贷款审批预测

案例背景

A公司是一家贷款公司，承接个人贷款业务。然而，贷款人员鱼龙混杂，能力不一。

- 如果贷款人能够正常还款，则A公司就可以从贷款人身上赚取利息。
- 如果贷款人还款逾期，则A公司就会承受一定程度的损失。

随着A公司的发展壮大，每年贷款的人数也越来越多，由于贷款人逾期还款的原因很多（个人人品，还款能力有限等），贷款业务对审核人员的要求也会较高。这既需要审核人员具有很强的行业背景知识，也需要一定的经验辅助。因此，大量的贷款审批要求A公司雇佣足够数量审批员工。这给A公司带来了一定的困扰：

- 廉价的审批人员专业能力不强，可能不能正确的发现逾期还款的贷款人员。
- 有经验的员工，成本又非常高。
- 人工审核需要一定的时间，可能无法做到及时性。

任务说明

我们的任务，是要根据历史贷款人的信息与还款结果，来建立有效的模型，这包括：

- 对贷款人员进行分类，预测其是否能够如期还款。
- 在分类的同时，还需要考虑其还款概率：
  - 如果贷款人能够如期还款的概率很高，则可以向贷款人提供贷款。
  - 如果贷款人存在一定的违约可能，则需要考虑是否向贷款人提供贷款。
  - 如果贷款人存在违约的可能性很高，则拒绝向贷款人提供贷款。
- 在分类的同时，根据其还款概率设立合适的阈值，从而权衡风险与利润，实现贷款利润最大化。

数据集描述

当贷款人申请贷款服务时，需要登记一些个人信息。A公司收集了大量的历史贷款人数据，主要的数据如下所示。关于更加详细的数据说明，请查看数据字典文件。

- id：贷款编号。
- member_id：会员编号。
- loan_amnt：借款人申请的贷款金额。
- funded_amnt：承诺给该贷款的总金额。
- funded_amnt_inv：投资者为该贷款承诺的总金额。
- term：贷款的偿还时间。
- int_rate：贷款的利率。
- installment：分期付款，每期还款的额度。
- grade：贷款等级。贷款利率越高，则等级越高。
- sub_grade：贷款子等级。
- emp_title：工作名称。
- emp_length：工作时间。
- home_ownership：房屋所有权状态。取值为：
  - RENT：出租
  - OWN：自由
  - MORTGAGE：按揭
  - OTHER：其他
- annual_inc：贷款人自报的年收入。
- verification_status：贷款人收入是否核实。
- issue_d：贷款月份。
- loan_status：贷款的当前状态。
- pymnt_plan：是否已经为贷款实施还款计划。
- url：贷款的url地址。
- desc：贷款人的贷款描述。
- purpose：贷款人贷款的用途。
- title：贷款人提供的标题。
- zip_code：邮政编码
- addr_state：贷款人所在的国家。
- dti：贷款人的总债务偿还总额与贷款人的月收入比值。
- delinq_2yrs：过去两年贷款人信用档案中逾期30天以上的拖欠事件。
- earliest_cr_line：贷款人最早报告的信贷额度开始的月份。
- inq_last_6mths：过去六个月的查询数目。
- open_acc：贷款人信用档案中的未结信用额度。
- pub_rec：贬损公共记录的数量。

- revol_bal：总信贷周转余额。

加载数据

通过pandas读取csv数据集文件。

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

sns.set(style="darkgrid", font_scale=1.2)
plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# skiprows：读取数据时跳过的行数。
#     数据的第一行是描述信息，因此使用skiprows跳过。第二行才是标题数据。
# id（贷款人编号）与next_pymnt_d（下一个预定的还款日期）列存在混合类型，显式指定列的类型，这样效率更高。
# Pandas默认以块的形式处理（解析）数据，从而降低内存消耗，但是可能会解析出混合类型。可以使用两种方式来处理：
#     1 显式通过dtype来设置列的类型。
#     2 将low_memory参数的值设置为False（默认为True）。

data = pd.read_csv("Loan.csv", skiprows=1, dtype={"id": np.str, "next_pymnt_d": np.str})
# 默认情况下，只显示20列数据。
print(pd.get_option("max_columns"))
# 查看数据集的形状。
print(data.shape)
# 显式指定最大列数。如果为None，指不限制最大显示列数。
pd.set_option("max_columns", 120)
data.head()
```

```
20
(42538, 111)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_g
0	1077501	1296599.0	5000.0	5000.0	4975.0	36 months	10.65%	162.87	B	B2
1	1077430	1314167.0	2500.0	2500.0	2500.0	60 months	15.27%	59.83	C	C4
2	1077175	1313524.0	2400.0	2400.0	2400.0	36 months	15.96%	84.33	C	C5
3	1076863	1277178.0	10000.0	10000.0	10000.0	36 months	13.49%	339.31	C	C1
4	1075358	1311748.0	3000.0	3000.0	3000.0	60 months	12.69%	67.79	B	B5

数据预处理

直观删除特征

通过观看数据集，可以直观上删除一些特征：

- 与贷款没有关联的特征，如id，memeber_id。
- 特征之间相关度极高，可以只保留一个即可。例如grade（贷款等级），sub_grade（贷款子等级）与int_rate（贷款利率），只保留int_rate即可。

```
# 删除与贷款无关特征。
irrelevant_coumns = ["id", "member_id", "funded_amnt", "funded_amnt_inv", "emp_title", "issue_d",
        "url", "desc", "zip_code", "addr_state", "last_credit_pull_d", "earliest_cr_line", "addr_state",
        "title", "last_pymnt_d"]
data.drop(irrelevant_coumns, axis=1, inplace=True)
# 删除相关性高的特征。
high_relevant = ["grade", "sub_grade"]
data.drop(high_relevant, axis=1, inplace=True)
```

缺失值处理

查看缺失值

- 可以通过info方法查看缺失值。
- 可以借助于isnull，notnull与sum等函数查看缺失值。

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42538 entries, 0 to 42537
Data columns (total 95 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   loan_amnt                       42535 non-null  float64
 1   term                            42535 non-null  object
 2   int_rate                        42535 non-null  object
 3   installment                     42535 non-null  float64
 4   emp_length                      41423 non-null  object
 5   home_ownership                  42535 non-null  object
 6   annual_inc                      42531 non-null  float64
 7   verification_status             42535 non-null  object
 8   loan_status                     42535 non-null  object
 9   pymnt_plan                      42535 non-null  object
 10  purpose                         42535 non-null  object
 11  dti                             42535 non-null  float64
 12  delinq_2yrs                     42506 non-null  float64
 13  inq_last_6mths                  42506 non-null  float64
 14  mths_since_last_delinq          15609 non-null  float64
 15  mths_since_last_record          3651 non-null   float64
 16  open_acc                        42506 non-null  float64
 17  pub_rec                         42506 non-null  float64
 18  revol_bal                       42535 non-null  float64
 19  revol_util                      42445 non-null  object
 20  total_acc                       42506 non-null  float64
 21  initial_list_status             42535 non-null  object
 22  out_prncp                       42535 non-null  float64
 23  out_prncp_inv                   42535 non-null  float64
 24  total_pymnt                     42535 non-null  float64
 25  total_pymnt_inv                 42535 non-null  float64
 26  total_rec_prncp                 42535 non-null  float64
 27  total_rec_int                   42535 non-null  float64
 28  total_rec_late_fee              42535 non-null  float64
 29  recoveries                      42535 non-null  float64
 30  collection_recovery_fee         42535 non-null  float64
 31  last_pymnt_amnt                 42535 non-null  float64
 32  next_pymnt_d                    2975 non-null   object
 33  collections_12_mths_ex_med      42390 non-null  float64
 34  mths_since_last_major_derog     0 non-null      float64
 35  policy_code                     42535 non-null  float64
 36  application_type                42535 non-null  object
 37  annual_inc_joint                0 non-null      float64
 38  dti_joint                       0 non-null      float64
 39  verification_status_joint       0 non-null      float64
 40  acc_now_delinq                  42506 non-null  float64
 41  tot_coll_amt                    0 non-null      float64
 42  tot_cur_bal                     0 non-null      float64
 43  open_acc_6m                     0 non-null      float64
 44  open_il_6m                      0 non-null      float64
 45  open_il_12m                     0 non-null      float64
 46  open_il_24m                     0 non-null      float64
 47  mths_since_rcnt_il              0 non-null      float64
```

```
 48  total_bal_il                  0 non-null      float64
 49  il_util                       0 non-null      float64
 50  open_rv_12m                   0 non-null      float64
 51  open_rv_24m                   0 non-null      float64
 52  max_bal_bc                    0 non-null      float64
 53  all_util                      0 non-null      float64
 54  total_rev_hi_lim              0 non-null      float64
 55  inq_fi                        0 non-null      float64
 56  total_cu_tl                   0 non-null      float64
 57  inq_last_12m                  0 non-null      float64
 58  acc_open_past_24mths          0 non-null      float64
 59  avg_cur_bal                   0 non-null      float64
 60  bc_open_to_buy                0 non-null      float64
 61  bc_util                       0 non-null      float64
 62  chargeoff_within_12_mths      42390 non-null  float64
 63  delinq_amnt                   42506 non-null  float64
 64  mo_sin_old_il_acct            0 non-null      float64
 65  mo_sin_old_rev_tl_op          0 non-null      float64
 66  mo_sin_rcnt_rev_tl_op         0 non-null      float64
 67  mo_sin_rcnt_tl                0 non-null      float64
 68  mort_acc                      0 non-null      float64
 69  mths_since_recent_bc          0 non-null      float64
 70  mths_since_recent_bc_dlq      0 non-null      float64
 71  mths_since_recent_inq         0 non-null      float64
 72  mths_since_recent_revol_delinq  0 non-null    float64
 73  num_accts_ever_120_pd         0 non-null      float64
 74  num_actv_bc_tl                0 non-null      float64
 75  num_actv_rev_tl               0 non-null      float64
 76  num_bc_sats                   0 non-null      float64
 77  num_bc_tl                     0 non-null      float64
 78  num_il_tl                     0 non-null      float64
 79  num_op_rev_tl                 0 non-null      float64
 80  num_rev_accts                 0 non-null      float64
 81  num_rev_tl_bal_gt_0           0 non-null      float64
 82  num_sats                      0 non-null      float64
 83  num_tl_120dpd_2m              0 non-null      float64
 84  num_tl_30dpd                  0 non-null      float64
 85  num_tl_90g_dpd_24m            0 non-null      float64
 86  num_tl_op_past_12m            0 non-null      float64
 87  pct_tl_nvr_dlq                0 non-null      float64
 88  percent_bc_gt_75              0 non-null      float64
 89  pub_rec_bankruptcies          41170 non-null  float64
 90  tax_liens                     42430 non-null  float64
 91  tot_hi_cred_lim               0 non-null      float64
 92  total_bal_ex_mort             0 non-null      float64
 93  total_bc_limit                0 non-null      float64
 94  total_il_high_credit_limit    0 non-null      float64
dtypes: float64(83), object(12)
memory usage: 30.8+ MB
```

缺失数据可视化

```
# 默认情况下，最多只显示60行数据。
print(pd.get_option("max_rows"))
# 如果需要显示完整，可以设置最多显示的行数。
# pd.set_option("max_rows", 200)
miss = data.isnull().sum(axis=0)
miss = pd.concat([miss, miss * 100 / data.shape[0]], axis=1)
miss.columns = ["miss_num", "miss_rate"]
miss.sort_values("miss_num", inplace=True, ascending=False)
display(miss)
```

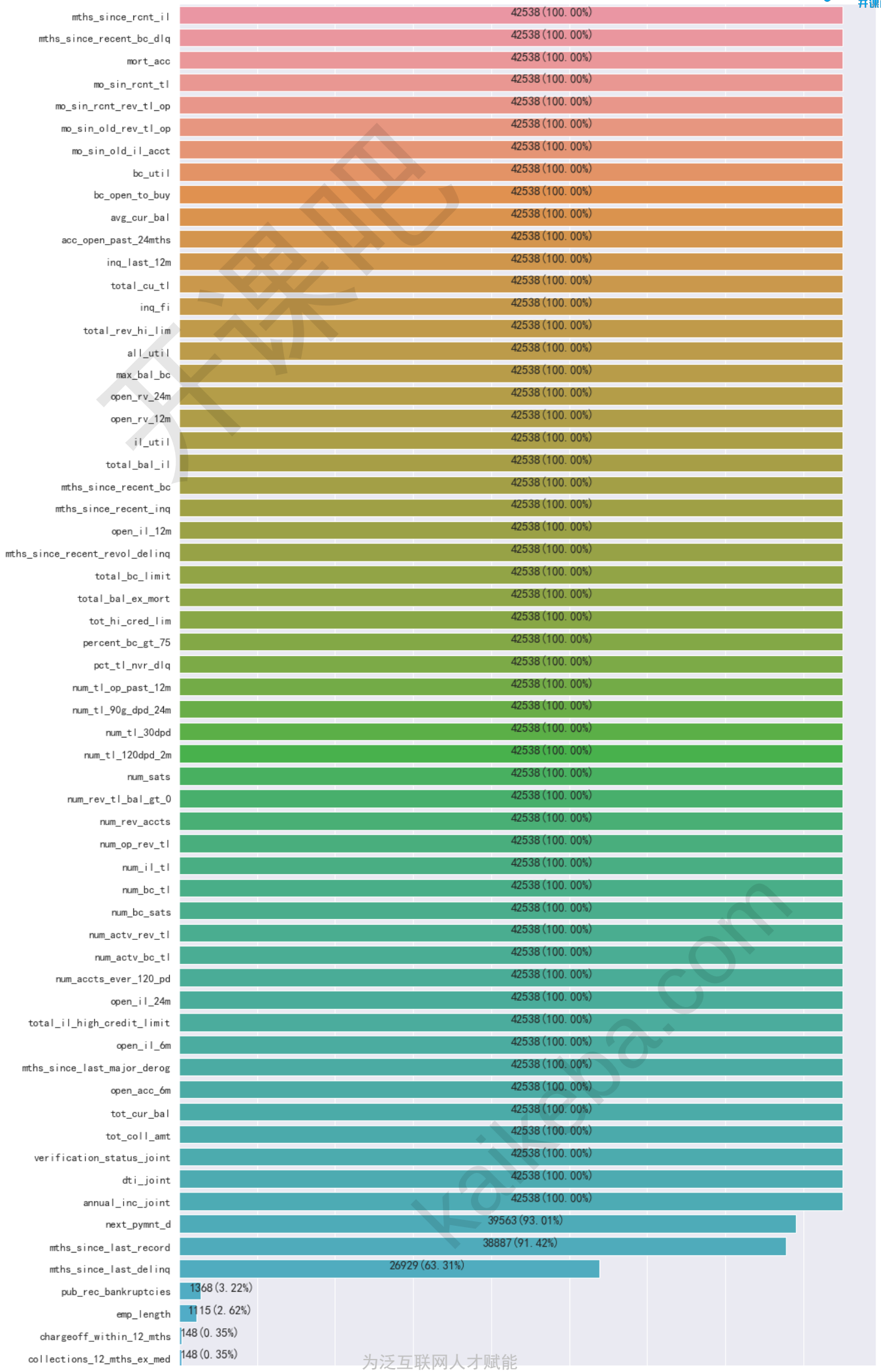
```
60
```

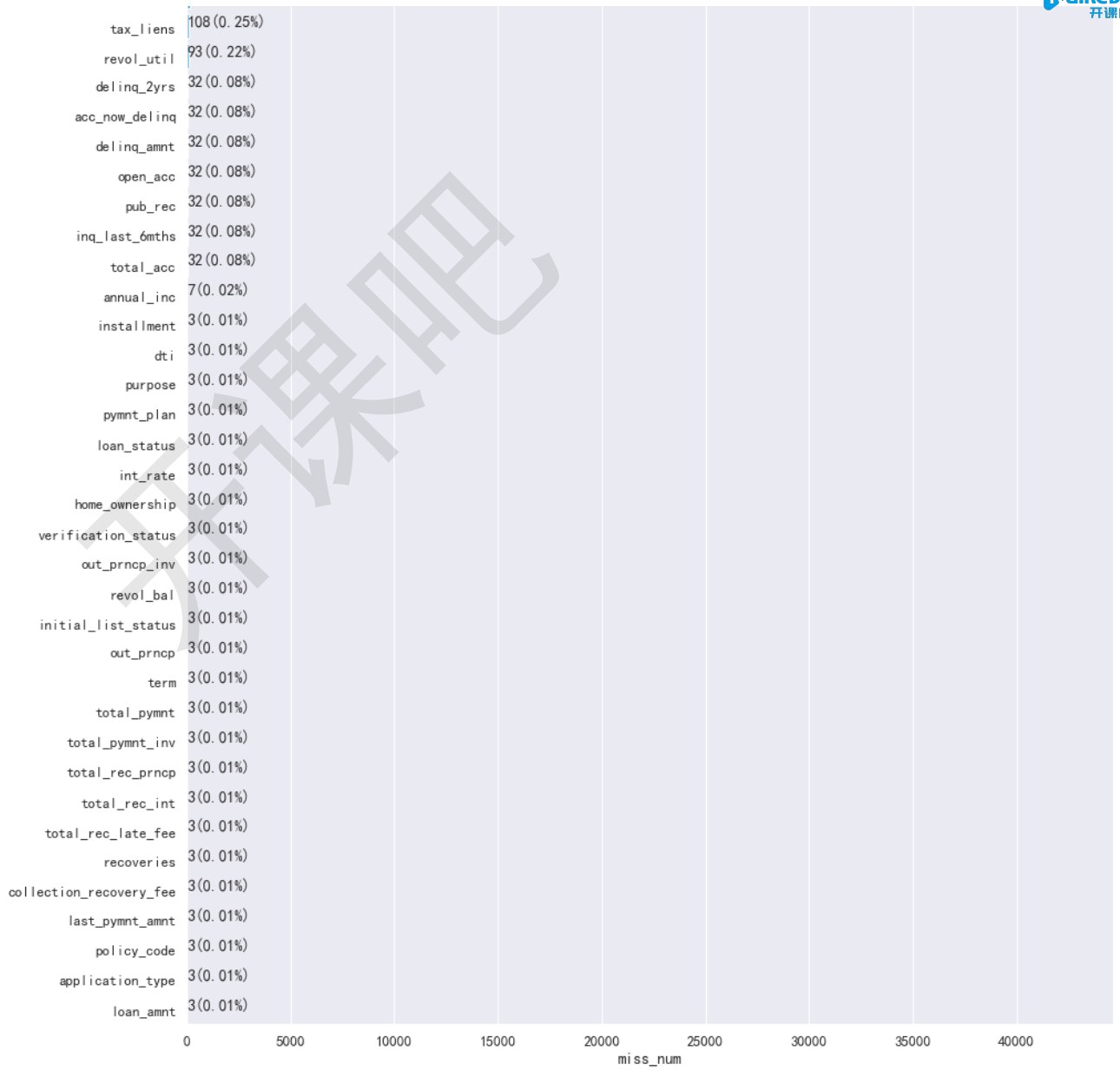
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	miss_num	miss_rate
mths_since_rcnt_il	42538	100.000000
mths_since_recent_bc_dlq	42538	100.000000
mort_acc	42538	100.000000
mo_sin_rcnt_tl	42538	100.000000
mo_sin_rcnt_rev_tl_op	42538	100.000000
...	...	...
collection_recovery_fee	3	0.007053
last_pymnt_amnt	3	0.007053
policy_code	3	0.007053
application_type	3	0.007053
loan_amnt	3	0.007053

95 rows × 2 columns

```
ax = sns.barplot(y=miss.index, x=miss["miss_num"])
figsize=(15, miss.shape[0] // 2)
ax.get_figure().set_size_inches(figsize)
for i in range(miss.shape[0]):
    num = miss["miss_num"].iloc[i]
    rate = miss["miss_rate"].iloc[i]
    ax.text(num / 2, i, f"{num}({rate:.2f}%)")
```





删除全部缺失数据

我们发现，数据集中有大量特征全部缺失，这样的特征没有任何意义，我们可以直接删除处理。

```
all_missing = miss[miss["miss_rate"] == 100].index
print("删除的特征：")
print(all_missing)
data.drop(all_missing, axis=1, inplace=True)
```

```
删除的特征：
Index(['mths_since_rcnt_il', 'mths_since_recent_bc_dlq', 'mort_acc',
       'mo_sin_rcnt_tl', 'mo_sin_rcnt_rev_tl_op', 'mo_sin_old_rev_tl_op',
       'mo_sin_old_il_acct', 'bc_util', 'bc_open_to_buy', 'avg_cur_bal',
       'acc_open_past_24mths', 'inq_last_12m', 'total_cu_tl', 'inq_fi',
       'total_rev_hi_lim', 'all_util', 'max_bal_bc', 'open_rv_24m',
       'open_rv_12m', 'il_util', 'total_bal_il', 'mths_since_recent_bc',
       'mths_since_recent_inq', 'open_il_12m',
       'mths_since_recent_revol_delinq', 'total_bc_limit', 'total_bal_ex_mort',
       'tot_hi_cred_lim', 'percent_bc_gt_75', 'pct_tl_nvr_dlq',
       'num_tl_op_past_12m', 'num_tl_90g_dpd_24m', 'num_tl_30dpd',
       'num_tl_120dpd_2m', 'num_sats', 'num_rev_tl_bal_gt_0', 'num_rev_accts',
       'num_op_rev_tl', 'num_il_tl', 'num_bc_tl', 'num_bc_sats',
       'num_actv_rev_tl', 'num_actv_bc_tl', 'num_accts_ever_120_pd',
       'open_il_24m', 'total_il_high_credit_limit', 'open_il_6m',
       'mths_since_last_major_derog', 'open_acc_6m', 'tot_cur_bal',
       'tot_coll_amt', 'verification_status_joint', 'dti_joint',
       'annual_inc_joint'],
      dtype='object')
```

二值化特征

对于缺失比率过多的特征（通常大于80%），特征原有的值，已经没有太多意义，我们可以将特征进行二值化处理，只记录该特征是否缺失。

```
miss_too_much = miss[(miss["miss_rate"] > 80) & (miss["miss_rate"] < 100)].index
print("缺失值过多的特征：")
print(miss_too_much)
```

```
缺失值过多的特征：
Index(['next_pymnt_d', 'mths_since_last_record'], dtype='object')
```

获取缺失值在80% ~ 100%之间的特征，进行二值化处理。

```
# next_pymnt_d：下一个计划的还款日期。
# mths_since_last_record：自上次记录以来的月数。
for c in miss_too_much:
    # 1与0只需要1个字节就可以存储。
    data[c] = data[c].isnull().astype(np.int8)
data.loc[:, miss_too_much].head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	next_pymnt_d	mths_since_last_record
0	1	1
1	1	1
2	1	1
3	1	1
4	0	1

填充并建立辅助特征

对于缺失比率在20% ~ 80%的特征，可以对缺失值进行填充，同时建立辅助特征。

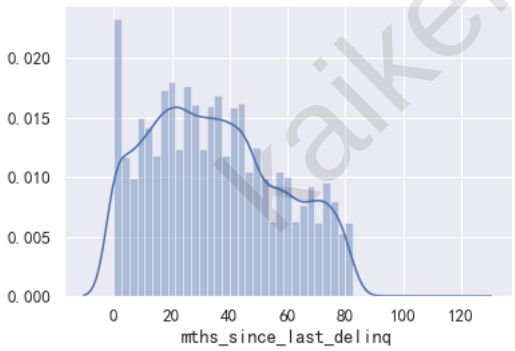
```
missing_medium = miss[(miss["miss_rate"] > 20) & (miss["miss_rate"] <= 80)].index
print("缺失值适中的特征：")
print(missing_medium)
```

```
缺失值适中的特征：
Index(['mths_since_last_delinq'], dtype='object')
```

我们首先来绘制数据的分布情况。

```
from scipy import stats

# 自借款人上次拖欠债务以来的月数。
sns.distplot(data["mths_since_last_delinq"])
stats.normaltest(data["mths_since_last_delinq"].dropna())
```

```
NormaltestResult(statistic=1979.824429462228, pvalue=0.0)
```



```
# 建立辅助特征，用来标记特征原有的值是否缺失。
data["mths_since_last_delinq_indicator"] = data["mths_since_last_delinq"].isnull().astype(np.int8)
# 计算中位数，并填充缺失值。
m = data["mths_since_last_delinq"].median()
data["mths_since_last_delinq"].fillna(m, inplace=True)
```

填充缺失特征

对于缺失比率在20%以内的特征，可以对缺失值进行填充，同时，如果缺失比率非常少，可以直接删除含有缺失值的记录。

```
missing_less = miss[(miss["miss_rate"] > 0) & (miss["miss_rate"] <= 20)].index
print("缺失值较少的特征：")
print(missing_less)
```

```
缺失值较少的特征：
Index(['pub_rec_bankruptcies', 'emp_length', 'chargeoff_within_12_mths',
       'collections_12_mths_ex_med', 'tax_liens', 'revol_util', 'delinq_2yrs',
       'acc_now_delinq', 'delinq_amnt', 'open_acc', 'pub_rec',
       'inq_last_6mths', 'total_acc', 'annual_inc', 'installment', 'dti',
       'purpose', 'pymnt_plan', 'loan_status', 'int_rate', 'home_ownership',
       'verification_status', 'out_prncp_inv', 'revol_bal',
       'initial_list_status', 'out_prncp', 'term', 'total_pymnt',
       'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int',
       'total_rec_late_fee', 'recoveries', 'collection_recovery_fee',
       'last_pymnt_amnt', 'policy_code', 'application_type', 'loan_amnt'],
      dtype='object')
```

通过之前的可视化，我们发现以上数据的缺失比例都非常小，这里我们直接删除。

```
data.dropna(inplace=True)
#缺失值处理完成后，我们再次来进行检查。
(data.isnull().sum() > 0).any()
```

```
False
```

重复值

可通过duplicated方法查看重复值数量。可以通过布尔数组提取元素的方式，查看重复值的具体信息。
可通过drop_duplicates方法删除重复值。

```
data.duplicated().sum()
# data.drop_duplicates(inplace=True)
```

```
0
```

数据转换与特征工程

机器学习模型接收数值类型的数据，因此，为了能够将数据输入模型中进行训练，需要将非数值类型转换为数值类型。

查看所有非数值类型

首先查看下，哪些列为非数值类型。

```
not_number = []
for k, v in data.dtypes.items():
    if not np.issubdtype(v, np.number):
        not_number.append(k)
print("非数值变量：")
print(not_number)
```

```
非数值变量：
['term', 'int_rate', 'emp_length', 'home_ownership', 'verification_status', 'loan_status', 'pymnt_plan', 'purpose', 'revol_util',
'initial_list_status', 'application_type']
```

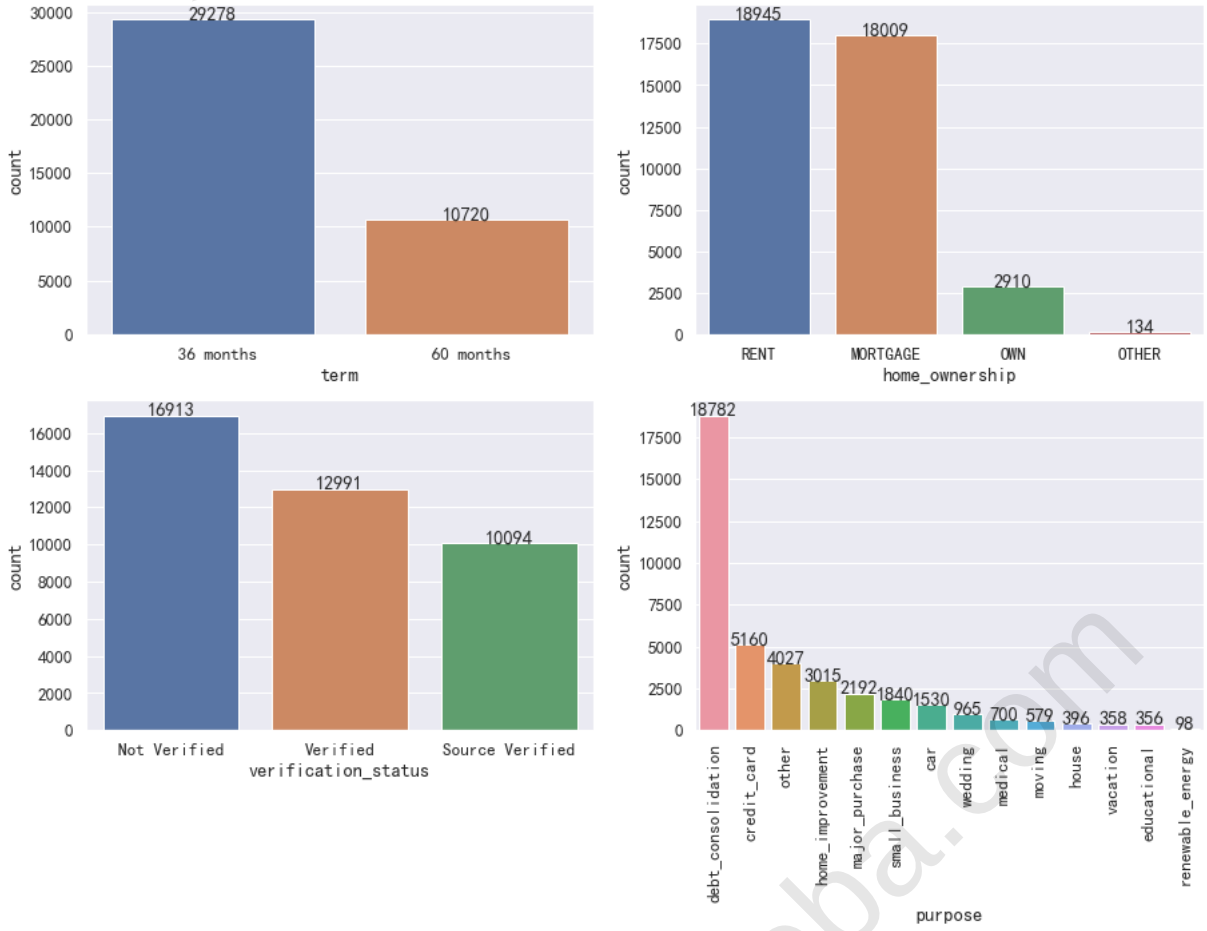
转换与提取特征

这里，我们分别对每一个非数值列，我们可以使用value_counts来查看其取值与每个取值的频数，提取有用的特征。

term，home_ownership，verification_status，purpose

这几列都是类别类型变量，我们可以使用one-hot编码来对其进行处理。

```
def plot_var(name_list):
    """绘制变量的每个类别的数量。

    Parameters
    ----------
    name : str
        变量的名称。

    """
    num = len(name_list)
    row, col = np.ceil(num / 2).astype(np.int32), 2
    fig, ax = plt.subplots(row, col)
    fig.set_size_inches(15, row * 5)
    ax = ax.ravel()
    for index, name in enumerate(name_list):
        v = data[name].value_counts()
        sns.countplot(x=name, data=data, order=v.index, ax=ax[index])
        # 在图像上绘制数值。
        for x, y in enumerate(v):
            t = ax[index].text(x, y, y)
            # 数值居中对齐。
            t.set_ha("center")
        if len(v) > 10:
            ax[index].set_xticklabels(ax[index].get_xticklabels(), rotation=90)


plot_var(["term", "home_ownership", "verification_status", "purpose"])
```




```
columns = ["term", "home_ownership", "verification_status", "purpose"]
dummy = pd.get_dummies(data[columns])
display(dummy.head())
data = pd.concat([data, dummy], axis=1)
data = data.drop(columns, axis=1)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	term_36 months	term_60 months	home_ownership_MORTGAGE	home_ownership_OTHER	home_ownership_OWN	home_ownership_RENT	verification_
0	1	0	0	0	0	1	0
1	0	1	0	0	0	1	0
2	1	0	0	0	0	1	1
3	1	0	0	0	0	1	0
4	0	1	0	0	0	1	0

int_rate与revol_util

int_rate与revol_util含有数值，但是具有%，可以去掉%值。

```
# 也可以使用map或者apply方法来实现相同的功能。
data["int_rate"] = data["int_rate"].str.replace("%", "").astype(np.float32)
data["revol_util"] = data["revol_util"].str.replace("%", "").astype(np.float32)
```

emp_length

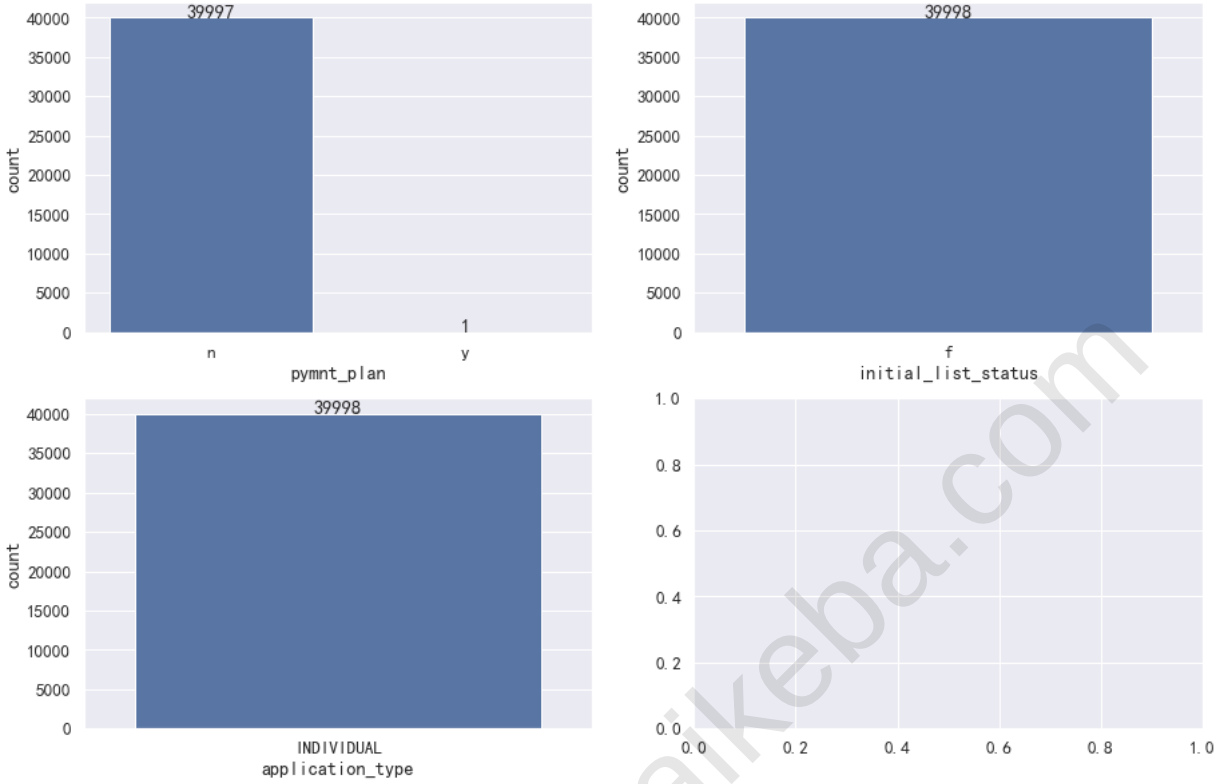
该列含有数值，这里需要转换为对应的数值类型。

```
map_dict = {
    "10+ years": 10, "9 years": 9, "8 years": 8,
    "7 years": 7, "6 years": 6, "5 years": 5,
    "4 years": 4, "3 years": 3, "2 years": 2,
    "1 year": 1, "< 1 year": 0
}
data["emp_length"] = data["emp_length"].map(map_dict)
```

pymnt_plan，initial_list_status，application_type

这三列数值（几乎）都是同一个值，对建模没有帮助，可以删除。

```
plot_var(["pymnt_plan", "initial_list_status", "application_type"])
```




```
data.drop(["pymnt_plan", "initial_list_status", "application_type"], axis=1, inplace=True)
```

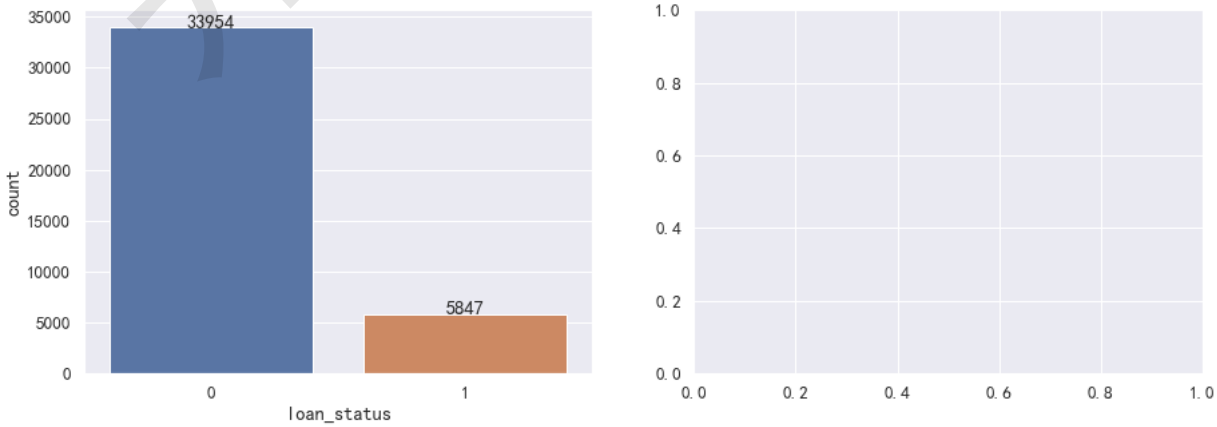
loan_status

loan_status列为标签列，表示还款状态。

```
data["loan_status"].value_counts()
```

```
Fully Paid                                            32447
Charged Off                                            5294
Does not meet the credit policy. Status:Fully Paid     1507
Does not meet the credit policy. Status:Charged Off     529
Current                                                 196
Late (31-120 days)                                       10
In Grace Period                                           9
Late (16-30 days)                                         5
Default                                                   1
Name: loan_status, dtype: int64
```

```python
def mapping(key):
    if "Fully Paid" in key:
        return 0
    else:
        return 1

# 选择正常还款与延期的记录。
data = data.loc[(data["loan_status"] != "Current") & (data["loan_status"] != "Default")]
data["loan_status"] = data["loan_status"].map(mapping)
plot_var(["loan_status"])
```



建立模型

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

lr = LogisticRegression(solver="liblinear")
y = data["loan_status"]
X = data.drop("loan_status", axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
lr.fit(X_train, y_train)
y_hat = lr.predict(X_test)
print("真实值：", y_test.values[:10])
print("预测值：", y_hat[:10])
```

```
真实值：  [0 0 1 0 0 0 0 1 0 0]
预测值：  [0 0 1 0 0 0 0 1 0 0]
```

分类模型评估

混淆矩阵

混淆矩阵，可以用来评估模型分类的正确性。该矩阵是一个方阵，矩阵的数值用来表示分类器预测的结果，包括真正例（True Positive），假正例（False Positive），真负例（True Negative），假负例（False Negative）。

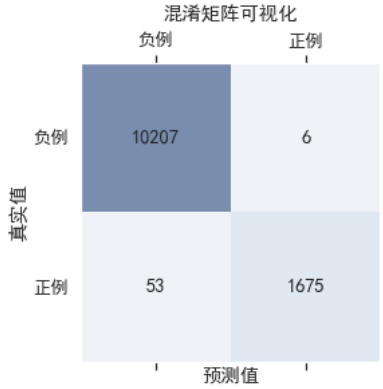
			预测值
		负例	正例
真实值	负例	真负例（TN）	假正例（FP）
	正例	假负例（FN）	真正例（TP）

混淆矩阵解释如下（1为正例，0为负例）。

```python
from sklearn.metrics import confusion_matrix

# 根据传入的真实值与预测值，创建混淆矩阵。
matrix = confusion_matrix(y_true=y_test, y_pred=y_hat)
print(matrix)
```

```
[[10207     6]
 [   53  1675]]
```

```python
mat = plt.matshow(matrix, cmap=plt.cm.Blues, alpha=0.5)
label = ["负例", "正例"]
# 获取当前的绘图对象。
ax = plt.gca()
# 可以一次性设置多个属性。
ax.set(xticks=np.arange(matrix.shape[1]), yticks=np.arange(matrix.shape[0]),
       xticklabels=label, yticklabels=label, title="混淆矩阵可视化\n",
       ylabel="真实值", xlabel="预测值")
for i in range(matrix.shape[0]):
    for j in range(matrix.shape[1]):
        plt.text(x=j, y=i, s=matrix[i, j], va="center", ha="center")
plt.grid(False)
# Matplotlib 3.1.1版本需要添加如下的代码，否则可视化显示不完整。（bug）
# a, b = ax.get_ylim()
# ax.set_ylim(a + 0.5, b - 0.5)
# plt.show()
```



课堂练习

关于混淆矩阵，说法正确的是（ ）。【不定项】
A 混淆矩阵可以用来评估分类模型。
B 混淆矩阵中，TP与TN的数值越大，则分类的效果越好。
C 混淆矩阵一行元素的和代表某个类别的实际数量。
D 混淆矩阵一列元素的和代表某个类别的预测数量。

评估指标

对于分类模型，我们可以提取如下的评估指标：

- 正确率（accuracy）
- 精准率（precision）
- 召回率（recall）
- F1（调和平均值）

正确率

正确率定义如下：

$$正确率 = \frac{TP+TN}{TP+TN+FP+FN}$$

使用正确率，可以很好的评估一个分类模型的好坏吗？
A 可以，没有问题。
B 不可以，可能会有局限性。

精准率

精准率定义如下：

$$精准率 = \frac{TP}{TP+FP}$$

召回率

召回率定义如下：

$$召回率 = \frac{TP}{TP+FN}$$

调和平均值F1

调和平均值F1定义如下：

$$F1 = \frac{2*precision*recall}{precision+recall}$$

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print("正确率：", accuracy_score(y_test, y_hat))
# 默认将1类别视为正例，可以通过pos_label参数指定。
print("精准率：", precision_score(y_test, y_hat))
print("召回率：", recall_score(y_test, y_hat))
print("F1调和平均值：", f1_score(y_test, y_hat))
# 我们也可以调用逻辑回归模型对象的score方法，也能获取正确率。
# 但是需要注意，score方法与正确率（accuracy_score）函数的参数是不同的。
print("score方法计算正确率：", lr.score(X_test, y_test))
```

```
正确率： 0.9950590402813835
精准率： 0.9964306960142773
召回率： 0.9693287037037037
F1调和平均值： 0.9826928718099149
score方法计算正确率： 0.9950590402813835
```

除此之外，我们也可以使用classification_report函数来查看模型的分类统计信息，该方法会返回字符串类型，给出相关的分类指标评估值。

```
from sklearn.metrics import classification_report

print(classification_report(y_true=y_test, y_pred=y_hat))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     10213
           1       1.00      0.97      0.98      1728

    accuracy                           1.00     11941
   macro avg       1.00      0.98      0.99     11941
weighted avg       1.00      1.00      1.00     11941
```

ROC曲线

ROC

ROC曲线（Receiver Operating Characteristic——受试者工作特征曲线），使用图形来描述二分类系统的性能表现。图形的纵轴为真正例率（TPR——True Positive Rate），横轴为假正例率（FPR——False Positive Rate）。其中，真正例率与假正例率定义为：

$$TPR = \frac{TP}{\text{真实正例数量}} = \frac{TP}{TP+FN}$$
$$FPR = \frac{FP}{\text{真实负例数量}} = \frac{FP}{FP+TN}$$

ROC曲线通过真正例率（TPR）与假正例率（FPR）两项指标，可以用来评估分类模型的性能。真正例率与假正例率可以通过移动分类模型的阈值而进行计算。随着阈值的改变，真正例率与假负例率也会随之发生改变，进而就可以在ROC曲线坐标上，形成多个点。

ROC曲线反映了FPR与TPR之间权衡的情况，通俗来说，即在TPR随着FPR递增的情况下，谁增长得更快，快多少的问题。TPR增长得越快，曲线越往上凸，模型的分类性能就越好。

ROC曲线如果为对角线，则可以理解为随机猜测。如果在对角线以下，则其性能比随机猜测还要差。如果ROC曲线真正例率为1，假正例率为0，即曲线为$x=0$与$y=1$构成的折线，则此时的分类器是最完美的。

AUC

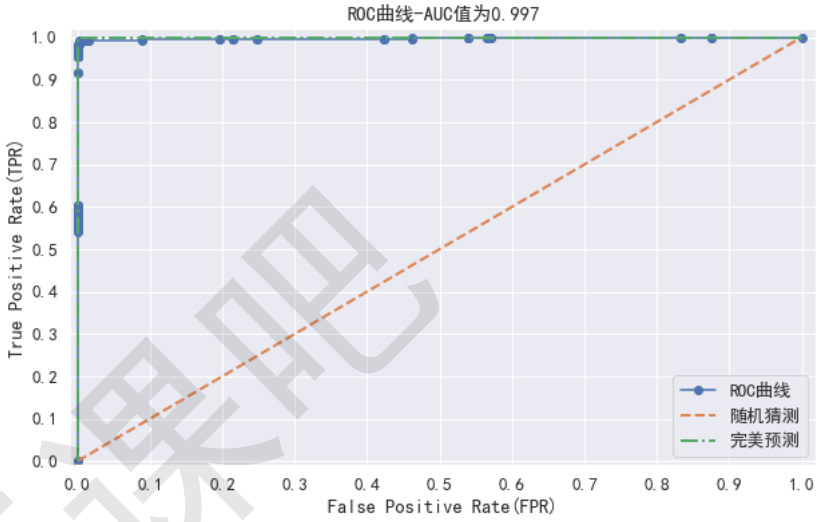
AUC（Area Under the Curve）是指ROC曲线下的面积，使用AUC值作为评价标准是因为有时候ROC曲线并不能清晰的说明哪个分类器的效果更好，而AUC作为数值可以直观的评价分类器的好坏，值越大越好。

```
from sklearn.metrics import roc_curve, auc, roc_auc_score

probo = lr.predict_proba(X_test)
# 返回ROC曲线相关值。返回FPR，TPR与阈值。当分值达到阈值时，将样本判定为正类，
# 否则判定为负类。
# y_true：二分类的标签值（真实值）。
# y_score：每个标签（数据）的分值或概率值。当该值达到阈值时，判定为正例，否则判定为负例。
# 在实际模型评估时，该值往往通过决策函数（decision_function）或者概率函数（predict_proba）获得。
# pos_label：指定正例的标签值。
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=probo[:, 1], pos_label=1)
# 对概率降序排列，然后从中选择若干元素作为阈值，每个阈值下，都可以计算一个tpr与fpr，
# 每个tpr与fpr对应ROC曲线上的一个点，将这些点进行连接，就可以绘制ROC曲线。
print(probo.shape, fpr.shape, tpr.shape, thresholds.shape)
print(thresholds[:10])
# auc与roc_auc_score函数都可以返回AUC面积值，但是注意，两个函数的参数是不同的。
print("AUC面积值：", auc(fpr, tpr))
print("AUC面积得分：", roc_auc_score(y_true=y_test, y_score=probo[:, 1]))
```

```
(11941, 2) (85,) (85,) (85,)
[2. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
AUC面积值： 0.9970317424052859
AUC面积得分： 0.9970317424052859
```

有了fpr与tpr的值，绘制ROC曲线是非常容易的，只不过是最简单的一个plot而已。

```
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, marker="o", label="ROC曲线")
plt.plot([0,1], [0,1], lw=2, ls="--", label="随机猜测")
plt.plot([0, 0, 1], [0, 1, 1], lw=2, ls="-.", label="完美预测")
plt.xlim(-0.01, 1.02)
plt.ylim(-0.01, 1.02)
plt.xticks(np.arange(0, 1.1, 0.1))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.grid(True)
plt.title(f"ROC曲线-AUC值为{auc(fpr, tpr):.3f}")
plt.legend()
plt.show()
```



P-R曲线

P-R(precision-recall)曲线，其横轴为召回率，纵轴为精准率。曲线上的点为在不同阈值下，精准率与召回率的对应关系。

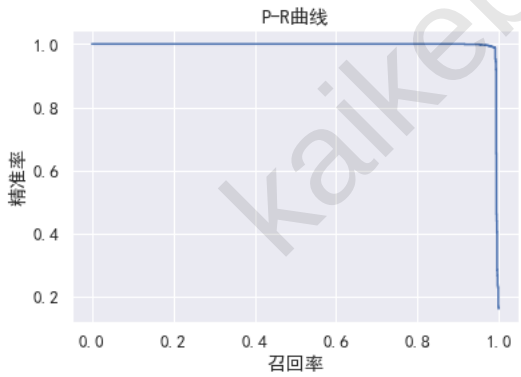
关于P-R曲线，说明如下：

- P-R曲线一定会过(0, 0)点。
  - 当阈值很大，P为0，TP与FP均为0。
- 精准率与召回率无法同时增大，一个增大时，另外一个可能就会降低。
  - 可能降低，不表示一定会降低。
  - 也可能保持不变。
- 随着召回率的增加，精准率一定会呈现下降的趋势。
- 当对精准率或召回率具有定量要求时，P-R曲线就会非常有用。

```
from sklearn.metrics import precision_recall_curve

# 计算在不同阈值下的精准率与召回率。
# y_true：每个样本的真实值。
# probas_pred：每个样本的概率值（或z值）。
# pos_label：指定正例类别。
# 函数会返回3个值：
# precision：每个样本在对应阈值下的精准率，最后一个元素值为1。
# recall：每个样本在对应阈值下的召回率。最后一个元素值为0。
# thresholds：升序排列的阈值数组。【说明：因为阈值是升序排列的，因此，可以推导出
# 精准率升序排列，召回率降序排列。】
precision, recall, thresholds = precision_recall_curve(y_test, probo[:, 1], pos_label=1)
# 阈值会从y_true参数的数组中来选取一部分。注意：阈值的数量比precision与recall少1。
print(precision.shape, recall.shape, thresholds.shape, y_test.shape)
plt.plot(recall, precision)
plt.xlabel("召回率")
plt.ylabel("精准率")
plt.title("P-R曲线")
```

```
(9693,) (9693,) (9692,) (11941,)
```

```
Text(0.5, 1.0, 'P-R曲线')
```



例如，我们要求贷款坏账率不超过5%，即召回率不低于95%，我们就可以找到在召回率不低于95%下的精准率，并找出能够使得F1值最大的阈值。

```
min_recall = 0.95
# 召回率降序排列，截取的一定是数组中的前半部分。
boundary_index = recall[recall >= min_recall].shape[0]
pre = precision[:boundary_index]
rec = recall[:boundary_index]
thr = thresholds[:boundary_index]
f1 = (2 * pre * rec) / (pre + rec)
index = f1.argmax()
print("最佳阈值：", thr[index])
print("最佳F1值：", f1.max())
print("最佳F1值时的精准率：", pre[index])
print("最佳F1值时的召回率：", rec[index])
```

```
最佳阈值： 0.13810438211219303
最佳F1值： 0.9898873158046807
最佳F1值时的精准率： 0.9884593190998269
最佳F1值时的召回率： 0.9913194444444444
```

课堂练习

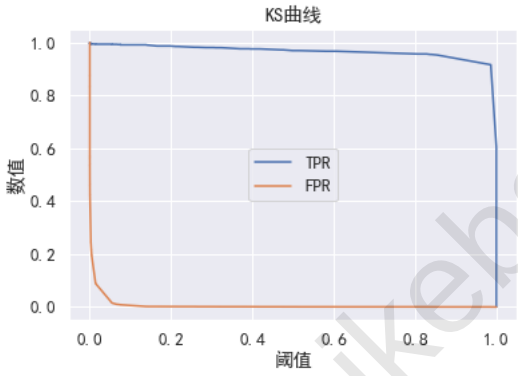
P-R曲线会不会过（1,0）这个点?
A 会。
B 不会。
C 可能会，可能不会。

KS曲线

KS曲线（Kolmogorov-Smirnov），可以用来衡量模型风险的区分能力。在KS曲线中，横轴为阈值，纵轴为TPR与FPR的数值。TRP与FPR差值最大的位置，就是模型区分能力最强的位置，也就是我们应该取得阈值的位置。

```
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=probo[:, 1], pos_label=1)
# roc_curve中，thresholds索引为0的元素值会大于1，为了能够正常显示，
# 将thresholds索引为0的元素修改为1。
thresholds[0] = 1
plt.plot(thresholds, tpr, label="TPR")
plt.plot(thresholds, fpr, label="FPR")
plt.legend()
plt.xlabel("阈值")
plt.ylabel("数值")
plt.title("KS曲线")
```

```
Text(0.5, 1.0, 'KS曲线')
```




```
diff = tpr - fpr
index = diff.argmax()
print("最大差值位置：", index)
print("对应的阈值：", thresholds[index])
```

```
最大差值位置： 53
对应的阈值： 0.13810438211219303
```

样本不均衡处理

不均衡现象

样本不均衡，是指在分类中，样本的类别数量相差较为悬殊。在实际的应用场景中，经常会出现样本不均衡的情况。例如，医院体检时，有病与没病的比例，没病的人会远远多于生病的人。
当使用不均衡的数据训练模型时，模型往往会特别"眷顾"类别数量多的样本数据，因为样本数量多的类别会得到更多的训练。

解决方案

关于解决方案，老梁提供辅助视频，供大家参考学习。