据库。

5.去重

这一步是为了去除冗余数据。





6.连接性能优化

①使用索引。

②在不影响结果的情况下，小表连接大表。

7.合并过程

然后根据前面字段的选择将多个表进行合并——先剔除不需要的列，再合并。

```
INSERT INTO credit_demo_phase2 (uid, credit_level, sex, marriage, education,
is_black, is_contact, card_sts, is_withdrw, is_transfer, is_deposit, is_purchase,
cred_limit, deposit, over_draft, dlay_amt, five_class, bankacct_bal, all_bal,
bad_bal, due_intr, norm_bal, delay_bal)
SELECT
c.uid,c.credit_level,c.sex,c.marriage,c.education,c.is_black,c.is_contact,d.card
_sts,d.is_withdrw,d.is_transfer,d.is_deposit,d.is_purchase,d.cred_limit,d.deposi
t,d.over_draft,d.dlay_amt,d.five_class,d.bankacct_bal,l.all_bal,l.bad_bal,l.due_
intr,l.norm_bal,l.delay_bal from credit_demo_phase1 c
left join djk_info_distinct d on c.uid = d.uid
left join liab_info_distinct l on d.uid = l.uid
INSERT INTO star_demo_phase2(uid, star_level, sex, marriage, education, is_black,
is_contact, all_bal, avg_mth, avg_qur, avg_year, sa_bal, td_bal, fin_bal,
sa_crd_bal, td_crd_bal, sa_td_bal, ntc_bal, td_3m_bal, td_6m_bal, td_1y_bal,
td_2y_bal, td_3y_bal, td_5y_bal, oth_td_bal, cd_bal, card_sts, is_withdrw,
is_transfer, is_deposit, is_purchase, cred_limit, deposit, over_draft, dlay_amt,
five_class, bankacct_bal)
SELECT
s.uid,s.star_level,s.sex,s.marriage,s.education,s.is_black,s.is_contact,a.all_ba
l,a.avg_mth,a.avg_qur,a.avg_year,a.sa_bal,a.td_bal,a.fin_bal,a.sa_crd_bal,a.td_c
rd_bal,a.sa_td_bal,a.ntc_bal,a.td_3m_bal,a.td_6m_bal,a.td_1y_bal,a.td_2y_bal,a.t
d_3y_bal,a.td_5y_bal,a.oth_td_bal,a.cd_bal,d.card_sts,d.is_withdrw,d.is_transfer
,d.is_deposit,d.is_purchase,d.cred_limit,d.deposit,d.over_draft,d.dlay_amt,d.fiv
e_class,d.bankacct_bal from star_demo_phase1 s
left join asset_info_distinct a on s.uid = a.uid
left join djk_info_distinct d on s.uid = d.uid
```

8.其它代码

```
//转mongoDB
# p1 2mongo
import pymongo
import csv

if __name__ == '__main__':
    client = pymongo.MongoClient("mongodb://localhost:27017/admin")
    db = client["star_database"]
    collection = db["star_info"]

    with open('客户星级.csv', 'r', encoding='UTF-8') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            collection.insert_one(row)
```

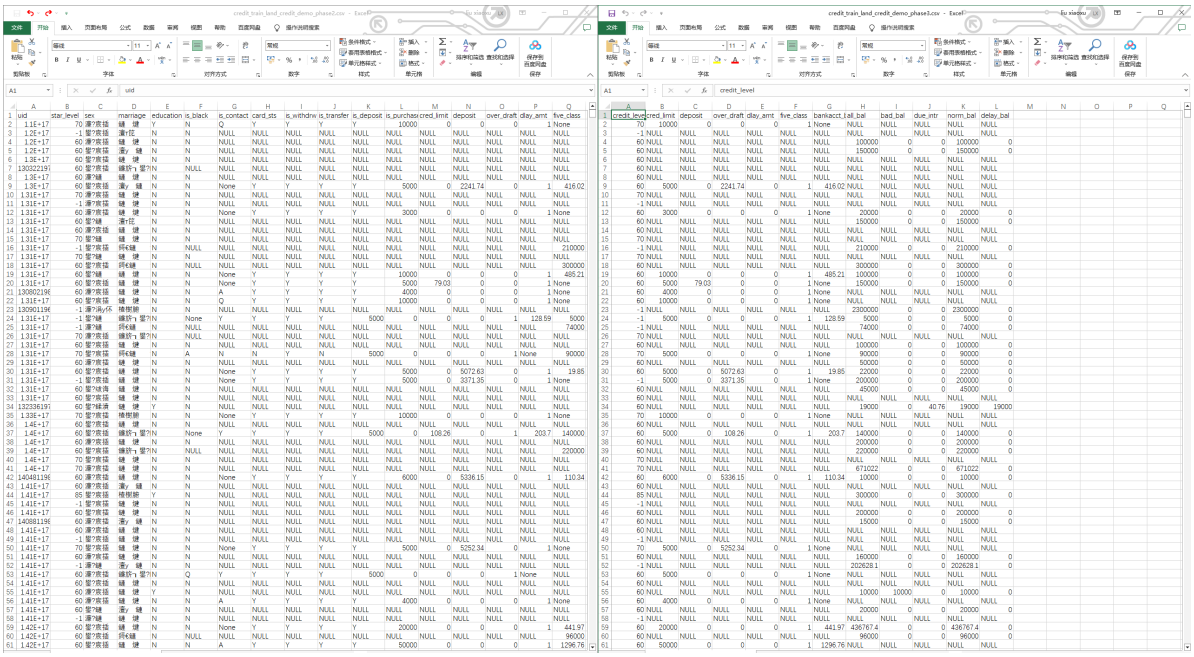
```
//转mysql
# p2 2mysql
import pymysql as mysql
import pandas as pd


def main():
    df = pd.read_csv('存款汇总--去重.csv')
    df = df.fillna(value='None')
    mydb = mysql.connect(
        host="localhost",
        user="root",
        password="root",
        port=3306,
        database="credit_train_land"
    )

    cursor = mydb.cursor()

    for index, row in df.iterrows():
        sql = "INSERT INTO asset_info (uid, all_bal, avg_mth, avg_qur, avg_year,
sa_bal, td_bal, fin_bal, sa_crd_bal, td_crd_bal, sa_td_bal, ntc_bal, td_3m_bal,
td_6m_bal, td_1y_bal, td_2y_bal, td_3y_bal, td_5y_bal, oth_td_bal, cd_bal) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s)"
        val = (row['uid'], row['all_bal'], row['avg_mth'], row['avg_qur'],
row['avg_year'], row['sa_bal'], row['td_bal'], row['fin_bal'], row['sa_crd_bal'],
row['td_crd_bal'], row['sa_td_bal'], row['ntc_bal'], row['td_3m_bal'],
row['td_6m_bal'], row['td_1y_bal'], row['td_2y_bal'], row['td_3y_bal'],
row['td_5y_bal'], row['oth_td_bal'], row['cd_bal'])
        cursor.execute(sql, val)

    mydb.commit()


if __name__ == '__main__':
    main()
```

4.数据盘点及其可视化10'（含扩展5'）

4.1数据盘点方法

0.准备工作

将数据库中的表选取可盘点列导出。



1.分位数

分位数解析直接调用pandas库。

```
#例子
import pandas as pd


def main():
    star_train = pd.read_csv('客户星级.csv')
    star_train.describe().to_csv('star_train_describe.csv')


if __name__ == '__main__':
    main()
```

2.频数

使用python生成可视化图表。

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


def main():
    df = pd.read_csv('credit_train_land_star_demo_phase3.csv')

    for col in df.columns:
        # 去除NaN
        col_data = df[col].dropna()
```

```
        # 去除无穷量
        if np.isinf(col_data).sum() > 0:
            col_data = col_data.replace([np.inf, -np.inf], np.nan).dropna()

        # 分别计算频数，分箱
        if col == 'sa_td_bal':
            freq, bins = np.histogram(col_data, bins=[0, 10, 100, 1000, 10000,
100000, 1000000, 10000000])
        else:
            continue

        # 绘制直方图
        plt.bar(range(len(freq)), freq)

        # 添加标注
        for i, v in enumerate(freq):
            plt.annotate(str(v), xy=(i, v), ha='center', va='bottom')

        # 设置X轴标签
        plt.xticks(range(len(bins) - 1), bins[:-1])

        plt.title('Histogram of {}'.format(col))
        plt.xlabel('Value')
        plt.ylabel('Frequency')
        plt.savefig('star_hist_{}.png'.format(col))


if __name__ == '__main__':
    main()
```

4.2分位数

得分-1的项不去除

4.2.1用户星级

	star_level	all_bal	avg_mth	avg_qur	avg_year	sa_bal	td_bal	fin_bal	sa_crd_bal	td_crd_bal	sa_td_bal	ntc_bal	td_3m_bal	td_6m_bal	td_1y_bal	td_2y_bal	td_3y_bal	td_5y_bal	oth_td_bal	cd_bal	cred_limit	deposit	over_draft	dlay_amt	five_class	bankacct_bal
count	290638	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	290642	15069	15069	15069	15069	15069	7992
mean	1.275843	34943.45	4149.799	0	34140.84	5007.748	29340.28	592.0686	4864.773	0	153.1267	245.4022	293.2378	327.2335	8417.631	1505.263	17834.43	567.0502	0	4375.004	9175.918	114.5343	933.7644	257.3518	1.103059	10683.36
std	1.555482	133668.3	17317.41	0	148232.5	30861.05	125162.4	16393.32	30302.09	0	6246.303	12390.28	11042.61	7566.962	38418.71	12170.4	91515.92	47139.45	0	93432.94	17917.16	4254.206	4594.979	3397.588	0.616178	49398.86
min	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
25%	1	27.66	3.84	0	73.44	11.0725	0	0	4.3	0	0	0	0	0	0	0	0	0	0	0	4000	0	0	0	1	7.3075
50%	1	727.485	92.54	0	1228.72	326.77	0	0	280.135	0	0	0	0	0	0	0	0	0	0	0	5000	0	0	0	1	369.375
75%	2	17913.85	2138.678	0	19409.7	2078.625	1000	0	1931.15	0	0	0	0	0	0	0	0	0	0	0	8000	0	50	0	1	4156.085
max	9	25320222	4785215	0	45972627	3341648	24000000	2000000	3341648	0	2550000	2135598	3000000	990000	5285800	700000	18000000	20000000	0	24000000	500000	390477.6	239674.8	239636.2	5	1678230

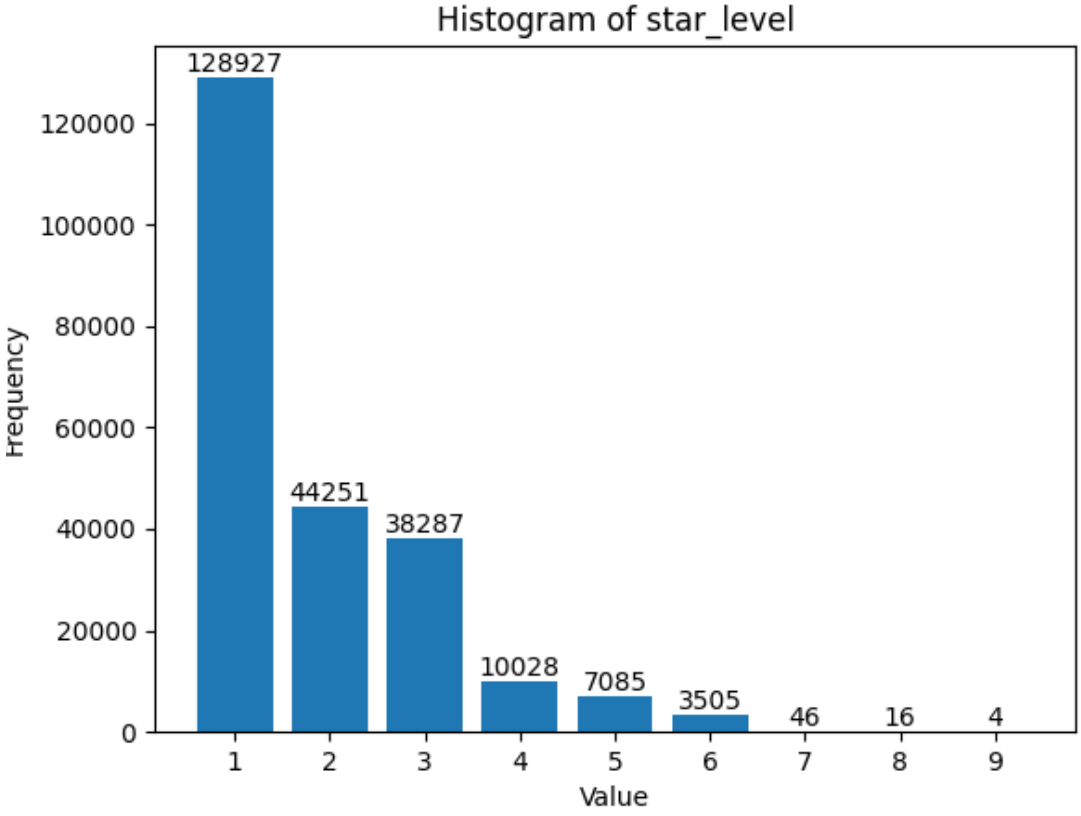
4.2.2信用等级

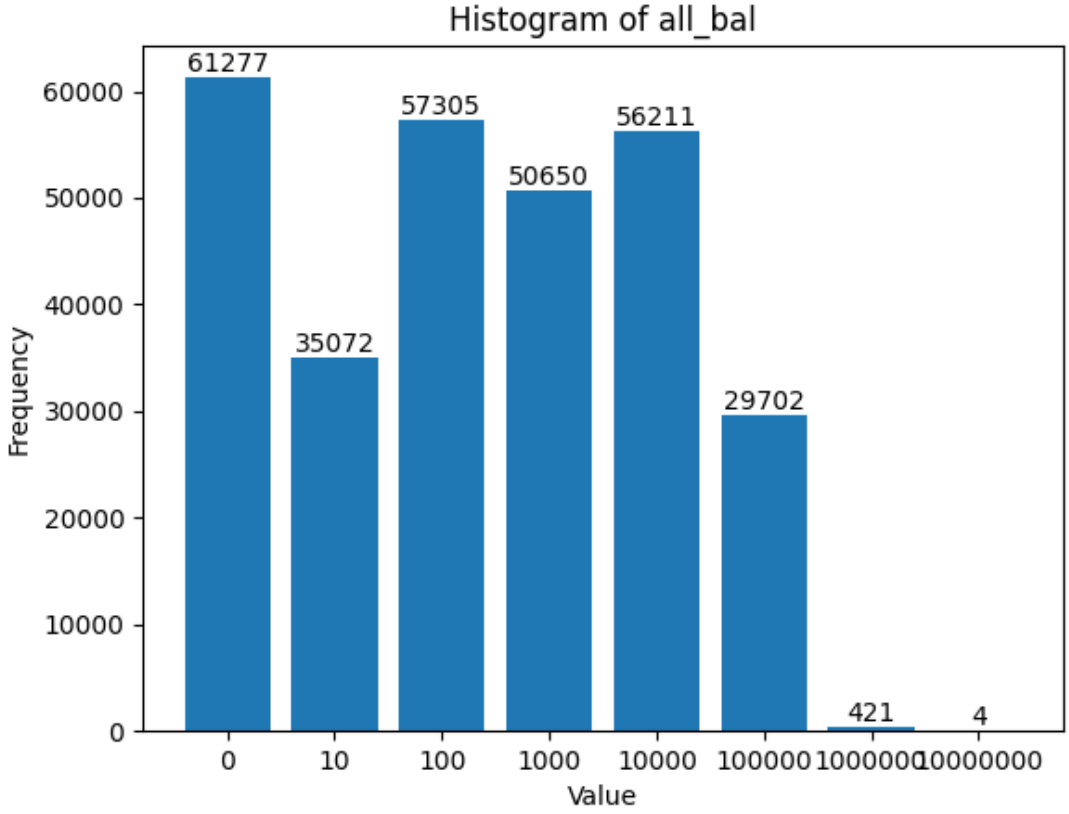
	star_level	cred_limit	deposit	over_draft	dlay_amt	five_class	bankacct_b	all_bal	bad_bal	due_intr	norm_bal	delay_bal
count	38309	6379	6379	6379	6379	6379	3377	15238	15238	15238	15238	15238
mean	52.01566	10059.3	186.2554	1289.377	333.6281	1.102681	12119.72	223946.2	13108.18	2697.85	222645.7	8374.476
std	27.84236	18332.7	6332.774	5852.446	4286.557	0.609034	44479.02	487257.3	123710.3	31924.08	487051.2	99641.38
min	-1	0	0	0	0	1	0	0	0	0	0	0
25%	60	4000	0	0	0	1	28.05	50000	0	0	50000	0
50%	60	5000	0	0	0	1	693.39	116776.1	0	0	111973.3	0
75%	70	10000	0	95.915	0	1	6814.32	290000	0	0	289988.8	0
max	85	500000	390477.6	239674.8	239636.2	5	1410537	20000000	7497978	1805879	20000000	7497978

4.3频数

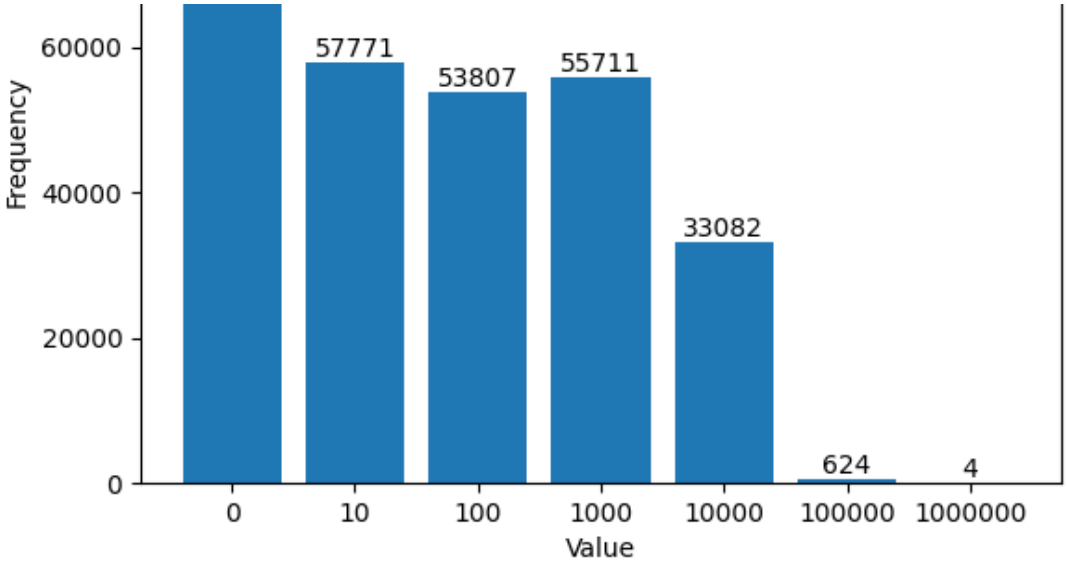
得分-1的项不去除

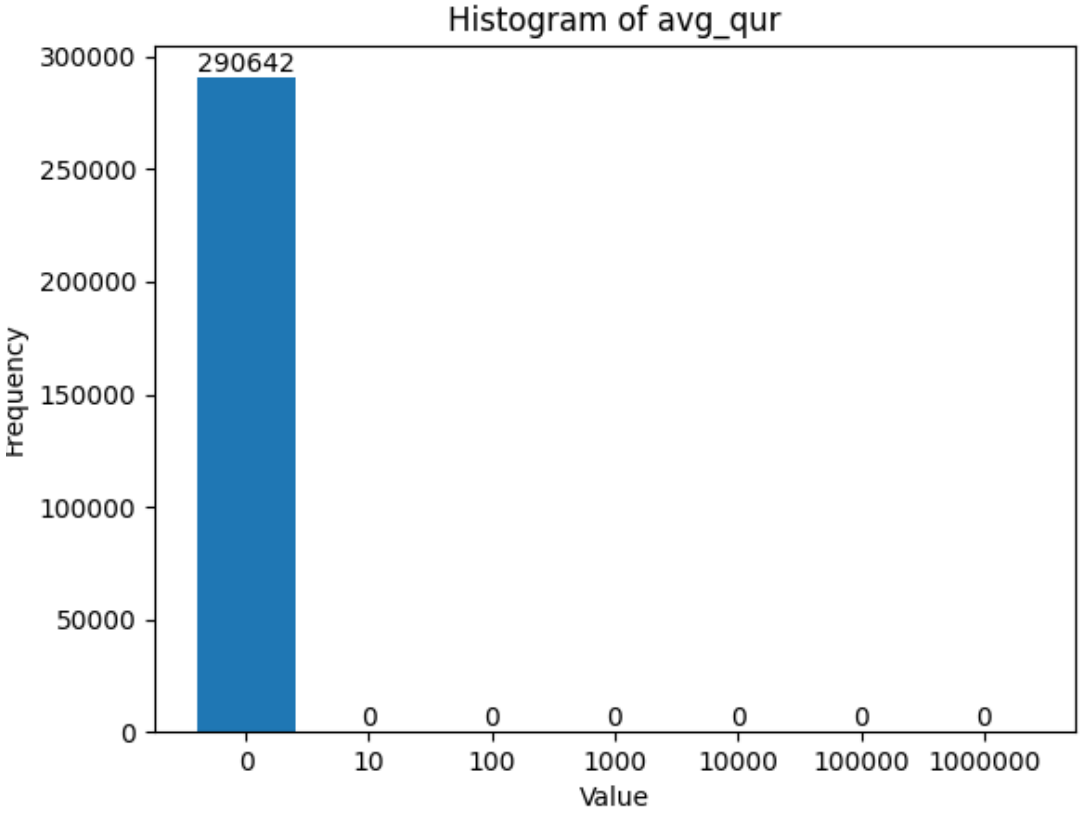
4.3.1用户星级

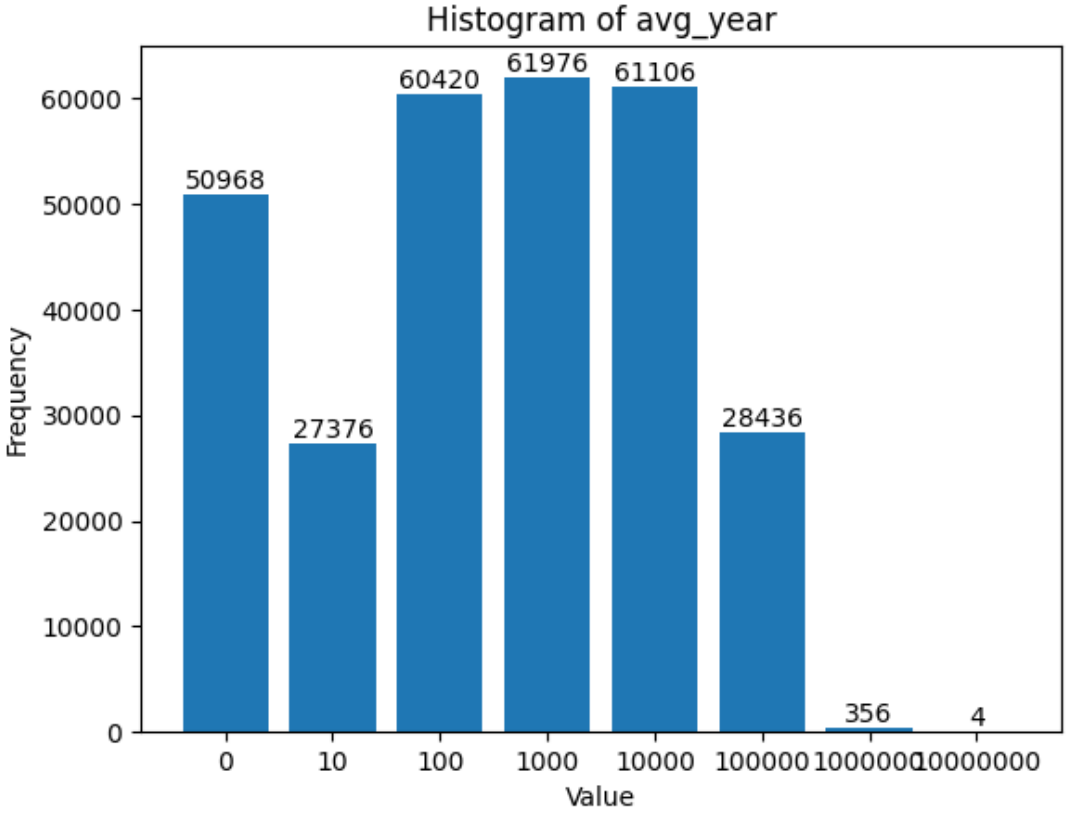

Histogram of star_level
Frequency
Value
128927
44251
38287
10028
7085
3505
46
16
4
0
20000
40000
60000
80000
100000
120000
1
2
3
4
5
6
7
8
9

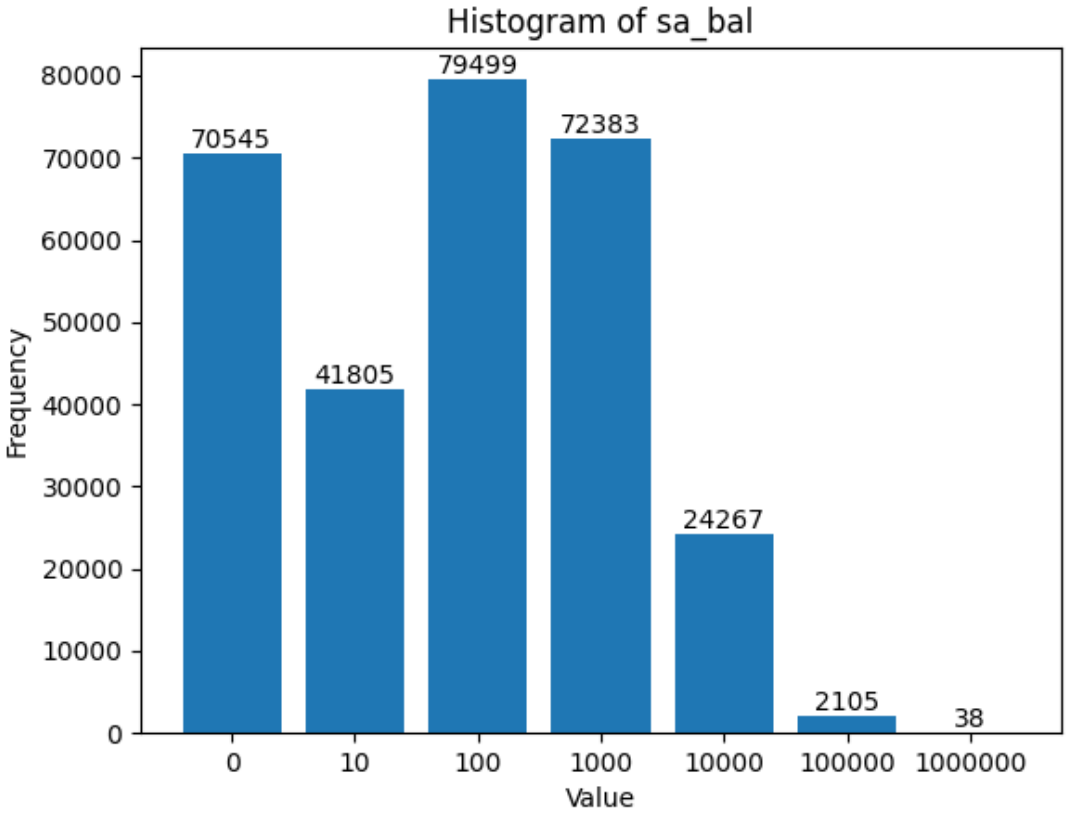

Histogram of all_bal
Frequency
Value
61277
35072
57305
50650
56211
29702
421
4
0
10000
20000
30000
40000
50000
60000
0
10
100
1000
10000
100000
1000000
10000000

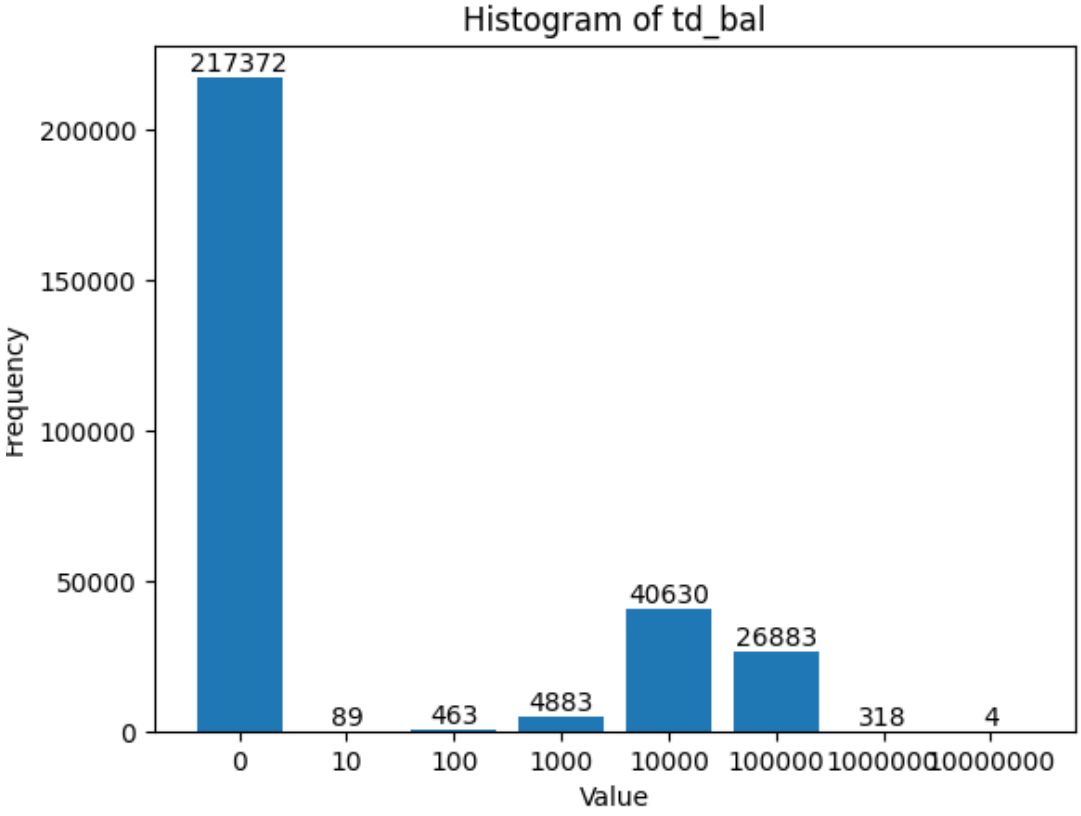

Histogram of avg_mth
89643
80000

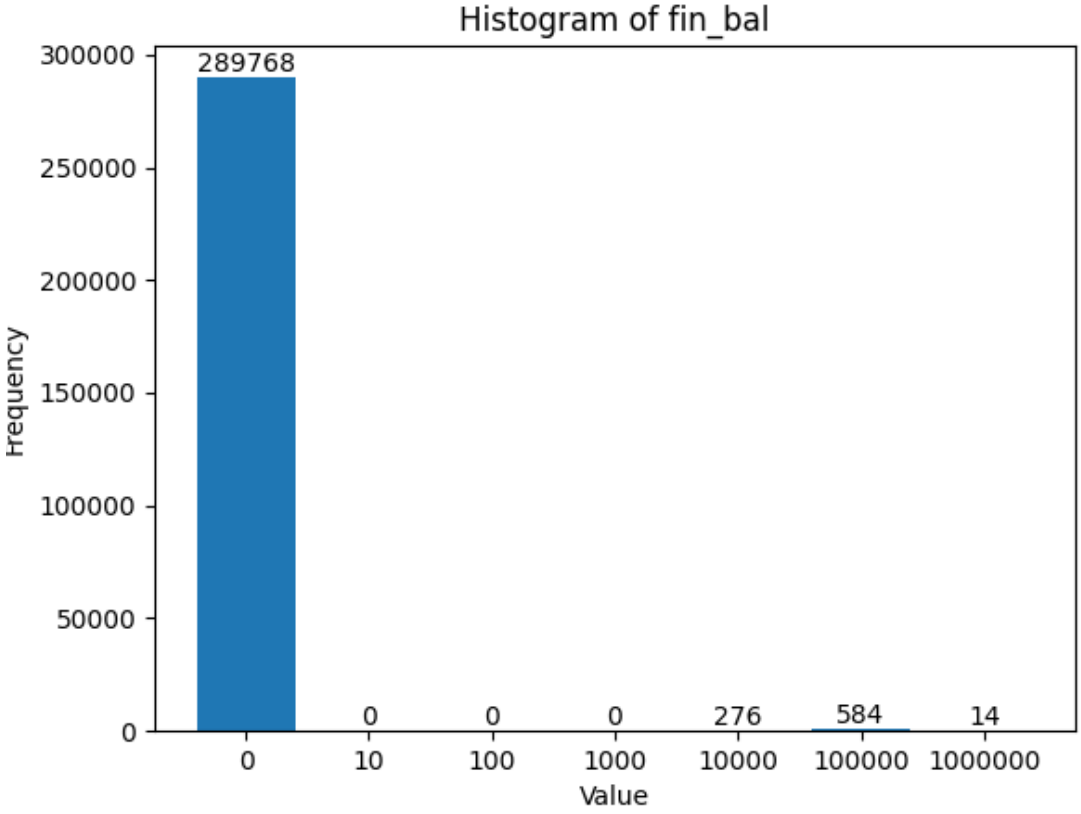

60000
40000
20000
0
Frequency
57771
53807
55711
33082
624
4
0
10
100
1000
10000
100000
1000000
Value

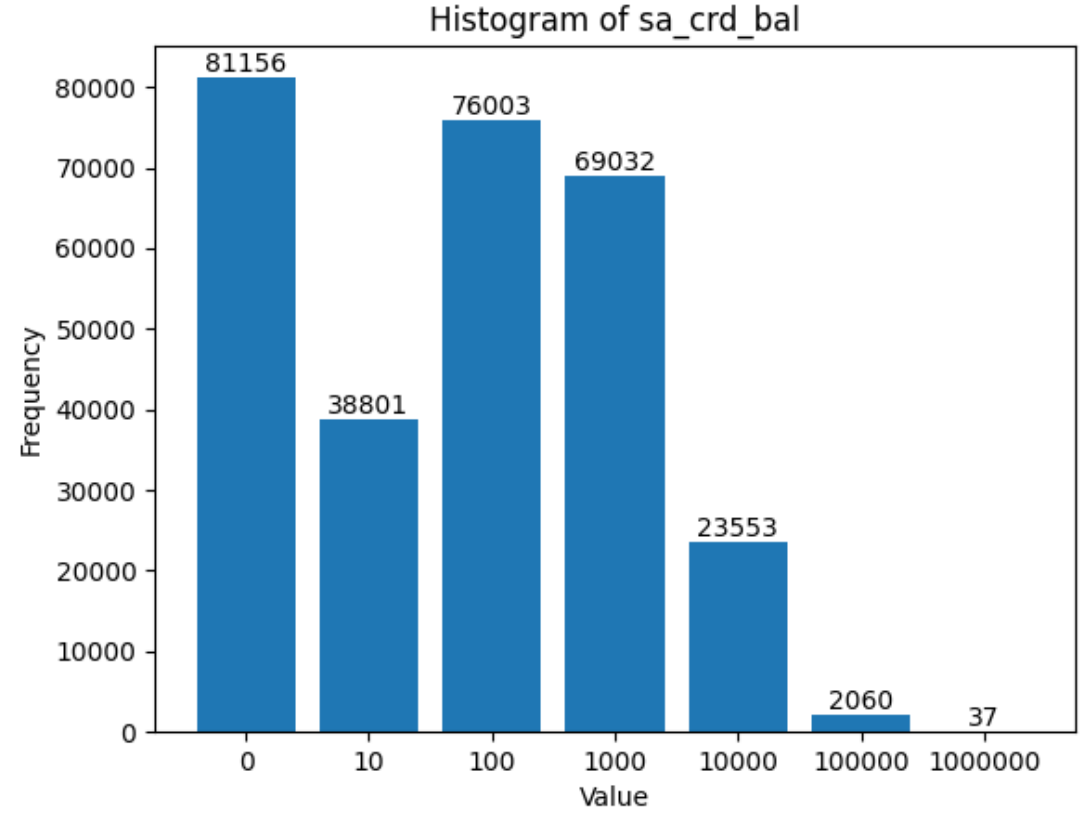

Histogram of avg_qur
300000
250000
200000
150000
100000
50000
0
Frequency
290642
0
0
0
0
0
0
0
10
100
1000
10000
100000
1000000
Value

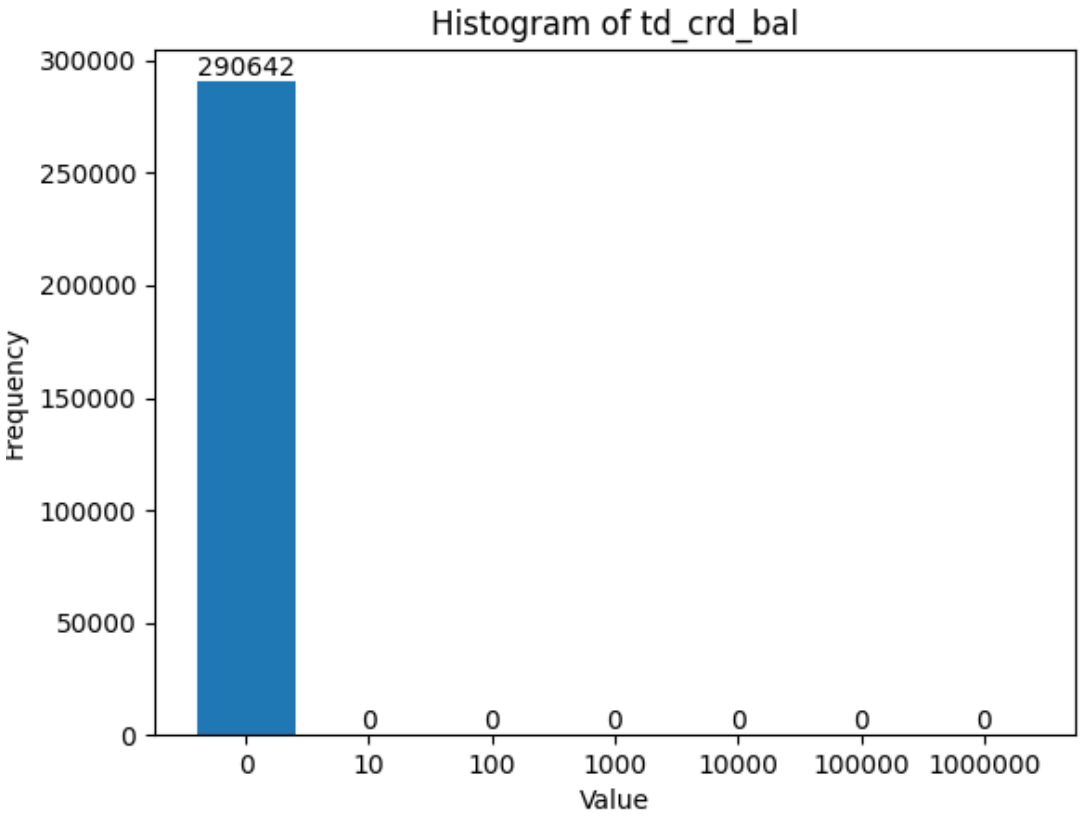

Histogram of avg_year
Frequency
Value
0
10000
20000
30000
40000
50000
60000
50968
27376
60420
61976
61106
28436
356
4
0
10
100
1000
10000
100000
1000000
10000000

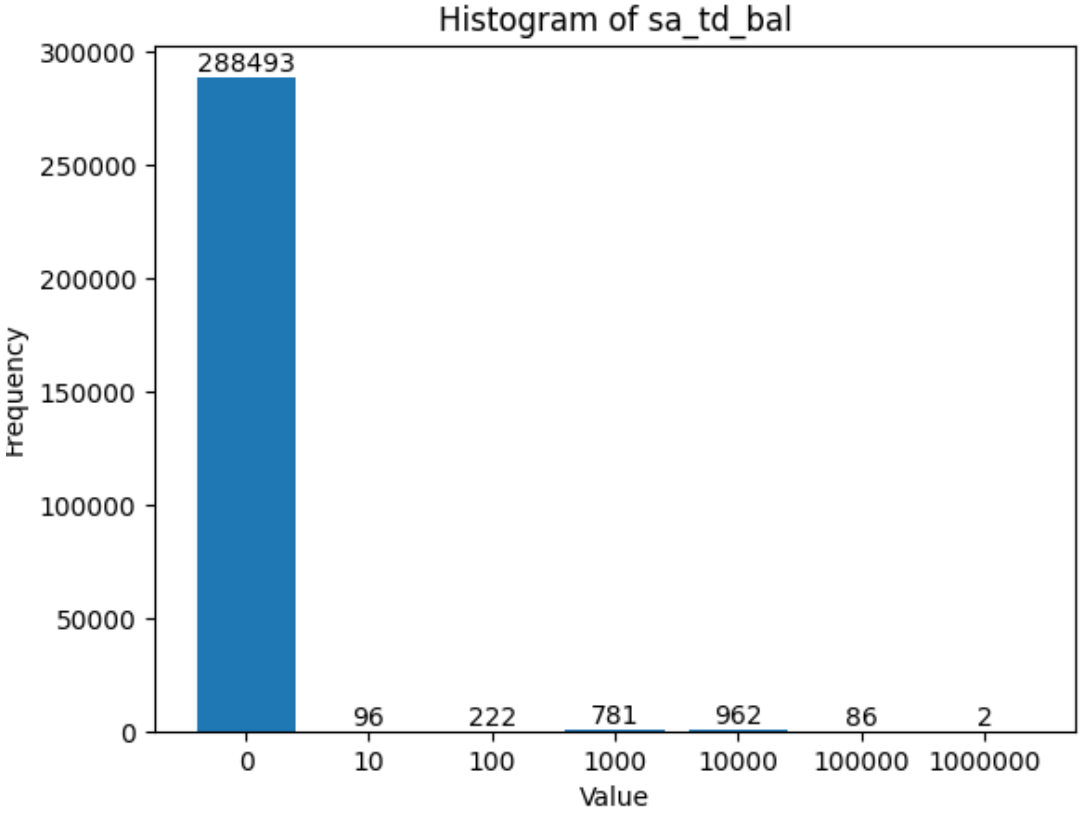

Histogram of sa_bal
Frequency
Value
0
10000
20000
30000
40000
50000
60000
70000
80000
70545
41805
79499
72383
24267
2105
38
0
10
100
1000
10000
100000
1000000

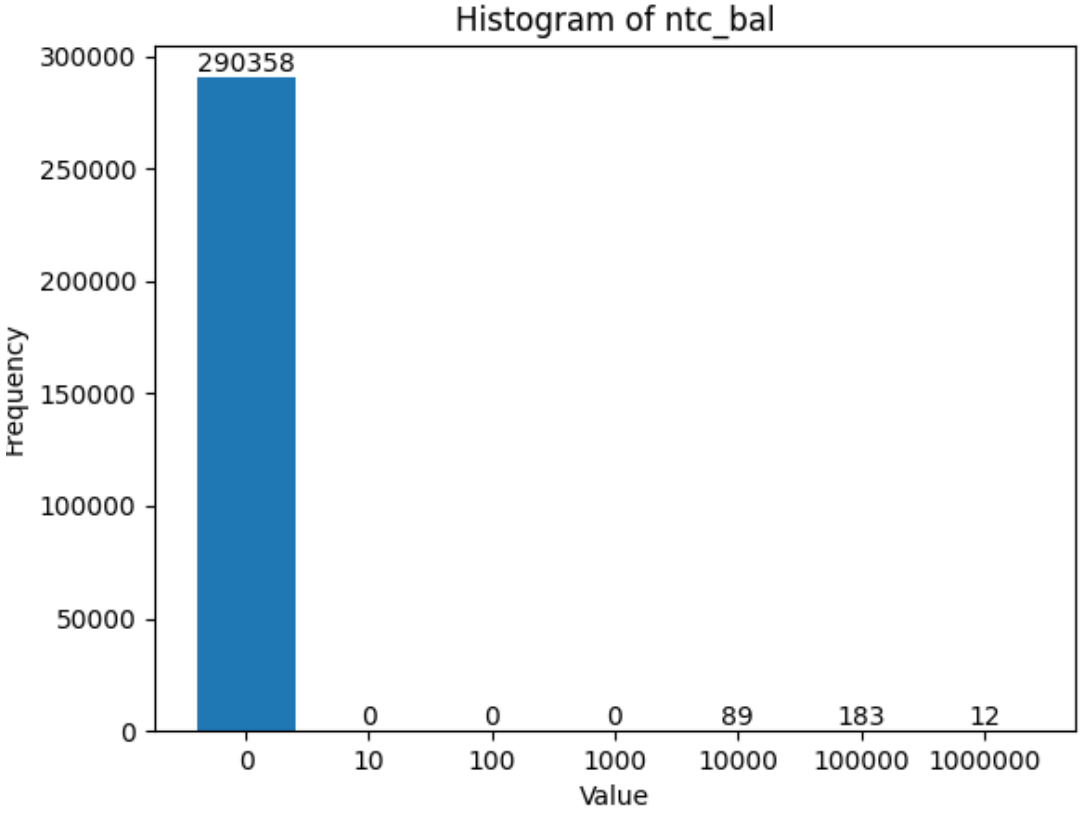

Histogram of td_bal
Frequency
Value
217372
89
463
4883
40630
26883
318
4
0
10
100
1000
10000
100000
1000000
10000000
0
50000
100000
150000
200000

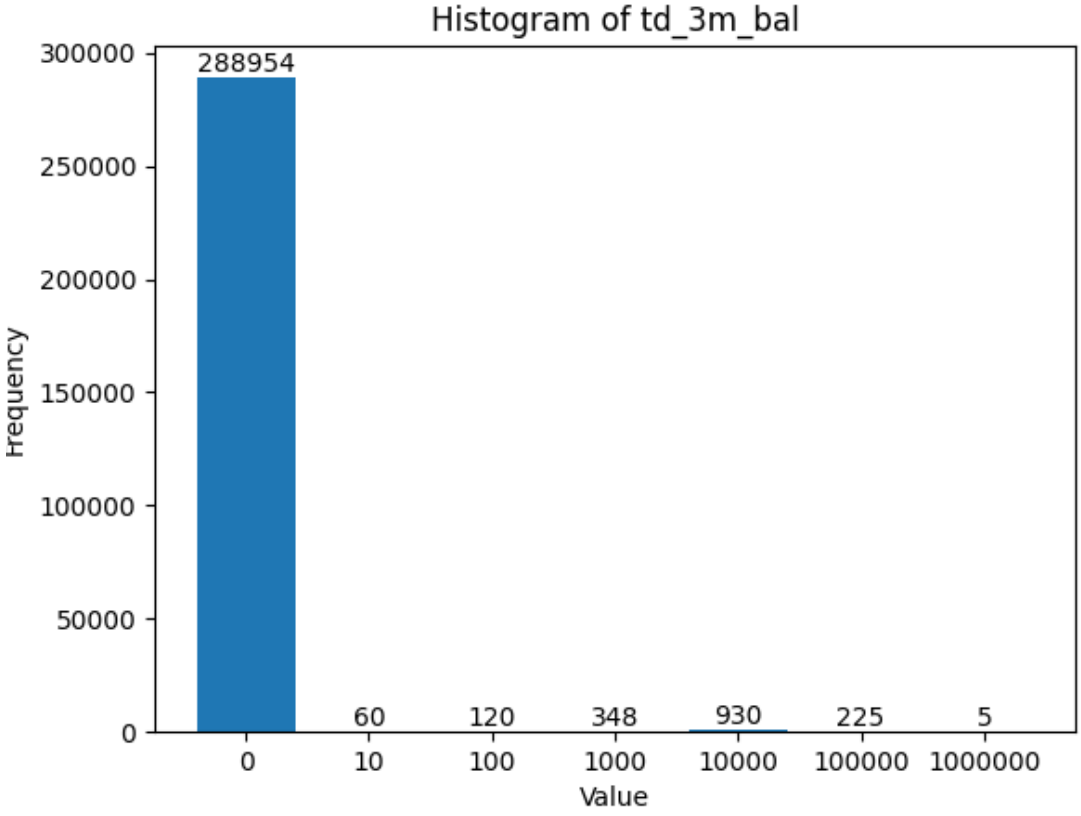

Histogram of fin_bal
Frequency
Value
289768
0
0
0
276
584
14
0
10
100
1000
10000
100000
1000000
0
50000
100000
150000
200000
250000
300000

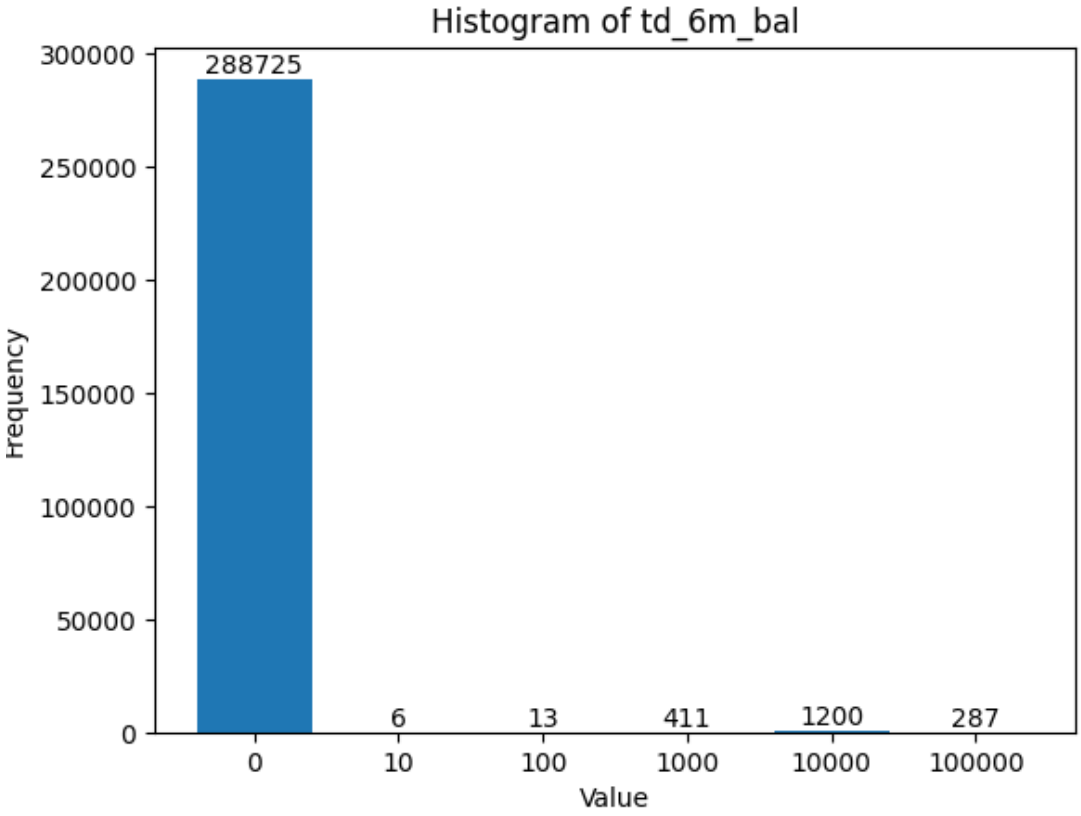

Histogram of sa_crd_bal
81156
76003
69032
38801
23553
2060
37
Frequency
0
10000
20000
30000
40000
50000
60000
70000
80000
0
10
100
1000
10000
100000
1000000
Value

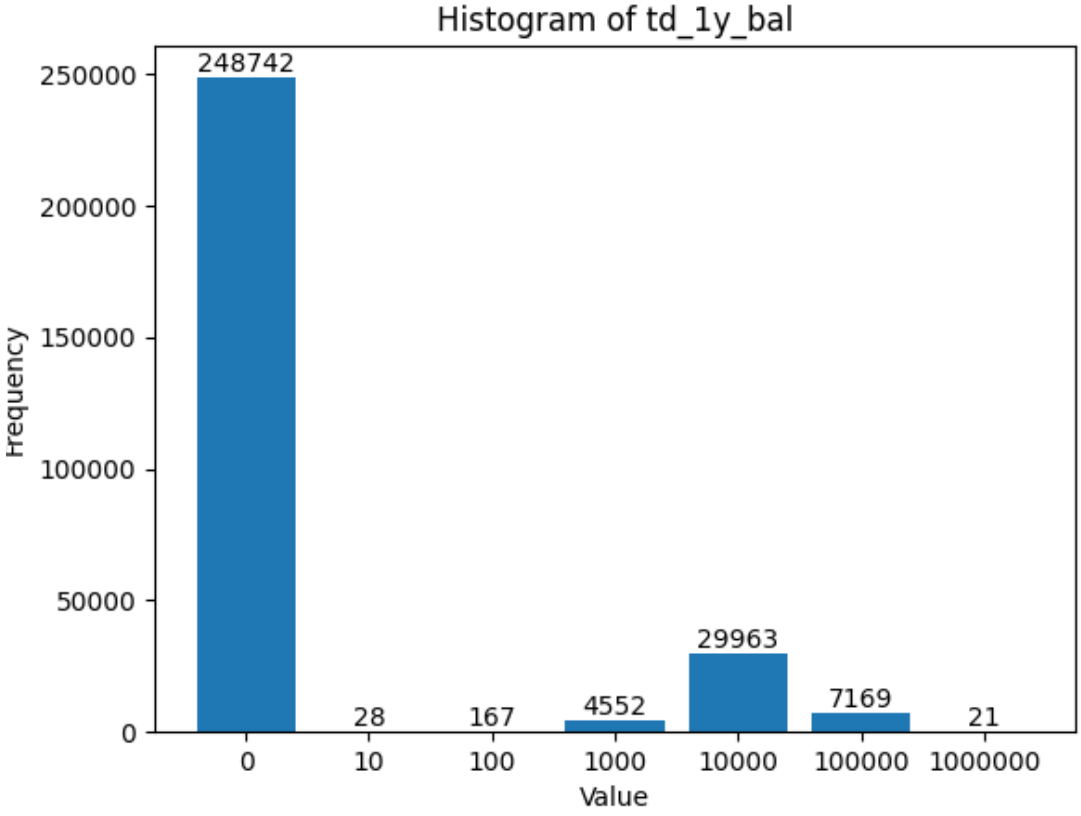

Histogram of td_crd_bal
290642
0
0
0
0
0
0
Frequency
0
50000
100000
150000
200000
250000
300000
0
10
100
1000
10000
100000
1000000
Value

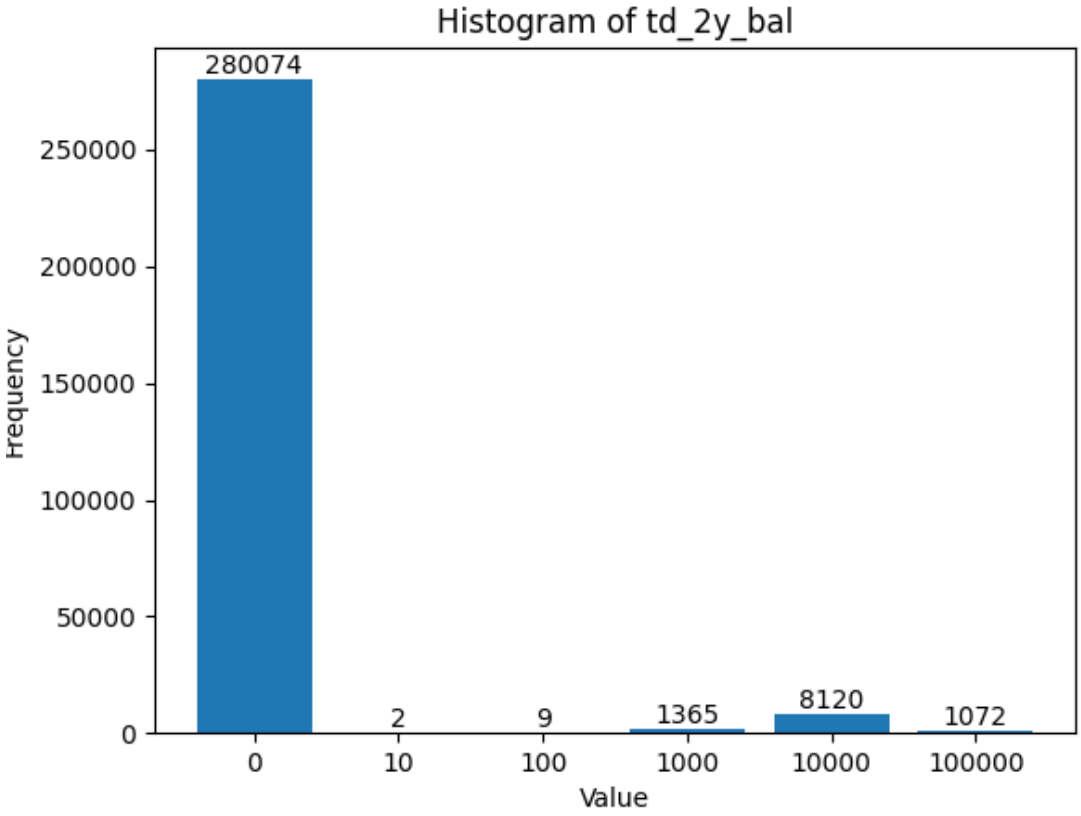

Histogram of sa_td_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
288493
96
222
781
962
86
2
0
10
100
1000
10000
100000
1000000

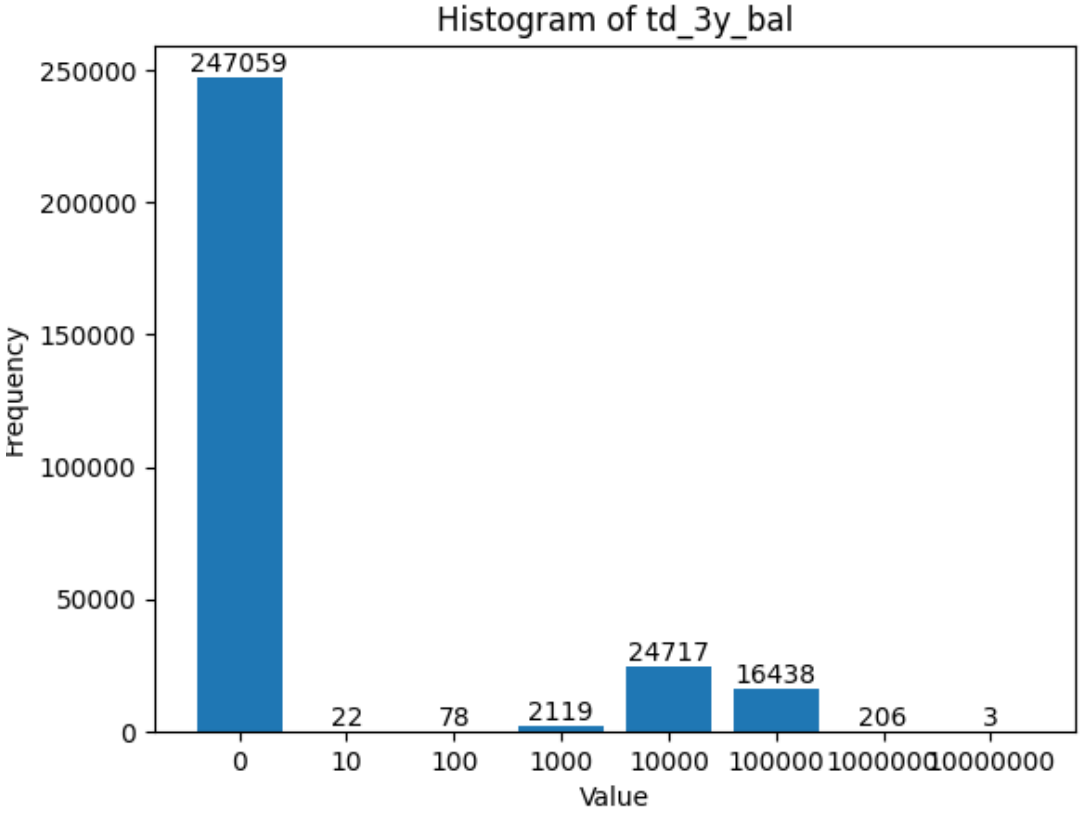

Histogram of ntc_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
290358
0
0
0
89
183
12
0
10
100
1000
10000
100000
1000000

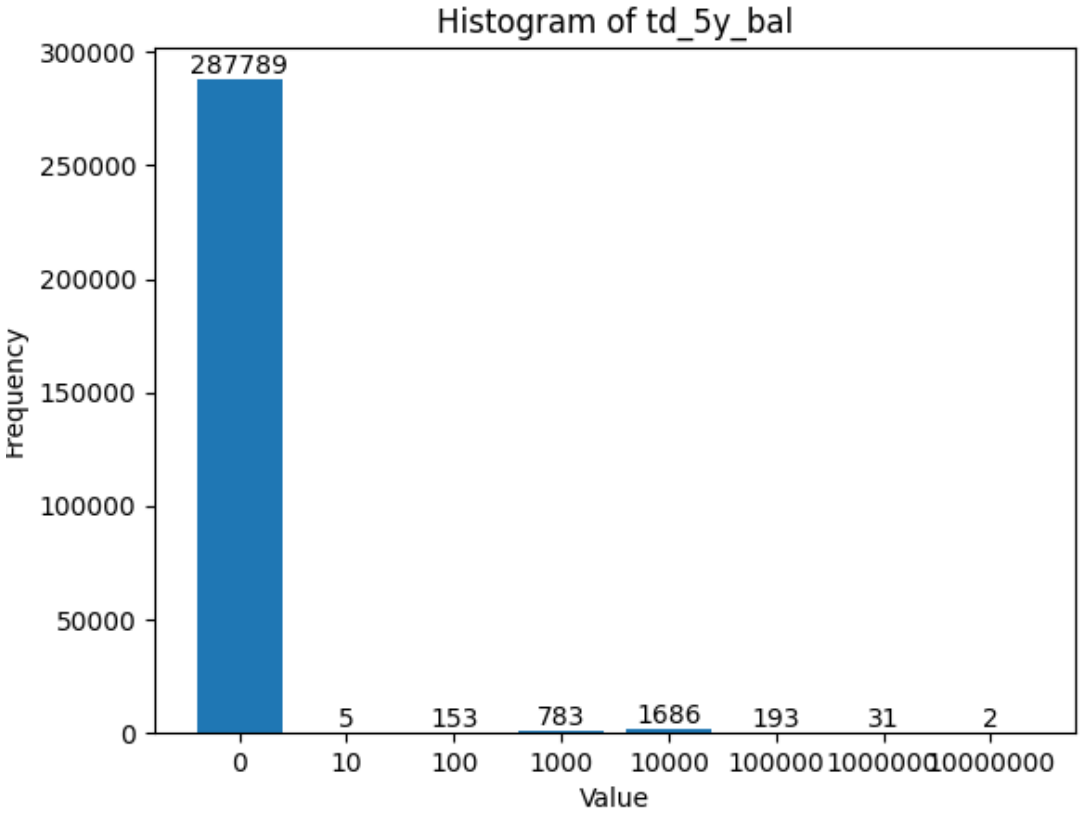

Histogram of td_3m_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
288954
60
120
348
930
225
5
0
10
100
1000
10000
100000
1000000

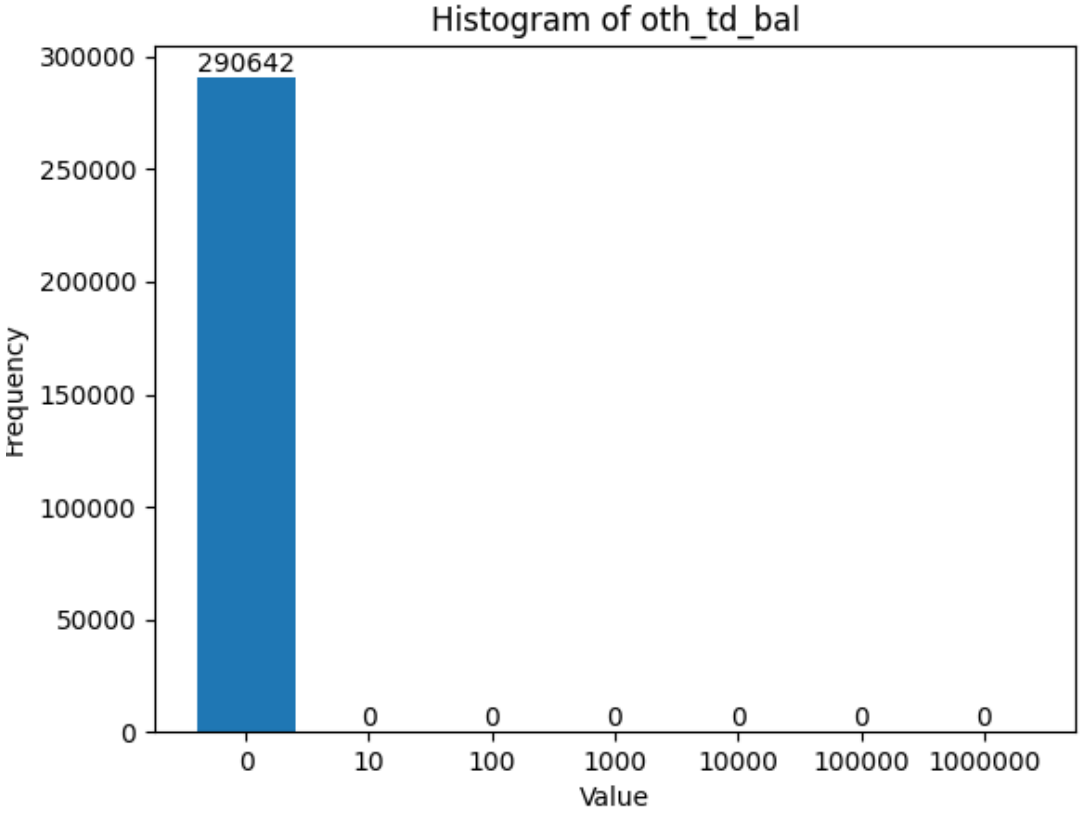

Histogram of td_6m_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
288725
6
13
411
1200
287
0
10
100
1000
10000
100000

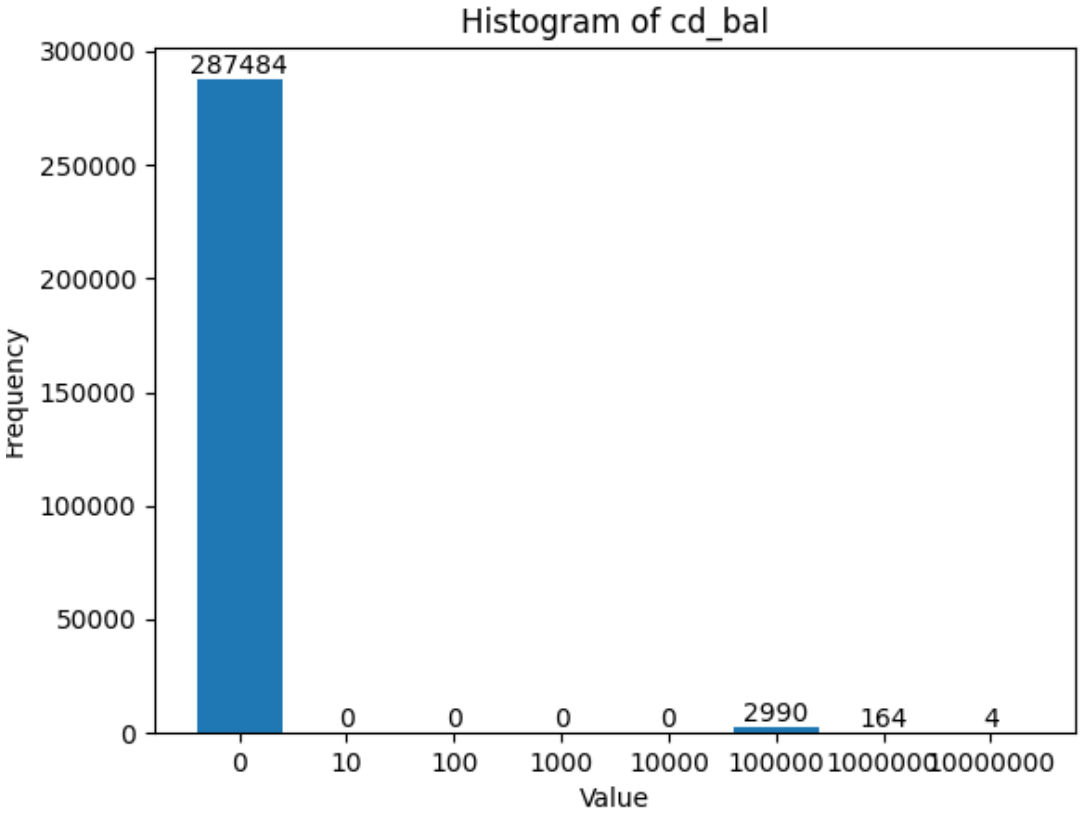

Histogram of td_1y_bal
Frequency
Value
248742
28
167
4552
29963
7169
21
0
10
100
1000
10000
100000
1000000
250000
200000
150000
100000
50000
0

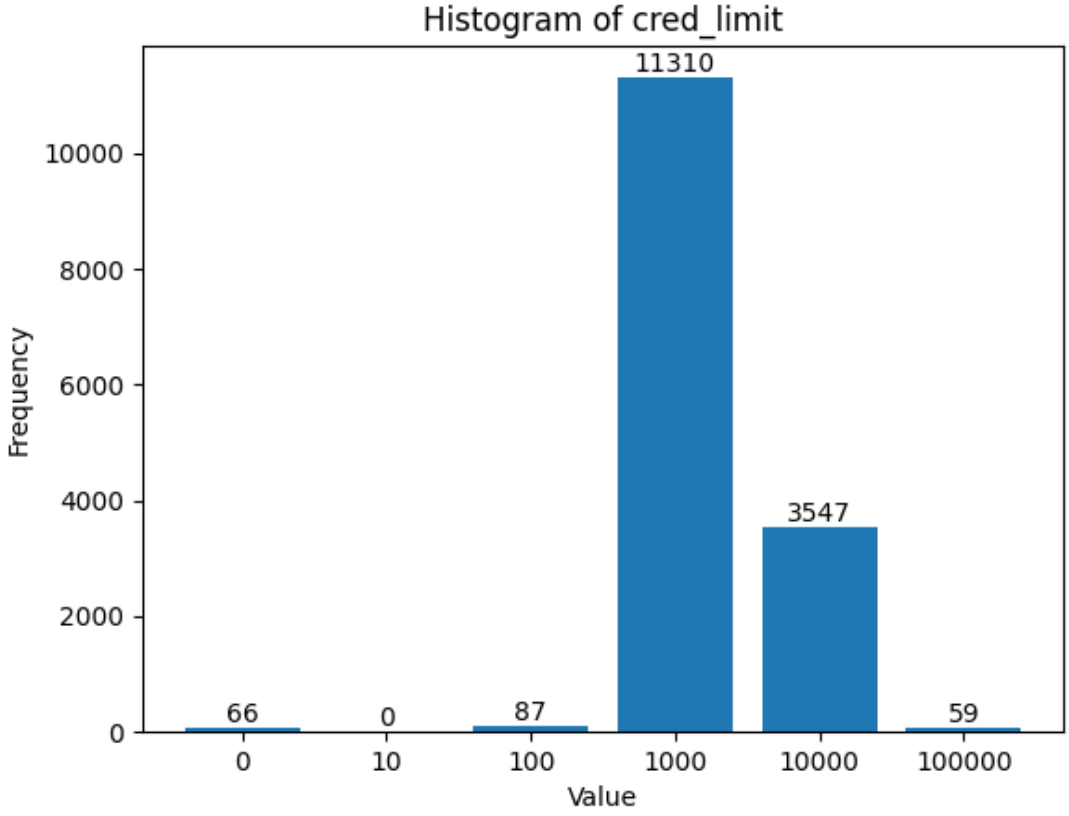

Histogram of td_2y_bal
Frequency
Value
280074
2
9
1365
8120
1072
0
10
100
1000
10000
100000
250000
200000
150000
100000
50000
0

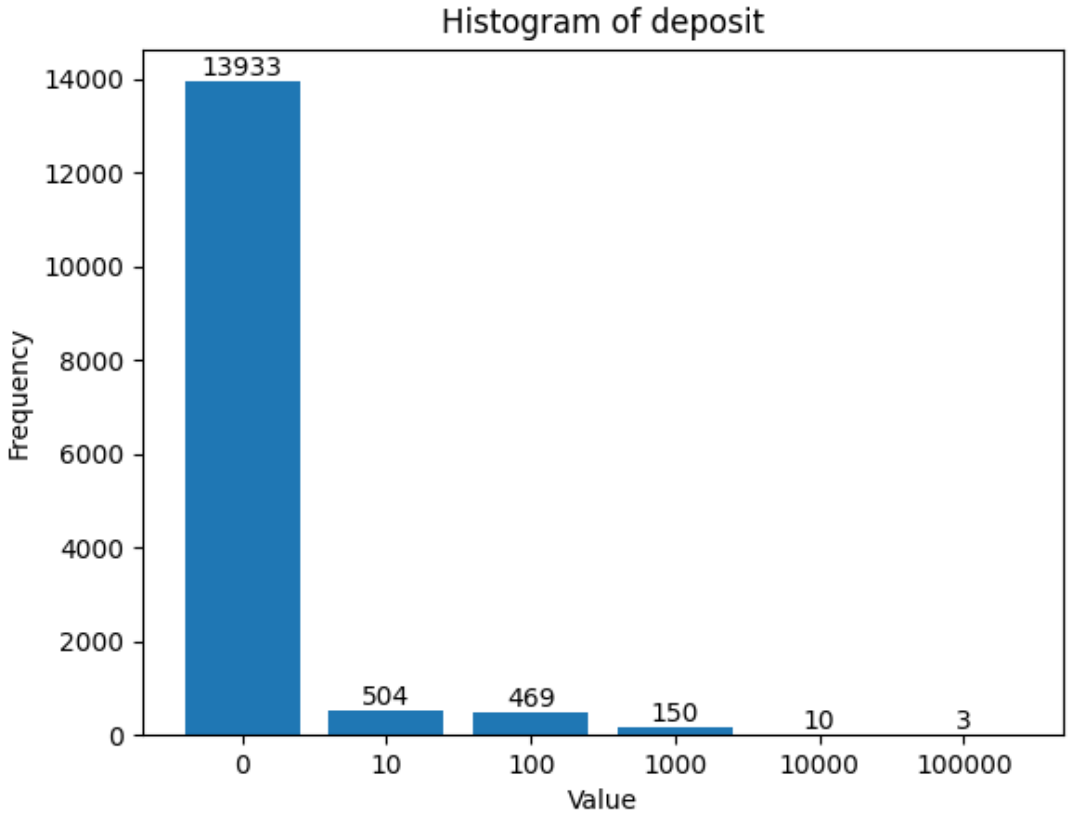

Histogram of td_3y_bal
Frequency
Value
247059
22
78
2119
24717
16438
206
3
0
10
100
1000
10000
100000
1000000
10000000
250000
200000
150000
100000
50000
0

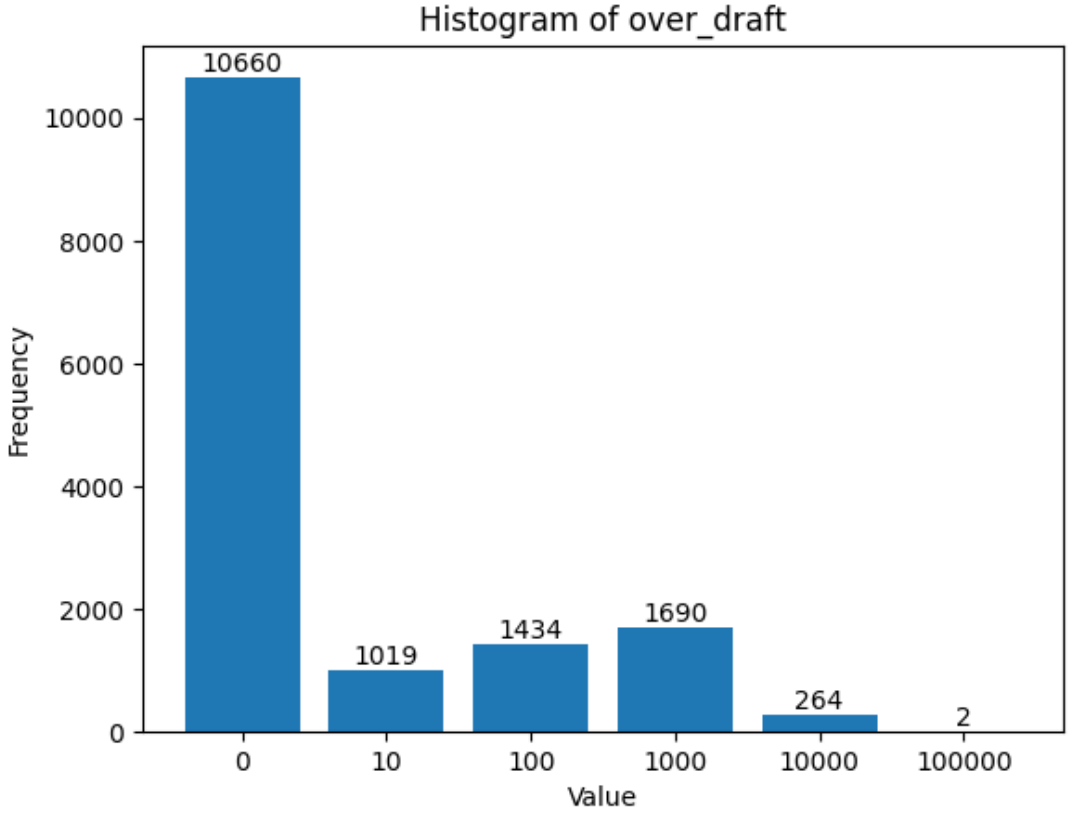

Histogram of td_5y_bal
Frequency
Value
287789
5
153
783
1686
193
31
2
0
10
100
1000
10000
100000
1000000
10000000
300000
250000
200000
150000
100000
50000
0

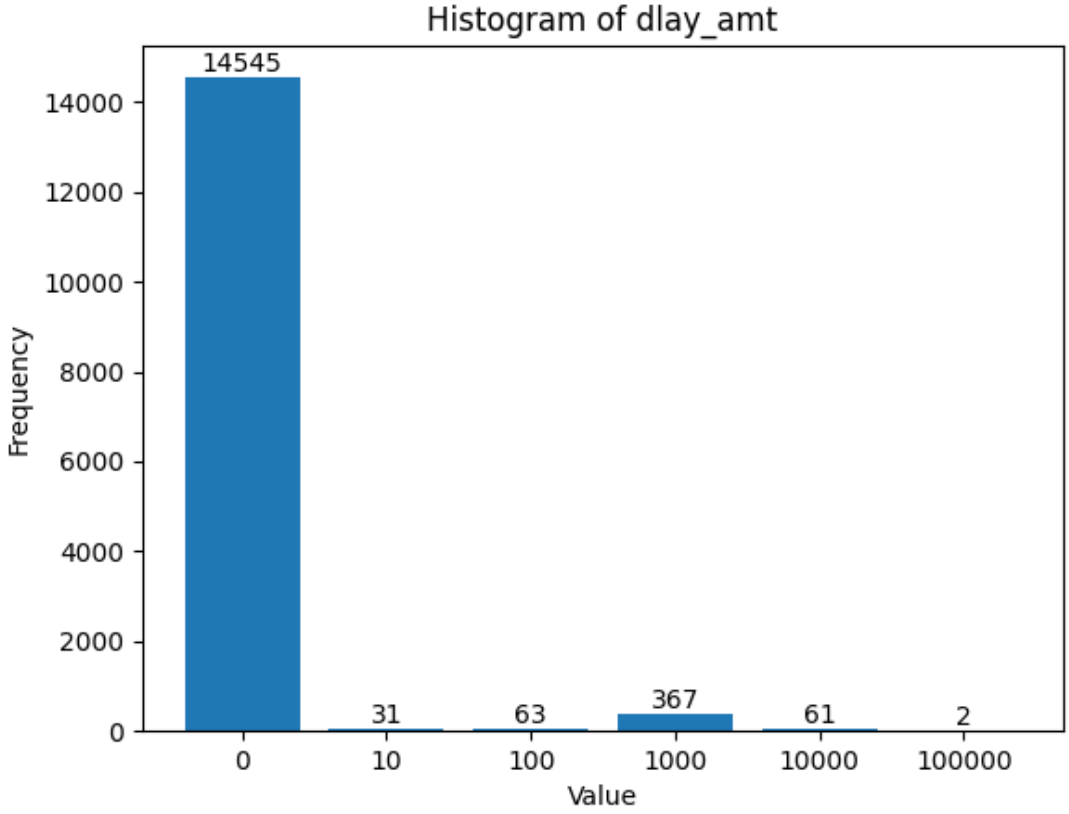

Histogram of oth_td_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
290642
0
0
0
0
0
0
0
10
100
1000
10000
100000
1000000

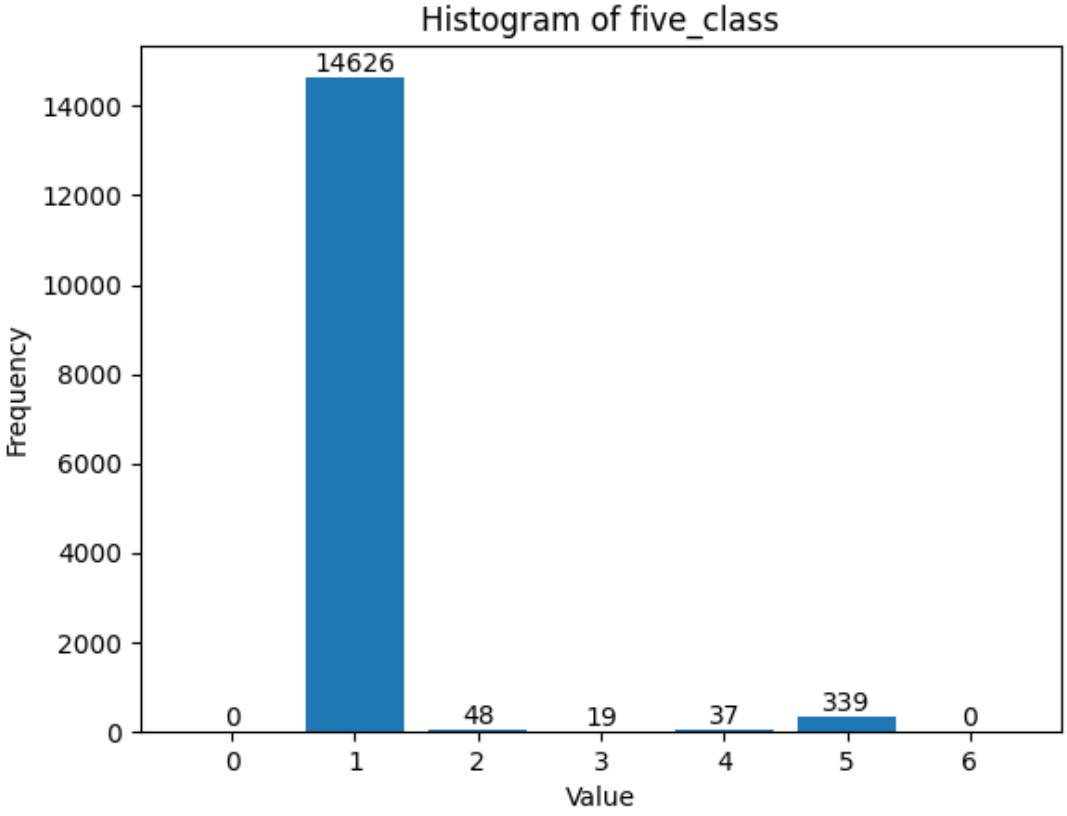

Histogram of cd_bal
Frequency
Value
300000
250000
200000
150000
100000
50000
0
287484
0
0
0
0
2990
164
4
0
10
100
1000
10000
100000
1000000
10000000

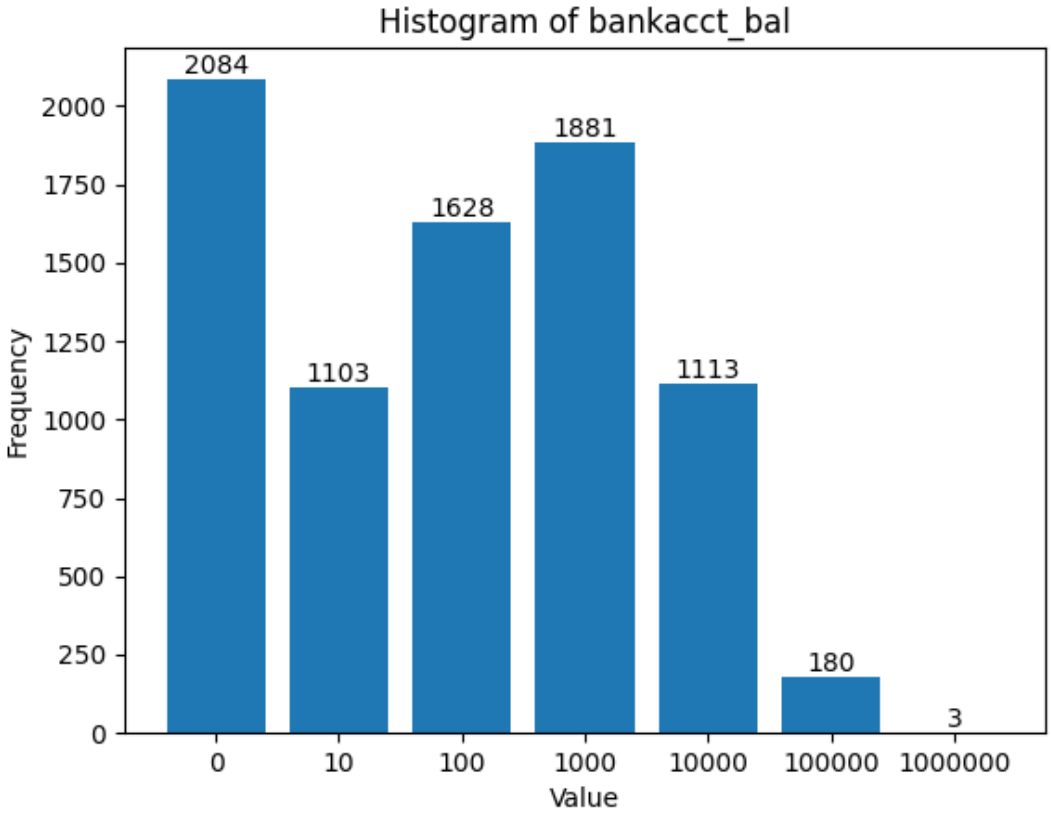

Histogram of cred_limit
Frequency
Value
0
2000
4000
6000
8000
10000
66
0
87
11310
3547
59
0
10
100
1000
10000
100000


Histogram of deposit
Frequency
Value
0
2000
4000
6000
8000
10000
12000
14000
13933
504
469
150
10
3
0
10
100
1000
10000
100000

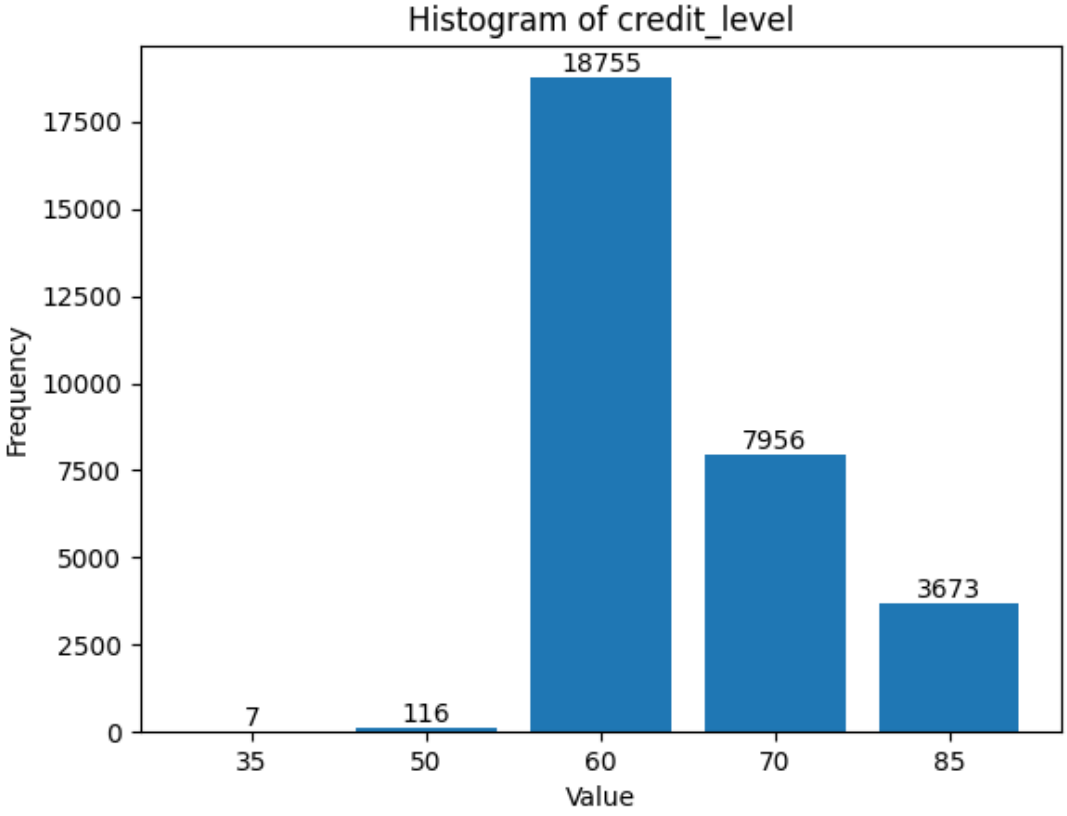

Histogram of over_draft
Frequency
Value
0
2000
4000
6000
8000
10000
10660
1019
1434
1690
264
2
0
10
100
1000
10000
100000

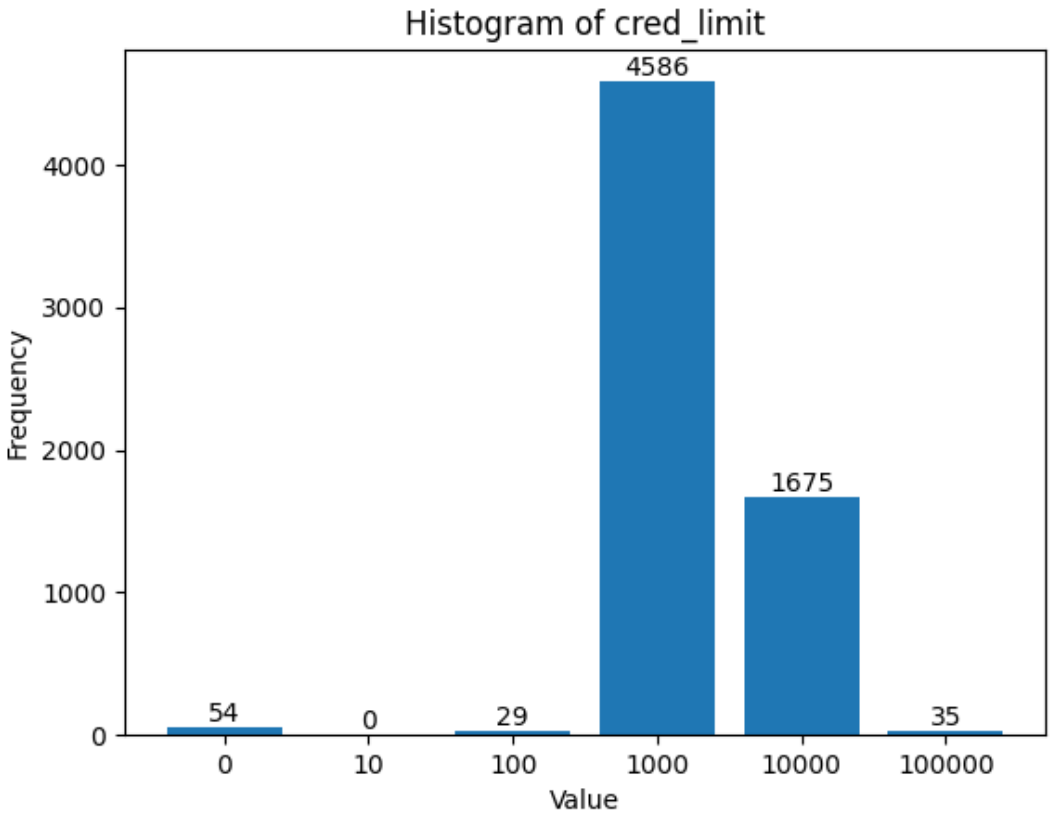

Histogram of dlay_amt
Frequency
Value
0
2000
4000
6000
8000
10000
12000
14000
14545
31
63
367
61
2
0
10
100
1000
10000
100000

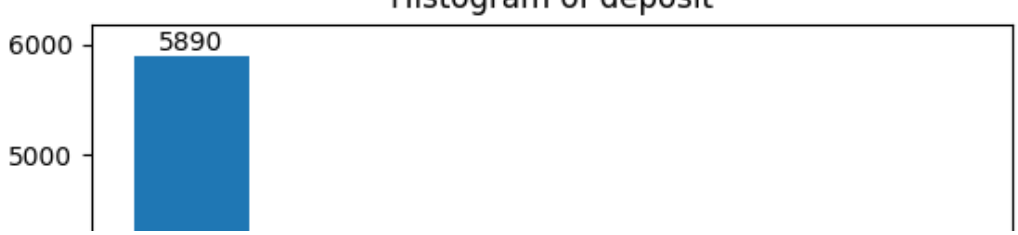

Histogram of five_class
Frequency
Value
14626
0
48
19
37
339
0
0
1
2
3
4
5
6

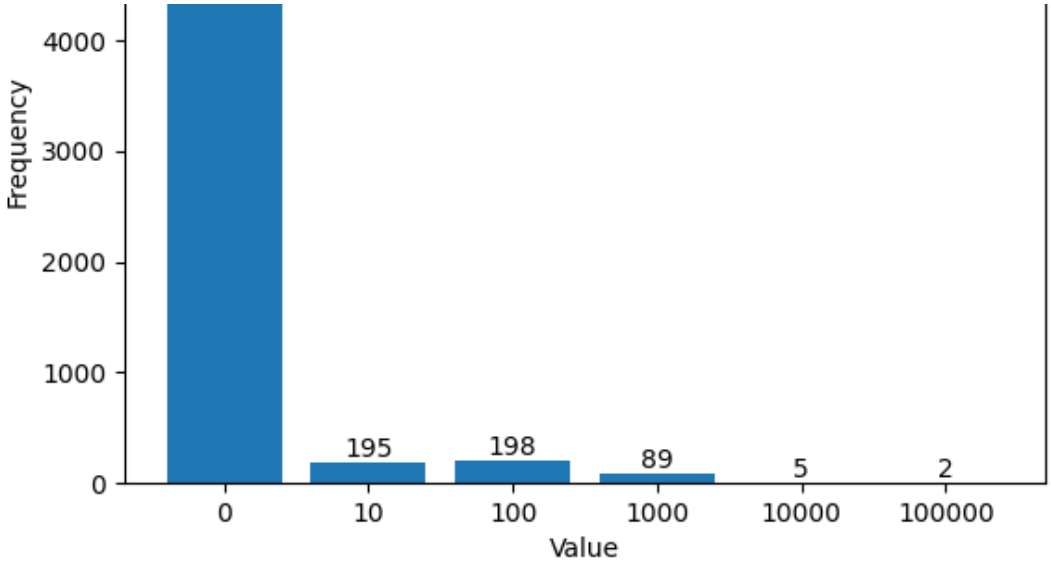

Histogram of bankacct_bal
Frequency
Value
2084
1103
1628
1881
1113
180
3
0
10
100
1000
10000
100000
1000000

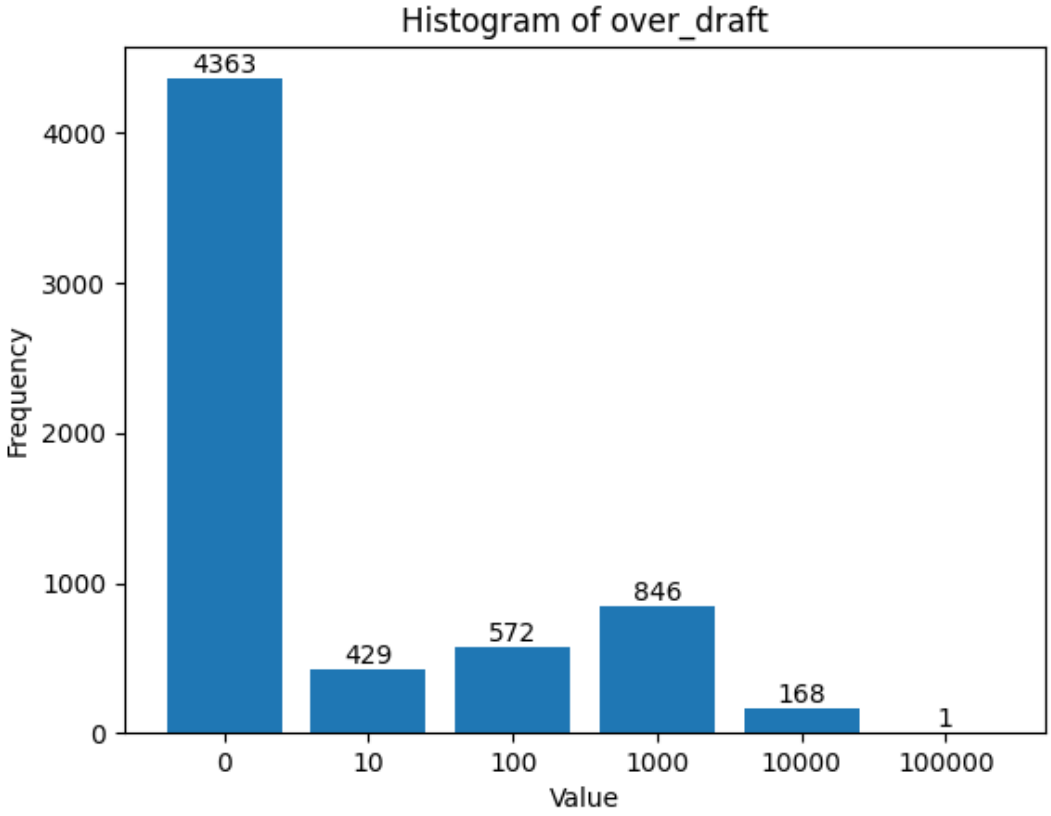
4.3.2信用等级

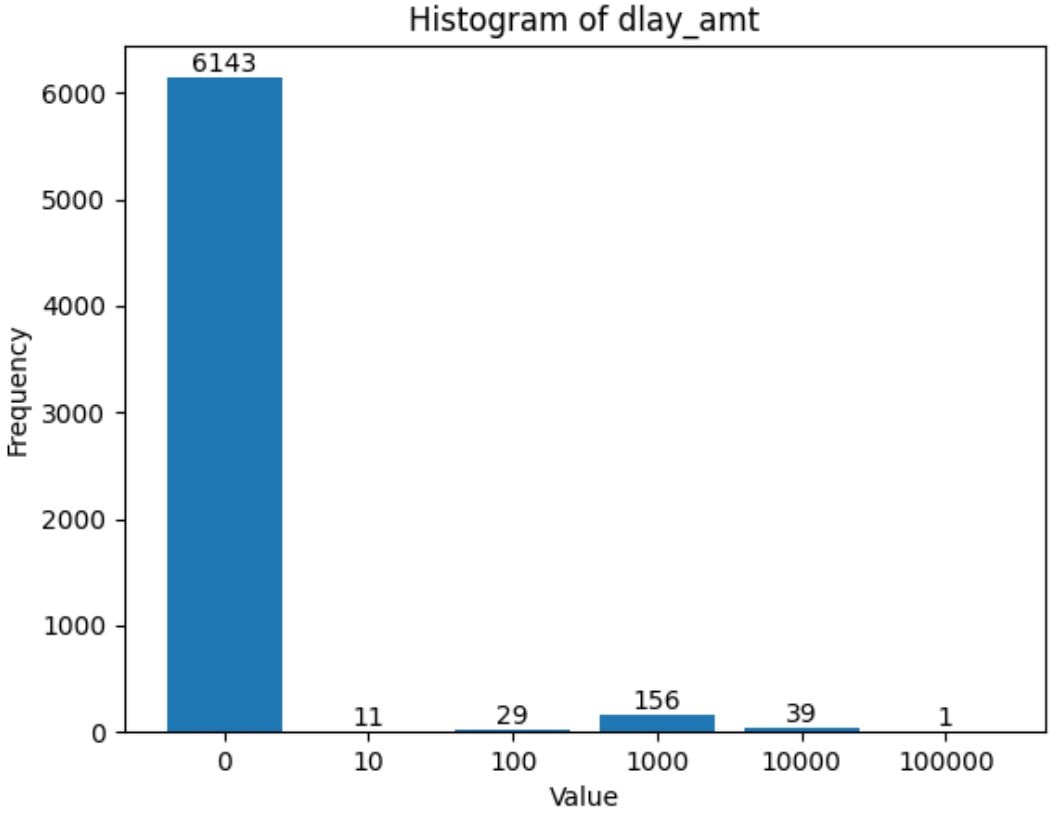

Histogram of credit_level
Frequency
Value
18755
7956
3673
7
116
35
50
60
70
85

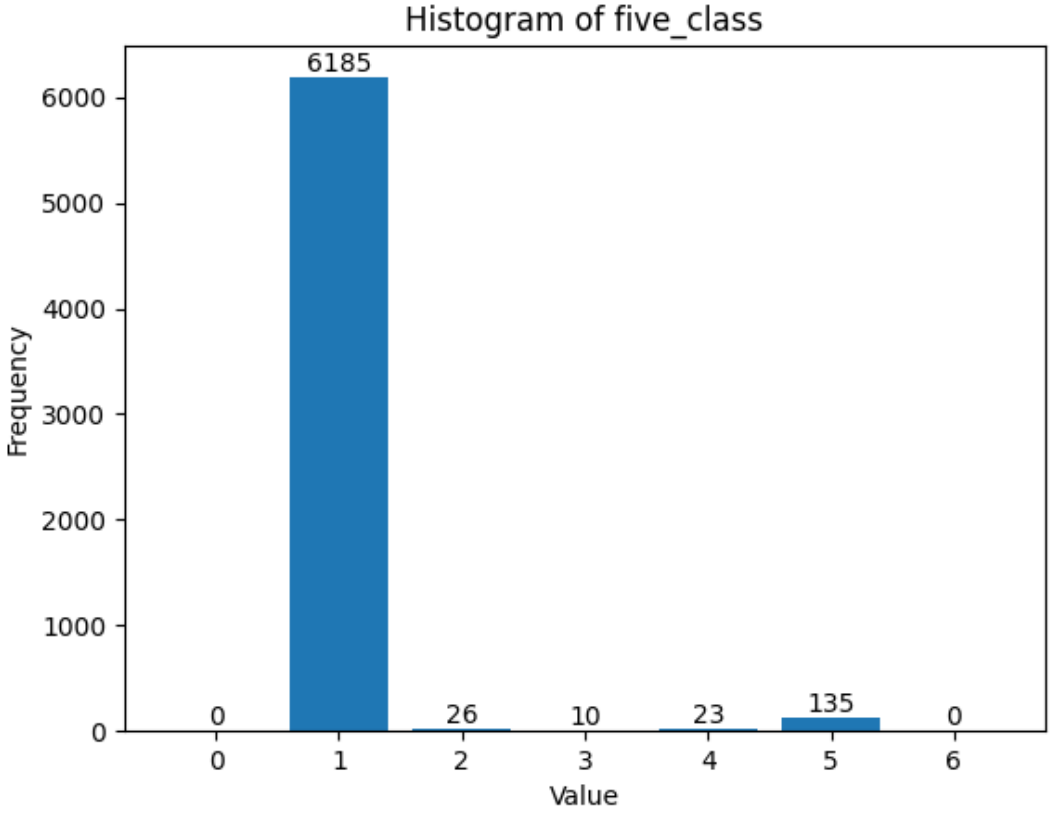

Histogram of cred_limit
Frequency
Value
4586
1675
54
0
29
35
0
10
100
1000
10000
100000

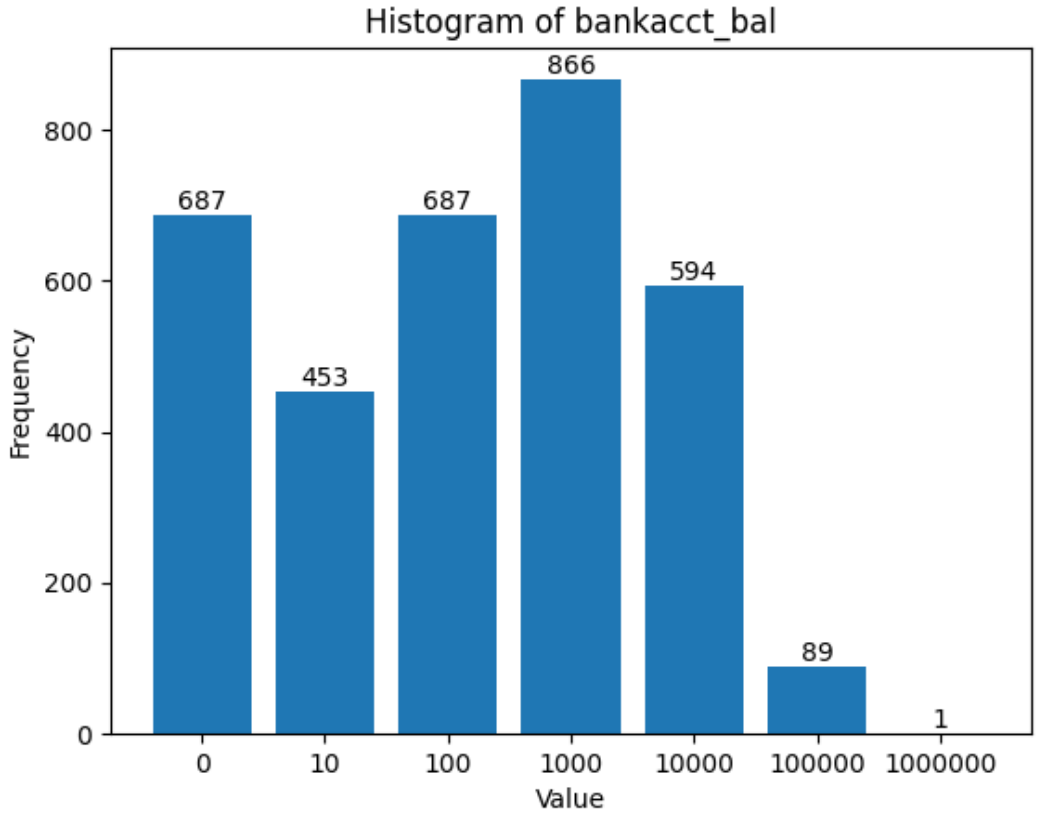

Histogram of deposit
5890

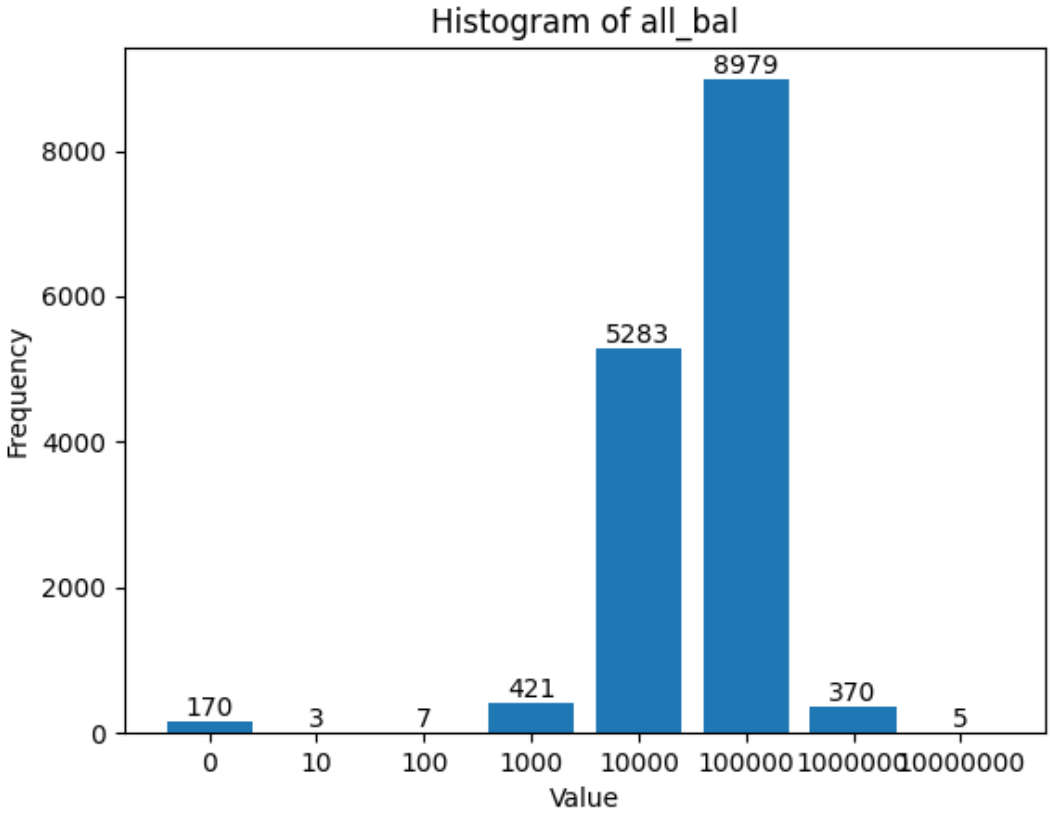

4000
3000
2000
1000
0
Frequency
195
198
89
5
2
0
10
100
1000
10000
100000
Value

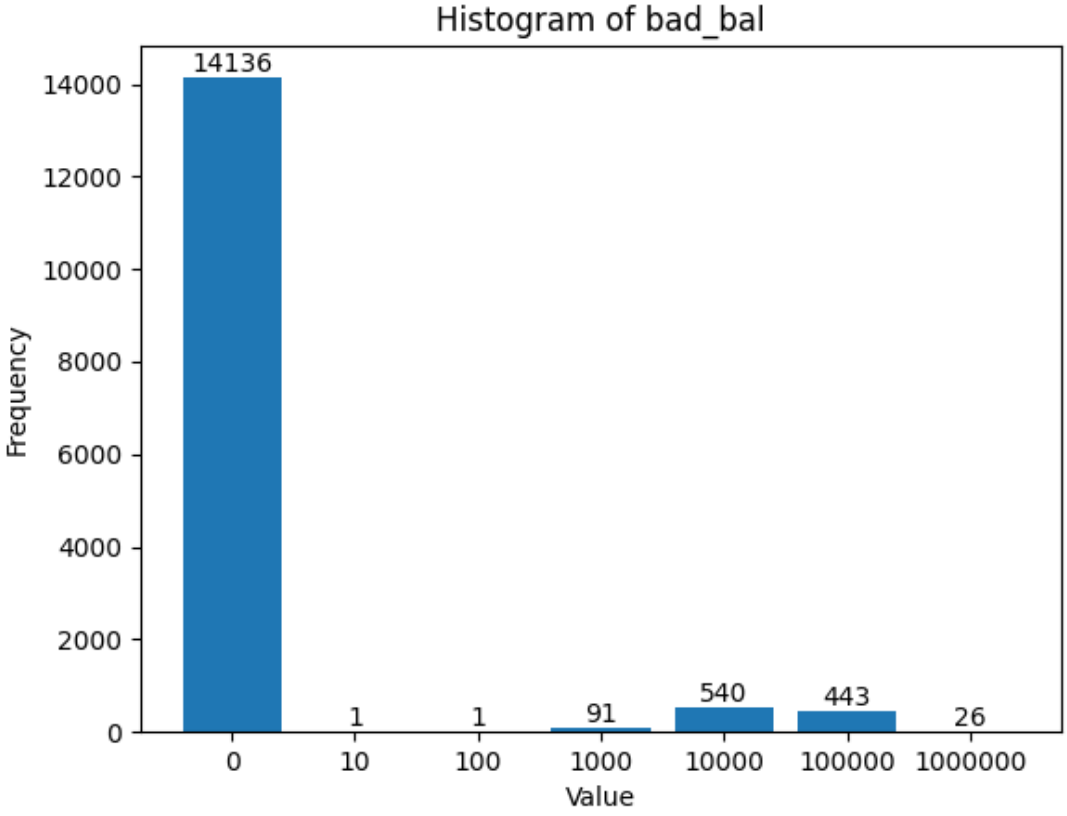

Histogram of over_draft
4363
4000
3000
2000
1000
0
Frequency
429
572
846
168
1
0
10
100
1000
10000
100000
Value

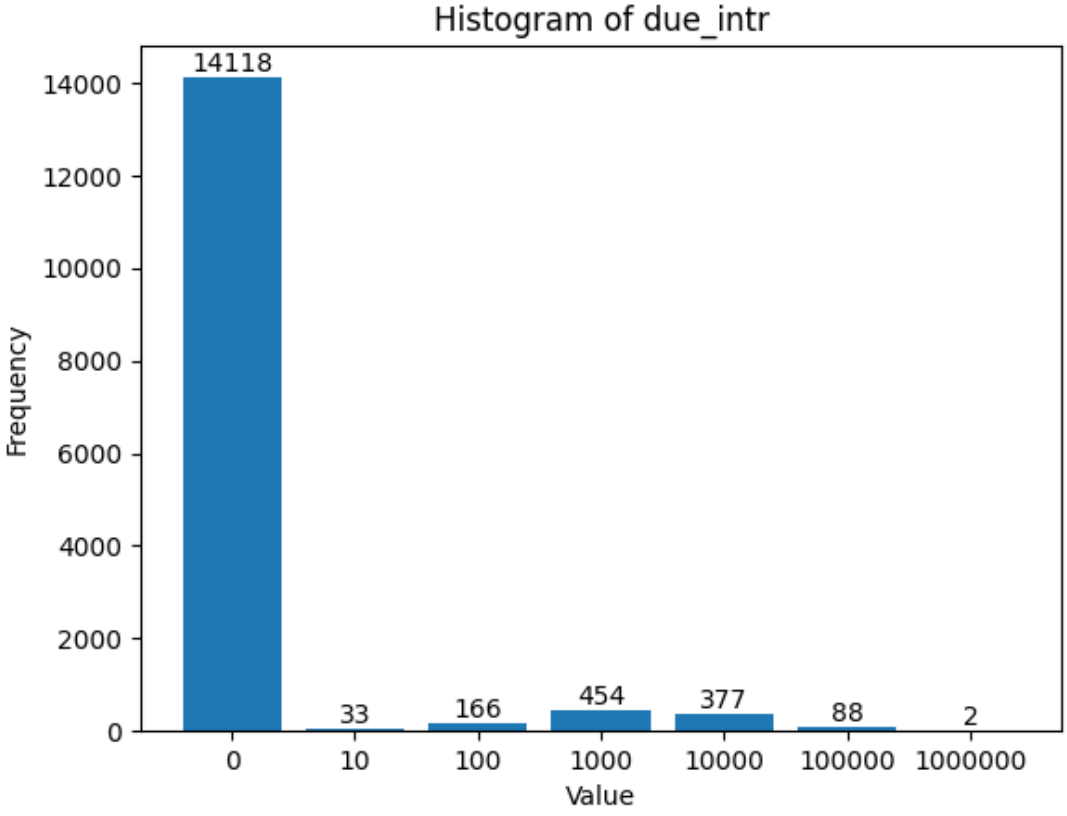

Histogram of dlay_amt
Frequency
Value
6143
11
29
156
39
1
0
10
100
1000
10000
100000
0
1000
2000
3000
4000
5000
6000

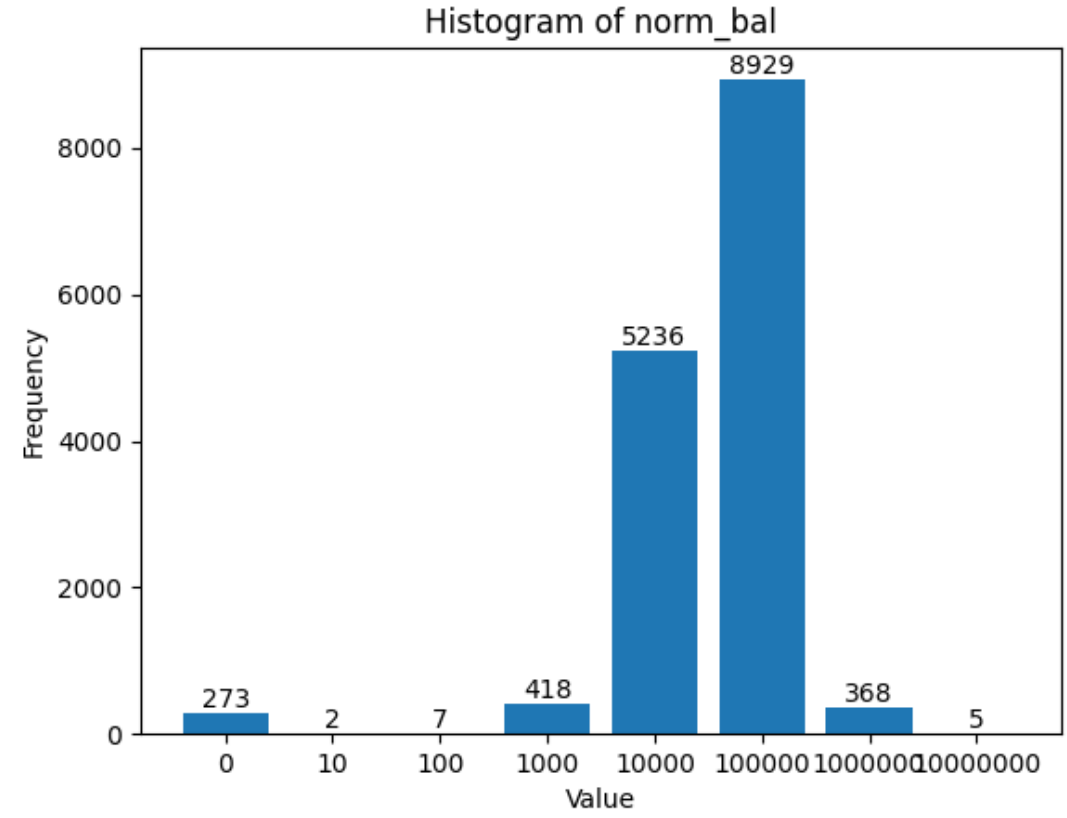

Histogram of five_class
Frequency
Value
0
6185
26
10
23
135
0
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
6000

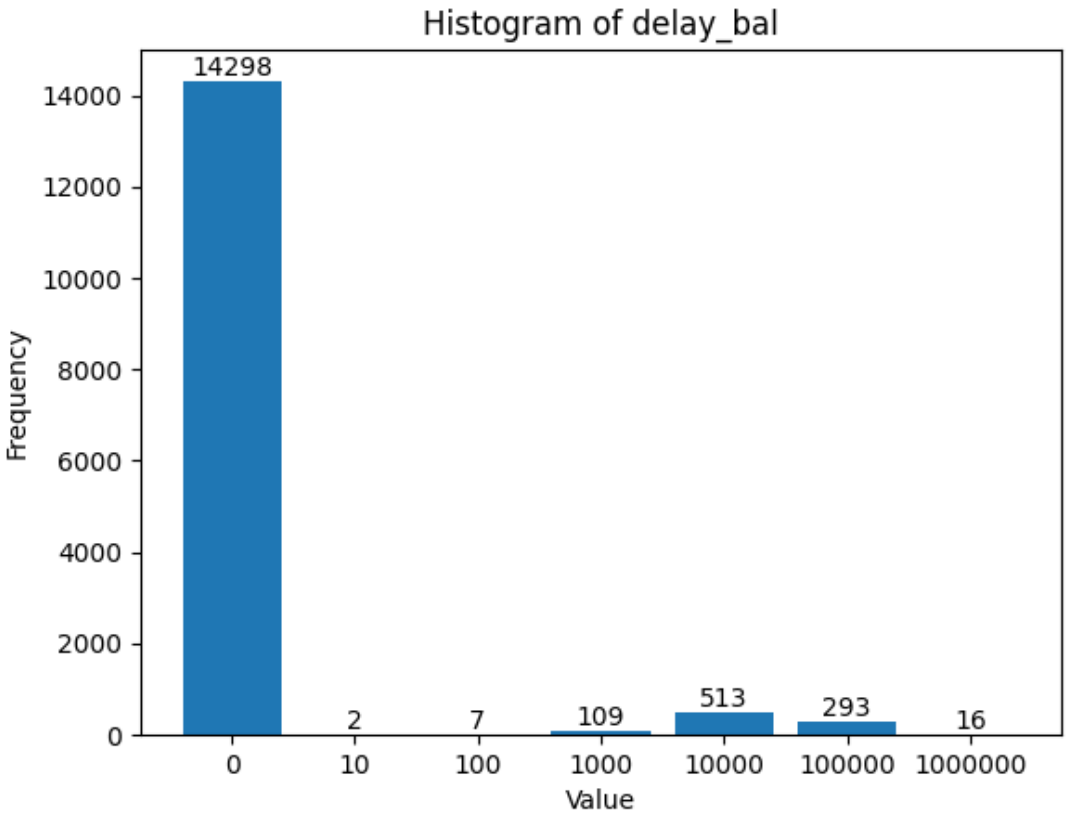

Histogram of bankacct_bal
Frequency
Value
0
200
400
600
800
687
453
687
866
594
89
1
0
10
100
1000
10000
100000
1000000


Histogram of all_bal
Frequency
Value
0
2000
4000
6000
8000
170
3
7
421
5283
8979
370
5
0
10
100
1000
10000
100000
1000000
10000000


Histogram of bad_bal
Frequency
Value
14136
1
1
91
540
443
26
0
10
100
1000
10000
100000
1000000
0
2000
4000
6000
8000
10000
12000
14000


Histogram of due_intr
Frequency
Value
14118
33
166
454
377
88
2
0
10
100
1000
10000
100000
1000000
0
2000
4000
6000
8000
10000
12000
14000





二、数据预处理25'

提交附件的不同phase对应如下：

一、数据收集

phase1　　信用等级/用户星级与个人基本信息组合，保证完整率近100%;

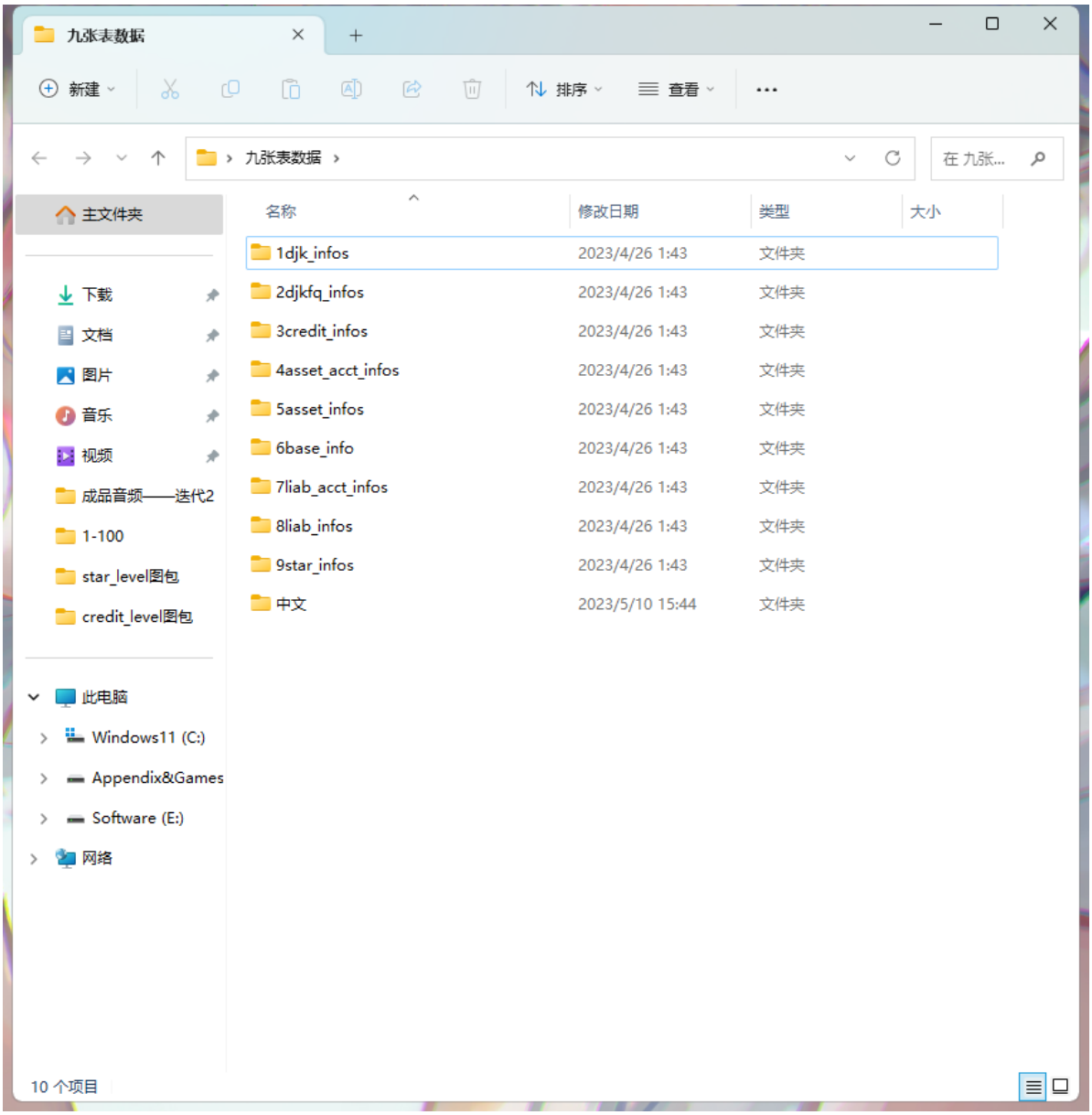
```
phase2          拟定要分析的属性（包括含-1的行）
phase3          能够量化分析频数的属性（包括含-1的行）
describe        数据盘点

二、数据预处理
phase4          phase2(不包括含-1的行)
missing_rate    缺失率
phase5          在phase4的基础上添加了缺失值处理
phase6          在phase5的基础上添加了异常值处理
phase7          综合phase4和phase6，将非数值恢复
trans           数据转换后
std             数据标准化后
```

1.去重0'

1.1hive->local CSV去重

使用来自作业2的方法，在虚拟机内去重并将结果移动到本地。

这种方法只能去除完全一致的行。



1.2local CSV->my SQL去重

使用my SQL的聚合语句可以做到去重的同时加以选择。这里的id是新创建的自增主码。

```
SELECT *
FROM djk_info
WHERE id IN (
  SELECT MIN(id)
  FROM djk_info
  GROUP BY uid
);
```

2.缺失值处理8'（含扩展3'）

2.1缺失率统计

2.1.1用户星级

字段	missing rate
uid	0.000000
star_level	0.000000
sex	0.000009
marriage	0.000009
education	0.000009
is_black	0.000009
is_contact	0.000009
all_bal	0.000047
avg_mth	0.000047
avg_qur	0.000047
avg_year	0.000047
sa_bal	0.000047
td_bal	0.000047
fin_bal	0.000047
sa_crd_bal	0.000047
td_crd_bal	0.000047
sa_td_bal	0.000047
ntc_bal	0.000047
td_3m_bal	0.000047
td_6m_bal	0.000047
td_1y_bal	0.000047
td_2y_bal	0.000047
td_3y_bal	0.000047
td_5y_bal	0.000047
oth_td_bal	0.000047
cd_bal	0.000047
card_sts	0.977950
is_withdrw	0.948102
is_transfer	0.948102
is_deposit	0.948102

字段	missing rate
is_purchase	0.948102
cred_limit	0.948102
deposit	0.948102
over_draft	0.948102
dlay_amt	0.948102
five_class	0.948102
bankacct_bal	0.972505

2.1.2信用等级

字段	missing rate
uid	0.000000
credit_level	0.000000
sex	0.000000
marriage	0.000000
education	0.000000
is_black	0.000000
is_contact	0.000000
card_sts	0.930541
is_withdrw	0.833383
is_transfer	0.833383
is_deposit	0.833383
is_purchase	0.833383
cred_limit	0.833383
deposit	0.833383
over_draft	0.833383
dlay_amt	0.833383
five_class	0.833383
bankacct_bal	0.912971
all_bal	0.601829
bad_bal	0.601829
due_intr	0.601829
norm_bal	0.601829
delay_bal	0.601829

2.2缺失值处理

2.2.1去除缺失值比例大于0.7的列

```
    df = pd.read_csv('credit_train_land_credit_demo_phase4.csv')
    df = df.loc[:, df.isnull().mean() < 0.7]
    df.to_csv('credit_train_land_credit_demo_phase5.csv')

    df = pd.read_csv('credit_train_land_star_demo_phase4.csv')
    df = df.loc[:, df.isnull().mean() < 0.7]
    df.to_csv('credit_train_land_star_demo_phase5.csv')
```

2.2.2去除有数值但是并无实际作用的列

在信用评估中不存在这样的字段；但是在星级评估中，avg_qur,oth_td_bal,td_crd_bal为全0，并没有实际意义，应该去除。

```
    columns = ['avg_qur', 'oth_td_bal', 'td_crd_bal']
    df = df.drop(columns, axis=1)
```

2.2.3使用机器学习算法补充空值

在信用等级评定中，即使是最为关键的列，缺失率也达到了60%（0.601829），我们使用KNN模型（K最近邻算法）来填充缺失值。

```
missing_cols = ['all_bal', 'bad_bal', 'due_intr', 'norm_bal', 'delay_bal']
    for target_col in missing_cols:
        feature_cols = ['credit_level', 'sex', 'marriage', 'education',
'is_black', 'is_contact']
        x = df.dropna()[feature_cols]
        y = df[target_col].dropna()
        knn_model = KNeighborsRegressor(n_neighbors=5)
        knn_model.fit(x, y)
        missing_rows = df[df[target_col].isnull()]
        predicted_values = knn_model.predict(missing_rows[feature_cols])
        df.loc[df[target_col].isnull(), target_col] = predicted_values
```

但是在此时，我们知道sex，marriage，education，is_black，is_contact并不是可解析的值，需要先做转换。

```
df.replace({'sex': {'男': 1, '女': 0}}, inplace=True)
    df.replace({'marriage': {'未婚': 0, '初婚': 1, '已婚': 2, '离婚': 3, '丧偶': 4,
'未说': 5}}, inplace=True)
    df.replace(
        {'education': {'初中': 0, '中专': 1, '大专': 2, '技术学校': 3, '高中': 4, '大
学本科': 5, '研究生': 6, '未知': 7,
                       '其他': 8}}, inplace=True)
    df.replace({'is_black': {'Y': 1, 'N': 0}}, inplace=True)
    df.replace({'is_contact': {'Y': 1, 'N': 0}}, inplace=True)
```

而对于星级评定，有两类缺失值：9e-6(2/232149)和4.7e-5(11/232149)，没有必要用KNN模型，直接替换即可。

```
# MEAN etc.
    df['sex'].fillna(df['sex'].mode().iloc[0], inplace=True)
    df['marriage'].fillna(df['marriage'].mode().iloc[0], inplace=True)
    df['education'].fillna(df['education'].mode().iloc[0], inplace=True)
    df['is_black'].fillna(df['is_black'].mode().iloc[0], inplace=True)
    df['is_contact'].fillna(df['is_contact'].mode().iloc[0], inplace=True)
    columns = ['avg_mth', 'avg_year', 'sa_bal', 'td_bal', 'fin_bal',
'sa_crd_bal', 'sa_td_bal', 'ntc_bal', 'td_3m_bal',
               'td_6m_bal', 'td_1y_bal', 'td_2y_bal', 'td_3y_bal', 'td_5y_bal',
'cd_bal', 'all_bal']
    for column in columns:
        df[column].fillna(df[column].median(), inplace=True)
```

3.异常值处理8'（含扩展3'）

3.1处理的异常值表

在处理之前，我们需要考虑异常值能出现在什么位置。

3.1.1用户星级

字段	分析
star_level	不会存在异常值
sex	不会存在异常值
marriage	不会存在异常值
education	不会存在异常值
is_black	不会存在异常值
is_contact	不会存在异常值
all_bal	存在异常值，但不在这一步处理
avg_mth	存在异常值，但不在这一步处理
avg_year	存在异常值，但不在这一步处理
sa_bal	存在异常值，但不在这一步处理
td_bal	存在异常值，但不在这一步处理
fin_bal	存在异常值，但不在这一步处理
sa_crd_bal	存在异常值，但不在这一步处理
sa_td_bal	存在异常值，但不在这一步处理
ntc_bal	存在异常值，但不在这一步处理
td_3m_bal	存在异常值，但不在这一步处理
td_6m_bal	存在异常值，但不在这一步处理
td_1y_bal	存在异常值，但不在这一步处理
td_2y_bal	存在异常值，但不在这一步处理
td_3y_bal	存在异常值，但不在这一步处理
td_5y_bal	存在异常值，但不在这一步处理
cd_bal	存在异常值，但不在这一步处理

3.1.2信用等级

不良余额指的是出现违约的贷款。一般而言，借款人若拖延还本利息达到三个月，贷款即被视为不良贷款。

欠息指的是贷款所欠的利息金额，一般不定。

正常余额指的是未违约的贷款。

逾期余额是不良贷款的一部分，属于不良贷款中回收希望最大的贷款。

字段	分析
credit_level	不会存在异常值
sex	不会存在异常值
marriage	不会存在异常值
education	不会存在异常值
is_black	不会存在异常值
is_contact	不会存在异常值
all_bal	假定没有异常值，异常值处理不在这一步。
bad_bal	会存在异常值（bad_bal如果有一定等于all_bal），在一阶段去除
due_intr	会存在异常值（due_intr不会大于bad_bal），在一阶段去除
norm_bal	会存在异常值（norm_bal在没有不良贷款的情况下等于all_bal，如果有不良贷款，这个值要么等于0，要么等于bad_bal），在一阶段去除
delay_bal	会存在异常值（delay_bal不会大于bad_bal），在一阶段去除

3.2一阶段

3.2.1信用等级

对于 bad_bal 字段，如果 bad_bal 不等于 0，则一定等于 all_bal，否则为异常值，需要将其设置为 all_bal。
对于 due_intr 字段，如果其大于 bad_bal，则为异常值，需要将其替换为介于 0-bad_bal 之间的随机数。
对于 norm_bal 字段，如果 bad_bal 不为 0，则 norm_bal 要么为 0，要么等于 bad_bal，否则为异常值，需要将其设置为 bad_bal。
对于 delay_bal 字段，如果其大于 bad_bal，则为异常值，需要将其替换为介于 0-bad_bal 之间的随机数。

```
import pandas as pd
import numpy as np


def main():
    df = pd.read_csv('credit_train_land_credit_demo_phase5.csv')

    # 处理 bad_bal 字段
    df.loc[df['bad_bal'] != 0, 'bad_bal'] = df['all_bal']

    # 处理 due_intr 字段
    mask = df['due_intr'] > df['bad_bal']
    rows = df.loc[mask].index
    df.loc[rows, 'due_intr'] = np.random.uniform(0, df.loc[mask, 'bad_bal'])

    # 处理 norm_bal 字段
    mask = (df['bad_bal'] != 0) & (df['norm_bal'] != 0) & (df['norm_bal'] !=
df['bad_bal'])
    rows = df.loc[mask].index
```

```
    df.loc[rows, 'norm_bal'] = df.loc[mask, 'bad_bal']

    # 处理 delay_bal 字段
    mask = df['delay_bal'] > df['bad_bal']
    rows = df.loc[mask].index
    df.loc[rows, 'delay_bal'] = np.random.uniform(0, df.loc[mask, 'bad_bal'])

    df.to_csv('credit_train_land_credit_demo_phase6.csv')


if __name__ == '__main__':
    main()
```

4.数据转换4'（含扩展4'）

```
转换前先将非数值型变量恢复
import pandas as pd


def main():
    df1 = pd.read_csv('credit_train_land_star_demo_phase4.csv', usecols=
['star_level', 'sex', 'marriage', 'education', 'is_black', 'is_contact'])
    df2 = pd.read_csv('credit_train_land_star_demo_phase6.csv', usecols=
['all_bal', 'avg_mth', 'avg_year', 'sa_bal', 'td_bal', 'fin_bal', 'sa_crd_bal',
'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal', 'td_2y_bal',
'td_3y_bal', 'td_5y_bal', 'cd_bal'])
    df_new1 = pd.concat([df1, df2], axis=1)
    df_new1.to_csv('credit_train_land_star_demo_phase7.csv')


if __name__ == '__main__':
    main()

import pandas as pd


def main():
    df1 = pd.read_csv('credit_train_land_star_demo_phase4.csv', usecols=
['credit_level', 'sex', 'marriage', 'education', 'is_black', 'is_contact'])
    df2 = pd.read_csv('credit_train_land_star_demo_phase6.csv', usecols=
['all_bal', 'bad_bal', 'due_intr', 'norm_bal', 'delay_bal'])
    df_new1 = pd.concat([df1, df2], axis=1)
    df_new1.to_csv('credit_train_land_credit_demo_phase7.csv')


if __name__ == '__main__':
    main()
```

```
cate_list = ['sex', 'marriage', 'education', 'is_black', 'is_contact']
    co_list1 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'bad_bal', 'due_intr', 'norm_bal', 'delay_bal']
    co_list2 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'avg_mth', 'avg_year', 'sa_bal', 'td_bal', 'fin_bal', 'sa_crd_bal',
'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal', 'td_2y_bal',
'td_3y_bal', 'td_5y_bal', 'cd_bal']
    df1 = pd.read_csv('credit_train_land_credit_demo_phase7.csv')
    df2 = pd.read_csv('credit_train_land_star_demo_phase7.csv')
    df1 = df1.iloc[:, 1:]
    df2 = df2.iloc[:, 1:]
    le = LabelEncoder()
    df1[cate_list] = df1[cate_list].apply(le.fit_transform)
    df2[cate_list] = df2[cate_list].apply(le.fit_transform)
    df1.to_csv('credit_trans.csv', index=False)
    df2.to_csv('star_trans.csv', index=False)
```

5.数据标准化5'

```
scaler = StandardScaler()
    df1[co_list1] = scaler.fit_transform(df1[co_list1])
    df2[co_list2] = scaler.fit_transform(df2[co_list2])
    df1.to_csv('credit_std.csv', index=False)
    df2.to_csv('star_std.csv', index=False)
```

三、特征工程20'

1.在特征工程前就已经删除的列

1.1用户星级

1.1.1数据收集时期

合并前，对选择字段步骤中一些实际上不利于模型训练的加以删除。新的字段选择如下：

在星级评定中，使用**客户星级，基本信息，存款汇总信息，贷记卡开户**；在信用评定中，使用**信用等级，基本信息，贷记卡开户，贷款汇总信息。**

其中，存款帐号信息，贷款账号信息存在冗余，且贷款账号信息数据量过大；贷记卡分期所含数据过少（约400条），不具有判断性。

1.1.2数据预处理时期

card_sts~bankacct_bal：缺失率过高

1.2信用等级

1.2.1数据收集时期

合并前，对选择字段步骤中一些实际上不利于模型训练的加以删除。新的字段选择如下：

在星级评定中，使用**客户星级，基本信息，存款汇总信息，贷记卡开户**；在信用评定中，使用**信用等级，基本信息，贷记卡开户，贷款汇总信息。**

其中，存款帐号信息，贷款账号信息存在冗余，且贷款账号信息数据量过大；贷记卡分期所含数据过少（约400条），不具有判断性。

1.2.2数据预处理时期

card_sts~bankacct_bal：缺失率过高

在星级评估中，avg_qur,oth_td_bal,td_crd_bal为全0，并没有实际意义，应该去除。

2.变量相关性7.5'

2.1用户星级

	sex	marriage	education	is_black	is_contact	all_bal	avg_mth	avg_year	sa_bal	td_bal	fin_bal	sa_crd_bal	sa_td_bal	ntc_bal	td_3m_bal	td_6m_bal	td_1y_bal	td_2y_bal	td_3y_bal	td_5y_bal	cd_bal
sex	1	0.046986	-0.02875	0.031354	0.009029	0.018598	0.016565	0.017254	0.01211	0.01676	0.000337	0.011142	0.001841	-0.00094	-0.00295	0.005497	0.016159	0.00834	0.016903	-0.00122	-0.00047
marriage	0.046986	1	-0.02336	0.010844	0.025579	-0.08519	-0.07624	-0.07318	-0.04018	-0.07922	-0.01339	-0.03951	-0.00889	-0.00532	-0.00782	-0.01476	-0.07589	-0.04001	-0.07181	-0.00125	-0.0122
education	-0.02875	-0.02336	1	-0.07659	0.01365	-0.00667	-0.00793	-0.00865	-0.01715	-0.00216	-0.00441	-0.01741	0.005895	-0.00269	-0.00178	0.001764	0.00938	0.005419	-0.00245	-0.0093	-0.01192
is_black	0.031354	0.010844	-0.07659	1	0.034211	-0.01939	-0.01749	-0.01648	0.003128	-0.02176	0.001204	0.00304	-0.00064	0.007334	0.004284	-0.00225	-0.01716	-0.01196	-0.02307	-0.00157	-0.00346
is_contact	0.009029	0.025579	0.01365	0.034211	1	-0.00088	-0.00079	-0.00075	-0.00037	-0.00083	-0.00012	-0.00036	-8.36E-05	-7.06E-05	-8.90E-05	-0.00016	-0.00075	-0.00043	-0.00075	-4.04E-05	-0.00017
all_bal	0.018598	-0.08519	-0.00667	-0.01939	-0.00088	1	0.978011	0.934644	0.337469	0.962213	0.175202	0.333627	0.066433	0.130859	0.119611	0.098949	0.402781	0.181103	0.79594	0.581957	0.75552
avg_mth	0.016565	-0.07624	-0.00793	-0.01749	-0.00079	0.978011	1	0.973872	0.315966	0.947029	0.153203	0.312763	0.060049	0.101621	0.107464	0.086256	0.35862	0.161254	0.734957	0.69124	0.797062
avg_year	0.017254	-0.07318	-0.00865	-0.01648	-0.00075	0.934644	0.973872	1	0.26305	0.915207	0.144188	0.259064	0.054835	0.087943	0.075828	0.078521	0.327829	0.149242	0.692917	0.720631	0.789398
sa_bal	0.01211	-0.04018	-0.01715	0.003128	-0.00037	0.337469	0.315966	0.26305	1	0.098008	0.046872	0.982321	0.015068	0.060717	0.034863	0.017702	0.042949	0.01468	0.057646	0.076876	0.08833
td_bal	0.01676	-0.07922	-0.00216	-0.02176	-0.00083	0.962213	0.947029	0.915207	0.098008	1	0.046732	0.098401	0.067071	0.122084	0.118586	0.099523	0.416457	0.190018	0.832124	0.603507	0.78264
fin_bal	0.000337	-0.01339	-0.00441	0.001204	-0.00012	0.175202	0.153203	0.144188	0.046872	0.046732	1	0.047206	0.001955	0.019488	0.004726	0.015284	0.032018	0.002431	0.047015	0.003392	0.033774
sa_crd_bal	0.011142	-0.03951	-0.01741	0.00304	-0.00036	0.333627	0.312763	0.259064	0.982321	0.098401	0.047206	1	0.014799	0.061453	0.035428	0.017285	0.0421	0.014774	0.057487	0.078433	0.089012
sa_td_bal	0.001841	-0.00889	0.005895	-0.00064	-8.36E-05	0.066433	0.060049	0.054835	0.015068	0.067071	0.001955	0.014799	1	0.020842	0.005032	0.008492	0.015604	0.009011	0.008196	0.000186	0.002252
ntc_bal	-0.00094	-0.00532	-0.00269	0.007334	-7.06E-05	0.130859	0.101621	0.087943	0.060717	0.122084	0.019488	0.061453	0.020842	1	0.01534	0.005145	0.009877	0.000304	0.020597	-0.00017	0.016308
td_3m_bal	-0.00295	-0.00782	-0.00178	0.004284	-8.90E-05	0.119611	0.107464	0.075828	0.034863	0.118586	0.004726	0.035428	0.005032	0.01534	1	0.05609	0.016836	0.009868	0.014143	0.001159	0.012295
td_6m_bal	0.005497	-0.01476	0.001764	-0.00225	-0.00016	0.098949	0.086256	0.078521	0.017702	0.099523	0.015284	0.017285	0.008492	0.005145	0.05609	1	0.050922	0.020974	0.024392	0.000666	0.018809
td_1y_bal	0.016159	-0.07589	0.00938	-0.01716	-0.00075	0.402781	0.35862	0.327829	0.042949	0.416457	0.032018	0.0421	0.015604	0.009877	0.016836	0.050922	1	0.120693	0.121237	0.003083	0.109523
td_2y_bal	0.00834	-0.04001	0.005419	-0.01196	-0.00043	0.181103	0.161254	0.149242	0.01468	0.190018	0.002431	0.014774	0.009011	0.000304	0.009868	0.020974	0.120693	1	0.073318	0.000296	0.019391
td_3y_bal	0.016903	-0.07181	-0.00245	-0.02307	-0.00075	0.79594	0.734957	0.692917	0.057646	0.832124	0.047015	0.057487	0.008196	0.020597	0.014143	0.024392	0.121237	0.073318	1	0.259267	0.593372
td_5y_bal	-0.00122	-0.00125	-0.0093	-0.00157	-4.04E-05	0.581957	0.69124	0.720631	0.076876	0.603507	0.003392	0.078433	0.000186	-0.00017	0.001159	0.000666	0.003083	0.000296	0.259267	1	0.803641
cd_bal	-0.00047	-0.0122	-0.01192	-0.00346	-0.00017	0.75552	0.797062	0.789398	0.08833	0.78264	0.033774	0.089012	0.002252	0.016308	0.012295	0.018809	0.109523	0.019391	0.593372	0.803641	1

	all_bal	avg_mth	avg_year	sa_bal	td_bal	td_5y_bal
avg_mth	0.978011	1	0.973872		0.947029	
avg_year	0.934644	0.973872	1		0.915207	0.720631
td_bal	0.962213	0.947029	0.915207		1	
sa_crd_bal				0.982321		
td_3y_bal	0.79594	0.734957			0.832124	
td_5y_bal			0.720631			1
cd_bal	0.75552	0.797062	0.789398		0.78264	0.803641

第一步：剔除sa_crd_bal。这是因为绝大多数人的活期余额都是卡内活期余额（知识域层面），而且sa_bal和sa_crd_bal与其它变量关系不大（矩阵域）。

第二步：剔除td_3y_bal和td_5y_bal。这是因为大多数人的大额存单都会存3-5年，会在td_bal上加以展现（知识域层面），而且与td_3y_bal和td_5y_bal有关的基本都和td_bal有关。

	all_bal	avg_mth	avg_year	td_bal
avg_mth	0.978011	1	0.973872	0.947029
avg_year	0.934644	0.973872	1	0.915207
td_bal	0.962213	0.947029	0.915207	1
cd_bal	0.75552	0.797062	0.789398	0.78264

第三步：使用PCA主成分分析对其进行降维。（见3.）

2.2信用等级

	sex	marriage	education	is_black	is_contact	all_bal	bad_bal	due_intr	norm_bal	delay_bal
sex	1	-0.01113	0.001975	0.033631	0.003894	0.1871	0.026092	0.033891	0.186867	-0.00446
marriage	-0.01113	1	-0.07409	0.119028	0.031645	-0.03868	0.01125	0.008639	-0.03887	0.012473
education	0.001975	-0.07409	1	-0.0496	-0.00358	0.00251	-0.01845	0.015239	0.002517	-0.00561
is_black	0.033631	0.119028	-0.0496	1	-0.00209	-0.06776	0.327337	0.277539	-0.07071	0.174689
is_contact	0.003894	0.031645	-0.00358	-0.00209	1	-0.00352	-0.0007	-0.00059	-0.00352	-0.00037
all_bal	0.1871	-0.03868	0.00251	-0.06776	-0.00352	1	0.150228	0.074873	0.999269	0.123952
bad_bal	0.026092	0.01125	-0.01845	0.327337	-0.0007	0.150228	1	0.61929	0.144053	0.719991
due_intr	0.033891	0.008639	0.015239	0.277539	-0.00059	0.074873	0.61929	1	0.063883	0.627766
norm_bal	0.186867	-0.03887	0.002517	-0.07071	-0.00352	0.999269	0.144053	0.063883	1	0.114042
delay_bal	-0.00446	0.012473	-0.00561	0.174689	-0.00037	0.123952	0.719991	0.627766	0.114042	1

	all_bal	**bad_bal**
norm_bal	0.999269	
delay_bal		0.719991

方法：剔除norm_bal（原因同上），使用PCA主成分分析对bad_bal和delay_bal进行降维。

3.PCA5'（含扩展5'）

```
import pandas as pd
from sklearn.decomposition import PCA


def main():
    df1 = pd.read_csv('credit_std.csv')
    co_list1 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'bad_bal', 'due_intr', 'norm_bal', 'delay_bal']
    df1[co_list1].corr().to_csv('credit_corr.csv', index=False)
    df2 = pd.read_csv('star_std.csv')
    co_list2 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'avg_mth', 'avg_year', 'sa_bal', 'td_bal', 'fin_bal', 'sa_crd_bal',
'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal', 'td_2y_bal',
'td_3y_bal', 'td_5y_bal', 'cd_bal']
    df2[co_list2].corr().to_csv('star_corr.csv', index=False)

    df1 = df1.drop('norm_bal', axis=1)
    df2 = df2.drop('sa_crd_bal', axis=1).drop('td_3y_bal',
axis=1).drop('td_5y_bal', axis=1)

    data = df1.iloc[:, [7, 9]]
    pca = PCA(n_components=1)
    data_pca = pca.fit_transform(data)
    df1['bad&delay'] = data_pca
    df1.drop(df1.columns[[7, 9]], axis=1, inplace=True)
    df1.to_csv('credit_after_pca.csv', index=False)

    data = df2.iloc[:, [6, 7, 8, 10, 18]]
    pca = PCA(n_components=0.99)
    data_pca = pca.fit_transform(data)
    df2['pca_1'] = data_pca[:, 0]
    df2['pca_2'] = data_pca[:, 1]
    df2['pca_3'] = data_pca[:, 2]
    df2.drop(df2.columns[[6, 7, 8, 10, 18]], axis=1, inplace=True)
    df2.to_csv('star_after_pca.csv', index=False)
```

```
    df1 = pd.read_csv('credit_after_pca.csv')
    co_list1 = ['credit_level', 'sex', 'marriage', 'education', 'is_black',
'is_contact', 'all_bal', 'due_intr', 'bad&delay']
    df1[co_list1].corr().to_csv('credit_corr_after_pca_corr.csv')
    df2 = pd.read_csv('star_after_pca.csv')
    co_list2 = ['star_level', 'sex', 'marriage', 'education', 'is_black',
'is_contact', 'sa_bal', 'fin_bal', 'sa_td_bal', 'ntc_bal', 'td_3m_bal',
'td_6m_bal', 'td_1y_bal', 'td_2y_bal', 'pca_1', 'pca_2', 'pca_3']
    df2[co_list2].corr().to_csv('star_corr_after_pca_corr.csv')


if __name__ == '__main__':
    main()
```

4.多重共线性7.5'

```
import pandas as pd
from sklearn.decomposition import PCA
from statsmodels.stats.outliers_influence import variance_inflation_factor

    df1 = pd.read_csv('credit_after_pca.csv')
    co_list1 = ['credit_level', 'sex', 'marriage', 'education', 'is_black',
'is_contact', 'all_bal', 'due_intr',
                'bad&delay']
    df1[co_list1].corr().to_csv('credit_corr_after_pca_corr.csv')
    df2 = pd.read_csv('star_after_pca.csv')
    co_list2 = ['star_level', 'sex', 'marriage', 'education', 'is_black',
'is_contact', 'sa_bal', 'fin_bal',
                'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal',
'td_2y_bal', 'pca_1', 'pca_2', 'pca_3']
    df2[co_list2].corr().to_csv('star_corr_after_pca_corr.csv')

    y = df1['credit_level']
    x = df1.drop(['credit_level'], axis=1)

    vif = pd.DataFrame()
    vif["VIF Factor"] = [variance_inflation_factor(x.values, i) for i in
range(x.shape[1])]
    vif["features"] = x.columns
    print(vif)

    y = df2['star_level']
    x = df2.drop(['star_level'], axis=1)

    vif = pd.DataFrame()
    vif["VIF Factor"] = [variance_inflation_factor(x.values, i) for i in
range(x.shape[1])]
    vif["features"] = x.columns
    print(vif)
```

	VIF Factor	features
0	1.005371	sex
1	1.018627	marriage
2	1.007965	education
3	1.009277	is_black
4	1.002155	is_contact
5	1.326804	sa_bal
6	1.092360	fin_bal
7	1.014771	sa_td_bal
8	1.056371	ntc_bal
9	1.052107	td_3m_bal
10	1.022307	td_6m_bal
11	1.461503	td_1y_bal
12	1.094913	td_2y_bal
13	1.388144	pca_1
14	1.507936	pca_2
15	1.290591	pca_3

	VIF Factor	features
0	1.041044	sex
1	1.021743	marriage
2	1.008709	education
3	1.132841	is_black
4	1.001074	is_contact
5	1.076819	all_bal
6	1.864691	due_intr
7	1.895788	bad&delay

四、特征选择0'

	star_level	sex	marriage	education	is_black	is_contact	sa_bal	fin_bal	sa_td_bal	ntc_bal	td_3m_bal	td_6m_bal	td_1y_bal	td_2y_bal	pca_1	pca_2	pca_3
star_level	1	0.006168	-0.20954	0.003974	-0.05004	-0.00249	0.224644	0.101974	0.046502	0.054283	0.054301	0.096707	0.424813	0.236233	0.477875	-0.51993	-0.08046
sex	0.006168	1	0.046986	-0.02875	0.031354	0.009029	0.01211	0.000337	0.001841	-0.00094	-0.00295	0.005497	0.016159	0.00834	0.014756	-0.02571	0.002025
marriage	-0.20954	0.046986	1	-0.02336	0.010844	0.025579	-0.04018	-0.01339	-0.00889	-0.00532	-0.00782	-0.01476	-0.07589	-0.04001	-0.06966	0.093991	0.011282
education	0.003974	-0.02875	-0.02336	1	-0.07659	0.01365	-0.01715	-0.00441	0.005895	-0.00269	-0.00178	0.001764	0.00938	0.005419	-0.00772	-0.01034	-0.0136
is_black	-0.05004	0.031354	0.010844	-0.07659	1	0.034211	0.003128	0.001204	-0.00064	0.007334	0.004284	-0.00225	-0.01716	-0.01196	-0.01677	0.021491	0.010702
is_contact	-0.00249	0.009029	0.025579	0.01365	0.034211	1	-0.00037	-0.00012	-8.36E-05	-7.06E-05	-8.90E-05	-0.00016	-0.00075	-0.00043	-0.00073	0.000903	0.000156
sa_bal	0.224644	0.01211	-0.04018	-0.01715	0.003128	-0.00037	1	0.046872	0.015068	0.060717	0.034863	0.017702	0.042949	0.01468	0.234808	-0.23579	0.345441
fin_bal	0.101974	0.000337	-0.01339	-0.00441	0.001204	-0.00012	0.046872	1	0.001955	0.019488	0.004726	0.015284	0.032018	0.002431	0.118005	-0.13904	0.199028
sa_td_bal	0.046502	0.001841	-0.00889	0.005895	-0.00064	-8.36E-05	0.015068	0.001955	1	0.020842	0.005032	0.008492	0.015604	0.009011	0.05373	-0.08639	-0.02303
ntc_bal	0.054283	-0.00094	-0.00532	-0.00269	0.007334	-7.06E-05	0.060717	0.019488	0.020842	1	0.01534	0.005145	0.009877	0.000304	0.098037	-0.13738	-0.08457
td_3m_bal	0.054301	-0.00295	-0.00782	-0.00178	0.004284	-8.90E-05	0.034863	0.004726	0.005032	0.01534	1	0.05609	0.016836	0.009868	0.092804	-0.1356	-0.09051
td_6m_bal	0.096707	0.005497	-0.01476	0.001764	-0.00225	-0.00016	0.017702	0.015284	0.008492	0.005145	0.05609	1	0.050922	0.020974	0.081506	-0.10173	-0.0458
td_1y_bal	0.424813	0.016159	-0.07589	0.00938	-0.01716	-0.00075	0.042949	0.032018	0.015604	0.009877	0.016836	0.050922	1	0.120693	0.343814	-0.36889	-0.19164
td_2y_bal	0.236233	0.00834	-0.04001	0.005419	-0.01196	-0.00043	0.01468	0.002431	0.009011	0.000304	0.009868	0.020974	0.120693	1	0.149925	-0.21552	-0.08242
pca_1	0.477875	0.014756	-0.06966	-0.00772	-0.01677	-0.00073	0.234808	0.118005	0.05373	0.098037	0.092804	0.081506	0.343814	0.149925	1	-1.27E-14	-6.96E-15
pca_2	-0.51993	-0.02571	0.093991	-0.01034	0.021491	0.000903	-0.23579	-0.13904	-0.08639	-0.13738	-0.1356	-0.10173	-0.36889	-0.21552	-1.27E-14	1	6.85E-15
pca_3	-0.08046	0.002025	0.011282	-0.0136	0.010702	0.000156	0.345441	0.199028	-0.02303	-0.08457	-0.09051	-0.0458	-0.19164	-0.08242	-6.96E-15	6.85E-15	1

	credit_leve	sex	marriage	education	is_black	is_contact	all_bal	due_intr	bad&delay
credit_leve	1	0.201711	-0.05938	0.020557	0.011009	-0.00379	0.172996	0.057346	0.024772
sex	0.201711	1	-0.01113	0.001975	0.033631	0.003894	0.1871	0.033891	0.011661
marriage	-0.05938	-0.01113	1	-0.07409	0.119028	0.031645	-0.03868	0.008639	0.012791
education	0.020557	0.001975	-0.07409	1	-0.0496	-0.00358	0.00251	0.015239	-0.01297
is_black	0.011009	0.033631	0.119028	-0.0496	1	-0.00209	-0.06776	0.277539	0.270675
is_contact	-0.00379	0.003894	0.031645	-0.00358	-0.00209	1	-0.00352	-0.00059	-0.00057
all_bal	0.172996	0.1871	-0.03868	0.00251	-0.06776	-0.00352	1	0.074873	0.147828
due_intr	0.057346	0.033891	0.008639	0.015239	0.277539	-0.00059	0.074873	1	0.67237
bad&delay	0.024772	0.011661	0.012791	-0.01297	0.270675	-0.00057	0.147828	0.67237	1

五、模型选择15'

1.自定义准确率函数

```
def accuracy_custom(y_true, y_pred, threshold=1):
    """
    自定义准确率函数，当预测值和真实值之间的差小于等于threshold时，认为预测正确
    :param y_true: 真实标签
    :param y_pred: 预测标签
    :param threshold: 误差阈值，默认为1
    :return: 准确率
    """
    n_samples = len(y_true)
    n_correct = sum(abs(y_true - y_pred) <= threshold)
    accuracy = n_correct / n_samples
    return accuracy
```

2.逻辑回归

```
lr = LogisticRegression(max_iter=500)  # 迭代到10000也只有0.02%的概率差
    lr.fit(x_train, y_train)
    y_pred_lr = lr.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_lr)
    accuracy2 = accuracy_custom(y_test, y_pred_lr)
    precision = precision_score(y_test, y_pred_lr, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_lr, average='macro')
    f1 = f1_score(y_test, y_pred_lr, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_lr)
    print('线性回归模型--误差为0准确率为：', accuracy, '，误差为1准确率为：', accuracy2,
'，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数
为：', kappa, "。")

lr = LogisticRegression(max_iter=500)  # 迭代到10000也只有0.02%的概率差
    lr.fit(x_train, y_train)
    y_pred_lr = lr.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_lr)
    accuracy2 = accuracy_custom(y_test, y_pred_lr, 15)
    precision = precision_score(y_test, y_pred_lr, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_lr, average='macro')
    f1 = f1_score(y_test, y_pred_lr, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_lr)
    print('线性回归模型--误差为0准确率为：', accuracy, '，误差为15准确率为：',
accuracy2, '，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，
Kappa系数为：', kappa, "。")
```

3.决策树

```
dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    y_pred_dt = dt.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_dt)
    accuracy2 = accuracy_custom(y_test, y_pred_dt)
    precision = precision_score(y_test, y_pred_dt, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_dt, average='macro')
```

```
    f1 = f1_score(y_test, y_pred_dt, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_dt)
    print('决策树模型--误差为0准确率为：', accuracy, '，误差为1准确率为：', accuracy2,
'，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数
为：', kappa, "。")

dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    y_pred_dt = dt.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_dt)
    accuracy2 = accuracy_custom(y_test, y_pred_dt, 15)
    precision = precision_score(y_test, y_pred_dt, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_dt, average='macro')
    f1 = f1_score(y_test, y_pred_dt, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_dt)
    print('决策树模型--误差为0准确率为：', accuracy, '，误差为15准确率为：', accuracy2,
'，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数
为：', kappa, "。")
```

4.随机森林

```
rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred_rf = rf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_rf)
    accuracy2 = accuracy_custom(y_test, y_pred_rf)
    precision = precision_score(y_test, y_pred_rf, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_rf, average='macro')
    f1 = f1_score(y_test, y_pred_rf, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_rf)
    print('随机森林模型--误差为0准确率为：', accuracy, '，误差为1准确率为：', accuracy2,
'，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数
为：', kappa, "。")

rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred_rf = rf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred_rf)
    accuracy2 = accuracy_custom(y_test, y_pred_rf, 15)
    precision = precision_score(y_test, y_pred_rf, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_rf, average='macro')
    f1 = f1_score(y_test, y_pred_rf, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_rf)
    print('随机森林模型--误差为0准确率为：', accuracy, '，误差为15准确率为：',
accuracy2, '，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，
Kappa系数为：', kappa, "。")
```

5.XGBoost

```
xgb = XGBClassifier()
    le = LabelEncoder()
    y_train = le.fit_transform(y_train)
```

```
    xgb.fit(x_train, y_train)
    y_pred_xgb = xgb.predict(x_test)
    y_pred_xgb += 1
    accuracy = accuracy_score(y_test, y_pred_xgb)
    accuracy2 = accuracy_custom(y_test, y_pred_xgb)
    precision = precision_score(y_test, y_pred_xgb, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_xgb, average='macro')
    f1 = f1_score(y_test, y_pred_xgb, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_xgb)
    print('XGBoost模型--误差为0准确率为：', accuracy, '，误差为1准确率为：', accuracy2,
'，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数
为：', kappa, "。")

xgb = XGBClassifier()
    le = LabelEncoder()
    y_train = le.fit_transform(y_train)
    xgb.fit(x_train, y_train)
    y_pred_xgb = xgb.predict(x_test)
    y_pred_xgb += 1
    y_pred_xgb = np.where(y_pred_xgb == 1, 35, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 2, 50, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 3, 60, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 4, 70, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 5, 85, y_pred_xgb)
    accuracy = accuracy_score(y_test, y_pred_xgb)
    accuracy2 = accuracy_custom(y_test, y_pred_xgb, 15)
    precision = precision_score(y_test, y_pred_xgb, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_xgb, average='macro')
    f1 = f1_score(y_test, y_pred_xgb, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_xgb)
    print('XGBoost模型--误差为0准确率为：', accuracy, '，误差为15准确率为：',
accuracy2, '，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，
Kappa系数为：', kappa, "。")
```

6.多模型拟合

```
w1 = 0.8278268360973509 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w2 = 0.8828774499246177 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w3 = 0.9021753176825329 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w4 = 0.9066982554382942 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    y_pred_weighted = (w1 * y_pred_lr + w2 * y_pred_dt + w3 * y_pred_rf + w4 *
y_pred_xgb)
    nearest_idx = np.abs(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]) -
y_pred_weighted.reshape(-1, 1)).argmin(axis=1)
    y_pred_weighted = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])[nearest_idx]
    accuracy = accuracy_score(y_test, y_pred_weighted)
    accuracy2 = accuracy_custom(y_test, y_pred_weighted)
    precision = precision_score(y_test, y_pred_weighted, average='macro',
zero_division=1)
    recall = recall_score(y_test, y_pred_weighted, average='macro')
    f1 = f1_score(y_test, y_pred_weighted, average='macro')
```

```
    kappa = cohen_kappa_score(y_test, y_pred_weighted)
    print('加权模型--误差为0准确率为：', accuracy, '，误差为1准确率为：', accuracy2, '，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数为：', kappa, "。")

w1 = 0.7286135693215339 / (0.7286135693215339 + 0.8964274008521796 + 0.8970829236315963 + 0.9062602425434284)
    w2 = 0.8964274008521796 / (0.7286135693215339 + 0.8964274008521796 + 0.8970829236315963 + 0.9062602425434284)
    w3 = 0.8970829236315963 / (0.7286135693215339 + 0.8964274008521796 + 0.8970829236315963 + 0.9062602425434284)
    w4 = 0.9062602425434284 / (0.7286135693215339 + 0.8964274008521796 + 0.8970829236315963 + 0.9062602425434284)
    y_pred_weighted = (w1 * y_pred_lr + w2 * y_pred_dt + w3 * y_pred_rf + w4 * y_pred_xgb)
    nearest_idx = np.abs(np.array([35, 50, 60, 70, 85]) - y_pred_weighted.reshape(-1, 1)).argmin(axis=1)
    y_pred_weighted = np.array([35, 50, 60, 70, 85])[nearest_idx]
    accuracy = accuracy_score(y_test, y_pred_weighted)
    accuracy2 = accuracy_custom(y_test, y_pred_weighted, 15)
    precision = precision_score(y_test, y_pred_weighted, average='macro', zero_division=1)
    recall = recall_score(y_test, y_pred_weighted, average='macro')
    f1 = f1_score(y_test, y_pred_weighted, average='macro')
    kappa = cohen_kappa_score(y_test, y_pred_weighted)
    print('加权模型--误差为0准确率为：', accuracy, '，误差为15准确率为：', accuracy2, '，精确率为：', precision, '，召回率为：', recall, '，F1分数为：', f1, '，Kappa系数为：', kappa, "。")
```

六、模型训练0'

详见五、模型选择与七、模型评估

七、模型评估15'

1.用户星级

1.1分数评定

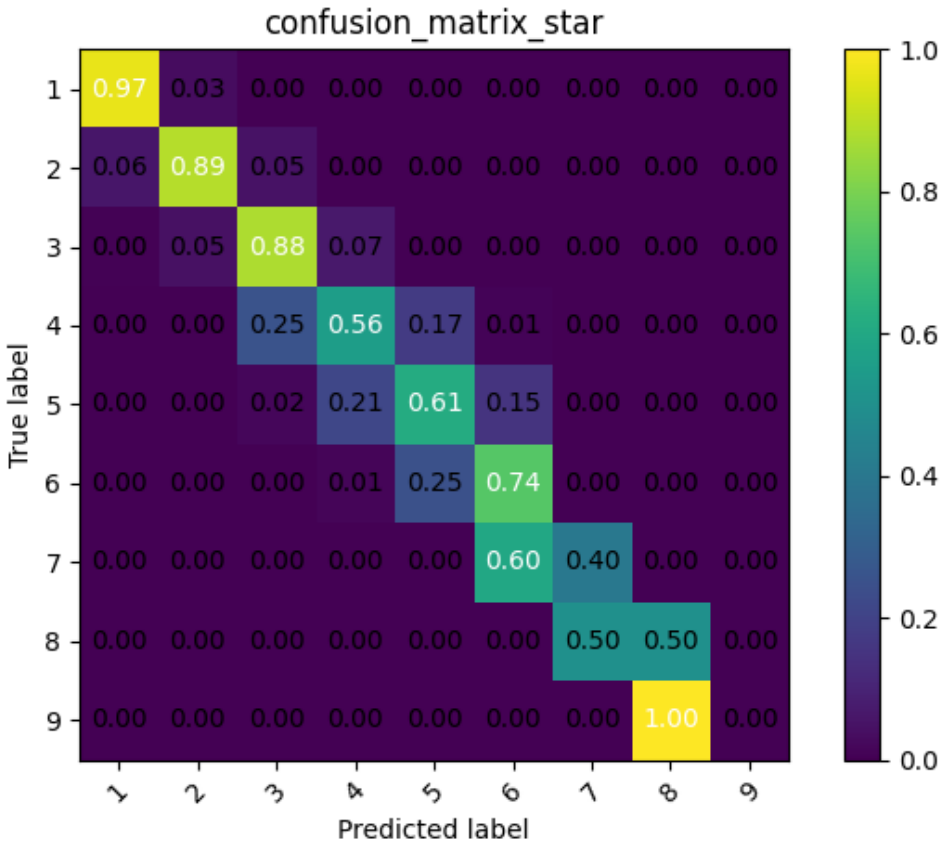
```
----------star_level----------
线性回归模型--误差为0准确率为： 0.8278268360973509 ，误差为1准确率为： 0.996338574197717 ，精确率为： 0.5252971427470274 ，召回率为： 0.4925117750889594 ，F1分数为： 0.49727996971560956 ，Kappa系数为： 0.7125656974136556 。
决策树模型--误差为0准确率为： 0.8826620719362481 ，误差为1准确率为： 0.9925479216024122 ，精确率为： 0.689253102521183 ，召回率为： 0.5834377346795075 ，F1分数为： 0.580435950790857 ，Kappa系数为： 0.8123061491757301 。
随机森林模型--误差为0准确率为： 0.9032952832220547 ，误差为1准确率为： 0.9956924402326083 ，精确率为： 0.712095024430682 ，召回率为： 0.5912191118123533 ，F1分数为： 0.5940741019680458 ，Kappa系数为： 0.8458930688809023 。
XGBoost模型--误差为0准确率为： 0.9066982554382942 ，误差为1准确率为： 0.996338574197717 ，精确率为： 0.7199758823977854 ，召回率为： 0.6430072303786168 ，F1分数为： 0.6226383657400738 ，Kappa系数为： 0.8515599308614064 。
加权模型--误差为0准确率为： 0.9048460047383158 ，误差为1准确率为： 0.996338574197717 ，精确率为： 0.7214727762146035 ，召回率为： 0.6156200731297458 ，F1分数为： 0.612890248041405 ，Kappa系数为： 0.8485080439872792 。
```

	准确率（误差0）	准确率（误差1）	精确率	召回率	F1	Kappa
线性回归	0.8278268360973509	0.996338574197717	0.5252971427470274	0.4925117750889594	0.49727996971560956	0.7125656974136556
决策树	0.8826620719362481	0.9925479216024122	0.689253102521183	0.5834377346795075	0.580435950790857	0.8123061491757301
随机森林	0.9032952832220547	0.9956924402326083	0.712095024430682	0.5912191118123533	0.5940741019680458	0.8458930688809023
XGBoost	0.9066982554382942	0.996338574197717	0.7199758823977854	0.6430072303786168	0.6226383657400738	0.8515599308614064
加权	0.9048460047383158	0.996338574197717	0.7214727762146035	0.6156200731297458	0.612890248041405	0.8485080439872792

1.2选择模型

选择加权模型，它在准确率第二的基础上，拥有最高的精确率，且召回率等与最高值相差不大，对于极端值效果更好。

1.3混淆矩阵



2.信用等级

2.1分数评定

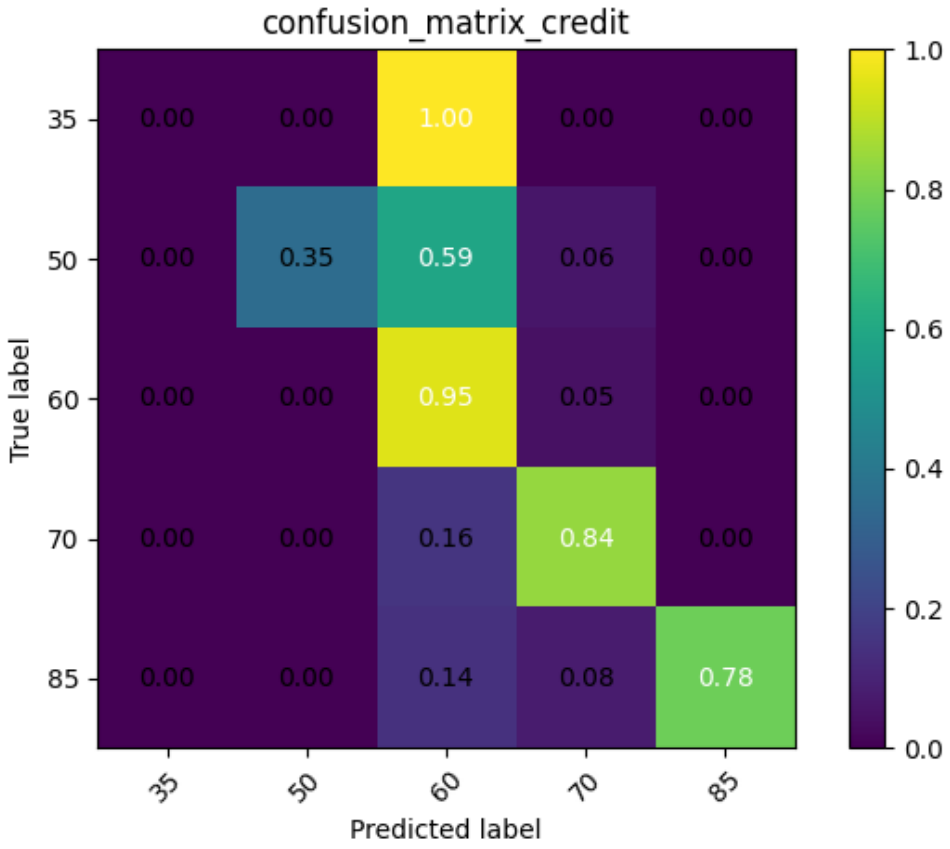
```
----------------------------------------------------------------credit_level----------------------------------------------------------------
线性回归模型——误差为0准确率为： 0.7286135693215339 ，误差为15准确率为： 0.8797115699770567 ，精确率为： 0.9127056227525335 ，召回率为： 0.2977475350306544 ，F1分数为： 0.29248026068429633 ，Kappa系数为： 0.3871142576964077 。
决策树模型——误差为0准确率为： 0.8941330711242216 ，误差为15准确率为： 0.9741068502130449 ，精确率为： 0.654946736526856 ，召回率为： 0.5868029165684463 ，F1分数为： 0.6137598129870839 ，Kappa系数为： 0.8008538360482014 。
随机森林模型——误差为0准确率为： 0.8947885939036382 ，误差为15准确率为： 0.9744346116027532 ，精确率为： 0.8865459636093671 ，召回率为： 0.5851568276196497 ，F1分数为： 0.6205803821574827 ，Kappa系数为： 0.8011565906745892 。
XGBoost模型——误差为0准确率为： 0.9062602425434284 ，误差为15准确率为： 0.9757456571615863 ，精确率为： 0.8908186096096481 ，召回率为： 0.5860478131806585 ，F1分数为： 0.6257909600178733 ，Kappa系数为： 0.8196119258151962 。
加权模型——误差为0准确率为： 0.8987217305801377 ，误差为15准确率为： 0.9803343166175025 ，精确率为： 0.8995670759197585 ，召回率为： 0.5852247131274819 ，F1分数为： 0.6253907610150444 ，Kappa系数为： 0.8078738857708815 。
```

	准确率（误差0）	准确率（误差1）	精确率	召回率	F1	Kappa
线性回归	0.7286135693215339	0.8797115699770567	0.9127056227525335	0.2977475350306544	0.29248026068429633	0.3871142576964077
决策树	0.8941330711242216	0.9741068502130449	0.654946736526856	0.5868029165684463	0.6137598129870839	0.8008538360482014
随机森林	0.8947885939036382	0.9744346116027532	0.8865459636093671	0.5851568276196497	0.6205803821574827	0.8011565906745892
XGBoost	0.9062602425434284	0.9757456571615863	0.8908186096096481	0.5860478131806585	0.6257909600178733	0.8196119258151962
加权	0.8987217305801377	0.9803343166175025	0.8995670759197585	0.5852247131274819	0.6253907610150444	0.8078738857708815

2.2选择模型

选择加权模型，它在准确率第二的基础上，拥有最高的准确率（误差1），拥有最高的精确率，且召回率等与最高值相差不大，对于极端值效果更好。

2.3混淆矩阵



八、模型应用10'

1.回忆所做的处理并用于未打分数据

1.1用户星级

在phase2的基础上，选择uid，star_level，sex，marriage，education，is_black，is_contact，sa_bal，fin_bal，sa_td_bal，ntc_bal，td_3m_bal，td_6m_bal，td_1y_bal，td_2y_bal，all_bal，avg_mth，avg_year，td_bal，cd_bal。

进行缺失值处理；

进行数据转换和标准化；

特征工程PCA；

摘取-1作为待测试项，其它作为训练项。

1.2信用等级

在phase2的基础上，选择uid，credit_level，sex，marriage，education，is_black，is_contact，all_bal，bad_bal，due_intr，delay_bal；

进行缺失值处理；

进行异常值处理；

进行数据转换和标准化；

特征工程PCA；

摘取-1作为待测试项，其它作为训练项。

2.处理

```
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier


def star():
    df = pd.read_csv('credit_train_land_star_demo_phase2.csv')
    all_list = ['uid', 'star_level', 'sex', 'marriage', 'education', 'is_black',
'is_contact', 'sa_bal', 'fin_bal',
                'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal',
'td_2y_bal', 'all_bal', 'avg_mth',
                'avg_year', 'td_bal', 'cd_bal']
    df = df.loc[:, all_list]

    df['sex'].fillna(df['sex'].mode().iloc[0], inplace=True)
    df['marriage'].fillna(df['marriage'].mode().iloc[0], inplace=True)
    df['education'].fillna(df['education'].mode().iloc[0], inplace=True)
    df['is_black'].fillna(df['is_black'].mode().iloc[0], inplace=True)
    df['is_contact'].fillna(df['is_contact'].mode().iloc[0], inplace=True)
    columns = ['avg_mth', 'avg_year', 'sa_bal', 'td_bal', 'fin_bal', 'sa_td_bal',
'ntc_bal', 'td_3m_bal', 'td_6m_bal',
               'td_1y_bal', 'td_2y_bal', 'cd_bal', 'all_bal']
    for column in columns:
        df[column].fillna(df[column].median(), inplace=True)

    cate_list = ['sex', 'marriage', 'education', 'is_black', 'is_contact']
    co_list = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'sa_bal', 'fin_bal',
               'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal',
'td_2y_bal', 'all_bal', 'avg_mth',
               'avg_year', 'td_bal', 'cd_bal']
    le = LabelEncoder()
    df[cate_list] = df[cate_list].apply(le.fit_transform)
    scaler = StandardScaler()
    df[co_list] = scaler.fit_transform(df[co_list])

    data = df.iloc[:, [15, 16, 17, 18, 19]]
    pca = PCA(n_components=0.99)
    data_pca = pca.fit_transform(data)
    df['pca_1'] = data_pca[:, 0]
    df['pca_2'] = data_pca[:, 1]
    df['pca_3'] = data_pca[:, 2]
    df.drop(df.columns[[15, 16, 17, 18, 19]], axis=1, inplace=True)

    df_neg = df.loc[df['star_level'] == -1]
    df_pos = df.loc[df['star_level'] != -1]
```

```
    co_list1 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'sa_bal', 'fin_bal',
                'sa_td_bal', 'ntc_bal', 'td_3m_bal', 'td_6m_bal', 'td_1y_bal',
'td_2y_bal', 'pca_1', 'pca_2', 'pca_3']
    x_train = df_pos[co_list1]
    y_train = df_pos['star_level']
    x_test = df_neg[co_list1]

    lr = LogisticRegression(max_iter=500)  # 迭代到10000也只有0.02%的概率差
    lr.fit(x_train, y_train)
    y_pred_lr = lr.predict(x_test)
    dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    y_pred_dt = dt.predict(x_test)
    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred_rf = rf.predict(x_test)
    xgb = XGBClassifier()
    le = LabelEncoder()
    y_train = le.fit_transform(y_train)
    xgb.fit(x_train, y_train)
    y_pred_xgb = xgb.predict(x_test)
    y_pred_xgb += 1
    w1 = 0.8278268360973509 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w2 = 0.8828774499246177 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w3 = 0.9021753176825329 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    w4 = 0.9066982554382942 / (0.8278268360973509 + 0.8828774499246177 +
0.9021753176825329 + 0.9066982554382942)
    y_pred_weighted = (w1 * y_pred_lr + w2 * y_pred_dt + w3 * y_pred_rf + w4 *
y_pred_xgb)
    nearest_idx = np.abs(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]) -
y_pred_weighted.reshape(-1, 1)).argmin(axis=1)
    y_pred_weighted = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])[nearest_idx]
    df_neg['star_level'] = y_pred_weighted

    df_neg.to_csv('result_star.csv')


def credit():
    df = pd.read_csv('credit_train_land_credit_demo_phase2.csv')
    all_list = ['uid', 'credit_level', 'sex', 'marriage', 'education',
'is_black', 'is_contact', 'all_bal', 'bad_bal', 'due_intr', 'delay_bal']
    df = df.loc[:, all_list]

    missing_cols = ['all_bal', 'bad_bal', 'due_intr', 'delay_bal']
    cate_list = ['sex', 'marriage', 'education', 'is_black', 'is_contact']
    co_list = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'bad_bal', 'due_intr', 'delay_bal']
    le = LabelEncoder()
    df[cate_list] = df[cate_list].apply(le.fit_transform)
    for target_col in missing_cols:
        feature_cols = ['credit_level', 'sex', 'marriage', 'education',
'is_black', 'is_contact']
        x = df.dropna()[feature_cols]
```

```
        y = df[target_col].dropna()
        knn_model = KNeighborsRegressor(n_neighbors=5)
        knn_model.fit(x, y)
        missing_rows = df[df[target_col].isnull()]
        predicted_values = knn_model.predict(missing_rows[feature_cols])
        df.loc[df[target_col].isnull(), target_col] = predicted_values

    df.loc[df['bad_bal'] != 0, 'bad_bal'] = df['all_bal']
    mask = df['due_intr'] > df['bad_bal']
    rows = df.loc[mask].index
    df.loc[rows, 'due_intr'] = np.random.uniform(0, df.loc[mask, 'bad_bal'])
    mask = df['delay_bal'] > df['bad_bal']
    rows = df.loc[mask].index
    df.loc[rows, 'delay_bal'] = np.random.uniform(0, df.loc[mask, 'bad_bal'])

    scaler = StandardScaler()
    df[co_list] = scaler.fit_transform(df[co_list])

    data = df.iloc[:, [8, 10]]
    pca = PCA(n_components=1)
    data_pca = pca.fit_transform(data)
    df['bad&delay'] = data_pca
    df.drop(df.columns[[8, 10]], axis=1, inplace=True)

    df_neg = df.loc[df['credit_level'] == -1]
    df_pos = df.loc[df['credit_level'] != -1]

    co_list1 = ['sex', 'marriage', 'education', 'is_black', 'is_contact',
'all_bal', 'due_intr', 'bad&delay']
    x_train = df_pos[co_list1]
    y_train = df_pos['credit_level']
    x_test = df_neg[co_list1]

    lr = LogisticRegression(max_iter=500)  # 迭代到10000也只有0.02%的概率差
    lr.fit(x_train, y_train)
    y_pred_lr = lr.predict(x_test)
    dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    y_pred_dt = dt.predict(x_test)
    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred_rf = rf.predict(x_test)
    xgb = XGBClassifier()
    le = LabelEncoder()
    y_train = le.fit_transform(y_train)
    xgb.fit(x_train, y_train)
    y_pred_xgb = xgb.predict(x_test)
    y_pred_xgb += 1
    y_pred_xgb = np.where(y_pred_xgb == 1, 35, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 2, 50, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 3, 60, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 4, 70, y_pred_xgb)
    y_pred_xgb = np.where(y_pred_xgb == 5, 85, y_pred_xgb)
    w1 = 0.7286135693215339 / (0.7286135693215339 + 0.8964274008521796 +
0.8970829236315963 + 0.9062602425434284)
    w2 = 0.8964274008521796 / (0.7286135693215339 + 0.8964274008521796 +
0.8970829236315963 + 0.9062602425434284)
```

```
    w3 = 0.8970829236315963 / (0.7286135693215339 + 0.8964274008521796 +
0.8970829236315963 + 0.9062602425434284)
    w4 = 0.9062602425434284 / (0.7286135693215339 + 0.8964274008521796 +
0.8970829236315963 + 0.9062602425434284)
    y_pred_weighted = (w1 * y_pred_lr + w2 * y_pred_dt + w3 * y_pred_rf + w4 *
y_pred_xgb)
    nearest_idx = np.abs(np.array([35, 50, 60, 70, 85]) -
y_pred_weighted.reshape(-1, 1)).argmin(axis=1)
    y_pred_weighted = np.array([35, 50, 60, 70, 85])[nearest_idx]
    df_neg['credit_level'] = y_pred_weighted

    df_neg.to_csv('result_credit.csv')


if __name__ == '__main__':
    star()
    credit()
```

九、结果

1.完成分数

完成了数据集成第三次作业要求的100分全部内容。

2.代码解释

```
feature.py 特征工程
missing_rate.py 统计缺失率并填充缺失值
outliers.py 异常值处理
predict.py 模型应用
show.py 数据盘点
showgraph.py 可视化
switch2mongo.py 写入mongo
switch2mysql.py 写入mysql
train.py 模型训练
transform.py 数据转化与标准化
```

3.中间数据

```
提交附件的不同phase对应如下：

一、数据收集
phase1          信用等级/用户星级与个人基本信息组合，保证完整率近100%;
phase2          拟定要分析的属性（包括含-1的行）
phase3          能够量化分析频数的属性（包括含-1的行）
describe        数据盘点

二、数据预处理
phase4          phase2(不包括含-1的行)
missing_rate    缺失率
phase5          在phase4的基础上添加了缺失值处理
phase6          在phase5的基础上添加了异常值处理
phase7          综合phase4和phase6，将非数值恢复
trans           数据转换后
```

```
std                 数据标准化后

三、特征工程
corr                相关系数
after_pca           pca后数据
corr_after_pca      pca后相关系数
```

4.结果

详见最终结果（result_credit.py result_star.py）