基于LendingClub数据的信贷分析和建模报告

一：课题分析

二：数据获取

三：数据探索
3．1主要特征含义理解
3．2特征分布
3．2．1目标特征分布
3．2．2分类变量的分布
3．2．3连续数值特征分布
3．2．4时序特征分布
3．2．5文字特征分布
3．2．6两两特征的协方差

四：数据预处理
4．1数据集划分
4．2特征缺失值识别与处理
4．2．1严重缺失值的处理
4．2．2缺失值填充
4．3同值性特征识别与处理
4．4特征格式变换
4．5文本特征处理
4．5．1工作机构分类
4．5．2借款人州地址分类
4．5．2．1 k-means聚类分析
4．5．2．2 等频分箱
4．6时序特征处理
4．7特征编码
4．8归一化处理
4．9警惕数据泄露
4．9．1警惕不恰当特征
4．9．2错误的交叉验证策略

五：特征工程
5．1特征衍生
5．2筛选变量
特征筛选的目的
5．2．1依据共线性筛选变量
5．2．2依据ＩＶ筛选变量
5．3特征分箱
5．3．1对无缺省值的连续型数值变量分箱
5．3．2对含有缺省值的连续型数值变量分箱

六：建模
6．1建立评分卡
6．2建立随机森林模型

七：总结

一：课题分析
课题：
研究小微企业主贷款的风险特征，提出可应用性强的风险评估模型构建方案。
课题分析：
小微企业贷款信用评估与小微企业主个人信用情况关系密切，本文将以 lendingclub 信贷平台上的公开数据作为小微企业主信贷数据模拟样本，构造一个简单明了的传统信贷申请评分卡（A卡），和一个解释性较差的黑箱预测模型，用于辅助决策。

二：数据获取
根据巴塞尔协议提供的经验，正常还款１２期以上的贷款人，其还款状态会趋于稳定，因此，我选择LendingClub平台(以下简称LC）２０１７年Ｑ１的数据，那么我们的样本到目前为止还款全部超过12个月，其数据更新度高，样本可以被有效利用。

```
In [53]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42538 entries, 0 to 42537
Columns: 145 entries, id to settlement_term
dtypes: float64(115), object(30)
memory usage: 47.1+ MB
```

由上图，样本共有４２５３８条数据，共有１４５个特征变量，样本量够大，对它们的分析具有统计意义。
此样本是经过ＬＣ平台依据一些条件（比如ＦＩＣＯ值）筛选过的美国借款人的样本，因此应用具有一定局限性。

三：数据探索
在着手处理数据之前，先了解基本的数据情况，为进一步的数据探索、数据预处理、特征工程和建模做准备。
３．１主要特征含义理解
主要特征摘录：

序号	变量简称	含义	数据类型	数据分类
1	loan_amnt	贷款申请金额	数值	信贷基本信息
2	funded_amnt	申请时可用于贷款的金额	数值	平台资产信息
3	term	借款周期	数值	借贷基本信息
4	int_rate	借款利率	数值	借贷基本信息
5	grade	LC指定的借贷等级	字符	平台指定信息
6	emp_length	工作年限	字符	借款人个人能力
7	home_ownership	住房性质	字符	借款人个人能力
8	annual_inc	年收入	数值	借款人个人能力
9	verification_status	收入来源是否核实	字符	平台指定信息
10	loan_status	借款状态	字符	目标变量
11	purpose	借款目的	字符	信贷基本信息
12	dti	负债率	数值	借款人个人能力
13	delinq_2yrs	过去 2 年逾期 3 0 天以上次数	数值	信贷历史
14	inq_last_6mths	过去 6 个月征信查询次数	数值	信用查询历史
15	open_acc	借款人信用档案中未结信用额度的数目	数值	信贷历史
16	pub_rec	借款人负面公共记录	数值	公共信息
17	revol_bal	总尚未结清信贷金额	数值	信贷历史
18	total_acc	当前借款人信用档案中总信用额度	数值	信贷历史
19	open_il_12m	过去 1 2 个月内所开分期付款账户数	数值	信贷历史
20	mths_since_rcnt_il	最近一次开分期账户距现在的月份数	数值	信贷历史
21	all_util	总信用余额	数值	信贷历史
22	total_rev_hi_lim	总高额授信/信用限额	数值	信贷历史
23	acc_now_deling	借款人现在有拖欠的账户数	数值	信贷历史
24	addr_state	借款人所在地	字符	借款人个人基本信息
25	ealiest_cr_line	借款人首开信用卡时间	时序	信贷历史
	…	…	…	…

3． 2 特征分布

3． 2． 1 目标特征分布

```
In [52]: # target
         df.loan_status.value_counts()

Out[52]: Fully Paid
           34116
         Charged Off
            5670
         Does not meet the credit policy. Status:Fully Paid
            1988
         Does not meet the credit policy. Status:Charged Off
             761
         Name: loan_status, dtype: int64
```

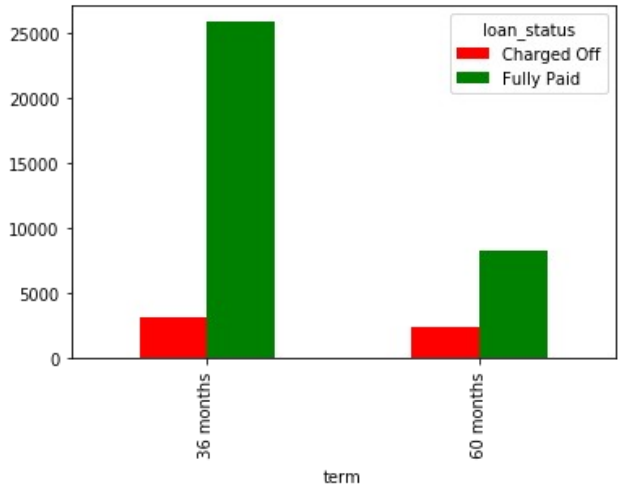
fully paid:完全结清 charged off:坏账注销

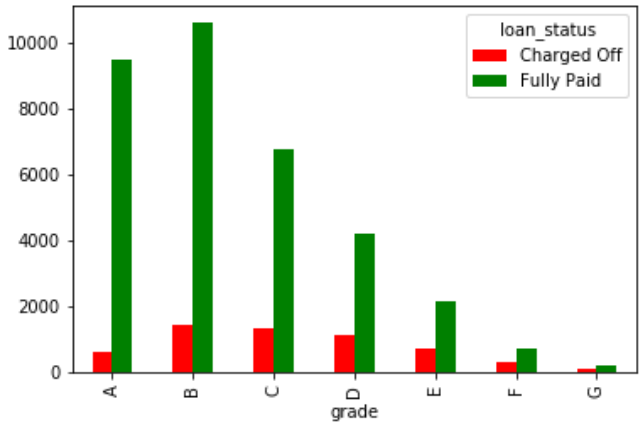
由上表可知，此样本集是一个不平衡数据集。在后续过程中需要考虑到这一点。

3． 2． 2 分类变量的分布

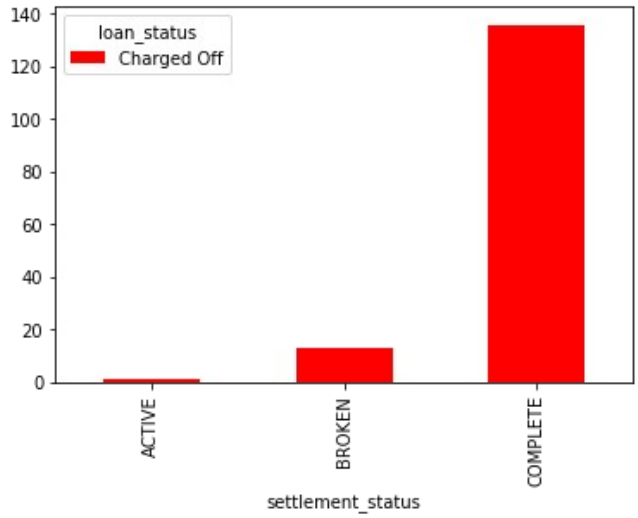
探索典型分类变量依据好坏样本两类的分布并进行可视化展示：

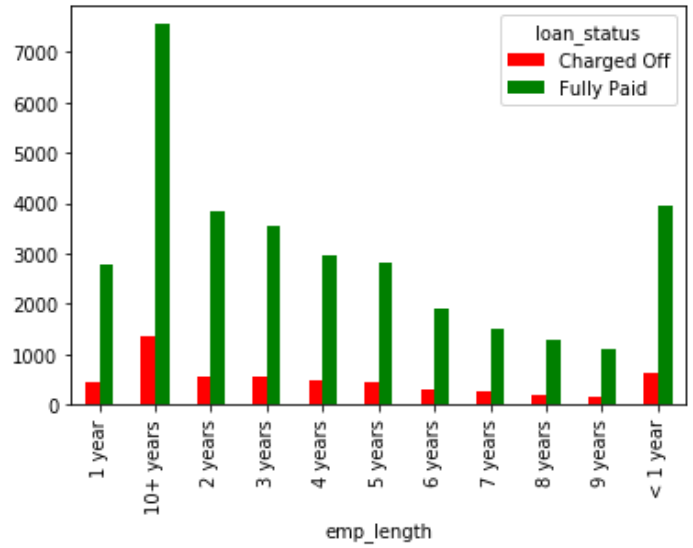
```
In [100]: # 分类特征的分布
cla_feas = ['term', 'grade', 'emp_length', 'home_ownership', 'verification_status', 'purpose',
            'debt_settlement_flag','settlement_status']
for fea in cla_feas:
    pvt = pd.pivot_table(df[['loan_status', fea]], index=fea, columns='loan_status', aggfunc=len)
    # plot pivot table with bar plot, green bar represent fully
    pvt.plot(kind='bar', color=['r', 'g'])
```



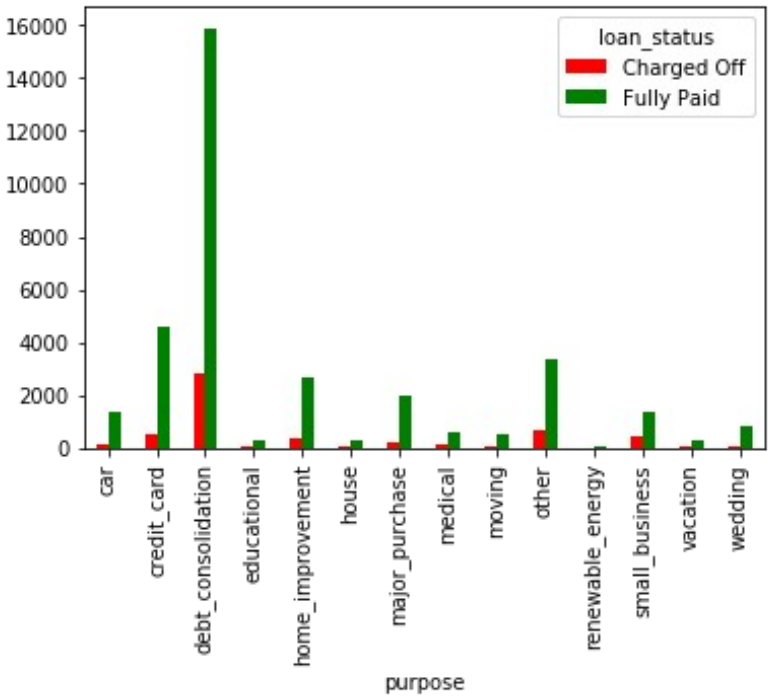


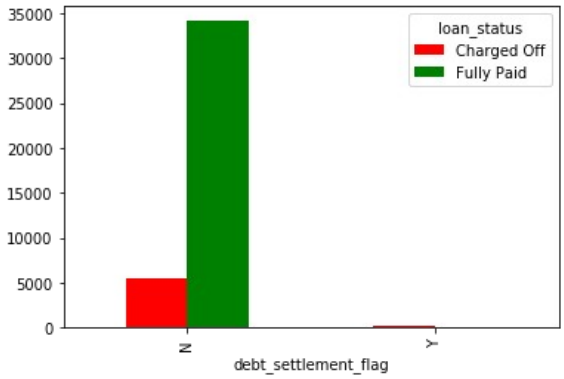
由上左图：
借款人多选择 3 6 期贷款，选择 6 0 期贷款的违约率要高一些。
由上右图：
我们需要首先理解一下 grade 这个特征。grade 是 L C 自评等级，不同的网贷平台投资人有不一样的风险收益偏好，L C 平台依据这种多样化的需求，将借贷人分成 A － G 七个等级。L C 使用复杂的算法对每笔贷款予以评级，这个评级和借款人的利率息息相关（这也说明，grade 与某些特征是存在关系的）。比如说，那些信用历史好，还款能力好的借款人利率偏低，约 7 %，其贷款等级通常为 A 级。从 A 到 G，贷款的风险越来越高，利率也越来越高。从图中，我们也可以看出这个趋势。
另外，无论是投资人还是借贷者，大多数都是选择较低风险较低收益的类型。



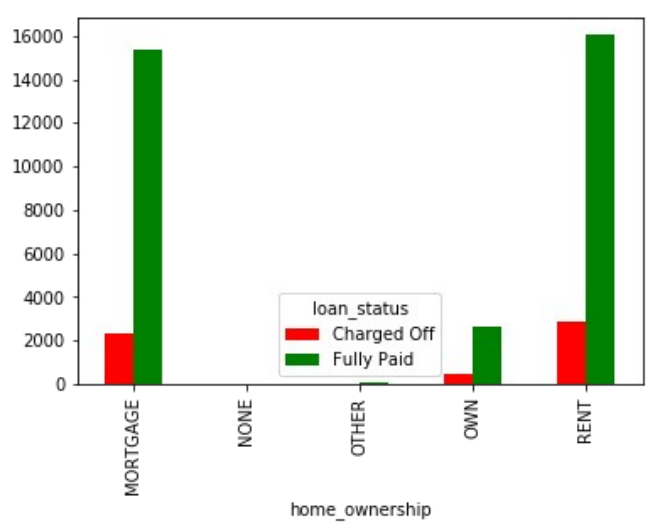


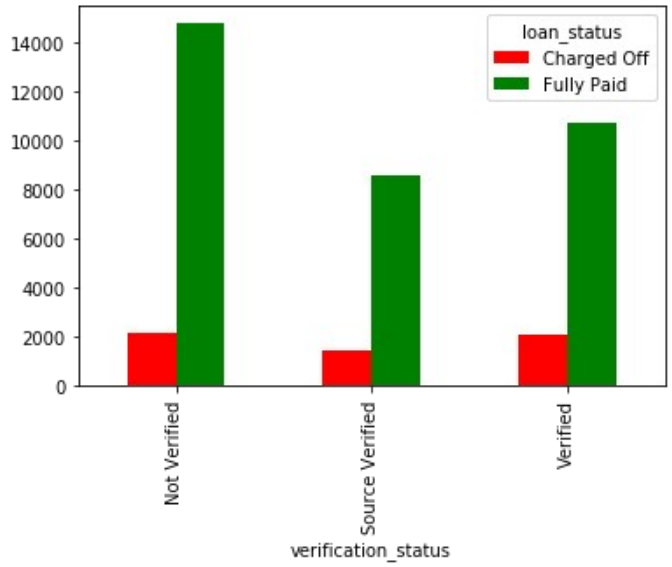
由上左图：
settlement_status（借款人结算计划的状态），只有违约注销了才会有这一项，属于严重缺失值。而且这是贷后指标，对贷前预测模型没有意义。
由上右图：
工作时长低于 1 年（包括 1 年）与 1 0 年以上的借款人最多，两者的违约比例相差不大，这张图打破了人们惯常以为的工作年限长就靠谱的想法。除此两类，在 2 — 9 年内，随着工作年限越长，贷款需求越少，可能是因为收入越来越稳定吧。





由上左图：
债务整合（举债还债）这类最多。另外 3 项是：住房改善、汽车、大宗采购，也就是基本生活需求。相对来说，借钱做生意的，违约率较高。
由上右图：
debt_settlement_flag 表示已注销的借款人是否与债务结算公司合作。属于贷后信息。



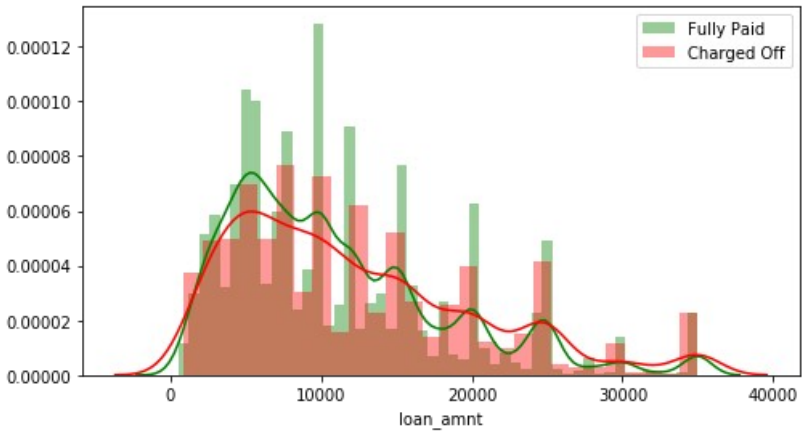


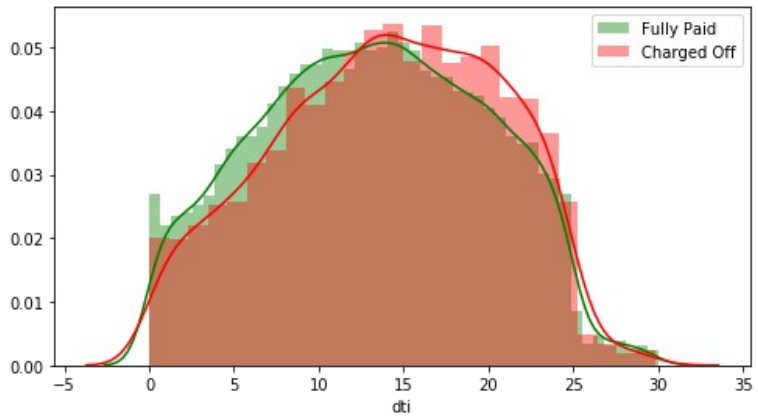
由上左图：
借款人住房按揭、租房最多，违约率不相上下；
由上右图：
表征收入或收入来源是否经过核实。大部分借款是经过核实的，经过简单计算可知，核实的借款违约率约为 0.15，未经核实的违约率约为 0.14.可以说明网上提交的申请数据还是比较诚实的。另外，也可以初步推断，这个变量的预测能力应该不是特别强。

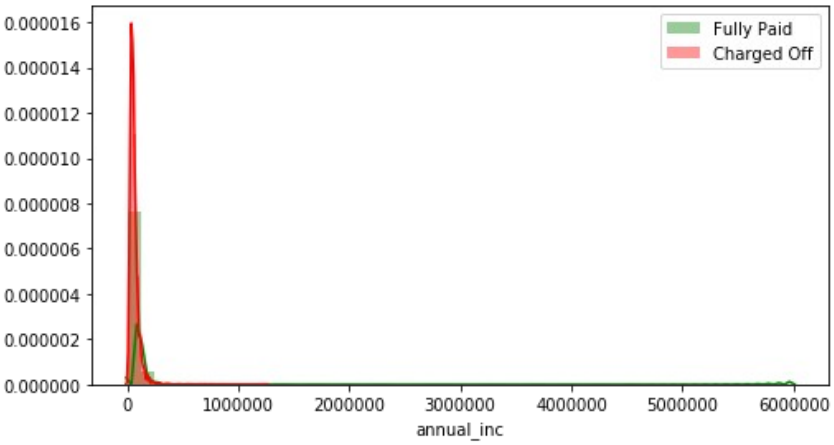
3．2．3 连续数值特征分布
探索连续变量的分布并可视化展示：

In [5]:
```
# 连续数值特征分布
# num_feas = ['loan_amnt', 'funded_amnt', 'installment', 'annual_inc',
#             'dti', 'delinq_2yrs', 'inq_last_6mths', 'pub_rec', 'total_acc']
num_feas = [i for i in df.columns[2:30] if df[i].dtype == 'float']
for i, fea in enumerate(num_feas):
    # set the whole figure size
    plt.figure(figsize=(8, 5*len(num_feas)))
    plt.subplot(len(num_feas), 1, i+1)
    df_temp_0 = df[fea][df.loan_status == 'Fully Paid']
    df_temp_1 = df[fea][df.loan_status == 'Charged Off']
    sns.distplot(df_temp_0.dropna(), color='g',)
    sns.distplot(df_temp_1.dropna(), color='r')
    plt.legend(['Fully Paid', 'Charged Off'])
```

数据集中有几十个连续特征，下面是示例图：







其余图示省略。
从这些图中可以初步得出如下信息：

- 贷款额度、分期付款金额成有些长尾的正态分布，说明贷款额度集中在中小额度，但是也有分散的大额度
- 年收入集中在 0 — 1 0 万刀以内，但是也有极高收入（最高达到 6 0 0 万）的借款人
- 负债率较符合正态分布，但是高负债率相较低负债率违约风险更大
- 分析过去 2 年内的违约次数分布，即便 1 次违约记录都没有，这次也可能会出现违约，
- 过去 6 个月内查询次数越多，违约的概率越大
- 迄今为止收到的付款总额或本金越少，违约率越高，这也显示了贷中监控的重要性，有问题及时预警。

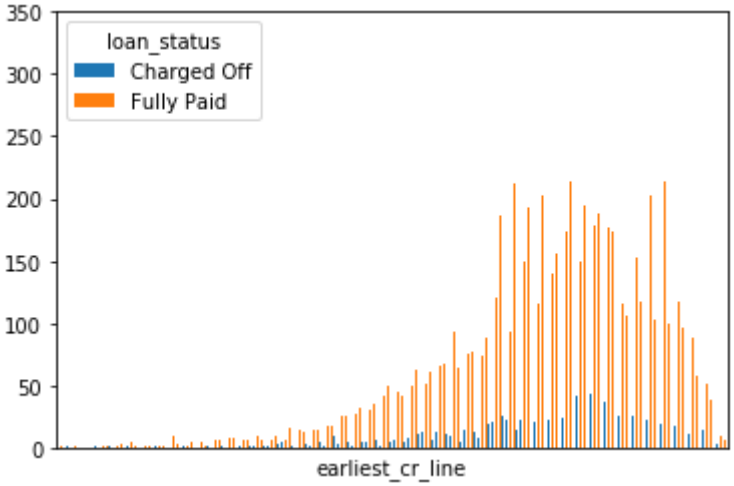
- 。。。

3．2．4 时序特征分布

全部时序特征：issue_d:（款发放月份），earliest_cr_line（首开信用卡时间），last_pymnt_d（最近一次收到还款的时间），last_credit_pull_d（LC撤回信贷最近的月份），其中第1，3，4项都是贷后数据。

In [78]:
```
# timestamp features distribution
df_time = df[['earliest_cr_line', 'loan_status']].dropna()
df_time.earliest_cr_line = [pd.datetime.strptime(str(j), '%b-%Y')
                            for j in df_time.earliest_cr_line.values]
pvt = pd.pivot_table(df_time, index='earliest_cr_line', columns='loan_status', aggfunc=len)
pvt.plot(kind='bar')
plt.xticks([])
```

首开信用卡时间分布图：



由上图可知：近期开信用卡的借款人居多，在时间维度上，借款人的分布类似正态分布。

3．2．5 文字特征分布

样本中的文本特征有：借款人所在地，借款描述，职位头衔

词云的优势是从大量文字样本中一眼看出文本的主要内容。

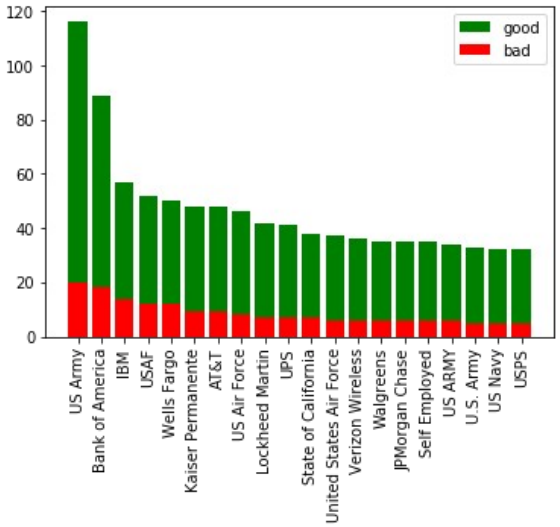
通过词云了解各个文本特征分布：

In [87]:
```
# text features distribution--wordcloud
text_feas = ['addr_state', 'desc', 'emp_title']
# collect words into a 3 strings
str_list = ['', '', '']
for i, j in enumerate(text_feas):
    for item in df[j].dropna().values:
        str_list[i] +=  str(item) + ' '

# make wordcloud: generate different size of word according to it's frequency
from wordcloud import WordCloud
for i in str_list:
    wordcloud = WordCloud().generate(i)
    plt.imshow(wordcloud)
    plt.axis('off')
```

按还款状态分类统计借款人所在州的分布：

In [45]:
```
# text features distribution
# emp_title
y_0 = df[df.loan_status == 'Fully Paid'].emp_title.value_counts()[:20].values
y_1 = df[df.loan_status == 'Charged Off'].emp_title.value_counts()[:20].values
ind = df[df.loan_status == 'Fully Paid'].emp_title.value_counts()[:20].index

plt.bar(ind, y_0, color='g')
plt.bar(ind, y_1,color='r')
# set xticks vertical
plt.xticks(ind,rotation='vertical')
plt.legend(['good', 'bad'])
```

输出：

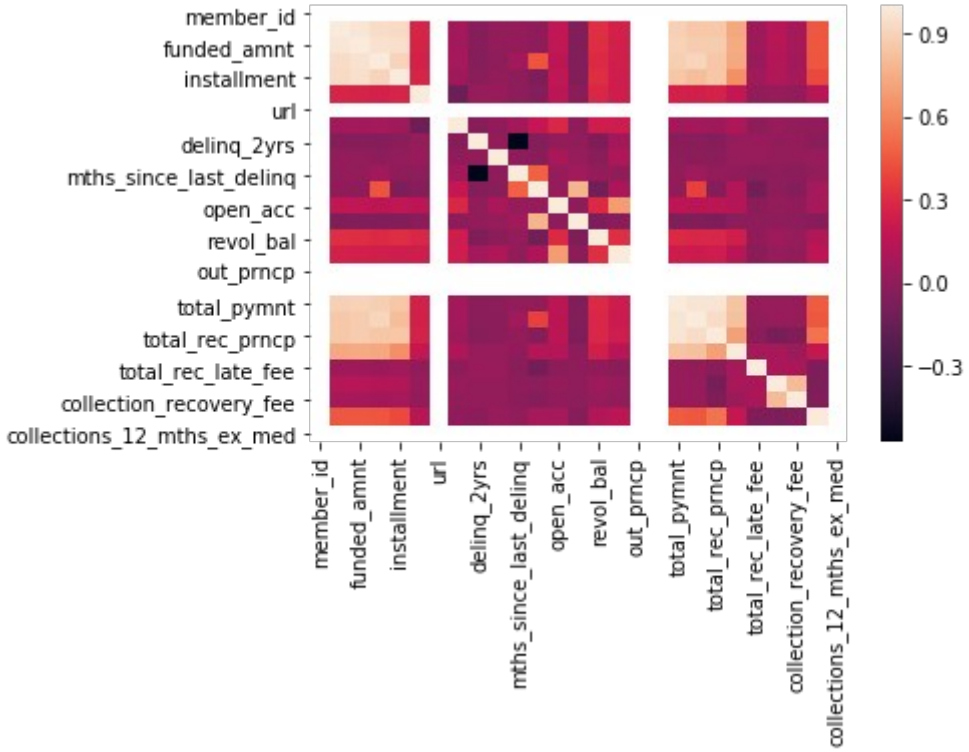


由上图可知：
借款人的区域来源，主要集中在加州、德州、纽约等等这些大州，后续数据处理可以考虑将这些主要区域提取出来。
借款描述：支付信用卡欠款
职务头衔：借款人的职务背景按照数量分布依次是公司、军队、医院。。。

3．2．6 两两特征的协方差
对样本特征的相关性有一个直观的理解：

In [94]:
```
# feature's corration
sns.heatmap(df.iloc[:, :50].corr())
```

输出基于协方差的热力图：



上图中浅颜色的部分是表示特征相关度高，除了对角线的区域，其它区域也分布着高相关性特征对，这说明样本集中某些特征之间存在强线性相关性，这个问题在选用某些机器学习模型（比如基于线性回归的模型族）时会显著影响模型性能，需要引起注意。

四：数据预处理

根据初步数据探索分析的结果，我们知道：

- •数据集存在严重的数据缺失问题
- •某些连续型特征用字符型特征表示，如百分比类的
- •部分特征存在明显的共线性关系

4．1 数据集划分

为了避免验证集/测试集数据被污染，最好在数据预处理之前进行训练集，验证集，测试集的划分。

训练集用于训练模型，验证集用于初步评估/优化模型,测试集进行最终的测试。

In [7]:
```
# split the data
x_col = df.columns.tolist()
x_col.remove('loan_status')
X = df[x_col]
y = df['loan_status']

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=0)
x_train, x_vali, y_train, y_vali = train_test_split(x_train, y_train, test_size=0.2, random_state=1)

df_train = pd.concat([x_train, y_train], axis=1)
df_vali = pd.concat([x_vali, y_vali], axis=1)
df_test = pd.concat([x_test, y_test], axis=1)
```

4．2特征缺失值识别与处理

4．2．1严重缺失值的处理
找到缺失超过60%的特征，并进一步了解缺失情况：

```
In [98]: # default values
         # select features which 60% of them is missing
         mis_feas = [i for i in df.columns if ((df[i].isnull().sum()) * 1.0 / df.shape[0]) > 0.6]
         # output missing rate to get a further understanding of missing info
         for i in mis_feas:
             mis_rate = (df[i].isnull().sum()) * 1.0 / df.shape[0]
             print(i, '\t', mis_rate)
```

截取输出结果的一部分：

列名	缺失值比例
mths_since_last_delinq	0.64663449454582
mths_since_last_record	0.92984969587292
debt_settlement_flag_date	0.996229829588297
settlement_status	0.996229829588297
settlement_date	0.996229829588297
settlement_amount	0.996229829588297
settlement_percentage	0.996229829588297
settlement_term	0.996229829588297
id	1
member_id	1
url	1
next_pymnt_d	1
mths_since_last_major_derog	1
…	…

从上表中可以看到，其中绝大部分特征都是完全缺失，对于这种，毫无疑问要删除。但是其中的mths_since_last_delinq（距离上次违约的月份数）、mths_since_last_record（距离上一次公共黑记录月份数）从经验判断应该对评估借款人的信用有帮助，所以，即便这2个属于严重缺失数据，也必须留下来。此外，严重缺失特征中还包括settlement（结算）信息，这属于贷后信息，不应包含在申请评分卡模型中，一并删除。
故：

```
In [9]: mis_feas.remove('mths_since_last_delinq')
        mis_feas.remove('mths_since_last_record')
        df.drop(mis_feas, axis=1, inplace=True)
        df_list = [df_train, df_vali, df_test]
        for i in df_list:
            i.drop(mis_feas, axis=1, inplace=True)
```

4．2．2缺失值填充

了解现有缺失值的情况，根据缺失比例，缺失值的具体信息含义，缺失特征性质，来决定处理策略：

In [133]:
```
# define missing value features to be filled
mis_feas_to_fil = [i for i in df.columns if df[i].isnull().sum() != 0]
for i in mis_feas_to_fil:
    mis_rate = (df[i].isnull().sum()) * 1.0 / df.shape[0]
    print(i, '\t', mis_rate)
```

删除严重缺失后的缺失情况		
列名	缺失值比例	应对对策
emp_title	0.062006736037802	众数填充
emp_length	0.027094958025436	众数填充
desc	0.325918664856985	将缺失值作为一类
title	0.000276479163525	众数填充
mths_since_last_delinq	0.64663449454582	将缺失值作为一类
mths_since_last_record	0.92984969587292	将缺失值作为一类
revol_util	0.001256723470568	众数填充
last_pymnt_d	0.001784547328206	删除（贷后数据）
last_credit_pull_d	5.02689388227015E-05	删除（贷后数据）
collections_12_mths_ex_med	0.001407530287036	删除（贷后数据）
chargeoff_within_12_mths	0.001407530287036	同一值
pub_rec_bankruptcies	0.017518725179712	众数填充
tax_liens	0.000980244307043	众数填充

上表中标红数据留后处理。
实施：

In [52]:
```
# drop after issued features
df.drop(['last_pymnt_d', 'last_credit_pull_d', 'collections_12_mths_ex_med'],
        axis=1,inplace=True)
# fill with mode number
to_fil_with_mode = ['emp_title', 'emp_length', 'title',
                    'revol_util','pub_rec_bankruptcies', 'tax_liens']

from scipy.stats import mode
for i in to_fil_with_mode:
    df[i][df[i].isnull()] = mode(df[i][df[i].notnull()])[0][0]
```

注：

- 由于接下来将用到逻辑回归模型，它对特征线性相关性较敏感，所以尽管用其它变量拟合缺失值对缺失值的填充会更符合实际情况，也没有采用这种方法。
- 为什么使用众数填充而不是用中位数或是均值填充？因为众数对于数值型变量和字符型变量都适用，而且也有统计意义。

4．3 同值性特征识别与处理

如果一个变量大部分的观测都是相同的特征，那么这个特征或者输入变量就是无法用来区分目标时间，一般来说，临界点在90%。但是最终的结果还是应该基于业务来判断。

In [26]:
```
# 同值性特征识别处理,将阈值设定为９０％
equi_fea = []
for i in df.columns:
    try:
        mode_value = mode(df[i])[0][0]
        mode_rate = mode(df[i])[1][0]*1.0 / df.shape[0]
        if mode_rate > 0.9:
            equi_fea.append([i, mode_value, mode_rate])
    except:
        pass
e = pd.DataFrame(equi_fea, columns=['col_name', 'mode_value', 'mode_rate'])
e.sort_values(by='mode_rate')
```

按众数占比从小到大给占比高于９０％的特征排序，并且依据特征内涵确定处理策略：

序号	col_name	mode_value	mode_rate	策略
7	collection_recovery_fee	0	0.903107	删除（贷后数据）
2	pub_rec	0	0.94669	大部分人都没有公共黑记录，合理
6	total_rec_late_fee	0	0.947771	删除（贷后数据）
13	pub_rec_bankruptcies	0	0.957648	大部分人都没有公共破产记录，合理
17	debt_settlement_flag	N	0.99623	删除（贷后数据）
11	chargeoff_within_12_mths	0	0.998592	绝大部分人近１年内都没有销账记录，合理，但是除了众数其它就是缺失值，无法应用，删除
15	hardship_flag	N	1	完全一样的值，没有意义，删除
14	tax_liens	0	1	
12	delinq_amnt	0	1	
0	pymnt_plan	n	1	
9	application_type	Individual	1	
16	disbursement_method	Cash	1	
5	out_prncp_inv	0	1	
4	out_prncp	0	1	
3	initial_list_status	f	1	
10	acc_now_delinq	0	1	
8	policy_code	1	1	

实施：

In [14]:
```
# 处理同一性数据
same_val_fea_to_drop = list(e.col_name.values)
for i in ['pub_rec', 'pub_rec_bankruptcies']:
    same_val_fea_to_drop.remove(i)
for i in df_list:
    i.drop(same_val_fea_to_drop, axis=1, inplace=True)
```

４．４特征格式变换
这一步将格式杂乱的特征进行规整.

In [15]:
```
# features to be regularized
for i in df_list:
    i.term = i.term.str.replace(' months', '').astype('float')
    i.int_rate = i.int_rate.str.replace('%', '').astype('float')
    i.earliest_cr_line = [pd.datetime.strptime(i, '%b-%Y') for i in i.earliest_cr_line]
    i.issue_d = [pd.datetime.strptime(i, '%b-%Y') for i in i.issue_d]
    i.revol_util = i.revol_util.str.replace('%', '').astype('float')
```

4．5 文本特征处理

经过上述步骤处理后，现有的文本类数据包括：emp_title(职务信息），desc(借款描述），title(标题）,addr_state(借款人地址），其中 desc/title 与 purpose 相关性较强，都是表明借款用途的信息。

emp_title 字面上看是职务头衔，但是实际内容是借款人所在机构，它类型多，且是文本型特征，但是根据数据探索阶段得到的结论，这个变量含有预测性的信息，用模型分箱相当耗费性能，所以根据经验考虑尝试机构类别将它分类。

addr_state 也包含预测性信息，可以尝试用卡方分箱或依据经验进行进行分类。

4．5．1 工作机构分类

工作机构分类，依据 A 政府机构类，B 银行类，F 医院类，E 学校类，C 自职业类，D 公司和其它类，G 退休类分类． 如果数据中的 emp_title 与某个上述 A-G 有交集，则将它划为该类，用字母字符表示； 缺省值为’H’。

In [33]:
```
# 工作机构分类，依据A政府机构类，B银行类，F医院类，E学校类，C自职业类，D公司和其它类，G退休类分类
# 如果df中的emp_title与某个上述A-G有交集，则将它划为该类，用字母字符表示
# 缺省值为'H'

A = ['board', 'general','american','u.s.' ,'army', 'force', 'us', 'states', 'corp', 'navy', 'united', 'department',
B = ['bank', 'morgan']
C = ['self']
D = 'OTHER'
E = ['college', 'school', 'university']
F = ['hospital', 'clinic', 'health', 'healthcare']
G = ['retired']
ls_letter = [0,1,2,4,5,6]
ls = [A,B,C,E,F,G]

def emp_classify(df1):
    for i in df1.emp_title.index:
        emp_list = []
        for j in range(len(ls)):
            emp_list.append((set(str(df1.emp_title[i]).lower().split()) & set(ls[j])))
        if emp_list.count(set()) != 6:
            sr_emp = pd.Series(emp_list)
            idx = sr_emp[sr_emp!=set()].index
            df1.emp_title[i] = ls_letter[idx[0]]
        else:
            df1.emp_title[i] = 3
    df1.emp_title[df1.emp_title.isnull()] = 7
```

注：由于下面要应用到随机森林模型，它只接受数值型和类别型变量，不接受‘object’或区间格式的变量，为了方便，将类别统一用数字表示。其它的格式变换也是如此。

4．5．2 借款人州地址分类

4．5．2．1 k-means 聚类分析

首先尝试使用 k-means 方式进行聚类，看看每一类中的州地址有没有什么共性：

In [228]:
```
# 尝试用k-means方法对所在州聚类
# k-means是基于距离值聚类，因此需要先归一化处理
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
df_addr_temp.iloc[:, :] = mm.fit_transform(df_addr_temp)

from sklearn.cluster import KMeans
km = KMeans(n_clusters=8,random_state=3)
km.fit(df_addr_temp.values)

df_addr_temp =  pd.concat([df_addr_temp, df_train.addr_state, pd.Series(km.labels_)], axis=1)

df_addr_temp.groupby([0,'addr_state'])['addr_state'].count()
```

从输出结果中并没有看出不同群体有什么共性。

4．5．2．2　等频分箱

用频数进行分类，使每一个分组内的样本数尽量相近

In [407]:
```
# 尝试用频数进行分类，使每一个分组内的样本数尽量相近,假设分8个样本，每个分组大约３３００个样本
S = [['CA'], ['NY'], ['FL', 'TX'], ['NJ', 'PA', 'IL'],['VA','GA','MA','OH'],['MD','AZ','WA','CT','CO','NC'],
 ['MI','MO','MN','NV','OR','WI','LA','SC','AL','OK']]

addr_list = df_train.addr_state.value_counts().index.tolist()

SS = []
for i in S:
    SS += i


S.insert(7, list(set(addr_list)-set(SS)))

addr_set = []
for i in S:
    addr_set.append(set(i))

addr_dict = {}
for i, j in zip(range(8), addr_set):
    addr_dict[i] = j

# 将addr_state进行分类转换
def trans_addr_func(df_to_trans):
    for i in df_to_trans.addr_state[10:]:
        for j in range(len(addr_dict)):
            if i in addr_dict[j]:
                df_to_trans.addr_state[df_to_trans.addr_state==i] = list(addr_dict.keys())[j]
```

4．6 时序特征处理

在4．4格式转换部分，将earliest_cr_line,issue_d的字符串格式数据，转换为标准datetime格式数据，方便后续的特征工程。

4．7 特征编码

逻辑回归和随机森林模型不接受字符型变量，因此需要将对此类变量进行编码。
常用的编码方式有类别标签法（不同的类别映射到不同的数值），哑变量编码法（对类别变量取哑变量）等等。考虑到评分卡模型的简洁性，在此选用类别标签法。

In [57]:
```
# encoding features:grade, emp_length, home_ownership, verification_status,purpose
for i in df_list:
    i.grade.replace({"A": 0,"B": 1,"C": 2, "D": 3, "E": 4,"F": 5,"G": 6}, inplace=True)
    i.emp_length.replace({"10+ years": 11,"9 years": 10,"8 years": 9,
                          "7 years": 8,"6 years": 7,"5 years": 6,"4 years":5,
                          "3 years": 4,"2 years": 3,"1 year": 2,"< 1 year": 1,
                          np.nan: 0}, inplace=True)
    i.home_ownership.replace({"MORTGAGE":0,"OTHER":1,"NONE":2,"OWN":3,"RENT":4}, inplace=True)
    i.verification_status.replace({"Not Verified":0,"Source Verified":1,"Verified":2}, inplace=True)
    i.purpose.replace({"credit_card":0,"home_improvement":1,"debt_consolidation":2,
                          "other":3,"major_purchase":4,"medical":5,"small_business":6,
                          "car":7,"vacation":8,"moving":9, "house":10,
                          "renewable_energy":11,"wedding":12, 'educational':13}, inplace=True)
    i.loan_status.replace({'Fully Paid':0, 'Charged Off':1}, inplace=True)
    i.term.replace({36.0:0, 60.0:1}, inplace=True)
```

4．8 归一化处理

逻辑回归模型基于线性回归，求参需要用到梯度下降法，为了加快迭代速度，不同特征的变化范围规模相差不宜过大，如果用数值直接带入逻辑回归模型，必须进行变量缩放。但是本文是用逻辑回归建立评分卡，会将数值变量进行分箱，所以这一步可以省略。

4．9 警惕数据泄露

数据泄露分为 2 种：不恰当特征导致的泄露、不恰当的交叉验证策略导致泄露

4．9．1 警惕不恰当特征

所有特征中，一旦在目标属性出现后，会随之更新或出现的属性，属于会泄露信息的属性，在这个数据集中，包括贷中、贷后特征。

删除此类特征（之前预处理步骤中已经删除了一些）：

In [18]:
```
# delete leaking features & not meaningful features
leak_feas = ['sub_grade','title','zip_code','recoveries','last_pymnt_amnt',
             'funded_amnt','funded_amnt_inv','total_pymnt','total_pymnt_inv',
              'total_rec_prncp','total_rec_int', 'desc']
for i in df_list:
        i.drop(leak_feas, axis=1, inplace=True)
```

4．9．2 错误的交叉验证策略：

尽量保证验证集数据的纯粹，不要让它参与到训练集的处理和模型构建当中，这意味着，要在预处理之前分割训练集和测试集。

五：特征工程

5．1 特征衍生

将时序变量衍生为月份值，将（贷款发放时间－首次使用信用卡时间）作为一个新的变量，表示信用历史(cre_hist)，单位是月份。

In [20]:
```
# deriving features, cre_hist = issue_d - earliest_cr_line
for i in df_list:
    i['cre_hist'] = [j.days for j in (i.issue_d - i.earliest_cr_line)/30]
    i.drop(['issue_d', 'earliest_cr_line'], axis=1, inplace=True)
```

5．2 筛选变量

5．2．1 依据共线性筛选变量

逻辑回归是基于线性回归模型，其前提假设是用于建模的特征之间不存在线性相关性，因此，它对共线性问题比较敏感。共线性的存在对模型稳定性有很大影响，并且也无法区分每个特征对目标变量的解释性。

依据 VIF（方差膨胀系数）筛选变量

每个特征的 VIF 计算是用其它特征对它进行回归拟合，如果这种拟合的解释性很强，说明它们之间存在多重共线性。

用 VIF 计算连续型变量的共线性：

In [23]:
```

# select number features
vif_feas = [i for i in df_train.columns if i not in
          ['term', 'mths_since_last_delinq', 'mths_since_last_record','addr_state','emp_title','loan_status']]

# calculate vif
from statsmodels.stats.outliers_influence import  variance_inflation_factor as vif
vif_ls = []
for i in range(len(vif_feas)):
    vif_ls.append([vif_feas[i], vif(df_train[vif_feas].values, i)])
vif_df = pd.DataFrame(vif_ls, columns=['col_name', 'vif'])
vif_df.sort_values(by='vif', ascending=False)
```

得到从大到小的 VIF 值排序：

	col_name	vif
1	int_rate	44.271965
0	loan_amnt	28.540607
2	installment	26.808813
12	open_acc	11.329237
16	total_acc	11.158463
3	grade	10.36559
15	revol_util	6.948412
18	cre_hist	6.628156
9	dti	6.463901
4	emp_length	4.30858
13	pub_rec	3.607757
17	pub_rec_bankruptcies	3.546608
6	annual_inc	3.037154
5	home_ownership	2.641862
7	verification_status	2.599168
14	revol_bal	2.587116
8	purpose	2.201331
11	inq_last_6mths	1.775502
10	delinq_2yrs	1.187509

用协方差计算线性相关性：

```
In [31]: # calculate correlation
         cor = df_train[vif_feas].corr()
         # get lower triangular matrix of cor
         cor.iloc[:, :] = np.tril(cor.values, k=-1)
         # stack columns of cor
         cor = cor.stack()
         cor[np.abs(cor)>0.7]

Out[31]: installment          loan_amnt    0.931526
         grade                int_rate     0.948791
         pub_rec_bankruptcies pub_rec      0.839870
         dtype: float64
```

根据经验，vif>10, cor>0.7, 变量之间的有显著性的相关性。
从上述关于协方差和 VIF 的分析结果中我们可以看到：
installment&loan_amnt，int_rate&grade，total_acc&open_acc，pub_rec_bankruptcies&pub_rec 之间存在线性关系，因为 pub_rec_bankruptcies&pub_rec 中大多数都是 0，所以其有相关可以解释。剩下的几对，我们可以尝试先删除每对中的一个, 删除 installment & grade, 再去检测相关系数。
此外，有几个处于边界地带的特征，暂时留下，特征工程中删除特征时要谨慎，因为删除特征，意味着弃用一些信息。

In [34]:
```
for i in ['installment', 'grade']:
    vif_feas.remove(i)

for i in df_list:
    i.drop(['installment', 'grade'], axis=1, inplace=True)
```

5．2．2 依据特征重要性筛选变量

筛选变量常用的方法有，基于正则化损失函数的线性模型，基于机器学习模型输出的特征重要性，基于 IV 值。

在此，为了方便进行评分卡建模，采用 IV 值.

定义 woe 和 iv:

In [404]:
```
#计算WOE和IV值
def CalcWOE(df,col, target):
    total=df.groupby([col])[target].count()
    total=pd.DataFrame({'total':total})
    bad=df.groupby([col])[target].sum()
    bad=pd.DataFrame({'bad':bad})
    regroup=total.merge(bad, left_index=True, right_index=True, how='left')
    regroup.reset_index(level=0, inplace=True)
    N=sum(regroup['total'])
    B=sum(regroup['bad'])
    regroup['good']=regroup['total']-regroup['bad']
    G=N-B
    regroup['bad_pcnt']=regroup['bad'].map(lambda x: x*1.0/B)
    regroup['good_pcnt']=regroup['good'].map(lambda x: x*1.0/G)
    # woe在这里显示不同特征不同属性对好样本的预测能力，这符合评分卡计分标准，分数越高，越可信
    regroup['WOE']=regroup.apply(lambda x: np.log(x.good_pcnt*1.0/x.bad_pcnt),axis=1)
    WOE_dict=regroup[[col,'WOE']].set_index(col).to_dict()
    IV=regroup.apply(lambda x:(x.good_pcnt-x.bad_pcnt)*np.log(x.good_pcnt*1.0/x.bad_pcnt),axis=1)
    IV_SUM=sum(IV)
    return {'WOE':WOE_dict,'IV_sum':IV_SUM,'IV':IV}
```

计算每个变量的 IV 值并按从大到小的顺序排序：

In [73]:
```
# 计算ＩＶ
df_train_boxed = df_train.copy()
```

In [74]:
```
# 输出IV
iv_list = []
for i in df_train_boxed.columns:
    iv_dict = CalcWOE(df_train_boxed,i, 'loan_status')
    iv_list.append(iv_dict['IV_sum'])

# 形成IV表
iv_df = pd.DataFrame({'iv_name':df_train_boxed.columns.values, 'iv':iv_list})

iv_df.sort_values('iv',ascending=False)
```

	iv_name	iv
11	delinq_2yrs	inf
21	loan_status	inf
5	home_ownership	inf
16	pub_rec	inf
2	int_rate	0.332112
1	term	0.154845
8	purpose	0.089021
18	revol_util	0.083983
12	inq_last_6mths	0.049551
6	annual_inc	0.042163
0	loan_amnt	0.029512
14	mths_since_last_record	0.023197
10	dti	0.020114
20	pub_rec_bankruptcies	0.0159
3	emp_title	0.015067
7	verification_status	0.015029
15	open_acc	0.014981
19	total_acc	0.013167
4	emp_length	0.010818
13	mths_since_last_delinq	0.009772
22	cre_hist	0.008439
9	addr_state	0.006799
17	revol_bal	0.005446

如果输出 IV 值是无穷，说明该特征中某些属性中缺失某类样本，这需要重新分箱，将这类样本添加到相邻类（对于连续型数值样本）或样本数量较少的那一类（对于分类样本）中去：

deling_2yrs:把 7.0, 8.0, 9.0, 11.0 划归到 7.0 那一类, 全算作 6.0

home_ownership:把 2 添加到 1 这一类

pub_rec：把 3.0，4.0，添加到 2.0 这一类

```
# 实施上述区间合并
for df_i in df_list:
    df_i.delinq_2yrs[df_i.delinq_2yrs.isin([7.0,8.0,9.0,11.0])] = 6.0
    df_i.home_ownership[df_i.home_ownership==2] = 1
    df_i.pub_rec[df_i.pub_rec.isin([3.0,4.0])] = 2.0
```

保留 IV 大于 0.15 的特征：

In [77]:
```
# 保留IV值大于0.015的变量
valid_feas = iv_df[iv_df.iv > 0.015].iv_name.tolist()
valid_feas.remove('loan_status')
```

In [78]:
```
valid_feas
```

Out[78]:
```
['loan_amnt',
 'term',
 'int_rate',
 'emp_title',
 'annual_inc',
 'verification_status',
 'purpose',
 'dti',
 'inq_last_6mths',
 'mths_since_last_record',
 'pub_rec',
 'revol_util',
 'pub_rec_bankruptcies']
```

5．3特征分箱

特征分箱就是把连续特征转化为离散特征，或者减小离散特征的离散性。

特征分箱有如下好处：

- 特征分箱后，特征被简化，也简化了模型，比如在逻辑回归评分卡模型中，评分卡被简化，基于决策树的模型中，决策树枝杈减少，降低了过拟合的风险，有效增加了模型的稳定性。
- 特征分箱可以将缺失值划为一类，比如此样本中无法被编码的公共记录缺失类。
- 特征离散化后对异常数据也有更强的容错性。比如假设年龄数据中出现1000岁，模型可以自动将其划分为>80岁一类，否则它会对对异常值敏感的模型如（逻辑回归）造成很大影响。
- 特征离散化之后，方便进一步进行非线性的特征衍生。

分箱的方法：

常用的分箱方法包括卡方分箱、等频或等距分箱、聚类、依据经验分箱等。

对连续型数值变量，在此采用有监督的最优分箱法——卡方分箱。

定义卡方分箱函数：

In [39]:
```
# 卡方分箱
def Chi2(df, total_col, bad_col,overallRate):
    df2=df.copy()
    df2['expected']=df[total_col].apply(lambda x: x*overallRate)
    combined=zip(df2['expected'], df2[bad_col])
    chi=[(i[0]-i[1])**2/i[0] for i in combined]
    chi2=sum(chi)
    return chi2


def ChiMerge_MaxInterval_Original(df, col, target,max_interval=5):
     colLevels=set(df[col])
    colLevels=sorted(list(colLevels))
    N_distinct=len(colLevels)
    if N_distinct <= max_interval:
        print("the row is cann't be less than interval numbers")
        return colLevels[:-1]
    else:
        total=df.groupby([col])[target].count()
        total=pd.DataFrame({'total':total})
        bad=df.groupby([col])[target].sum()
        bad=pd.DataFrame({'bad':bad})
        regroup=total.merge(bad, left_index=True, right_index=True, how='left')
        regroup.reset_index(level=0, inplace=True)
        N=sum(regroup['total'])
        B=sum(regroup['bad'])
        overallRate=B*1.0/N
        groupIntervals=[[i] for i in colLevels]
        groupNum=len(groupIntervals)
        N=sum(regroup['total'])
        B=sum(regroup['bad'])
        overallRate=B*1.0/N
        groupIntervals=[[i] for i in colLevels]
        groupNum=len(groupIntervals)
        while(len(groupIntervals)>max_interval):
            chisqList=[]
            for interval in groupIntervals:
                df2=regroup.loc[regroup[col].isin(interval)]
                chisq=Chi2(df2,'total','bad',overallRate)
                chisqList.append(chisq)
            min_position=chisqList.index(min(chisqList))
            if min_position==0:
                combinedPosition=1
            elif min_position==groupNum-1:
                combinedPosition=min_position-1
            else:
                if chisqList[min_position-1]<=chisqList[min_position + 1]:
                    combinedPosition=min_position-1
                else:
                    combinedPosition=min_position+1
            #合并箱体
            groupIntervals[min_position]=groupIntervals[min_position]+groupIntervals[combinedPosition]
            groupIntervals.remove(groupIntervals[combinedPosition])
            groupNum=len(groupIntervals)
        groupIntervals=[sorted(i) for i in groupIntervals]
        print(groupIntervals)
        cutOffPoints=[i[-1] for i in groupIntervals[:-1]]
        return cutOffPoints
```

5．3．1 对无缺省值的连续型数值变量分箱
对连续型数值变量实施分箱，得到切割点：

	loan_amnt	int_rate	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	cre_hist
0	4450	6.03	26004	2.53	2	1098.1	10.17	3	55
1	6000	7.75	37225	4.18	3	2838.2	21.4	4	104
2	7725	9.99	40560	6.45	5	4632.6	29.7	14	186
3	9750	13.99	48156	9.65	6	6577	42	26	220
4	12200	16.01	51669	12.47	15	8841	55.3	36	314
5	14275	17.49	62851	15.62	20	11465	68	43	376
6	17800	18.99	74143.68	19.31	26	14833	80.5	50	425
7	22875	20.3	89092	21.03	30	19816.2	92.7	60	473
8	29900	21.67	115275	23.87	32	29139.5	96.5	69	525

注：上表中的切割点不包括最大点，最小点。

In [32]:
```
# by the sequence of num_feas_to_box:
cut_points_list = [
   [4450.0, 6000.0, 7725.0, 9750.0, 12200.0, 14275.0, 17800.0, 22875.0, 29900.0],
    [6.03, 7.75, 9.99, 13.99, 16.01, 17.49, 18.99, 20.3, 21.67],
    [26004.0, 37225.0, 40560.0, 48156.0, 51669.0, 62851.0, 74143.68, 89092.0, 115275.0],
    [2.53, 4.18, 6.45, 9.65, 12.47, 15.62, 19.31, 21.03, 23.87],
    [2.0, 3.0, 5.0, 6.0, 15.0, 20.0, 26.0, 30.0, 32.0],
    [1098.1, 2838.2, 4632.6,6577.0,8841.0,11465.0,14833.0,19816.2,29139.5],
    [10.17, 21.4, 29.7, 42.0, 55.3, 68.0, 80.5, 92.7, 96.5],
    [3.0, 4.0, 14.0, 26.0, 36.0, 43.0, 50.0, 60.0, 69.0],
    [55, 104, 186, 220, 314, 376, 425, 473, 525]]

cut_points_dict = {}
for i in range(len(num_feas_to_box)):
    cut_points_dict[num_feas_to_box[i]] = cut_points_list[i]
```

制作切割点字典：
实施分箱：

In [37]:
```
# 用卡方分箱得到的分割点进行特征分箱，要注意，所有数据集中同样的特征区间都必须是相同的
def box_col_to_df(df_to_box, col, cut_points):
    bins = [-10.0] + cut_points + [100000000.0]
    # 如果有重复切割点,duplicates='drop'去重
    df_to_box[col] = pd.cut(df_to_box[col], bins=bins,include_lowest=True,duplicates='drop',
                            labels=range(len(bins)-1))
```

5．3．2 对含有缺省值的连续型数值变量分箱
对两个比较特殊的变量mths_since_last_delinq’和’mths_since_last_record’，它们也是属于连续型特征变量，但是却存在缺失值，分箱策略是将缺失值作为一类，其他类进行卡方分箱。
得到2者的切割点：
‘mths_since_last_delinq’：
[19.0, 33.0, 38.0, 63.0]

’mths_since_last_record’:
[46.0, 68.0, 79.0, 82.0]

实施分箱：

In [36]:
```
# 对mths_since_last_record & mths_since_last_delinq分箱编写函数
def box_mth_col(df_to_box, mth_col, bins):
    bins = [0.0]+[1.0] + bins + [150.0]
    df_to_box[mth_col][df_to_box[mth_col].notnull()] = pd.cut(
        df_to_box[mth_col][df_to_box[mth_col].notnull()], bins=bins, include_lowest=True,
        labels=range(len(bins)-1))
    df_to_box[mth_col][df_to_box[mth_col].isnull()] = -1
```

将所有变量分箱后的效果如下：

Out[65]:

	loan_amnt	term	int_rate	emp_title	emp_length	home_ownership	annual_inc	verification_status	purpose	addr_state	...	mths_
26539	4	1.0	8	3	6	4	5	2	2	3	...	
19139	7	1.0	3	0	7	0	3	2	2	0	...	
18702	6	0.0	1	3	11	0	5	0	2	3	...	
31778	4	0.0	4	1	5	0	4	0	6	2	...	
25665	6	0.0	2	1	11	4	6	0	2	1	...	
19986	6	1.0	5	4	11	0	5	2	2	3	...	
19666	3	0.0	3	1	11	4	0	2	2	0	...	

六：建模

经过之前的数据处理和特征工程，得到了分箱后的，规整的数据，并且找到了对评分卡来说，高预测性能的特征。

这部分我会建立 1 个传统的评分卡和 1 个较复杂解释性较差但预测性较好的随机森林模型。

6．1 建立评分卡

在 5．2．2 节，得到了每个特征中不同属性的 woe 值，它的含义是该分类对“好结果”的贡献度。

建立评分卡的步骤如下：

- 用 woe 值替换相应位置的属性值
- 建立逻辑回归模型
- 根据回归结果计分

实施：

In [114]:
```
# 用woe替换相应位置的属性值:
df_card_train = df_train[valid_feas+ ['loan_status']]
df_card_test = df_test[valid_feas+ ['loan_status']]

for i in range(len(valid_feas)):
    df_card_train[valid_feas[i]].replace(iv_dict[valid_feas[i]]['WOE']['WOE'],inplace=True)
    df_card_test[valid_feas[i]].replace(iv_dict[valid_feas[i]]['WOE']['WOE'],inplace=True)

# 构建逻辑回归模型:
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(df_card_train.drop('loan_status', axis=1), df_card_train['loan_status'])
```

评估模型的预测性能：

In [130]:
```
from sklearn.metrics import roc_auc_score, roc_curve
auc = roc_auc_score(LR.predict(df_card_test.drop('loan_status', axis=1)),df_card_test['loan_status'])
fpr, tpr, thre = roc_curve(LR.predict(df_card_test.drop('loan_status', axis=1)),df_card_test['loan_status'])
ks = max(tpr-fpr)
```

In [133]:
```
print('auc:{}    ks:{}'.format(auc,ks))
```

```
auc:0.6725580476685414    ks:0.3451160953370828
```

auc 约为 0.672,ks 值约为 0.35,在评分卡建模中，ks 值大于 0.3，说明这是一个基本可用的模型。

输出评分卡：

In [159]:
```
# 输出评分卡
import statsmodels.api as sm

#df_card_train = sm.add_constant(df_card_train)
logit=sm.Logit(df_card_train['loan_status'].values, df_card_train.drop('loan_status',axis=1).values).fit()

B=20/np.log(2)
A=600+20*np.log(1/60)/np.log(2)
basescore=round(A-B*logit.params[0],0)
scorecard=[]
#features.remove('loan_status')
for j, i in enumerate(valid_feas+['loan_amnt']):
    woe = iv_dict[i]['WOE']['WOE']
    interval=[]
    scores=[]
    for key,value in woe.items():
        score=round(-(value*logit.params[j]*B))
        scores.append(score)
        interval.append(key)
    data=pd.DataFrame({'interval':interval,'scores':scores})
    scorecard.append(data)
```

```
Optimization terminated successfully.
         Current function value: 0.377603
         Iterations 7
```

所得评分卡见附录——评分表.pdf。

6．2 建立随机森林模型

评分卡的优势在于简单明了，但是它无法包含更多的信息。随机森林的是以决策树为弱学习模型通过 bagging 方法构造出的强学习模型，它能容纳更多的信息，同时通过多模型投票，又能很好的避免过拟合的影响，它正好弥补了这一缺陷。这一模型可以作为评分卡的参考。

In [207]:
```
# 用随机森林方法进行建模
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(df_train.drop('loan_status', axis=1).values, df_train['loan_status'].values)
```

Out[207]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

用网格搜索的方式优化逻辑回归森林模型：

回归森林模型中，n_estimators 表示底层决策树个数，一般来说，树的个数越多，模型的稳定性越强，但是它的增大要受限于计算性能。

In [210]:
```
# 网格搜索优化模型，对随机森林中的决策树个数(n_estimators)进行网格遍历
from sklearn.model_selection import GridSearchCV
RF = RandomForestClassifier()
parameters = {'n_estimators':[5,10, 20,35,50,100]}
gs = GridSearchCV(estimator=RF, param_grid=parameters, scoring='roc_auc',cv=5,n_jobs=-1)
grid_result = gs.fit(df_train.drop('loan_status', axis=1).values, df_train.loan_status.values)
```

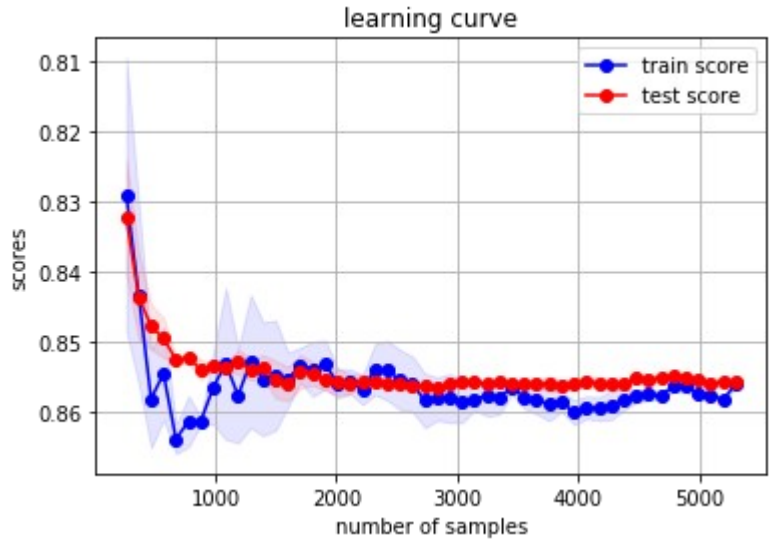
In [211]:
```
grid_result.best_score_
```

Out[211]:
```
0.6731332811015402
```

参数遍历之后，最优 n_estimator 取值为 100，此模型比评分卡模型的性能稍好一点。

用学习曲线判断模型的拟合状态：

In [228]:
```
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=1,
                        train_sizes=np.linspace(.05, 1., 20), verbose=0, plot=True):

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, verbose=verbose)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    if plot:
        plt.figure()
        plt.title(title)
        if ylim is not None:
            plt.ylim(*ylim)

        plt.gca().invert_yaxis()

    midpoint = ((train_scores_mean[-1] + train_scores_std[-1]) + (test_scores_mean[-1] - test_scores_std[-1])) / 2
    diff = (train_scores_mean[-1] + train_scores_std[-1]) - (test_scores_mean[-1] - test_scores_std[-1])
    return midpoint, diff

plot_learning_curve(RF, 'learning curve',df_test.drop('loan_status',axis=1).values, df_test.loan_status.values,train
```



模型在训练和测试样本上，随着训练样本的增加，贴合很好，并无过拟合现象。

七　总结

在评估小微企业信贷风险时，个人信用评分只是其中一个环节，还应该综合考虑借贷人的经营情况,出借方的风险偏好等其他因素，构建风控策略和风控系统。

２０１９年４月２３日

张爽