German 信用评分卡

数据获取

1）导包

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
matplotlib.rc("font", family='Microsoft YaHei')
plt.rcParams["font.sans-serif"]=['SimHei']
plt.rcParams["axes.unicode_minus"]=False
plt.rcParams.update({'font.size':13 }) #全局字体大小
plt.rcParams["font.family"]=['SimHei']
plt.rcParams['axes.unicode_minus']=False
```

2）按照格式读取数据集合

```
column = ['账户状态','续存期数','信用记录','目的','信用额度','储蓄情况',
          '工作年限','分期利率','性别及婚姻情况','担保情况','现居时间',
          '财产情况','年龄','其他分期计划','住房','信用卡数量','工作状况',
          '法人数量','是否有电话','外籍工作者','违约']
```

```
df = pd.read_csv('./数据/german.data',sep=' ',header=None)
df.columns = column
df
```

	账户状态	续存期数	信用记录	目的	信用额度	储蓄情况	工作年限	分期利率	性别及婚姻情况	担保情况	...	财产情况	年龄	其他分期计划	住房	信用卡数量	工作状况	法人数量	是否有电话	外籍工作者	违约
0	A11	6	A34	A43	1169	A65	A75	4	A93	A101	...	A121	67	A143	A152	2	A173	1	A192	A201	1
1	A12	48	A32	A43	5951	A61	A73	2	A92	A101	...	A121	22	A143	A152	1	A173	1	A191	A201	2
2	A14	12	A34	A46	2096	A61	A74	2	A93	A101	...	A121	49	A143	A152	1	A172	2	A191	A201	1
3	A11	42	A32	A42	7882	A61	A74	2	A93	A103	...	A122	45	A143	A153	1	A173	2	A191	A201	1
4	A11	24	A33	A40	4870	A61	A73	3	A93	A101	...	A124	53	A143	A153	2	A173	2	A191	A201	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	A14	12	A32	A42	1736	A61	A74	3	A92	A101	...	A121	31	A143	A152	1	A172	1	A191	A201	1
996	A11	30	A32	A41	3857	A61	A73	4	A91	A101	...	A122	40	A143	A152	1	A174	1	A192	A201	1
997	A14	12	A32	A43	804	A61	A75	4	A93	A101	...	A123	38	A143	A152	1	A173	1	A191	A201	1
998	A11	45	A32	A43	1845	A61	A73	4	A93	A101	...	A124	23	A143	A153	1	A173	1	A192	A201	2
999	A12	45	A34	A41	4576	A62	A71	3	A93	A101	...	A123	27	A143	A152	1	A173	1	A191	A201	1

1000 rows × 21 columns

数据处理

1）目标变量'违约'的处理

```
#将'逾期'标签变量转为0，1（0：好；1：差）
df['违约'] = df['违约']-1
df
```

财产情况	年龄	其他分期计划	住房	信用卡数量	工作状况	法人数量	是否有电话	外籍工作者	违约
A121	67	A143	A152	2	A173	1	A192	A201	0
A121	22	A143	A152	1	A173	1	A191	A201	1
A121	49	A143	A152	1	A172	2	A191	A201	0
A122	45	A143	A153	1	A173	2	A191	A201	0
A124	53	A143	A153	2	A173	2	A191	A201	1
...	...	...	...	...	...	...	...	...	...
A121	31	A143	A152	1	A172	1	A191	A201	0
A122	40	A143	A152	1	A174	1	A192	A201	0
A123	38	A143	A152	1	A173	1	A191	A201	0
A124	23	A143	A153	1	A173	1	A192	A201	1
A123	27	A143	A152	1	A173	1	A191	A201	0

2）数据集整体信息

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   账户状态     1000 non-null   object
 1   续存期数     1000 non-null   int64
 2   信用记录     1000 non-null   object
 3   目的       1000 non-null   object
 4   信用额度     1000 non-null   int64
 5   储蓄情况     1000 non-null   object
 6   工作年限     1000 non-null   object
 7   分期利率     1000 non-null   int64
 8   性别及婚姻情况  1000 non-null   object
 9   担保情况     1000 non-null   object
 10  现居时间     1000 non-null   int64
 11  财产情况     1000 non-null   object
 12  年龄       1000 non-null   int64
 13  其他分期计划   1000 non-null   object
 14  住房       1000 non-null   object
 15  信用卡数量    1000 non-null   int64
 16  工作状况     1000 non-null   object
 17  法人数量     1000 non-null   int64
 18  是否有电话    1000 non-null   object
 19  外籍工作者    1000 non-null   object
 20  违约       1000 non-null   int64
dtypes: int64(8), object(13)
memory usage: 164.2+ KB
```

从上得知各列并没有缺失值

3）按照类型划分变量

```
#数值型变量
numerical = list(df.describe().columns)
#类别型变量
category = list(set(column)-set(numerical))
```

```
df_numerical = df.loc[:,numerical]
df_numerical.head()
```

	续存期数	信用额度	分期利率	现居时间	年龄	信用卡数量	法人数量	违约
0	0	0	4	4	3	2	1	0
1	2	1	2	2	0	1	1	1
2	0	0	2	3	1	1	2	0
3	2	1	2	4	1	1	2	0
4	1	1	3	4	2	2	2	1

```
category.append('违约')
df_category = df.loc[:,category]
df_category
```

	工作状况	住房	担保情况	其他分期计划	信用记录	财产情况	账户状态	是否有电话	外籍工作者	储蓄情况	性别及婚姻情况	工作年限	目的	违约
0	A173	A152	A101	A143	A34	A121	A11	A192	A201	A65	A93	A75	A43	0
1	A173	A152	A101	A143	A32	A121	A12	A191	A201	A61	A92	A73	A43	1
2	A172	A152	A101	A143	A34	A121	A14	A191	A201	A61	A93	A74	A46	0
3	A173	A153	A103	A143	A32	A122	A11	A191	A201	A61	A93	A74	A42	0
4	A173	A153	A101	A143	A33	A124	A11	A191	A201	A61	A93	A73	A40	1

4）对于类别型变量进行位符清洗

```
#对原本的类别型数据进行清洗，避免出现位符占位情况
df.loc[:,category[:-1]].applymap(lambda x:x.replace('\n','') )
df.loc[:,category[:-1]].applymap(lambda x:x.replace('\t','') )
df.loc[:,category[:-1]].applymap(lambda x:x.replace(' ','') )
```

	工作状况	住房	担保情况	其他分期计划	信用记录	财产情况	账户状态	是否有电话	外籍工作者	储蓄情况	性别及婚姻情况	工作年限	目的
0	A173	A152	A101	A143	A34	A121	A11	A192	A201	A65	A93	A75	A43
1	A173	A152	A101	A143	A32	A121	A12	A191	A201	A61	A92	A73	A43
2	A172	A152	A101	A143	A34	A121	A14	A191	A201	A61	A93	A74	A46
3	A173	A153	A103	A143	A32	A122	A11	A191	A201	A61	A93	A74	A42
4	A173	A153	A101	A143	A33	A124	A11	A191	A201	A61	A93	A73	A40

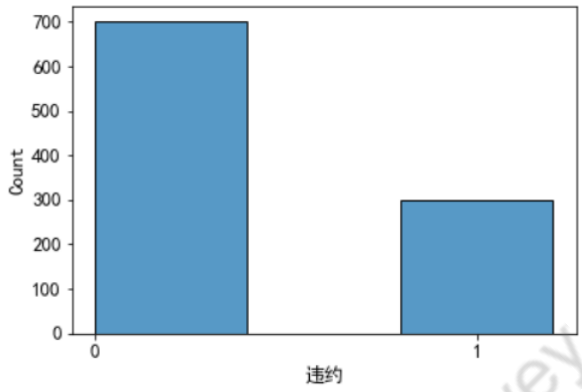
EDA

常用的探索性数据分析方法有：直方图、散点图和箱线图等。例如：客户年龄分布图可以看到年龄变量大致呈正态分布，符合统计分析的假设。

一、数据集描述信息以及变量的数据分布可视化

1）目标变量'违约'的分布直方图

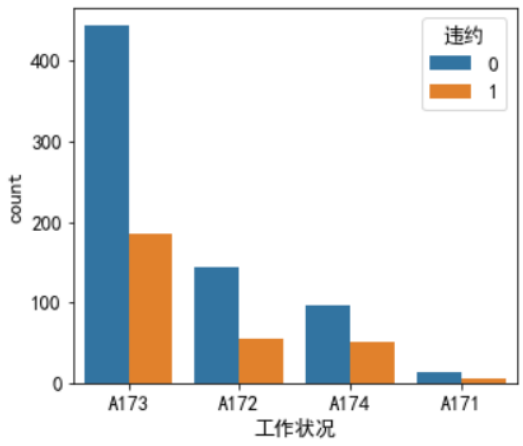
```
sns.histplot(data = df['违约'],x=df['违约'].values,binwidth=0.4)
plt.xticks(ticks=[0,1],labels=['0','1'])
plt.xlabel('违约')
plt.show()
```

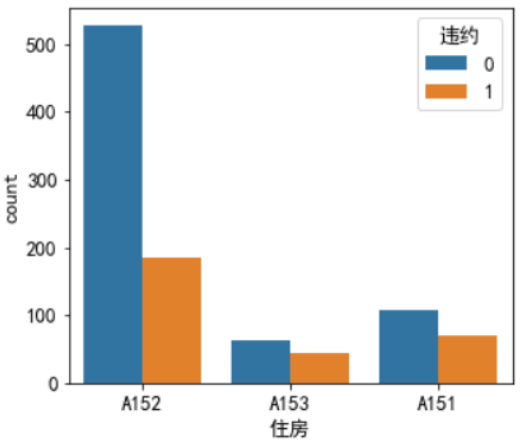


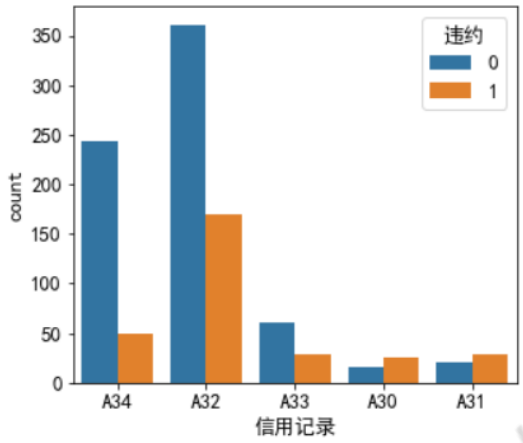
2）类别型变量的分布柱状图

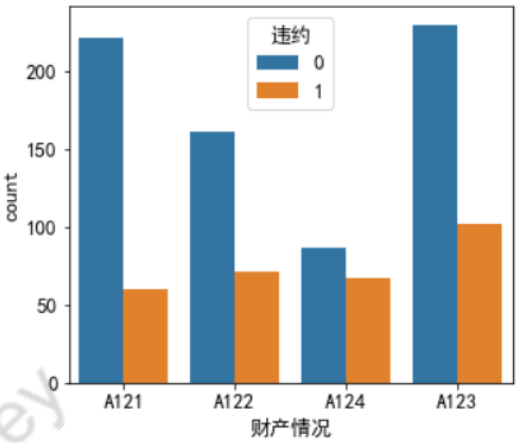
循环画图代码：

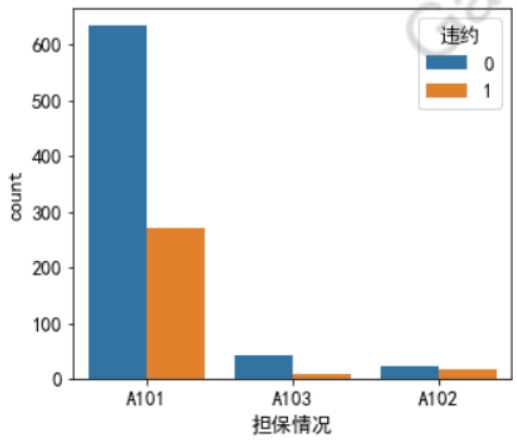
```
#类别型变量
plt.figure(figsize=(23,20))
i=1
for temp in category[:-1]:
    plt.subplot(4,4,i)
    sns.countplot(data=df_category,x=temp,hue='违约')
    i+=1
plt.show()
```

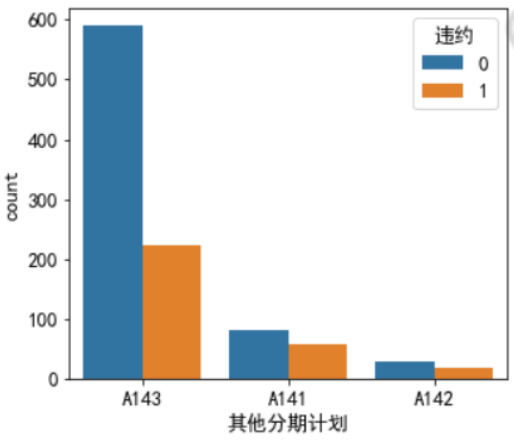

违约
0
1
count
A173
A172
A174
A171
工作状况

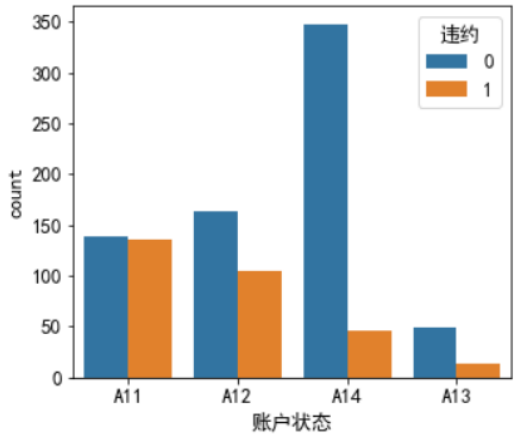

违约
0
1
count
A152
A153
A151
住房

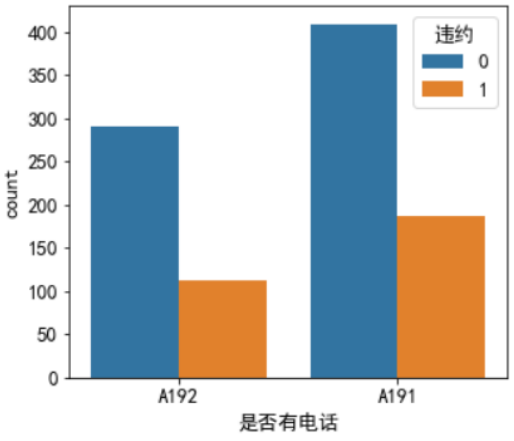

违约
0
1
count
A34
A32
A33
A30
A31
信用记录

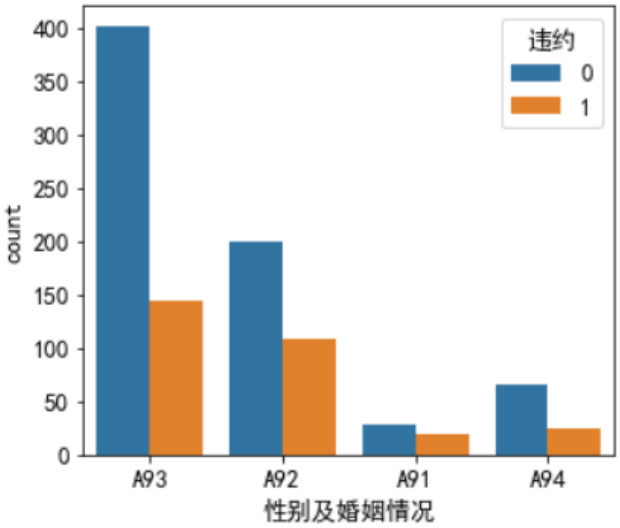

违约
0
1
count
A121
A122
A124
A123
财产情况

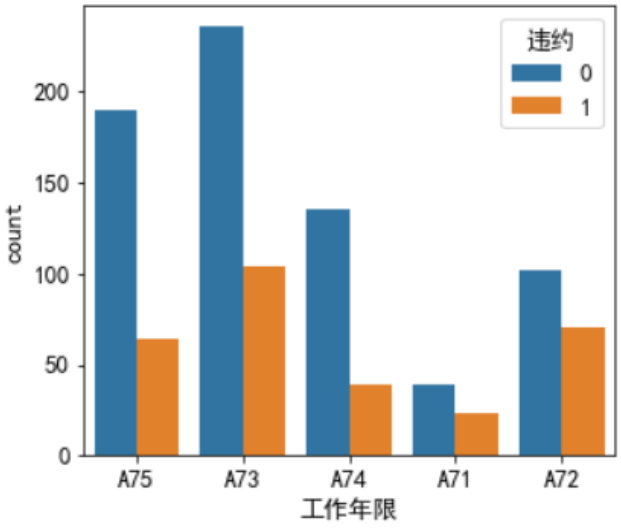

违约
0
1
count
A101
A103
A102
担保情况

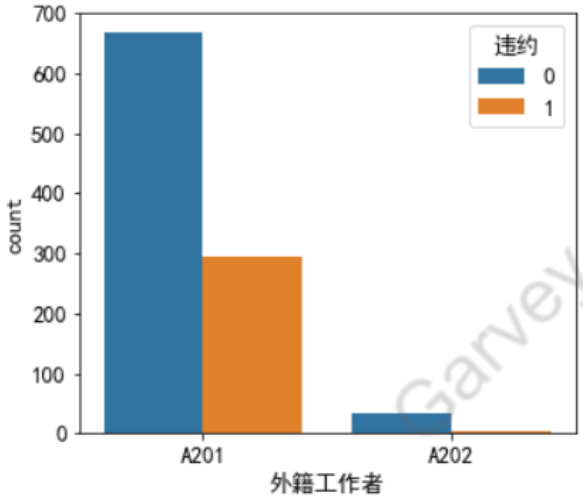

违约
0
1
count
A143
A141
A142
其他分期计划

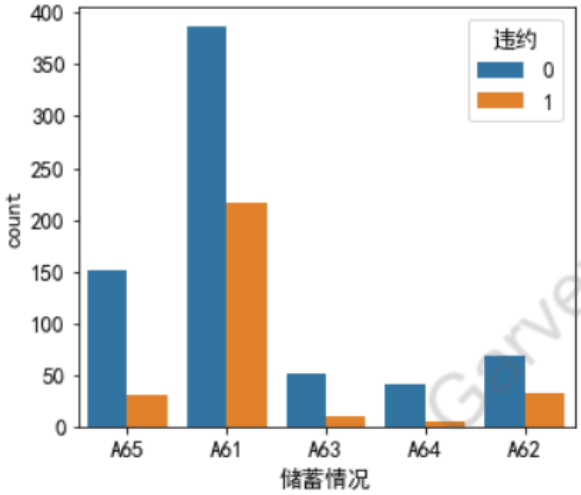

违约
0
1
count
A11
A12
A14
A13
账户状态

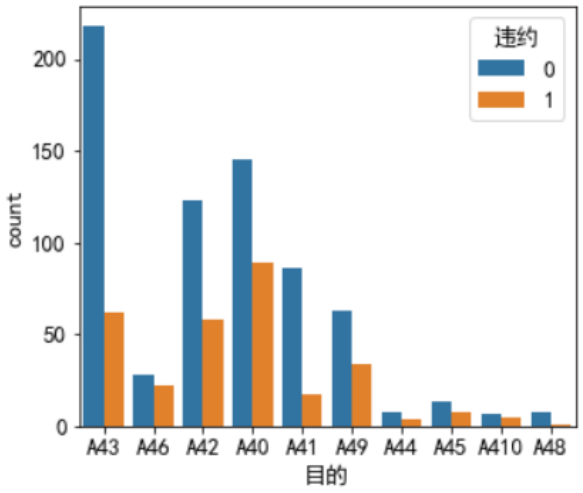

违约
0
1
count
A192
A191
是否有电话


违约
0
1
count
A93
A92
A91
A94
性别及婚姻情况


违约
0
1
count
A75
A73
A74
A71
A72
工作年限


违约
0
1
count
A201
A202
外籍工作者

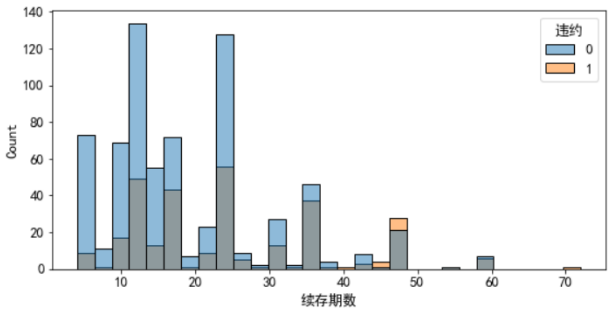

违约
0
1
count
A65
A61
A63
A64
A62
储蓄情况

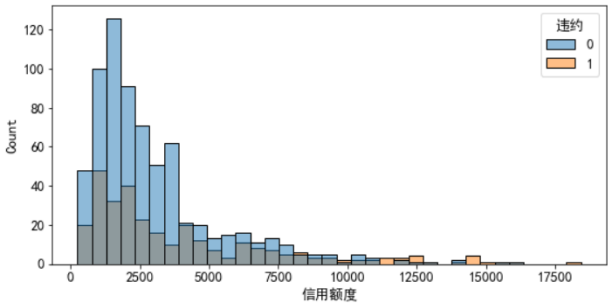

违约
0
1
count
A43
A46
A42
A40
A41
A49
A44
A45
A410
A48
目的

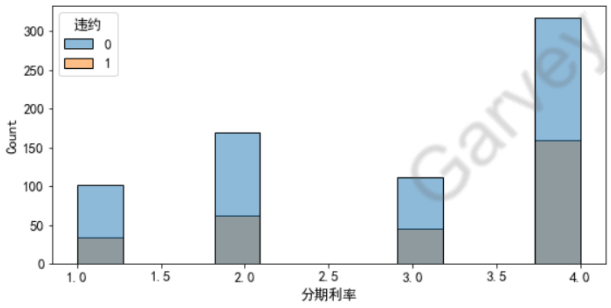
3）数值型变量的分布柱状图

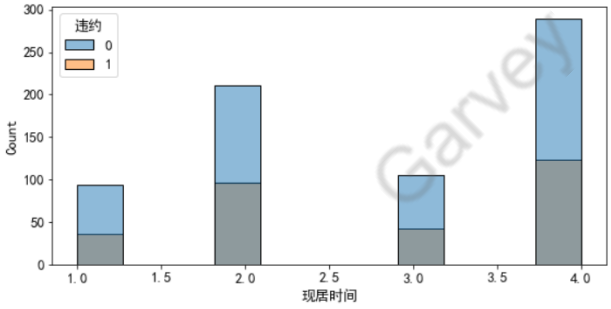
循环画图代码:

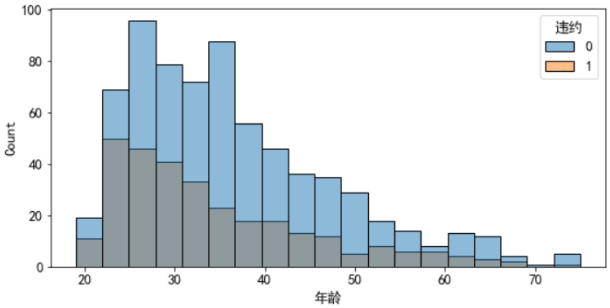
```
#数值型变量分布（直方图）
plt.figure(figsize=(20,20))
i=1
for temp in numerical[:-1]:
    plt.subplot(4,2,i)
    sns.histplot(data = df_numerical,x=temp,hue='违约')
    i+=1
plt.show()
```

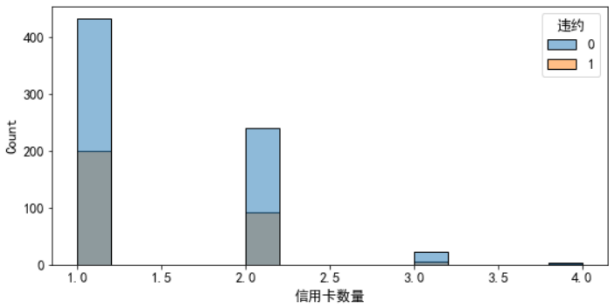


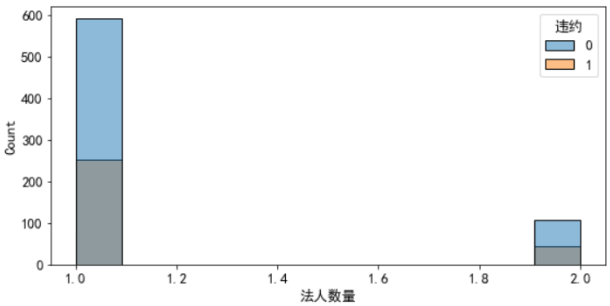






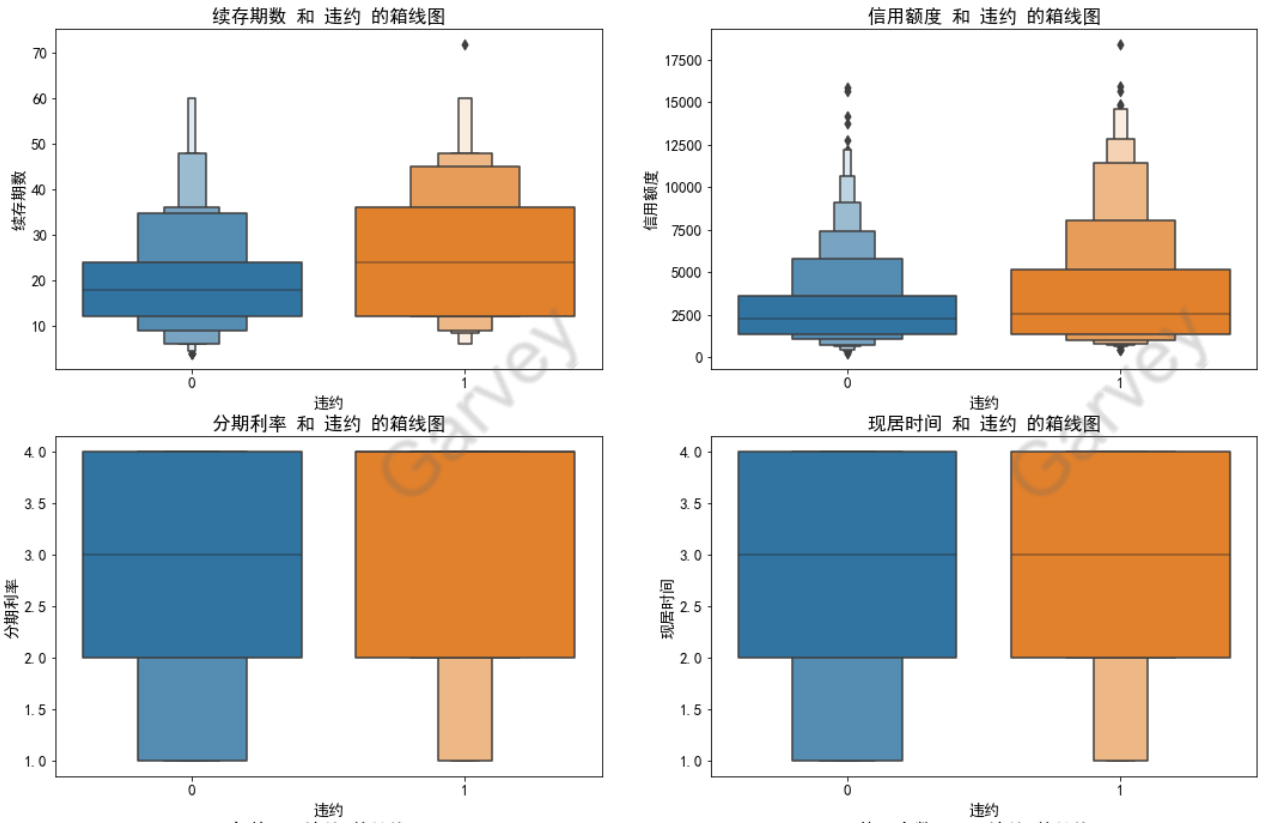


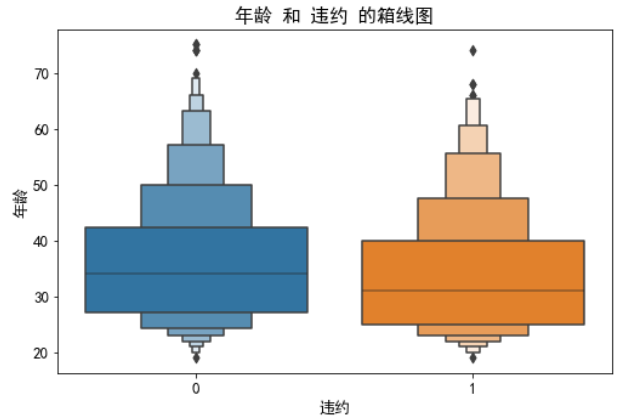


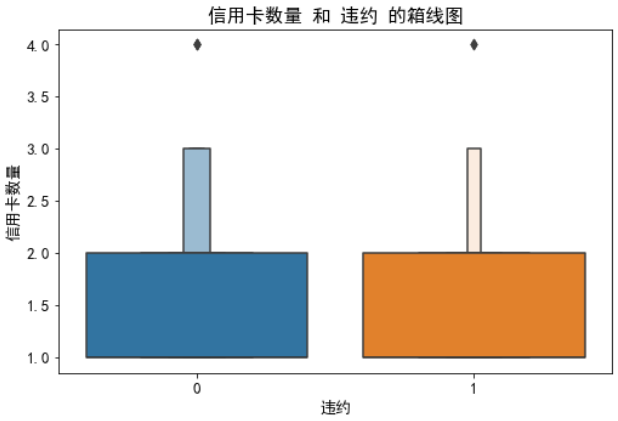


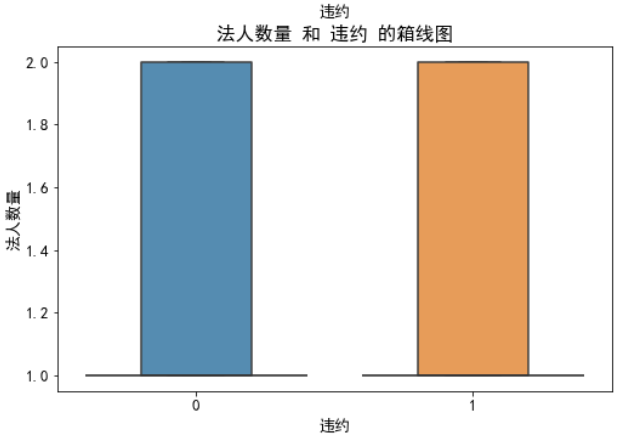
4）数值型变量的箱线图

```
#数值型变量分布(箱线图)
plt.figure(figsize=(18,24))
i=1
for temp in numerical[:-1]:
    plt.subplot(4,2,i)
    sns.boxenplot(data = df_numerical,y=temp,x='违约')
    plt.title(f'{temp} 和 违约 的箱线图')
    i+=1
plt.show()
```









上述变量的分布柱状图和箱线图将为后续的变量分箱提供一定的参考。

5）数据集的描述（即对数值型变量的描述）

```
df.describe()
```

	续存期数	信用额度	分期利率	现居时间	年龄	信用卡数量	法人数量	违约
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	20.903000	3271.258000	2.973000	2.845000	35.546000	1.407000	1.155000	0.300000
std	12.058814	2822.736876	1.118715	1.103718	11.375469	0.577654	0.362086	0.458487
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000000	0.000000
25%	12.000000	1365.500000	2.000000	2.000000	27.000000	1.000000	1.000000	0.000000
50%	18.000000	2319.500000	3.000000	3.000000	33.000000	1.000000	1.000000	0.000000
75%	24.000000	3972.250000	4.000000	4.000000	42.000000	2.000000	1.000000	1.000000
max	72.000000	18424.000000	4.000000	4.000000	75.000000	4.000000	2.000000	1.000000

数值型变量的分箱处理

1）数值型变量的等距分箱函数

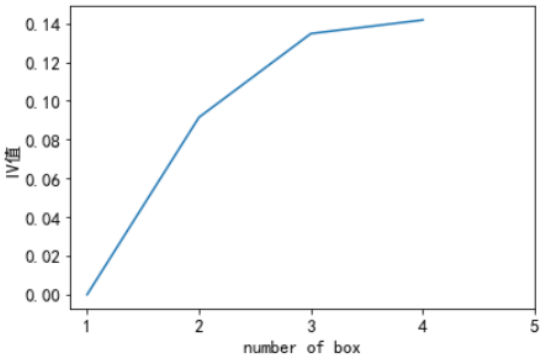
```
# 数值型变量的等距分箱
def binning_num(df,target,col_list,n=10):
    """
    df:数据集
    target:目标变量的字段名
    col_list:变量list集合
    n:分箱个数 默认为10
    return:
    iv_value:list形式，里面存储每个变量的IV值
    """
    total = df[target].count()
    bad = df[target].sum() #计数违约为1
    good = total-bad       #总数-违约的
    iv_value=[]
    bucket=0
    for col in col_list:
        bucket = pd.cut(df[col],n) #分箱操作
        d1 = df.groupby(bucket)   #按照分箱进行分组
        d2 = pd.DataFrame()       #构造dataframe
        d2['min_bin'] = d1[col].min()  #最小值
        d2['max_bin'] = d1[col].max()  #最大值
        d2['total'] = d1[target].count()  #目标变量分组汇总数
        d2['totalrate'] = d2['total']/total
        d2['bad'] = d1[target].sum()
        d2['badrate'] = d2['bad']/d2['total']
        d2['good'] = d2['total'] - d2['bad']
        d2['goodrate'] = d2['good']/d2['total']
        d2['badattr'] = d2['bad']/bad                    #bad%
        d2['goodattr'] = (d2['total']-d2['bad'])/good    #good%
        d2['woe'] = np.log(d2['badattr']/d2['goodattr'])
        d2['bin_iv'] = (d2['badattr']-d2['goodattr'])*d2['woe']
        d2['IV'] = d2['bin_iv'].sum()
        iv = d2['bin_iv'].sum().round(4)
        iv_value.append(iv)

    return iv_value
```

2）**数值型变量分箱（对于分箱的区间划分，为了方便后续使用评分卡，我采取结合 IV 值-分箱数量的折线图和上述变量的分布图、箱线图；同时对于区间划分采取整数，这可能导致后续的 woe 和 iv 与现在最优值的有差别！）**

续存期数:

```
#续存期数
iv = []
x = []
for i in range(1,5):
    iv.append(binning_num(df.loc[:,['续存期数','违约']],target='违约',n=i,col_list=['续存期数'])[0])
    x.append(i)
plt.plot(x,iv)
plt.xticks(range(1,6))
plt.ylabel('IV值')
plt.xlabel('number of box')
plt.show()
```



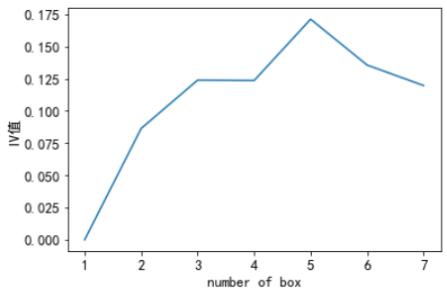

```
pd.cut(df['续存期数'],4).unique()
```

```
[(3.932, 21.0], (38.0, 55.0], (21.0, 38.0], (55.0, 72.0]]
Categories (4, interval[float64, right]): [(3.932, 21.0] < (21.0, 38.0] < (38.0, 55.0] < (55.0, 72.0]]
```

1、续存期数：'0':0-20 ; '1':20-40 ; '2':40-55 ; '3':55以上

信用额度:

```
#信用额度
iv = []
x = []
for i in range(1,8):
    iv.append(binning_num(df.loc[:,['信用额度','违约']],target='违约',n=i,col_list=['信用额度'])[0])
    x.append(i)
plt.plot(x,iv)
plt.xticks(range(1,8))
plt.ylabel('IV值')
plt.xlabel('number of box')
plt.show()
```




```
pd.cut(df['信用额度'],5).unique()
```

```
[(231.826, 3884.8], (3884.8, 7519.6], (7519.6, 11154.4], (11154.4, 14789.2], (14789.2, 18424.0]]
Categories (5, interval[float64, right]): [(231.826, 3884.8] < (3884.8, 7519.6] < (7519.6, 11154.4] < (11154.4, 14789.2] < (14789.2, 18424.0]]
```

2、信用额度:'0':0-4000 ; '1':4000-8000 ; '2':8000-12000 ; '3':12000-15000 ; '4':15000以上

```
df['信用额度'] = pd.cut(df['信用额度'],[0,4000,8000,12000,15000,19000],labels=['0','1','2','3','4'])
```

分期利率、现居时间：

3、分期利率:只有[4, 2, 3, 1]四种利率，直接转为字符串进行分箱

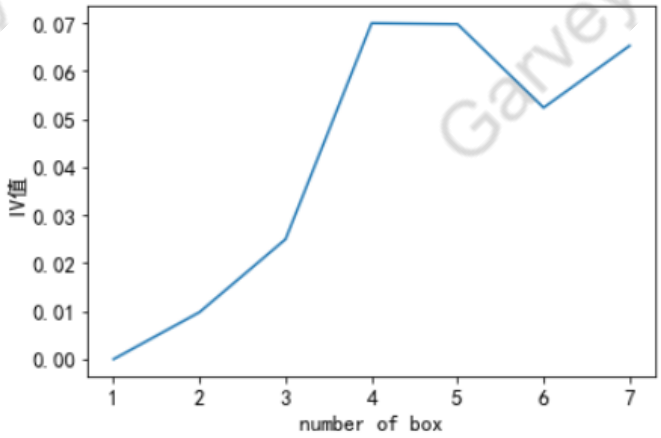
```
df['分期利率'] = df['分期利率'].astype('object')
```

4、现居时间: 只有[4, 2, 3, 1]四种现居时间，直接转为字符串进行分箱

```
df['现居时间'] = df['现居时间'].astype('object')
```

年龄：

```
#年龄
iv = []
x = []
for i in range(1,8):
    iv.append(binning_num(df.loc[:,['年龄','违约']],target='违约',n=i,col_list=['年龄'])[0])
    x.append(i)
plt.plot(x,iv)
plt.xticks(range(1,8))
plt.ylabel('IV值')
plt.xlabel('number of box')
plt.show()
```



5、年龄 '0':18-35 ; '1':35-50 ; '2':50-60 ; '3':60以上

```
df['年龄'] = pd.cut(df['年龄'],[18,35,50,60,80],labels=['0','1','2','3'])
```

信用卡数量、法人数量：

6、信用卡数量:只有[4, 2, 3, 1]四种信用卡数量，直接转为字符串进行分箱

```
df['信用卡数量'] = df['信用卡数量'].astype('object')
```

7、法人数量: 只有[1,2]四种法人数量，直接转为字符串进行分箱

```
df['法人数量'] = df['法人数量'].astype('object')
```

3）函数准备以及数据集替换

```
# 备份df数据集
df_ = df.copy()
df_after = df.copy()
```

WOE_todf(将数据集替换成变量对应的 woe 值):

```
#将 woe 值替换进 df_after
def WOE_todf(varList,df,df_after):
    '''
    varList: 表示需要计算 WOE 并替换的变量列表，
    df即是存储对于目标变量和特征变量的数据集合，
    df_after:是把 woe值替换进去的数据集
    '''
    for var in varList:  #遍历varList
        data = df.groupby([var,'违约'])[['违约']].count().rename(columns={'违约':'数量'})
        data = data.reset_index()
        good_sum = data[data.违约==0]['数量'].sum()
        bad_sum = data[data.违约==1]['数量'].sum()
        woe = {}
        for i in df[var].unique():
            good_temp = data[(data[var]==i) & (data['违约']==0)]['数量'].sum()
            bad_temp = data[(data[var]==i) & (data['违约']==1)]['数量'].sum()
            woe[i] = np.log(((good_temp/good_sum)/(bad_temp/bad_sum)))  #woe值

            #将对变量var分箱好的woe值替换回df数据集合,对woe值精确到4位并填入原数据集
            df_after.loc[:,[var]] = df_after.loc[:,[var]].replace(i,round(woe[i],4))

    return df_after
```

WOE（计算变量对应的 woe 值（是一个字典形式）和 IV 值）

```
def WOE(var,df):
    #var: 表示需要计算 WOE和 IV值的变量，df即是存储对于目标变量和特征变量的数据集合
    data = df.groupby([var,'违约'])[['违约']].count().rename(columns={'违约':'数量'})
    data = data.reset_index()
    good_sum = data[data.违约==0]['数量'].sum()
    bad_sum = data[data.违约==1]['数量'].sum()
    woe = {}
    IV = 0
    for i in df[var].unique():
        good_temp = data[(data[var]==i) & (data['违约']==0)]['数量'].sum()
        bad_temp = data[(data[var]==i) & (data['违约']==1)]['数量'].sum()
        woe[i] = np.log(((good_temp/good_sum)/(bad_temp/bad_sum)))
        IV += ((good_temp/good_sum)-(bad_temp/bad_sum)) * woe[i]

    return woe,IV
```

IV（单独计算变量的 IV 值）

```
def IV(var,df):      #var: 表示需要计算 IV值的变量，df即是存储对于目标变量和特征变量的数据集合
    data = df.groupby([var,'违约'])[['违约']].count().rename(columns={'违约':'数量'})
    data = data.reset_index()
    good_sum = data[data.违约==0]['数量'].sum()
    bad_sum = data[data.违约==1]['数量'].sum()
    woe = {}
    IV = 0
    for i in df[var].unique():
        good_temp = data[(data[var]==i) & (data['违约']==0)]['数量'].sum()
        bad_temp = data[(data[var]==i) & (data['违约']==1)]['数量'].sum()
        woe[i] = np.log(((good_temp/good_sum)/(bad_temp/bad_sum)))
        IV += ((good_temp/good_sum)-(bad_temp/bad_sum)) * woe[i]

    return IV
```

调用 WOE_todf 函数，生成替换完成的 dataframe

```
#调用 WOE_todf 函数，输出将 woe 值替换完成的dataframe
df_after = WOE_todf(col[:-1],df_,df_after)
df_after
```

	账户状态	续存期数	信用记录	目的	信用额度	储蓄情况	工作年限	分期利率	性别及婚姻情况	担保情况	...	财产情况	年龄	其他分期计划	住房	信用卡数量	工作状况	法人数量	是否有电话	外籍工作者	违约
0	-0.8181	0.3050	0.7337	0.4101	0.2059	0.7042	0.2356	-0.1573	0.1655	0.0005	...	0.4610	0.4055	0.1212	0.1942	0.1157	0.0228	-0.0028	0.0986	-0.0349	0
1	-0.4014	-0.9968	-0.0883	0.4101	-0.3606	-0.2714	-0.0321	0.1555	-0.2353	0.0005	...	0.4610	-0.1694	0.1212	0.1942	-0.0749	0.0228	-0.0028	-0.0647	-0.0349	1
2	1.1763	0.3050	0.7337	-0.6061	0.2059	-0.2714	0.3944	0.1555	0.1655	0.0005	...	0.4610	0.3194	0.1212	0.1942	-0.0749	0.0972	0.0154	-0.0647	-0.0349	0
3	-0.8181	-0.9968	-0.0883	-0.0956	-0.3606	-0.2714	0.3944	0.1555	0.1655	0.5878	...	-0.0286	0.3194	0.1212	-0.4726	-0.0749	0.0228	0.0154	-0.0647	-0.0349	0
4	-0.8181	-0.1828	-0.0852	-0.3592	-0.3606	-0.2714	-0.0321	0.0645	0.1655	0.0005	...	-0.5861	-0.0417	0.1212	-0.4726	0.1157	0.0228	0.0154	-0.0647	-0.0349	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	1.1763	0.3050	-0.0883	-0.0956	0.2059	-0.2714	0.3944	0.0645	-0.2353	0.0005	...	0.4610	-0.1694	0.1212	0.1942	-0.0749	0.0972	-0.0028	-0.0647	-0.0349	0
996	-0.8181	-0.1828	-0.0883	0.7738	0.2059	-0.2714	-0.0321	-0.1573	-0.4418	0.0005	...	-0.0286	0.3194	0.1212	0.1942	-0.0749	-0.2044	-0.0028	0.0986	-0.0349	0
997	1.1763	0.3050	-0.0883	0.4101	0.2059	-0.2714	0.2356	-0.1573	0.1655	0.0005	...	-0.0342	0.3194	0.1212	0.1942	-0.0749	0.0228	-0.0028	-0.0647	-0.0349	0
998	-0.8181	-0.9968	-0.0883	0.4101	0.2059	-0.2714	-0.0321	-0.1573	0.1655	0.0005	...	-0.5861	-0.1694	0.1212	-0.4726	-0.0749	0.0228	-0.0028	0.0986	-0.0349	1
999	-0.4014	-0.9968	0.7337	0.7738	-0.3606	-0.1396	-0.3192	0.0645	0.1655	0.0005	...	-0.0342	-0.1694	0.1212	0.1942	-0.0749	0.0228	-0.0028	-0.0647	-0.0349	0

1000 rows × 21 columns

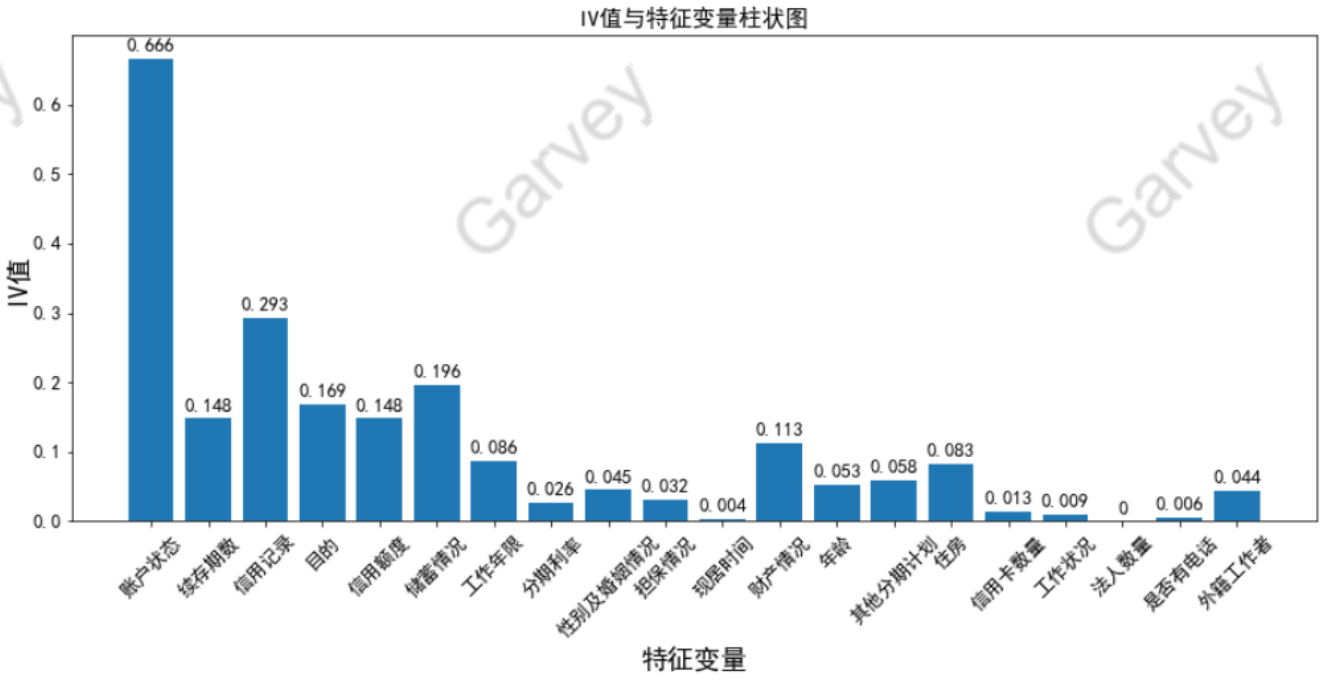
逻辑模型建立和检验

1）变量选择

计算各变量的 IV 值，判断变量是否进入模型

```
#计算各变量的IV值，判断变量是否进入模型
iv_dict = {}  #iv集合
for k in col[:-1]:
    iv_dict[k] = IV(k,df)
```

```
#绘图
x = iv_dict.keys()
y=[]
for val in iv_dict.values():
    y.append(round(val,3))
plt.figure(figsize=(15,6))
ax = plt.subplot(1,1,1)
rects = ax.bar(x,y)
ax.bar_label(rects,padding=3)
plt.ylabel('IV值',fontsize=18)
plt.xlabel('特征变量',fontsize=18)
plt.title('IV值与特征变量柱状图')
plt.xticks(rotation=45)
plt.show()
```



按照 IV 值升序排序:

```
#升序排序
lt = sorted(iv_dict.items(),key=lambda s:s[1])
```

```
#iv z值大于 0.08 的特征变量
lt = lt[-9:]
lt
```

```
[('住房', 0.08329343361549926),
 ('工作年限', 0.086433631026641),
 ('财产情况', 0.11263826240979673),
 ('续存期数', 0.14760905122423662),
 ('信用额度', 0.14792743500157343),
 ('目的', 0.16919506567307835),
 ('储蓄情况', 0.1960095569042267),
 ('信用记录', 0.2932335473908263),
 ('账户状态', 0.6660115033513336)]
```

```
tar = []
for a in range(len(lt)):
    tar.append(lt[a][0])
print(f"IV值大于0.08的变量：\n{tar},\n长度为：{len(tar)}")
```

```
IV值大于0.08的变量：
['住房', '工作年限', '财产情况', '续存期数', '信用额度', '目的', '储蓄情况', '信用记录', '账户状态'],
长度为：9
```

引入目标变量并且新建 df_info(包含可进入模型的变量)

```
tar.append('违约')  #引入目标变量
df_info = df_after.loc[:,tar]
df_info
```

	住房	工作年限	财产情况	续存期数	信用额度	目的	储蓄情况	信用记录	账户状态	违约
0	0.1942	0.2356	0.4610	0.3050	0.2059	0.4101	0.7042	0.7337	-0.8181	0
1	0.1942	-0.0321	0.4610	-0.9968	-0.3606	0.4101	-0.2714	-0.0883	-0.4014	1
2	0.1942	0.3944	0.4610	0.3050	0.2059	-0.6061	-0.2714	0.7337	1.1763	0
3	-0.4726	0.3944	-0.0286	-0.9968	-0.3606	-0.0956	-0.2714	-0.0883	-0.8181	0
4	-0.4726	-0.0321	-0.5861	-0.1828	-0.3606	-0.3592	-0.2714	-0.0852	-0.8181	1
...	...	...	...	...	...	...	...	...	...	...
995	0.1942	0.3944	0.4610	0.3050	0.2059	-0.0956	-0.2714	-0.0883	1.1763	0
996	0.1942	-0.0321	-0.0286	-0.1828	0.2059	0.7738	-0.2714	-0.0883	-0.8181	0
997	0.1942	0.2356	-0.0342	0.3050	0.2059	0.4101	-0.2714	-0.0883	1.1763	0
998	-0.4726	-0.0321	-0.5861	-0.9968	0.2059	0.4101	-0.2714	-0.0883	-0.8181	1
999	0.1942	-0.3192	-0.0342	-0.9968	-0.3606	0.7738	-0.1396	0.7337	-0.4014	0

1000 rows × 10 columns

划分数据集

```
#划分数据集
from sklearn.model_selection import train_test_split
X = df_info.iloc[:,:-1]
Y = df_info.iloc[:,-1]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=110)
```

2）LogisticRegression 建模

```
from sklearn.linear_model import LogisticRegression
```

```
#网络搜索查找最优参数
lr_param = {'C': [0.01, 0.1, 0.2, 0.5, 1, 1.5, 2],
                'class_weight': [{1: 1, 0: 1}, {1: 2, 0: 1}, {1: 3, 0: 1}, 'balanced'],
                'max_iter': range(10,100,10)}
```

```
#逻辑回归模型
from sklearn.model_selection import GridSearchCV
lr_gsearch = GridSearchCV(
        estimator=LogisticRegression(random_state=0, fit_intercept=True, penalty='l2'),
        param_grid=lr_param, cv=5, scoring='f1', n_jobs=-1, verbose=2)
```

```
#执行超参数优化
lr_gsearch.fit(X_train, Y_train)
print('logistic model best_score_ is {0},and best_params_ is {1}'.format(lr_gsearch.best_score_,
                                                                         lr_gsearch.best_params_))
```

```
Fitting 5 folds for each of 252 candidates, totalling 1260 fits
logistic model best_score_ is 0.6265609316246388,and best_params_ is {'C': 1, 'class_weight': 'balanced', 'max_iter': 20}
```

```
#用最优参数，初始化Logistic模型
LR = LogisticRegression(C=lr_gsearch.best_params_['C'], penalty='l2',
                                  class_weight=lr_gsearch.best_params_['class_weight'],
                                  max_iter=lr_gsearch.best_params_['max_iter'])
```

```
#逻辑回归模型
LR = LogisticRegression(class_weight='balanced',max_iter=20,C=1)
LR.fit(X_train,Y_train)
```

```
LogisticRegression
LogisticRegression(C=1, class_weight='balanced', max_iter=20)
```

3）模型评价

准备

```
#分类模型评价
from sklearn.metrics import classification_report
#混淆矩阵
from sklearn.metrics import confusion_matrix
#roc,P-R
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score
```

准确率：

```
print(f"逻辑回归的准确率：{(round(LR_b.score(X_test,Y_test),4))*100}%")
```

逻辑回归的准确率：75.0%

评价报告：

```
# 逻辑回归模型的评价
print(f"模型评价:\n{classification_report(Y_test,LR.predict(X_test))}")
```

模型评价：

	precision	recall	f1-score	support
0	0.87	0.76	0.81	144
1	0.54	0.71	0.62	56
accuracy			0.75	200
macro avg	0.71	0.74	0.72	200
weighted avg	0.78	0.75	0.76	200

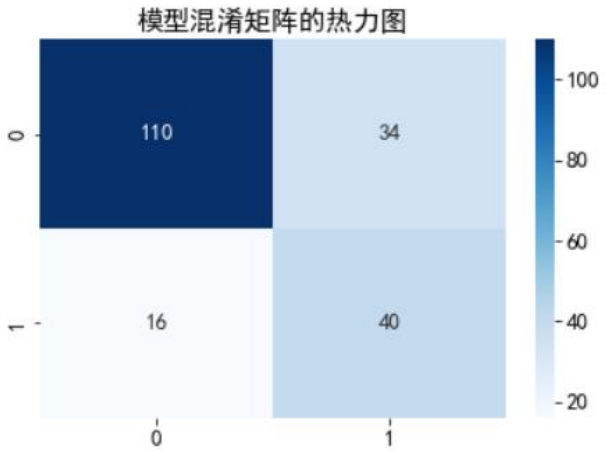
混淆矩阵：

```
#混淆矩阵
cnf_matrix = confusion_matrix(Y_test,LR.predict(X_test))
print('混淆矩阵：\n',cnf_matrix)
```

混淆矩阵：

```
[[110  34]
 [ 16  40]]
```

```
sns.heatmap(cnf_matrix,annot=True,cmap=plt.cm.Blues,fmt='g')
plt.title('模型混淆矩阵的热力图')
plt.show()
```

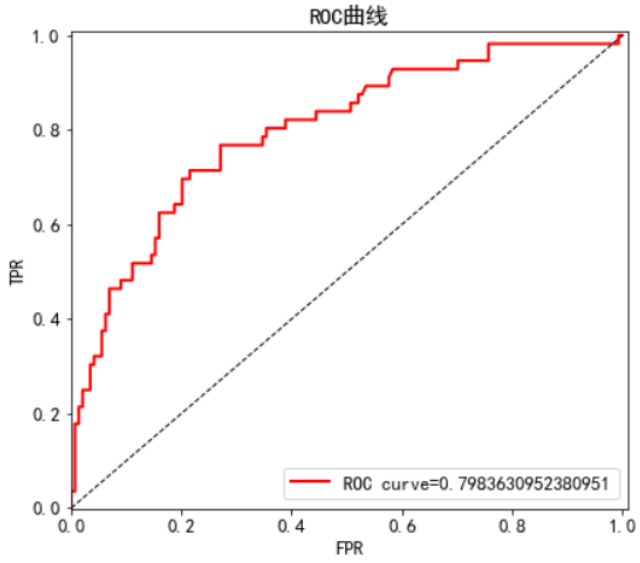


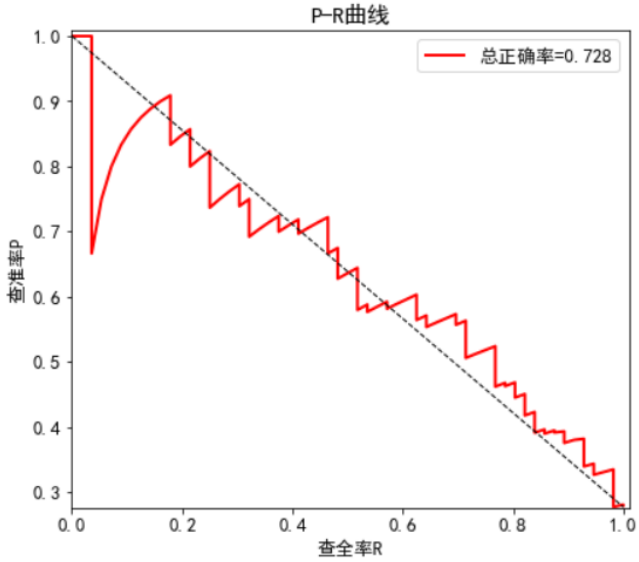
ROC 曲线和 P-R 曲线:

```
#ROC曲线和P-R曲线
#FTR:假阳性率，TPR：真阳性率
FPR,TPR,thresholds = roc_curve(Y_test,LR.predict_proba(X_test)[:,1],pos_label=1)
roc_auc = auc(FPR,TPR)  #auc值
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
plt.plot(FPR,TPR,color = 'r',linewidth=2,label=f'ROC curve={roc_auc}')
plt.plot([0,1],[0,1],color='k',linewidth=1,linestyle='--')
plt.xlim([-0.001,1.01])
plt.ylim([-0.001,1.01])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC曲线')
plt.legend(loc = 'best')

plt.subplot(1,2,2)
PRE,REC,thresholds = precision_recall_curve(Y_test,LR.predict_proba(X_test)[:,1],pos_label=1)
plt.plot(REC,PRE,color = 'r',linewidth=2,label=f'总正确率={accuracy_score(Y,LR.predict(X))}')
plt.plot([0,1],[1,PRE.min()],color='k',linewidth=1,linestyle='--')
plt.xlim([-0.001,1.01])
plt.ylim([PRE.min()-0.001,1.01])
plt.xlabel('查全率R')
plt.ylabel('查准率P')
plt.title('P-R曲线')
plt.legend(loc = 'best')
plt.show()

print("AUC为：",roc_auc)
print("总准确率为：",accuracy_score(Y_test,LR.predict(X_test)))
```






```
AUC为： 0.7983630952380951
总准确率为： 0.75
```

生成评分卡（如下图）

1）逻辑回归模型系数：β0,β1,…,β p

```
#系数,即是θ1,…,θp
LR.coef_[0]
```

```
array([-0.52358358, -1.00367417, -0.73806486, -0.85375767, -0.61080522,
       -0.6651978 , -0.62570778, -0.07664987, -0.6395133 , -0.71652088,
       -0.89459694, -0.60370162, -0.60191614, -0.79853633])
```

```
#截距，即是θ0
LR.intercept_
```

```
array([-0.83542124])
```

得分函数等

```
def Score(probability):
    score = A-B*np.log(probability/(1-probability))
    return score

#批量获取得分
def List_score(pos_probablity_list):
    list_score=[]
    for probability in pos_probablity_list:
        score=Score(probability)
        list_score.append(score)
    return list_score

P0 = 600
PDO = 20
theta0 = 1.0/60
B = PDO/np.log(2)
A = P0 + B*np.log(theta0)
print("A:",A)
print("B:",B)
```

```
A: 481.8621880878296
B: 28.85390081777927
```

特征变量：

```
tar.pop() #去除引入的目标变量'违约'
tar
```

```
['住房', '工作年限', '财产情况', '续存期数', '信用额度', '目的', '储蓄情况', '信用记录', '账户状态']
```

除β0 外的**系数,即是β1,⋯,β p**

```
#系数,即是β1,⋯,βp
beta_lt_b = LR_b.coef_[0]
beta_lt_b
```

```
array([-0.64521407, -0.94071371, -0.10364738, -0.73151993, -0.77521383,
       -0.96191618, -0.52434953, -0.7237851 , -0.80315633])
```

上述各特征变量的分箱 woe 值

```
#获得各特征变量各分箱的woe值
woedict = {}
for var_ in tar:
    woedict[var_] = WOE(var_,df)[0]
woedict
```

```
{'住房': {'A152': 0.19415601444095756,
  'A153': -0.4726044109457929,
  'A151': -0.4044452202074189},
 '工作年限': {'A75': 0.2355660713127669,
  'A73': -0.03210324538441743,
  'A74': 0.39441527192157944,
  'A71': -0.3192304301867068,
  'A72': -0.4708202891522916},
 '财产情况': {'A121': 0.46103495926297494,
  'A122': -0.028573372444056,
  'A124': -0.5860823611235859,
  'A123': -0.034191364748279426},
 '续存期数': {'0': 0.30498584507342397,
  '2': -0.9968295943581674,
  '1': -0.18278249250158538,
  '3': -0.8472978603872037},
 '信用额度': {'0': 0.20585205420414882,
  '1': -0.36064259754902595,
  '2': -0.8064758658669485,
  '3': -1.6357552207514738,
  '4': -1.252762968495368},
 '目的': {'A43': 0.41006281735679384,
  'A46': -0.6061358035703156,
```

基础分数

```
print(f'基础分数：{round(A-beta0*B)}')
```

基础分数：482

模型应用

```
#获取全部X_test数据预测概率
probablity_list=LR_b.predict_proba(X_test)
print('pos_probablity_list\n', probablity_list)
```

```
pos_probablity_list
 [[0.95669163 0.04330837]
 [0.88418558 0.11581442]
 [0.13941647 0.86058353]
 [0.81207718 0.18792282]
 [0.74619525 0.25380475]
 [0.690142   0.309858  ]
 [0.87932488 0.12067512]
 [0.53433683 0.46566317]
 [0.51371053 0.48628947]
 [0.62650733 0.37349267]
 [0.85609612 0.14390388]
 [0.59428308 0.40571692]
 [0.91499402 0.08500598]
 [0.60219777 0.39780223]
 [0.51665928 0.48334072]
```

```
#获取全部X_test数据预测为坏用户的概率
pos_probablity_list=LR_b.predict_proba(X_test)[:,1]
print('probablity_list\n', pos_probablity_list)
```

```
probablity_list
 [0.04330837 0.11581442 0.86058353 0.18792282 0.25380475 0.309858
 0.12067512 0.46566317 0.48628947 0.37349267 0.14390388 0.40571692
 0.08500598 0.39780223 0.48334072 0.28916098 0.36589637 0.8274339
 0.24274274 0.16949149 0.7709261  0.30921483 0.39741491 0.76128497
 0.16889543 0.52643423 0.37417799 0.54805046 0.34201278 0.96306882
 0.30027751 0.19923204 0.42660356 0.41936437 0.10575475 0.33350081
 0.07676625 0.30174028 0.55192961 0.77236258 0.28457533 0.4199758
 0.15516845 0.61361727 0.47314886 0.23847052 0.37455779 0.68922412
 0.443327   0.28266086 0.51586439 0.17452051 0.10954307 0.15366862
 0.81550889 0.74546054 0.14860707 0.64230597 0.75572365 0.27729898
 0.3385875  0.215068   0.22631429 0.79588679 0.30921483 0.15024865
 0.81724097 0.074245   0.24476911 0.39907542 0.11075938 0.67359837
 0.09536958 0.34535848 0.39059387 0.61191379 0.074245   0.43708856
 0.4170894  0.74612468 0.61221492 0.32763247 0.1654148  0.77714429
```

```
#获取所有x数据的坏客户预测概率
pos_probablity_list=[i[1] for i in probablity_list]
pos_probablity_list
```

```
[0.04330837092968998,
 0.11581442063359097,
 0.8605835254433225,
 0.1879228225846488,
 0.25380474856816365,
 0.3098579989535778,
 0.12067512465973897,
 0.4656631724086704,
 0.4862894679210038,
 0.37349266821289556,
 0.14390388227493817,
 0.4057169221245572,
 0.08500598247368102
```

```
#获得所有客户的分数
list_score=List_score(pos_probablity_list)
list_predict=LR_b.predict(X_test)
```

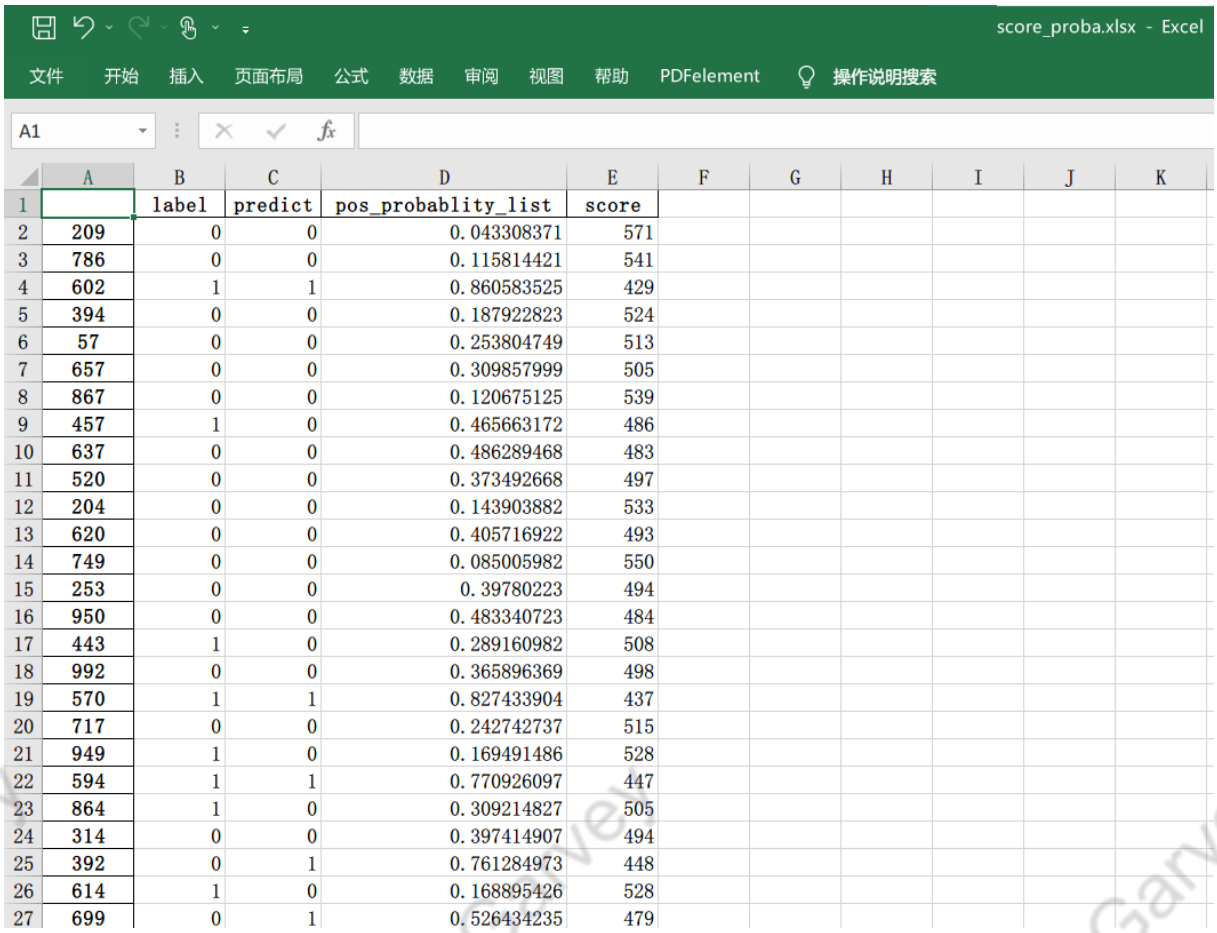
```
#分数取整
list_score = [round(i) for i in list_score]
list_score
```

```
[571,
 541,
 429,
 524,
 513,
 505,
 539,
 486,
 483,
 497
```

```
#构造dataframe
df_results = pd.DataFrame({'label':Y_test,'predict':list_predict,
                           'pos_probablity_list':pos_probablity_list,
                           'score':list_score})
```

```
#结果写入excel
df_results.to_excel('score_proba.xlsx')
```

部分结果：



score_proba.xlsx - Excel

文件 开始 插入 页面布局 公式 数据 审阅 视图 帮助 PDFelement 操作说明搜索

A1

	A	B	C	D	E	F	G	H	I	J	K
1		label	predict	pos_probablity_list	score						
2	209	0	0	0.043308371	571						
3	786	0	0	0.115814421	541						
4	602	1	1	0.860583525	429						
5	394	0	0	0.187922823	524						
6	57	0	0	0.253804749	513						
7	657	0	0	0.309857999	505						
8	867	0	0	0.120675125	539						
9	457	1	0	0.465663172	486						
10	637	0	0	0.486289468	483						
11	520	0	0	0.373492668	497						
12	204	0	0	0.143903882	533						
13	620	0	0	0.405716922	493						
14	749	0	0	0.085005982	550						
15	253	0	0	0.39780223	494						
16	950	0	0	0.483340723	484						
17	443	1	0	0.289160982	508						
18	992	0	0	0.365896369	498						
19	570	1	1	0.827433904	437						
20	717	0	0	0.242742737	515						
21	949	1	0	0.169491486	528						
22	594	1	1	0.770926097	447						
23	864	1	0	0.309214827	505						
24	314	0	0	0.397414907	494						
25	392	0	1	0.761284973	448						
26	614	1	0	0.168895426	528						
27	699	0	1	0.526434235	479						

评分卡转换：

```
#评分卡转换
i=-1
k=0 #计数行，用于append进df1
df2 = pd.DataFrame(columns=['变量','类别','分数'])  #新建dataframe
for var in woedict.items():
    #var[0] :变量
    #var[1] :变量的woe字典
    i+=1
    for temp in var[1].keys():

        #var[1][temp] :变量内各个分箱的woe值
        #beta_lt[i] :各个变量对应的beta值
        point = -(B * beta_lt_b[i] * var[1][temp]) #计算变量分箱的评分
        point = round(point)                       #取整
        df2.loc[k] = [var[0],temp,point]           #写入dataframe
        k+=1   #行定位下移
```

转换结果写入 excel

```
df2.to_excel('German信用评分卡(iv大于0.8,LR_b平衡).xlsx')
```

实验结论：

综上所述，根据模型预测以及结合变量的 IV 值，得出['住房', '工作年限', '财产情况', '续存期数', '信用额度', '目的', '储蓄情况', '信用记录', '账户状态']这 9 个变量对预测函数的影响较大，即是对于目标变量'违约'的预测评分影响较大，具体的各变量、各类别评分见评分表。