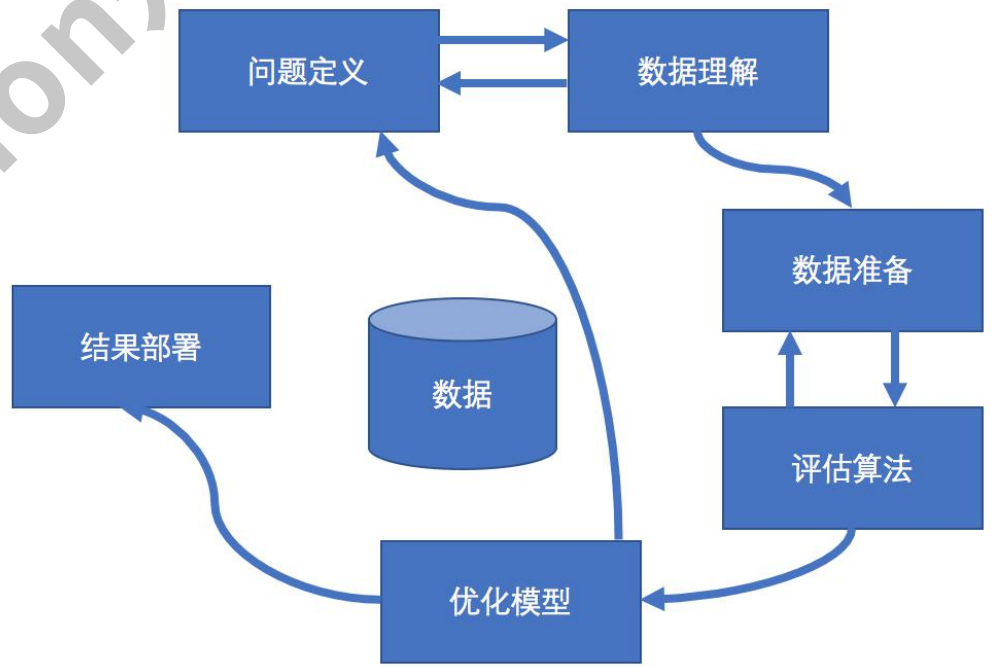
Python开发者日
让开发者紧跟技术潮流

Python在金融领域的应用

- 讲师：Kevin
- 《机器学习—Python 实践》与《深度学习—基于Keras的Python实践》作者，世界500强企业的数据分析团队 Leader，主要负责银行客户的复杂系统开发，在Python的Web开发、数据分析，机器学习与深度学习方面有多年的实践经验。同时是电子工业出版社大数据技术图书的专家委员。
- 作品：

目录

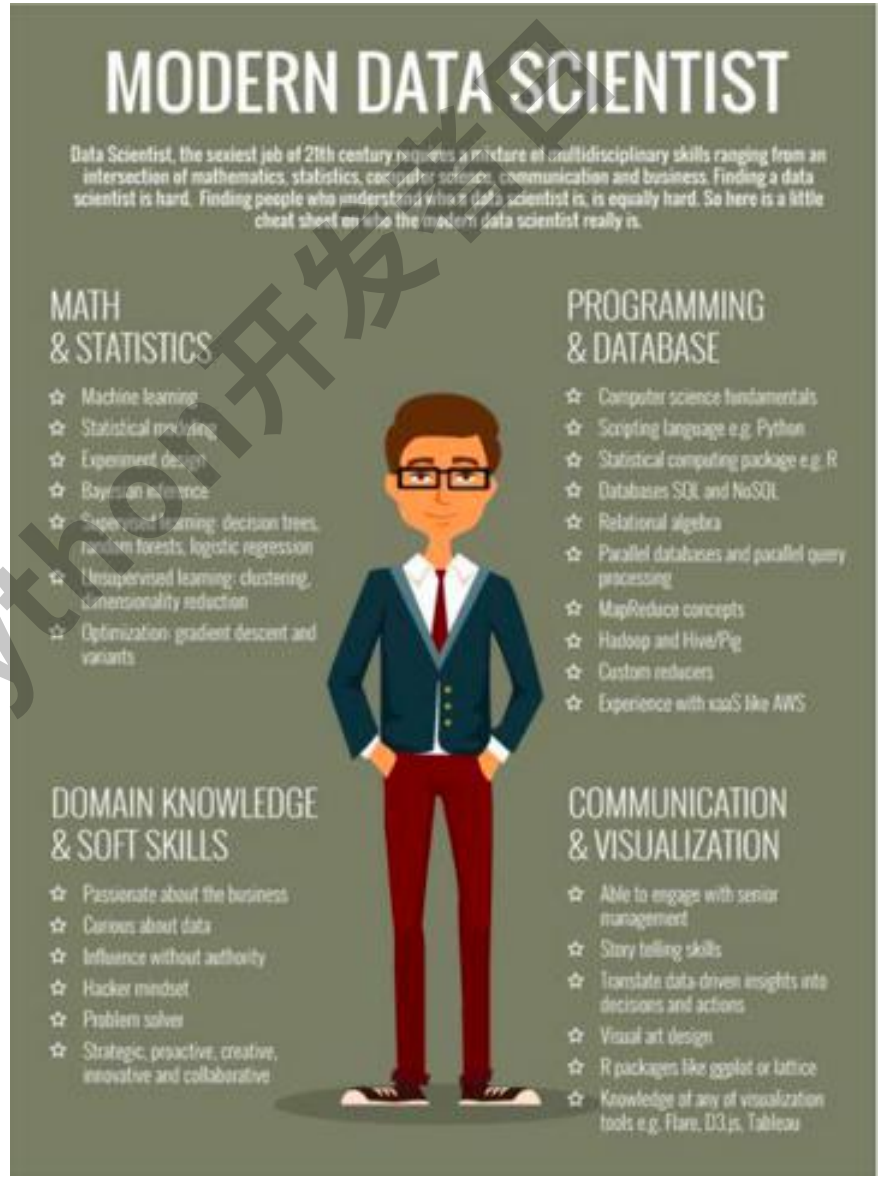
- 2020年技术发展趋势
- 数据科学家必备技能
- 机器学习 - 信用评分卡
  - ✓信用评分卡简介
  - ✓信用评分卡应用场景
  - ✓项目流程
  - ✓常用算法及评估方法
  - ✓Python实现



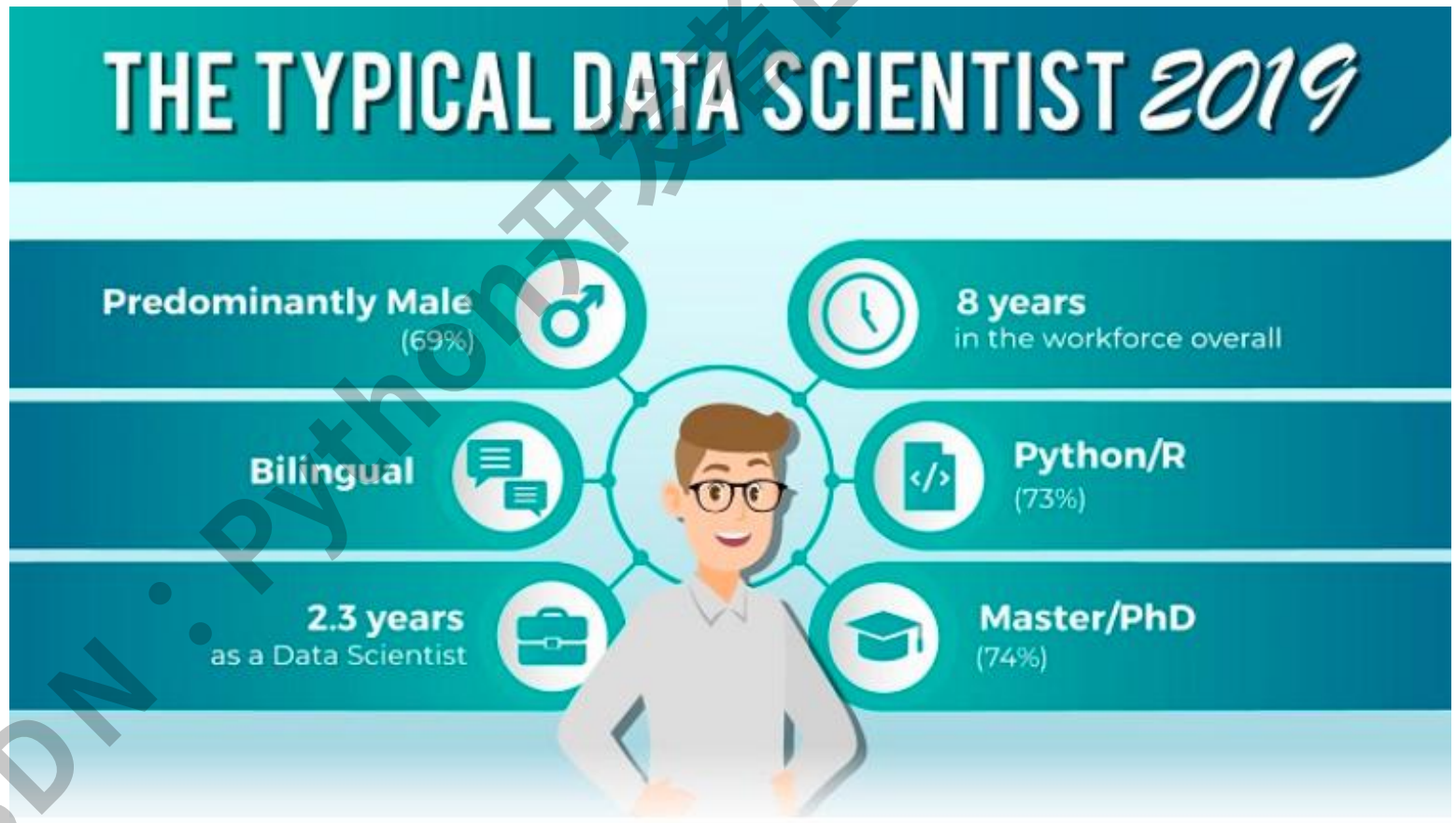
5 predictions for technology in 2020

- **Hyper-effective Security:** By 2020, 70 percent of mobile consumers will access their devices biometrically. IDC said this will reduce password memorization by 50 percent. Consider how much time this will free up for activities infinitely more valuable than changing and remembering passwords and worrying about data breaches.
- **3-D Printing:** By 2020, additive manufacturing technologies will enable produce-on-demand scenarios for more than ten percent of all consumer product purchases. And, over one-third of these products will be completely customized to match the buyer's needs and price sensitivity. Move over Savile Row, there's a new consumer in charge.
- **Self-driving Cars:** By 2020, 30 percent of new cars will have a self-driving mode. Pioneered by Google, self-driving cars are showing up on everyone's wish list including former partner Uber which just announced a new strategic partnership with Carnegie Mellon University.
- **Internet of Things:** By 2020 consumers will interact with over 150 sensor-enabled devices every day (think: connected cars, building, homes and wearables), and 25 percent will be disposable. Here is my take on wearables summarizing perspectives from experts on a recent SAP's Coffee Break with Game-Changers radio broadcast.
- **Cognitive systems (smart machines) that observe, learn and offer suggestions to people:** By 2020, 60 percent of device interactions will be passive, allowing people to use information from intelligent systems and machine learning.

数据科学家的四种能力



数据科学家的典型特征



Python数据科学的环境

- 推荐Anaconda Python

  https://www.anaconda.com/distribution/

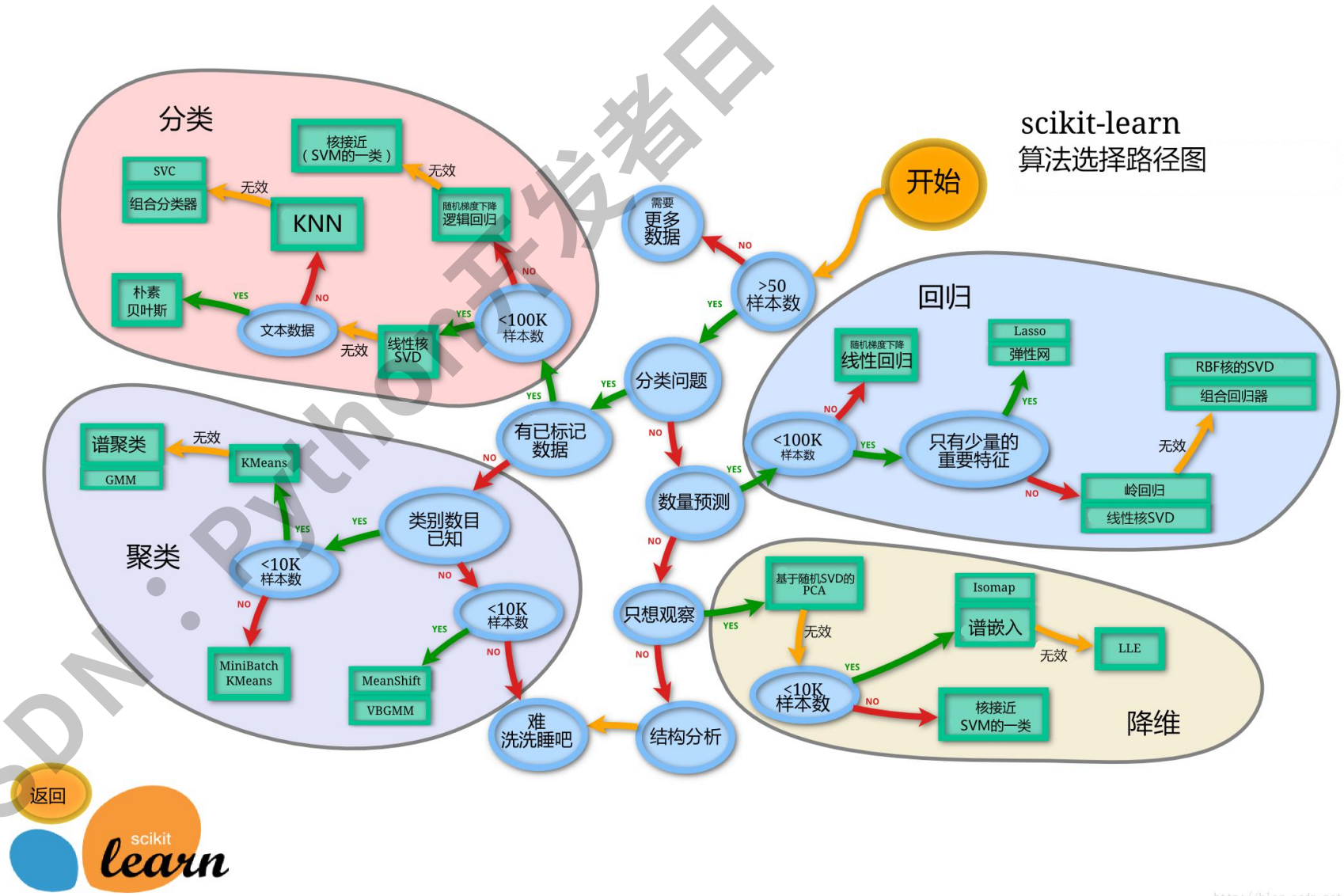
- Python官网安装包

  https://www.python.org/downloads/

- 常用的工具，及IDE
  - PyCharm
  - VSCode
  - JupyterNotebook

Python数据科学常用库

- Scrapy
- Pandas
- Numpy
- Scipy
- Matplotlib\Seaborn\Boken
- Scikit-Learn
- NLTK
- Pyspark
- Keras
- Tensorflow
- Pytorch



信用评分卡

信用评分卡简介

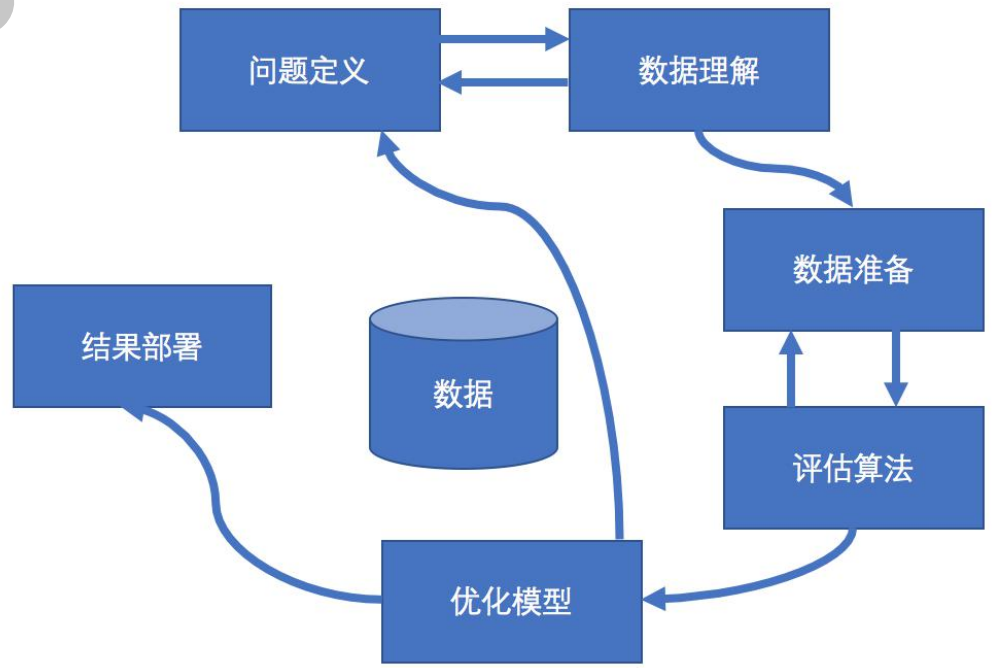
- 在银行、消费金融公司等各种贷款业务机构，普遍使用信用评分，对客户实行打分制，以期对客户有一个优质与否的评判。
- 信用评分卡还分A,B,C卡通常分为三类：
  - A卡（Application score card）申请评分卡
  - B卡（Behavior score card）行为评分卡
  - C卡（Collection score card）催收评分卡
- 评分机制的区别在于：
  - 使用的时间不同。分别侧重贷前、贷中、贷后；
  - .数据要求不同。A卡一般可做贷款0-1年的信用分析，B卡则是在申请人有了一定行为后，有了较大数据进行的分析，一般为3-5年，C卡则对数据要求更大，需加入催收后客户反应等属性数据。
  - 每种评分卡的模型会不一样。在A卡中常用的有逻辑回归，AHP等，而在后面两种卡中，常使用多因素逻辑回归，精度等方面更好。

信用评分卡的目的及意义

- 由于零售信贷业务具有笔数多、单笔金额小、数据丰富的特征，决定了需要对其进行智能化、概率化的管理模式。信用评分模型运用现代的数理统计模型技术，通过对借款人信用历史记录和业务活动记录的深度数据挖掘、分析和提炼，发现蕴藏在纷繁复杂数据中、反映消费者风险特征和预期信贷表现的知识和规律，并通过评分的方式总结出来，作为管理决策的科学依据。
- 减少对信用审查人员的依赖，良好风险管理。尤其对中小企业融资，可以提供高效率的审批。
- 信用评分卡具有客观性，它是根据从大量数据中提炼出来的预测信息和行为模式制定的，反映了借款人信用表现的普遍性规律，在实施过程中不会因审批人员的主观感受、个人偏见、个人好恶和情绪等改变，减少了审批员过去单凭人工经验进行审批的随意性和不合理性。
- 信用评分卡具有一致性，在实施过程中前后一致，无论是哪个审批员，只要用同一个评分卡，其评估和决策的标准都是一样的。
- 信用评分卡具有准确性，它是依据大数原理、运用统计技术科学地发展出来的，预测了客户各方面表现的概率，使银行能比较准确地衡量风险、收益等各方面的交换关系，找出适合自己的风险和收益的最佳平衡点。
- 运用信用评分卡可以极大地提高审批效率。由于信用评分卡是在申请处理系统中自动实施，只要输入相关信息，就可以在几秒中内自动评估新客户的信用风险程度，给出推荐意见，帮助审批部门更好地管理申请表的批核工作，对于业务批量巨大、单笔业务金额较小的产品特别适合。

信用评分卡模型使用场景

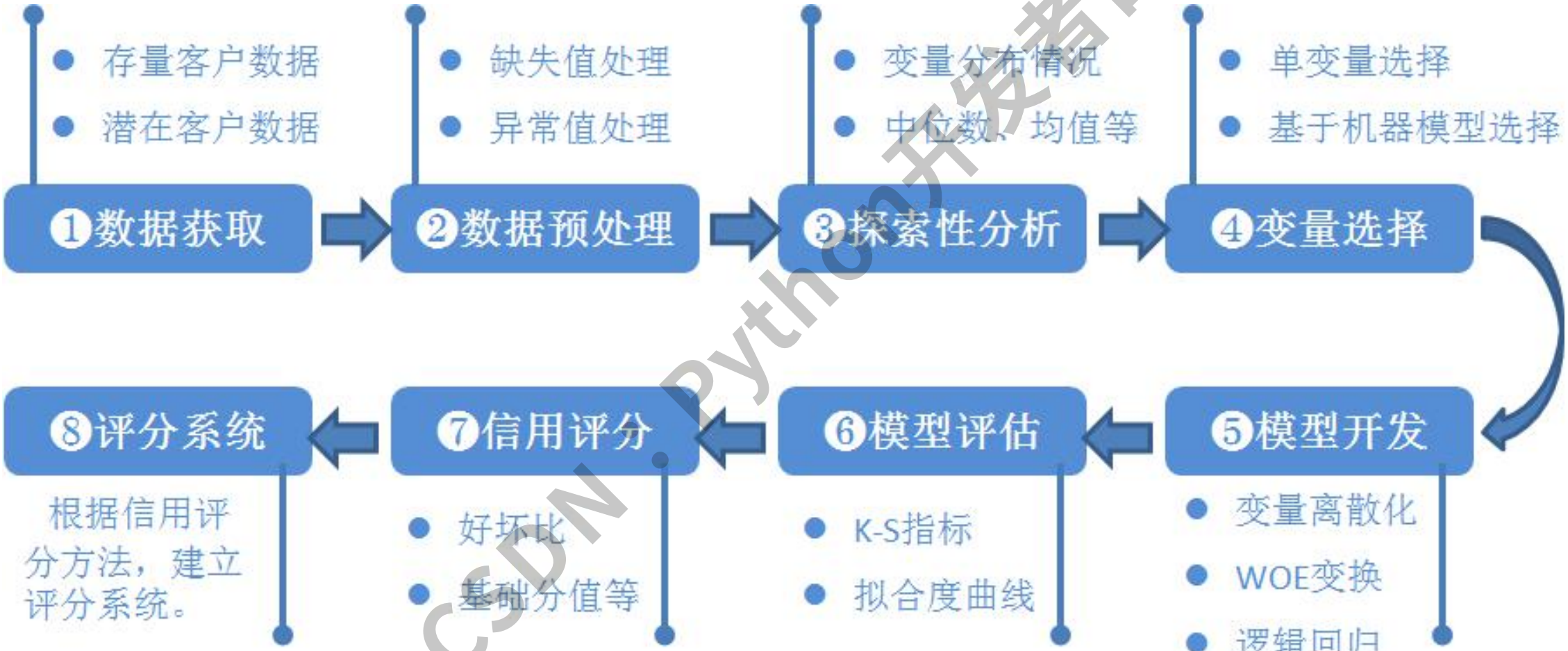
- 在客户获取期，建立信用局风险评分，预测客户带来违约风险的概率大小；
- 在客户申请处理期，建立申请风险评分模型，预测客户开户后一定时期内违约拖欠的风险概率，有效排除了信用不良客户和非目标客户的申请；
- 在帐户管理期，建立催收评分模型，对逾期帐户预测催收策略反应的概率，从而采取相应的催收措施。



运用信用评分卡需要注意的问题

- **开展贷款业务的历史要长** - 评分卡的发展必须以历史数据为依据，如果公司开展小贷业务的历史太短，数据不充分，则不具备开发评分卡的条件。
- **发展信用评分卡需要大量的数据，而且数据的质量要好** - 如果数据很少，不具有代表性或数据质量很差，有很多错误，那么基于该数据的评分卡就不会准确，那么申请评分卡的发展就会受到制约。
- **数据的保存要完整** - 小贷公司必须把历史上各个时期申请贷款的客户申请表信息、当时的信用报告记录等数据保存起来，不仅所有被批准的客户的数据要保存，被拒绝的申请者数据也应该保存，以进行模型的表现推测。而且，保存的数据不仅要足以提炼出各种预测变量，还要能够辨别其表现（好、坏等）。
- **信用评分卡只是提供了决策依据，不是决策本身** - 信用评分卡并不能告诉审批人员某个客户一定是好的或坏的，它只是告诉我们一定的概率，因此，对于有些客户的申请审批决定就必须综合信用报告等其它信息作出判断。
- **一张申请评分卡很难满足整个人群，需要针对不同人群建立单独的评分卡** – 比如多家分公司，存在着较大的地域差别，各地区经济发展也存在着较大差别，客户消费习惯有较大差异，如果使用一张申请评分卡就会造成信用评分的不真实。
- **时间越久，信用评分卡的有效性会降低** - 因为经济环境、市场状况和申请者、借款人的构成、业务的来源渠道在不断变化，使得样本人群的特质和属性发生改变，特别是在经济高速发展的阶段（或是股市大牛市阶段），人群的生活方式、消费习惯、经济状况等变化很快，申请评分模型在应用一段时间后通常会与初期模型产生偏移，所以需要适当重新调整，必要时还要重新开发，以保证信用评分卡的有效性。

项目流程



数据获取

海量征信数据 全方面构建用户信用画像

用户身份信息
居住地、婚姻状况、子女情况、工作单位、职位、房产、收入、联系人数据

用户认证数据
公积金、社保、运营商通讯、学历数据、职业数据（猎聘、脉脉和领英）

人行征信报告
贷款信息
信贷交易信息
个人公共信息

婚恋社交数据
家庭情况
房产情况
学历情况
生活作息
爱情规划
爱情账户等级
婚恋社交信用度评
婚恋社交人脉图谱

消费收支数据
线上电商和线下
银联消费数据，
银行卡收支数据
，
航旅出行数据

互金和银行黑灰名单
信贷逾期名单、司法不良名单、多头申请多头负债名单、团伙欺诈名单、百合&世纪佳缘婚恋社交黑名单

芝麻信用
信用评分、行业关注名单、申请欺诈评分、欺诈信息验证、欺诈关注清单、企业信用评分

用户行为数据
申请表单填写时间
借款协议页面停留时间

设备相关数据
设备指纹、设备硬件信息、GPS定位信息、设备安装APP数据

算法模型

常用算法模型

模型	解释	复杂度	应用场景
Logistics回归	影响程度大小与显著性，解释力度强，但只是线性，没有顾及到非线性，预测精度较低		申请评分、流失预测
决策树	1、描述性，重建用户场景，可做变量提取与用户画像 2、树的结构不稳定，可以得出变量重要性，可以作为变量筛选	叶子的数量	流失模式识别
随机森林	随机森林比决策树在变量筛选中，变量排序比较优秀		
神经网络	1、不可解释，内部使用，预测精度较高。可以作为初始模型的金模型（用以评估在给定数据条件下，逻辑回归可达到的最精确程度） 2、线性（逻辑回归）+非线性关系，可用于行为评分的预测模型（行为评分对模型可解释性不强），可用于申请评分的金模型 3、使用场景：先做一个神经网络，让预测精度（AUC）达到最大时，再用逻辑回归	迭代次数	申请评分的金模型； 行为评分的预测模型

模型评估方法

决策类评估——混淆矩阵指标
排序类评估——ROC指标

指标	描述	Scikit-learn函数
Precision	精准度	from sklearn.metrics import precision_score
Recall	召回率	from sklearn.metrics import recall_score
F1	F1值	from sklearn.metrics import f1_score
Confusion Matrix	混淆矩阵	from sklearn.metrics import confusion_matrix
ROC	ROC曲线	from sklearn.metrics import roc
AUC	ROC曲线下的面积	from sklearn.metrics import auc

Python实现

- 数据获取
- 数据预处理
  - 导入数据
  - 了解数据集
  - 缺失值处理
  - 异常值处理
  - 数据切分
- 探索性分析
- 变量选择
  - 分箱处理
  - WOE分析
  - 相关性分析和IV筛选
- 模型分析
  - WOE转换
  - Logisic模型建立
  - 模型检验
- 信用评分
  - 评分标准
  - 部分评分
- 自动评分系统

数据获取

- 数据来自Kaggle的Give Me Some Credit，有15万条的样本数据，数据属于个人消费类贷款，只考虑信用评分最终实施时能够使用到的数据应从如下一些方面获取数据：
  - 基本属性：包括了借款人当时的年龄。
  - 偿债能力：包括了借款人的月收入、负债比率。
  - 信用往来：两年内35-59天逾期次数、两年内60-89天逾期次数、两年内90天或高于90天逾期的次数。
  - 财产状况：包括了开放式信贷和贷款数量、不动产贷款或额度数量。
  - 贷款属性：暂无。
  - 其他因素：包括了借款人的家属数量（不包括本人在内）。
  - 时间窗口：自变量的观察窗口为过去两年，因变量表现窗口为未来两年。

数据预处理

- 导入数据并了解数据基本情况

```
# 导入数据
data = pd.read_csv('data/scoring/cs-training.csv', index_col=0)
# 查看数据的统计信息
data.describe().to_csv('data/scoring/data_describe.csv')
```

Column1	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
count	150000	150000	150000	150000	150000	120269	150000	150000	150000	150000	146076
mean	0.06684	6.048438055	52.29520667	0.421033333	353.0050758	6670.221237	8.45276	0.265973333	1.01824	0.240386667	0.757222268
std	0.249745531	249.7553706	14.77186586	4.192781272	2037.818523	14384.67422	5.14595099	4.169303788	1.129770985	4.155179421	1.115086071
min	0	0	0	0	0	0	0	0	0	0	0
25%	0	0.029867442	41	0	0.175073832	3400	5	0	0	0	0
50%	0	0.154180737	52	0	0.366507841	5400	8	0	1	0	0
75%	0	0.559046248	63	0	0.868253773	8249	11	0	2	0	1
max	1	50708	109	98	329664	3008750	58	98	54	98	20

- 从数据可以看出，MonthlyIncome和NumberOfDependents存在缺失值。

数据预处理 - 缺失值处理

这种情况在现实问题中非常普遍，这会导致一些不能处理缺失值的分析方法无法应用，因此，在信用风险评级模型开发的第一步就要进行缺失值处理。缺失值处理的方法，包括如下几种：

- 直接删除含有缺失值的样本
- 根据样本之间的相似性填补缺失值
- 根据变量之间的相关关系填补缺失值

变量MonthlyIncome缺失率比较大，所以根据变量之间的相关关系填补缺失值，采用随机森林法：

数据预处理 - 缺失值处理

变量MonthlyIncome缺失率比较大，所以根据变量之间的相关关系填补缺失值，采用随机森林法：

```
# 用随机森林对缺失值预测填充函数
def set_missing(df):
    # 把已有的数值型特征取出来
    process_df = df.iloc[:,[5,0,1,2,3,4,6,7,8,9]]
    # 分成已知该特征和未知该特征两部分
    known = process_df[process_df.MonthlyIncome.notnull()].as_matrix()
    unknown = process_df[process_df.MonthlyIncome.isnull()].as_matrix()
    # X为特征属性值
    X = known[:, 1:]
    # y为结果标签值
    y = known[:, 0]
    # fit到RandomForestRegressor之中
    rfr = RandomForestRegressor(random_state=0, n_estimators=200, max_depth=3, n_jobs=-1)
    rfr.fit(X, y)
    # 用得到的模型进行未知特征值预测
    predicted = rfr.predict(unknown[:, 1:]).round(0)
    print(predicted)
    # 用得到的预测结果填补原缺失数据
    df.loc[(df.MonthlyIncome.isnull()), 'MonthlyIncome'] = predicted
    return df

#用随机森林填补比MonthlyIncome的缺失值
data = set_missing(data)
```

数据预处理 - 缺失值处理

NumberOfDependents变量缺失值比较少，直接删除，对总体模型不会造成太大影响。对缺失值处理完之后，删除重复项。

```
#删除比NumberOfDependents的缺失值
data = data.dropna()
#删除重复项
data = data.drop_duplicates()
data.to_csv('data/scoring/NoMissingData.csv', index=False)
```

异常值处理

缺失值处理完毕后，还需要进行异常值处理。异常值是指明显偏离大多数抽样数据的数值，比如个人客户的年龄为0时，通常认为该值为异常值。找出样本总体中的异常值，通常采用离群值检测的方法。
首先，发现变量age中存在0，显然是异常值，直接剔除：

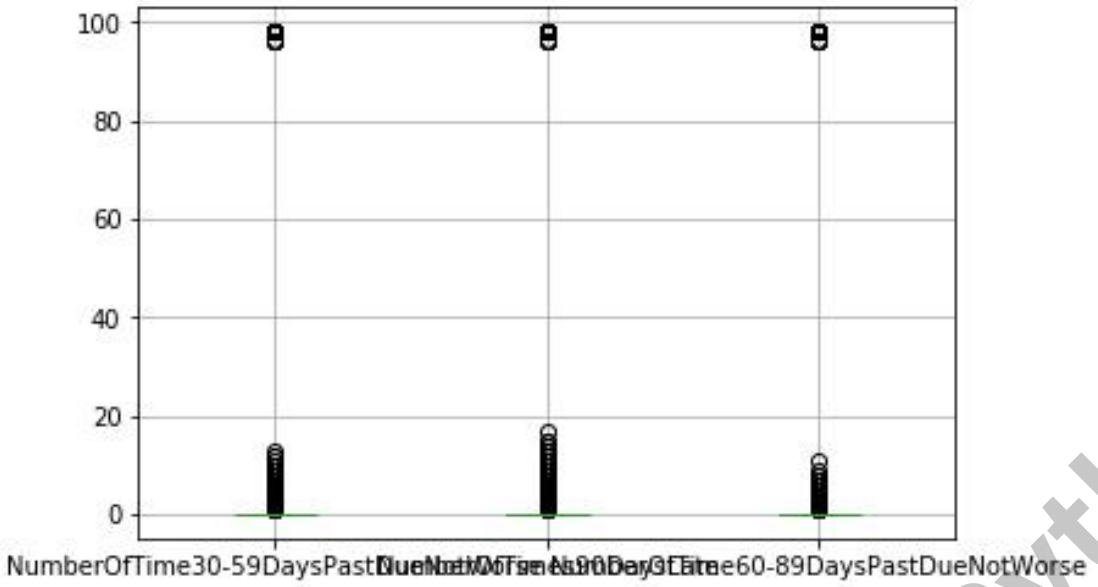
```
 # 年龄等于0的异常值进行剔除
data = data[data['age'] > 0]
```

异常值处理

对于变量NumberOfTime30-59DaysPastDueNotWorse、NumberOfTimes90DaysLate、NumberOfTime60-89DaysPastDueNotWorse这三个变量，由下面的箱线图可以看出，均存在异常值，并通过unique函数来确认异常值。

```
%matplotlib inline
import matplotlib.pyplot as plt
# 对NumberOfTime30-59DaysPastDueNotWorse、NumberOfTimes90DaysLate、NumberOfTime60-89DaysPastDueNotWorse画箱线图
data.boxplot(column=['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTimes90DaysLate',
                     'NumberOfTime60-89DaysPastDueNotWorse'])
plt.show()
print(pd.unique(data['NumberOfTime30-59DaysPastDueNotWorse']))
print(pd.unique(data['NumberOfTimes90DaysLate']))
print(pd.unique(data['NumberOfTime60-89DaysPastDueNotWorse']))
```

异常值处理



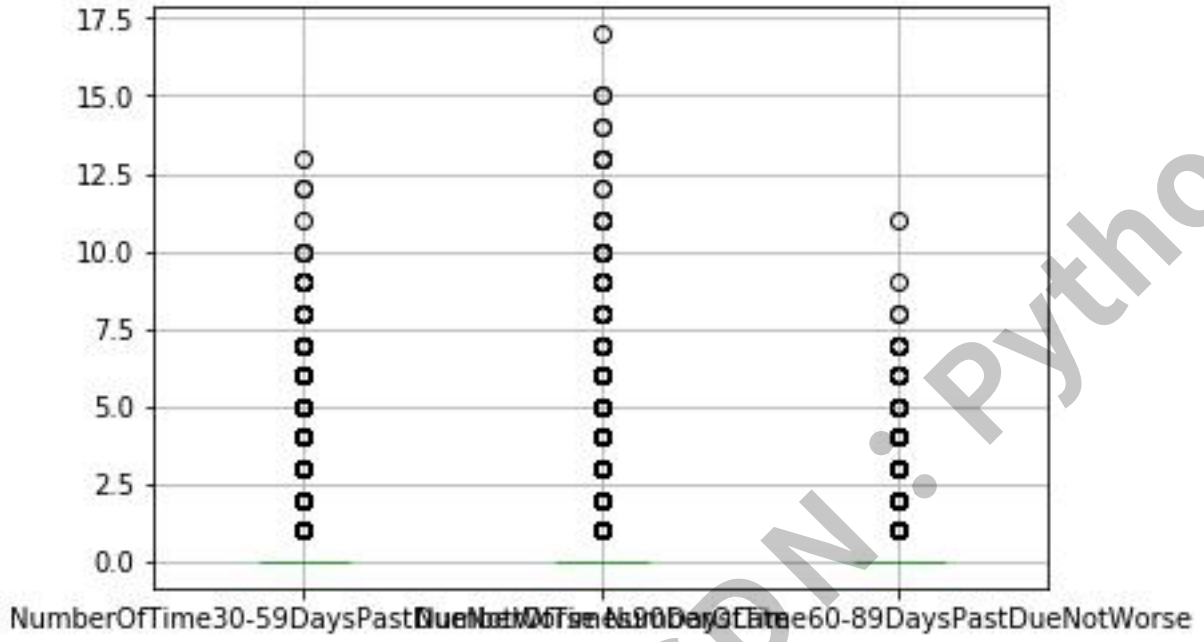

```
[ 2  0  1  3  4  5  7 10  6 98 12  8  9 96 13 11]
[ 0  1  3  2  5  4 98 10  9  6  7  8 15 96 11 13 14 17 12]
[ 0  1  2  5  3 98  4  6  7  8 96 11  9]
```

由unique函数可以得知均存在96、98两个异常值，因此予以剔除。

```
#剔除异常值
data = data[data['NumberOfTime30-59DaysPastDueNotWorse'] < 90]
data.boxplot(column=['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTimes90DaysLate',
                     'NumberOfTime60-89DaysPastDueNotWorse'])
plt.show()
```

异常值处理

同时会发现剔除其中NumberOfTime30-59DaysPastDueNotWorse变量的96、98值，其他变量的96、98两个值也会相应被剔除。



异常值处理

数据集中好客户为0，违约客户为1。考虑到正常的理解，能正常履约并支付利息的客户为1，所以将其取反。

```
#变量SeriousDlqin2yrs取反
data['SeriousDlqin2yrs'] = 1 - data['SeriousDlqin2yrs']
data.to_csv('data/scoring/CleanedData.csv', index=False)
```

数据切分

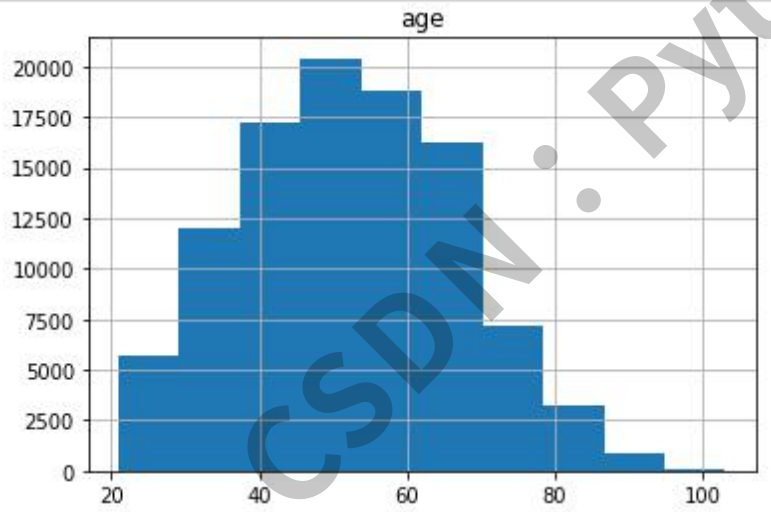
为了验证模型的拟合效果，将数据集进行切分，分成训练集和测试集。

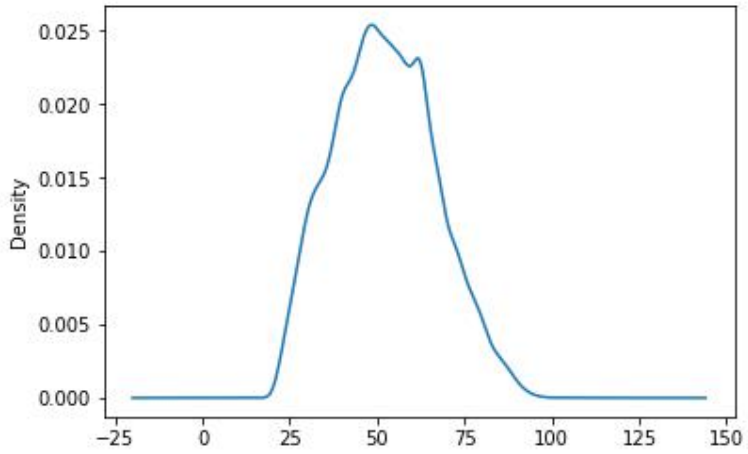
```
from sklearn.model_selection import train_test_split
Y = data['SeriousDlqin2yrs']
X = data.iloc[:, 1:]
#测试集占比30%
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

train = pd.concat([Y_train, X_train], axis=1)
test = pd.concat([Y_test, X_test], axis=1)
classTest = test.groupby('SeriousDlqin2yrs')['SeriousDlqin2yrs'].count()
train.to_csv('data/scoring/TrainData.csv',index=False)
test.to_csv('data/scoring/TestData.csv',index=False)
classTest
```

在建立模型之前，通常会对现有的数据进行探索性数据分析（Exploratory Data Analysis）。EDA是指对已有的数据(特别是调查或观察得来的原始数据)在尽量少的先验假定下进行探索。常用的探索性数据分析方法有：直方图、密度图、散点图和箱线图等。

首先看一下客户年龄的分布情况：

```
train.hist(column=['age'])
plt.show()
train['age'].plot(kind='density')
plt.show()
```

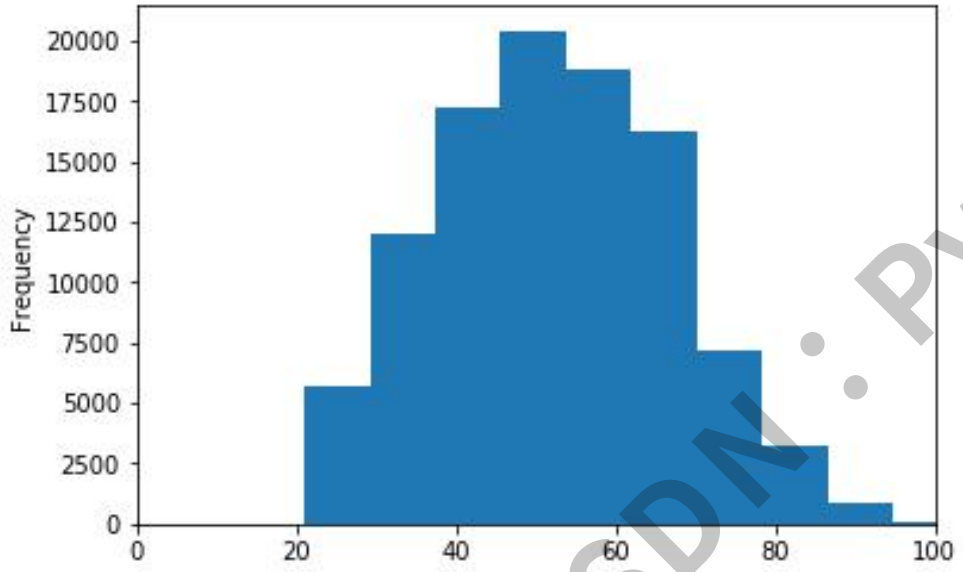


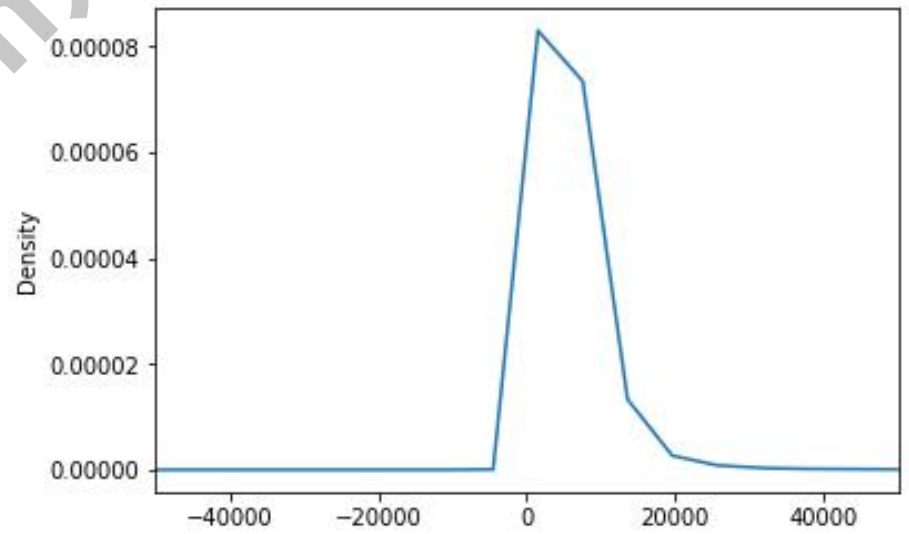


可以看到年龄变量大致呈正态分布，符合统计分析的假设。

同样对客户的月收入状况也做一下同样的简单分析。

```
train['age'].plot(kind='hist', xlim=(0, 100))
plt.show()
train['MonthlyIncome'].plot(kind='density', xlim=(-50000, 50000))
plt.show()
```





可以看到月收入也基本符合正态分布，也符合统计分析需求。其他的字段可以根据情况逐个字段进行分析。

分箱处理

- 变量分箱（binning）是对连续变量离散化（discretization）的一种称呼。信用评分卡开发中一般有常用的等距分段、等深分段、最优分段。其中等距分段（Equval length intervals）是指分段的区间是一致的，比如年龄以十年作为一个分段；等深分段（Equal frequency intervals）是先确定分段数量，然后令每个分段中数据数量大致相等；最优分段（Optimal Binning）又叫监督离散化（supervised discretizaion），使用递归划分（Recursive Partitioning）将连续变量分为分段，背后是一种基于条件推断查找较佳分组的算法。
- 通常选择对连续变量进行最优分段，在连续变量的分布不满足最优分段的要求时，再考虑对连续变量进行等距分段。

分箱处理

自定义最优分段分箱函数， 对于数据集中的RevolvingUtilizationOfUnsecuredLines、age、DebtRatio和MonthlyIncome进行最优分类。

```
from scipy import stats
# 自动分箱函数
def mono_bin(Y, X, n=20):
    r = 0
    good = Y.sum()
    bad = Y.count() - good
    while np.abs(r) < 1:
        d1 = pd.DataFrame({"X": X, "Y": Y, "Bucket": pd.qcut(X, n, duplicates='drop')})
        d2 = d1.groupby('Bucket', as_index = True)
        r, p = stats.spearmanr(d2.mean().X, d2.mean().Y)
        n = n - 1
    d3 = pd.DataFrame(d2.X.min(), columns = ['min'])
    d3['min']=d2.min().X
    d3['max'] = d2.max().X
    d3['sum'] = d2.sum().Y
    d3['total'] = d2.count().Y
    d3['rate'] = d2.mean().Y
    d3['woe'] = np.log((d3['rate'] / (1 - d3['rate'])) / (good / bad))
    d4 = (d3.sort_values(by='min')).reset_index(drop=True)
    return d4
```

分箱处理

使用自定义的最优分段分箱函数对age进行分类

```
age_bin = mono_bin(data['SeriousDlqin2yrs'], data['age'])
age_bin
```

	min	max	sum	total	rate	woe
0	21	30	8913	10054	0.886513	-0.581713
1	31	34	7469	8349	0.894598	-0.498725
2	35	38	8691	9617	0.903712	-0.398150
3	39	41	8271	9095	0.909401	-0.330979
4	42	43	5675	6224	0.911793	-0.301592
5	44	46	9543	10381	0.919276	-0.204774
6	47	48	6816	7399	0.921206	-0.178478
7	49	50	6890	7471	0.922233	-0.164243
8	51	53	9850	10639	0.925839	-0.112859
9	54	55	6360	6810	0.933921	0.011217
10	56	58	9614	10151	0.947099	0.247659
11	59	61	9290	9765	0.951357	0.336060
12	62	63	6792	7081	0.959187	0.519755
13	64	66	7602	7856	0.967668	0.761513
14	67	70	7927	8133	0.974671	1.012835
15	71	76	8039	8247	0.974779	1.017203
16	77	107	7906	8082	0.978223	1.167574

分箱处理

使用自定义的最优分段分箱函数对RevolvingUtilizationOfUnsecuredLines进行分类

```
RevolvingUtilizationOfUnsecuredLines_bin = mono_bin(data['SeriousDlqin2yrs'],
                                                    data['RevolvingUtilizationOfUnsecuredLines'])
RevolvingUtilizationOfUnsecuredLines_bin
```

	min	max	sum	total	rate	woe
0	0.000000	0.031125	35659	36339	0.981287	1.322345
1	0.031128	0.158089	35590	36338	0.979415	1.225098
2	0.158100	0.558255	34499	36338	0.949392	0.294389
3	0.558278	50708.000000	29900	36339	0.822807	-1.101834

分箱处理

使用自定义的最优分段分箱函数对DebtRatio进行分类

```
DebtRatio_bin = mono_bin(data['SeriousDlqin2yrs'], data['DebtRatio'])
DebtRatio_bin
```

	min	max	sum	total	rate	woe
0	0.000000	0.235948	45593	48452	0.940993	0.131963
1	0.235953	0.544862	45434	48451	0.937731	0.074679
2	0.544864	329664.000000	44621	48451	0.920951	-0.181979

分箱处理

使用自定义的最优分段分箱函数对MonthlyIncome进行分类

```
MonthlyIncome_bin = mono_bin(data['SeriousDlqin2yrs'], data['MonthlyIncome'])
MonthlyIncome_bin
```

	min	max	sum	total	rate	woe
0	0.0	3400.0	44952	48760	0.921903	-0.168828
1	3401.0	6850.0	44600	48145	0.926368	-0.105123
2	6851.0	3008750.0	46096	48449	0.951433	0.337716

分箱处理

自定义等距分段分箱函数， 对不满足最优分段分箱的变量进行等距分段。

```
#自定义等距分段分箱函数
def self_bin(Y, X, cat):
    good = Y.sum()
    bad = Y.count() - good
    d1 = pd.DataFrame({'X': X,'Y': Y,'Bucket': pd.cut(X, cat)})
    d2 = d1.groupby('Bucket', as_index=True)
    d3 = pd.DataFrame(d2.X.min(), columns=['min'])
    d3['min'] = d2.min().X
    d3['max'] = d2.max().X
    d3['sum'] = d2.sum().Y
    d3['total'] = d2.count().Y
    d3['rate'] = d2.mean().Y
    d3['woe'] = np.log((d3['rate'] / (1 - d3['rate'])) / (good / bad))
    d3['goodattribute'] = d3['sum'] / good
    d3['badattribute'] = (d3['total'] - d3['sum']) / bad
    d4 = (d3.sort_values(by='min'))
    return d4
```

分箱处理

使用自定义的最优分段分箱函数对NumberOfTime30-59DaysPastDueNotWorse进行分类

```
cutx3 = [ninf, 0, 1, 3, 5, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberOfTime30-59DaysPastDueNotWorse'], cutx3)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 0.0]	0	0	117077	122020	0.959490	0.527540	0.863094	0.509273
(0.0, 1.0]	1	1	13381	15744	0.849911	-0.903415	0.098645	0.243458
(1.0, 3.0]	2	3	4467	6279	0.711419	-1.735033	0.032931	0.186689
(3.0, 5.0]	4	5	606	1075	0.563721	-2.381042	0.004467	0.048321
(5.0, inf]	6	13	117	236	0.495763	-2.654269	0.000863	0.012260

分箱处理

使用自定义的最优分段分箱函数对NumberOfOpenCreditLinesAndLoans进行分类

```
cutx6 = [ninf, 1, 2, 3, 5, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberOfOpenCreditLinesAndLoans'], cutx6)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 1.0]	0	1	4438	5322	0.833897	-1.023817	0.032717	0.091078
(1.0, 2.0]	2	2	5577	6162	0.905063	-0.382525	0.041114	0.060272
(2.0, 3.0]	3	3	7853	8519	0.921822	-0.169958	0.057892	0.068617
(3.0, 5.0]	4	5	22082	23622	0.934807	0.025661	0.162789	0.158665
(5.0, inf]	6	58	95698	101729	0.940715	0.126966	0.705488	0.621368

分箱处理

使用自定义的最优分段分箱函数对NumberOfTimes90DaysLate进行分类

```
cutx7 = [ninf, 0, 1, 3, 5, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberOfTimes90DaysLate'], cutx7)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 0.0]	0	0	131008	137449	0.953139	0.375256	0.965794	0.663610
(0.0, 1.0]	1	1	3396	5130	0.661988	-1.965152	0.025035	0.178652
(1.0, 3.0]	2	3	1041	2178	0.477961	-2.725530	0.007674	0.117144
(3.0, 5.0]	4	5	142	417	0.340528	-3.298263	0.001047	0.028333
(5.0, inf]	6	17	61	180	0.338889	-3.305569	0.000450	0.012260

分箱处理

使用自定义的最优分段分箱函数对NumberRealEstateLoansOrLines进行分类

```
cutx8 = [ninf, 0,1, 2, 3, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberRealEstateLoansOrLines'], cutx8)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 0.0]	0	0	48757	53172	0.916968	-0.235478	0.359438	0.454873
(0.0, 1.0]	1	1	48477	51191	0.946983	0.245347	0.357373	0.279621
(1.0, 2.0]	2	2	29410	31155	0.943990	0.187261	0.216811	0.179786
(2.0, 3.0]	3	3	5812	6230	0.932905	-0.005120	0.042846	0.043066
(3.0, inf]	4	54	3192	3606	0.885191	-0.594782	0.023531	0.042654

分箱处理

使用自定义的最优分段分箱函数对NumberOfTime60-89DaysPastDueNotWorse进行分类

```
cutx9 = [ninf, 0, 1, 3, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberOfTime60-89DaysPastDueNotWorse'], cutx9)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 0.0]	0	0	130993	138127	0.948352	0.272953	0.965683	0.735009
(0.0, 1.0]	1	1	3905	5647	0.691518	-1.830095	0.028788	0.179477
(1.0, 3.0]	2	3	688	1415	0.486219	-2.692457	0.005072	0.074902
(3.0, inf]	4	11	62	165	0.375758	-3.144914	0.000457	0.010612

分箱处理

使用自定义的最优分段分箱函数对NumberOfDependents进行分类

```
cutx10 = [ninf, 0, 1, 2, 3, 5, pinf]
self_bin(data['SeriousDlqin2yrs'], data['NumberOfDependents'], cutx10)
```

Bucket	min	max	sum	total	rate	woe	goodattribute	badattribute
(-inf, 0.0]	0.0	0.0	81248	86234	0.942181	0.153553	0.598962	0.513703
(0.0, 1.0]	1.0	1.0	24370	26291	0.926933	-0.096812	0.179656	0.197919
(1.0, 2.0]	2.0	2.0	17929	19500	0.919436	-0.202612	0.132173	0.161859
(2.0, 3.0]	3.0	3.0	8646	9479	0.912122	-0.297501	0.063738	0.085823
(3.0, 5.0]	4.0	5.0	3241	3605	0.899029	-0.450836	0.023893	0.037503
(5.0, inf]	6.0	20.0	214	245	0.873469	-0.705330	0.001578	0.003194

WOE分析

- WoE分析，是对指标分箱、计算各个档位的WoE值并观察WoE值随指标变化的趋势。WOE的全称是“Weight of Evidence”，即证据权重。WOE是对原始自变量的一种编码形式。其中WoE的数学定义是:

$$WOE_i = \ln(\frac{py_i}{pn_i}) = \ln(\frac{\#y_i / \#y_T}{\#n_i / \#n_T})$$

ln(goodattribute/badattribute)

- 从这个公式中可以体会到，WOE表示的实际上是“当前分组中响应客户占所有响应客户的比例”和“当前分组中没有响应的客户占所有没有响应的客户的比例”的差异。
- WOE也可以这么理解，他表示的是当前这个组中响应客户和未响应客户的比值，和所有样本中这个比值的差异。这个差异是用这两个比值的比值，再取对数来表示的。WOE越大，这种差异越大，这个分组里的样本响应的可能性就越大，WOE越小，差异越小，这个分组里的样本响应的可能性就越小。
- 在进行分析时，需要对各指标从小到大排列，并计算出相应分档的WoE值。其中正向指标越大，WoE值越小；反向指标越大，WoE值越大。正向指标的WoE值负斜率越大，反响指标的正斜率越大，则说明指标区分能力好。WoE值趋近于直线，则意味指标判断能力较弱。若正向指标和WoE正相关趋势、反向指标同WoE出现负相关趋势，则说明此指标不符合经济意义，则应当予以去除。
- woe函数实现在在定义的分段函数里面已经包含，这里不再重复。

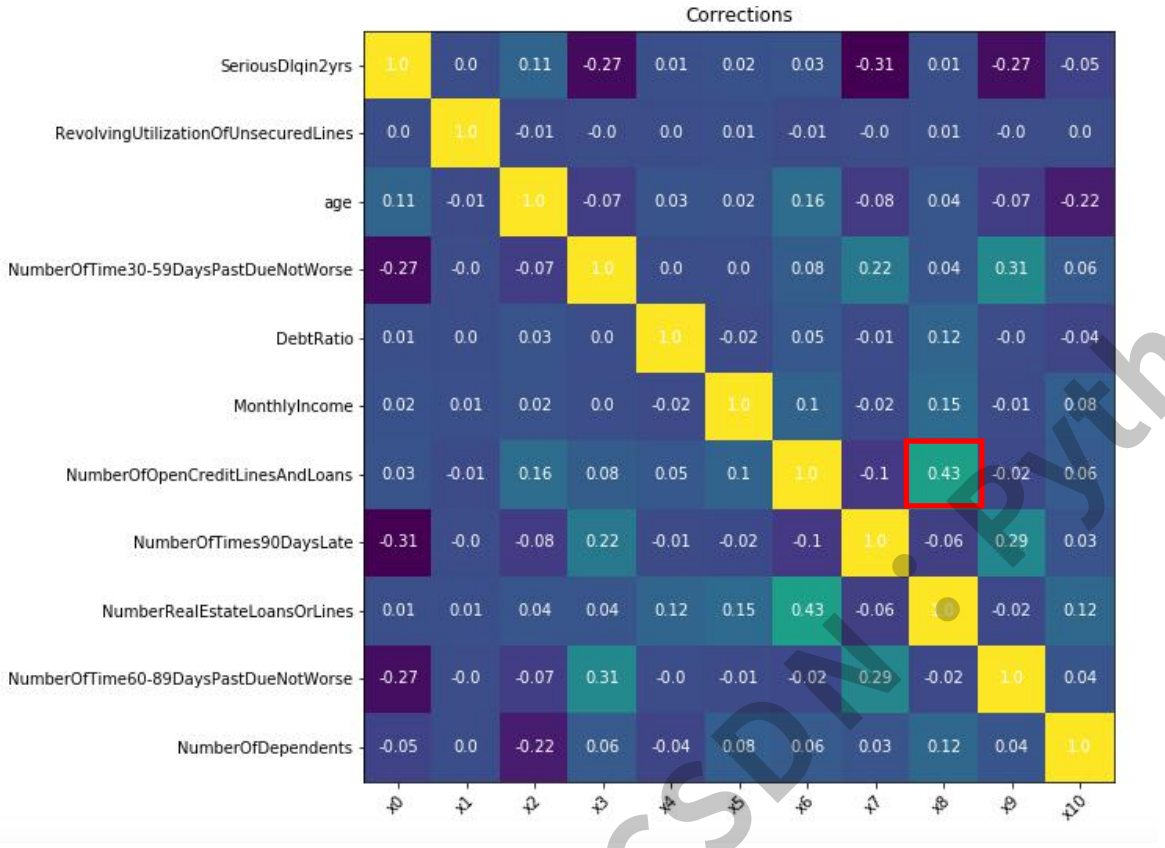
相关性分析

清洗后的数据看一下变量间的相关性。注意，这里的相关性分析只是初步的检查，进一步检查模型的VI（证据权重）作为变量筛选的依据。实现代码如下：

```
#计算各变量的相关性系数
corr = data.corr()
#x轴标签
xticks = ['x0','x1','x2','x3','x4','x5','x6','x7','x8','x9','x10']
#y轴标签
yticks = list(corr.index)
fig, ax = plt.subplots()
fig.set_size_inches(10, 10)
im = ax.imshow(corr)

# We want to show all ticks...
ax.set_xticks(np.arange(len(xticks)))
ax.set_yticks(np.arange(len(yticks)))
# ... and label them with the respective list entries
ax.set_xticklabels(xticks)
ax.set_yticklabels(yticks)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(yticks)):
    for j in range(len(xticks)):
        text = ax.text(j, i, np.round(corr.values[i, j], decimals=2), ha="center", va="center", color="w")

ax.set_title("Corrections")
fig.tight_layout()
plt.show()
```

相关性分析

计算相关性的结果如下：



由图可以看出，各变量之间的相关性是非常小的。NumberOfOpenCreditLinesAndLoans和NumberRealEstateLoansOrLines的相关性系数为0.43。

IV筛选

- IV的全称是Information Value，中文意思是信息价值，或者信息量。
- 在用逻辑回归、决策树等模型方法构建分类模型时，经常需要对自变量进行筛选。比如有200个候选自变量，通常情况下，不会直接把200个变量直接放到模型中去进行拟合训练，而是会用一些方法，从这200个自变量中挑选一些出来，放进模型，形成入模变量列表。那么怎么去挑选入模变量呢？
- 挑选入模变量过程是个比较复杂的过程，需要考虑的因素很多，比如：变量的预测能力，变量之间的相关性，变量的简单性（容易生成和使用），变量的强壮性（不容易被绕过），变量在业务上的可解释性（被挑战时可以解释的通）等等。但是，其中最主要和最直接的衡量标准是变量的预测能力。
- “变量的预测能力”这个说法很笼统，很主观，非量化，在筛选变量的时候我们总不能说：“我觉得这个变量预测能力很强，所以他要进入模型”吧？因此，需要一些具体的量化指标来衡量每自变量的预测能力，并根据这些量化指标的大小，来确定哪些变量进入模型。IV就是这样一种指标，它可以用来衡量自变量的预测能力。类似的指标还有信息增益、基尼系数等等。
- IV的计算公式（ IV=sum((goodattribute - badattribute)*ln(goodattribute / badattribute)) ）

$$IV_i = (py_i - pn_i) * WOE_i = (py_i - pn_i) * \ln(\frac{py_i}{pn_i}) = (\frac{\#y_i}{\#y_T} - \frac{\#n_i}{\#n_T}) * \ln(\frac{\#y_i / \#y_T}{\#n_i / \#n_T})$$

$$IV = \sum_i^n IV_i$$

IV筛选

通过IV值判断变量预测能力的标准是：

- < 0.02: unpredictive
- 0.02 to 0.1: weak
- 0.1 to 0.3: medium
- 0.3 to 0.5: strong
- > 0.5: suspicious

IV的实现放在自定义的分段函数里面:

```
from scipy import stats
# 自动最优分段分箱函数
def mono_bin(Y, X, n=20):
    r = 0
    good = Y.sum()
    bad = Y.count() - good
    while np.abs(r) < 1:
        d1 = pd.DataFrame({"X": X, "Y": Y, "Bucket": pd.qcut(X, n, duplicates='drop')})
        d2 = d1.groupby('Bucket', as_index = True)
        r, p = stats.spearmanr(d2.mean().X, d2.mean().Y)
        n = n - 1
    d3 = pd.DataFrame(d2.X.min(), columns = ['min'])
    d3['min']=d2.min().X
    d3['max'] = d2.max().X
    d3['sum'] = d2.sum().Y
    d3['total'] = d2.count().Y
    d3['rate'] = d2.mean().Y
    d3['woe'] = np.log((d3['rate'] / (1 - d3['rate'])) / (good / bad))
    d3['goodattribute'] = d3['sum'] / good
    d3['badattribute'] = (d3['total'] - d3['sum']) / bad
    iv = ((d3['goodattribute'] - d3['badattribute']) * d3['woe']).sum()
    d4 = (d3.sort_values(by='min')).reset_index(drop=True)

    cut=[]
    cut.append(ninf)
    for i in range(1, n + 1):
        qua = X.quantile(i / (n + 1))
        cut.append(round(qua, 4))
    cut.append(pinf)
    woe = list(d4['woe'].round(3))
    return d4, iv, cut, woe
```

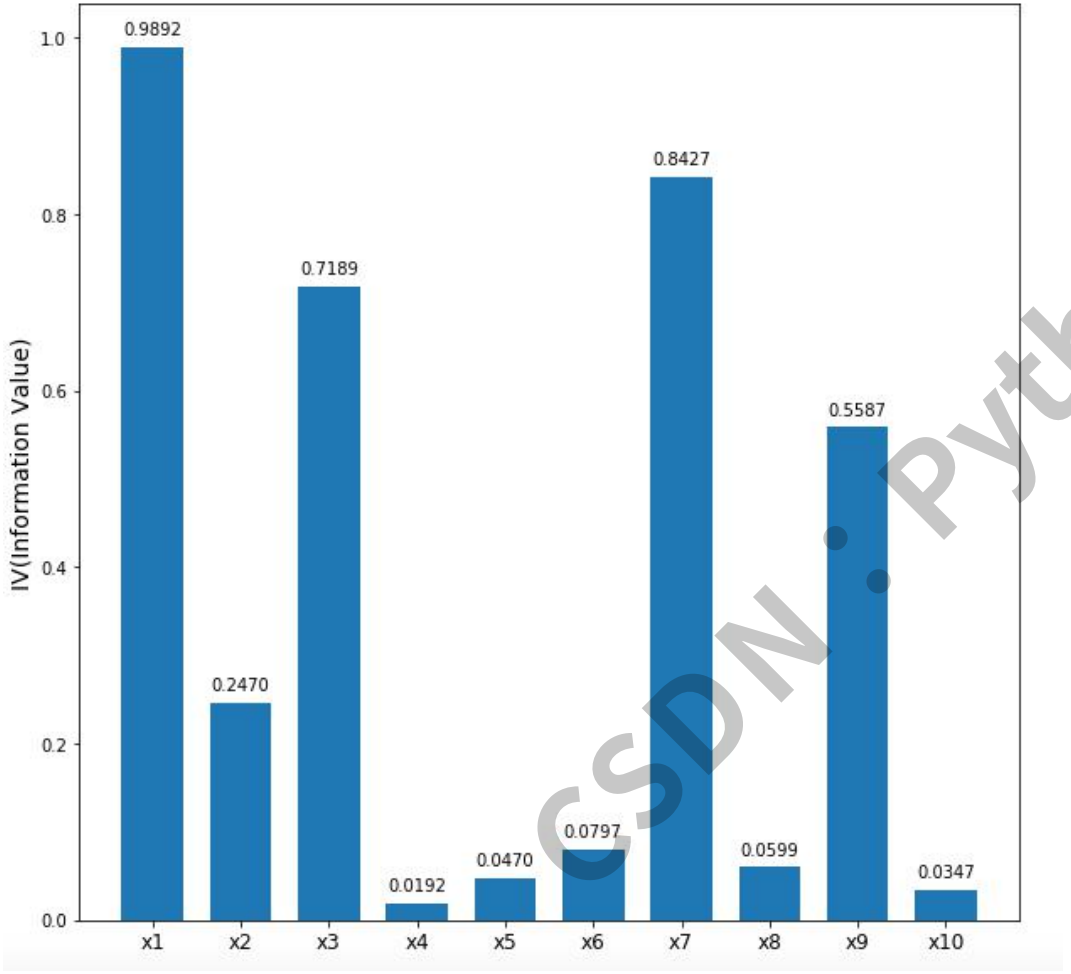
IV筛选

计算IV的值后，通过柱状体来看一下各个变量的IV分布。

```
#各变量IV
ivlist = [ivx1, ivx2, ivx3, ivx4, ivx5, ivx6, ivx7, ivx8, ivx9, ivx10]
#x轴的标签
index=['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10']
fig = plt.figure(1)
fig.set_size_inches(10, 10)
ax = fig.add_subplot(1, 1, 1)
x = np.arange(len(index)) + 1
#生成柱状图
ax.bar(x, ivlist, width=0.7)
ax.set_xticks(x)
ax.set_xticklabels(index, rotation=0, fontsize=12)
ax.set_ylabel('IV(Information Value)', fontsize=14)
#在柱状图上添加数字标签
for a, b in zip(x, ivlist):
    plt.text(a, b + 0.01, '%.4f' % b, ha='center', va='bottom', fontsize=10)
plt.show()
```

IV筛选

结果如下，可以看出， DebtRatio、MonthlyIncome、NumberOfOpenCreditLinesAndLoans、NumberRealEstateLoansOrLines和NumberOfDependents变量的IV值明显较低，所以予以删除。



模型分析

- 证据权重（Weight of Evidence,WOE）转换可以将Logistic回归模型转变为标准评分卡格式。引入WOE转换的目的并不是为了提高模型质量，只是一些变量不应该被纳入模型，这或者是因为它们不能增加模型值，或者是因为与其模型相关系数有关的误差较大。
- 建立标准信用评分卡也可以不采用WOE转换。这种情况下，Logistic回归模型需要处理更大数量的自变量。这样不仅会增加建模程序的复杂性，但最终得到的评分卡都是一样的。
- 在建立模型之前，需要将筛选后的变量转换为WoE值，便于信用评分。

已经能获取了每个变量的分箱数据和woe数据，只需要根据各变量数据进行替换，实现代码如下：

```
#替换成woe函数
def replace_woe(series, cut, woe):
    list=[]
    i = 0
    while i < len(series):
        value = series[i]
        j = len(cut) - 2
        m = len(cut) - 2
        while j >= 0:
            if value >= cut[j]:
                j = -1
            else:
                j -= 1
                m -= 1
        list.append(woe[m])
        i += 1
    return list
```

WOE转换

使用替换函数对数据进行转换，实现代码如下：

```
# Training数据替换成woe
from pandas import Series
data = pd.read_csv('data/scoring/TrainData.csv')
data['RevolvingUtilizationOfUnsecuredLines'] = Series(replace_woe(data['RevolvingUtilizationOfUnsecuredLines'].values,
                                                                  cutx1, woex1))
data['age'] = Series(replace_woe(data['age'].values, cutx2, woex2))
data['NumberOfTime30-59DaysPastDueNotWorse'] = Series(replace_woe(data['NumberOfTime30-59DaysPastDueNotWorse'].values,
                                                                  cutx3, woex3))
data['DebtRatio'] = Series(replace_woe(data['DebtRatio'].values, cutx4, woex4))
data['MonthlyIncome'] = Series(replace_woe(data['MonthlyIncome'].values, cutx5, woex5))
data['NumberOfOpenCreditLinesAndLoans'] = Series(replace_woe(data['NumberOfOpenCreditLinesAndLoans'].values,
                                                             cutx6, woex6))
data['NumberOfTimes90DaysLate'] = Series(replace_woe(data['NumberOfTimes90DaysLate'].values, cutx7, woex7))
data['NumberRealEstateLoansOrLines'] = Series(replace_woe(data['NumberRealEstateLoansOrLines'].values, cutx8, woex8))
data['NumberOfTime60-89DaysPastDueNotWorse'] = Series(replace_woe(data['NumberOfTime60-89DaysPastDueNotWorse'].values,
                                                                  cutx9, woex9))
data['NumberOfDependents'] = Series(replace_woe(data['NumberOfDependents'].values, cutx10, woex10))
data.to_csv('data/scoring/TrainWoeData.csv', index=False)
```

Logisic模型建立

使用scikit-learn来构建逻辑回归函数模型，代码如下：

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score
#导入Training数据
data_train = pd.read_csv('data/scoring/TrainWoeData.csv')

#应变量
Y_train = data_train['SeriousDlqin2yrs']
#自变量，剔除对因变量影响不明显的变量
X_train = data_train.drop(['SeriousDlqin2yrs', 'DebtRatio', 'MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans',
                'NumberRealEstateLoansOrLines', 'NumberOfDependents'], axis=1)

logit = LogisticRegression()
kfold = KFold(n_splits=10, random_state=7)
result = cross_val_score(logit, X_train, Y_train, cv=kfold, scoring='accuracy')
print(result.mean())
```

```
#训练模型
model = LogisticRegression()
model.fit(X_train, Y_train)
```

模型检验

到这里，建模部分基本结束了。需要验证一下模型的预测能力如何。使用在建模开始阶段预留的test数据进行检验。首先使用scikit-learn的classification_report来看一下模型的准确度。

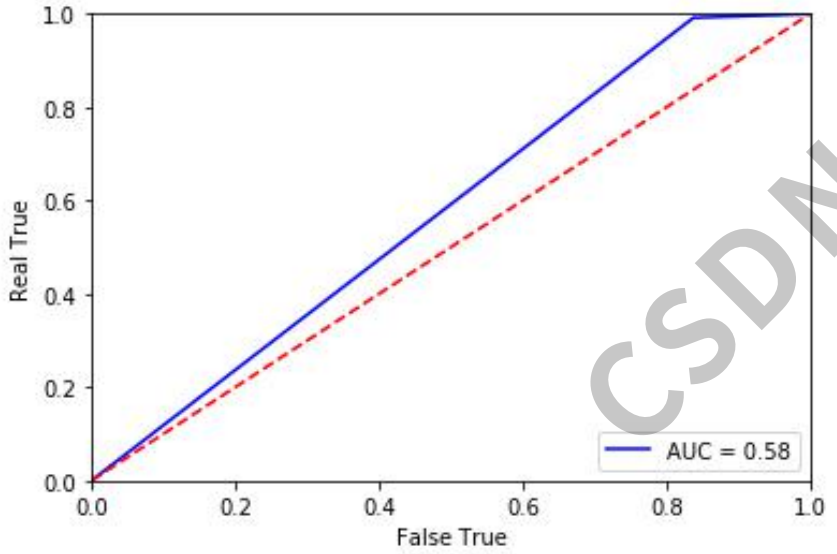
```
#导入Test数据
data_test = pd.read_csv('data/scoring/TestWoeData.csv')
#应变量
Y_test = data_test['SeriousDlqin2yrs']
#自变量，剔除对因变量影响不明显的变量
X_test = data_test.drop(['SeriousDlqin2yrs', 'DebtRatio', 'MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans',
                'NumberRealEstateLoansOrLines', 'NumberOfDependents'], axis=1)
#用模型预测数据
Y_predict = model.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_predict))
```

```
              precision    recall  f1-score   support

           0       0.58      0.16      0.25      6813
           1       0.94      0.99      0.97     94934

   micro avg       0.94      0.94      0.94    101747
   macro avg       0.76      0.58      0.61    101747
weighted avg       0.92      0.94      0.92    101747
```

模型检验

通过ROC曲线和AUC来评估模型的拟合能力。

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, threshold = roc_curve(Y_test, Y_predict)
rocauc = auc(fpr, tpr)
#生成ROC曲线
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % rocauc)
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('Real True')
plt.xlabel('False True')
plt.show()
```



在 Logistic 回归模型中，将概率发生比的对数表示成特征变量的线性组合，公式如下：

$$\begin{aligned}\text{Logit}(pi)&=\log(p\_good/p\_bad)\\&=\log(odds)\\&=age\_woe*b\ age+\\&\ status\_woe*b\ car+\\&\ +a\end{aligned}\tag{6}$$

每个属性对应的分值可以通过下面的公式计算：WOE 乘该变量的回归系数，加上回归截距，再乘上比例因子，最后加上偏移量：

$$(woe_i*\beta_i+\frac{a}{n})*factor+\frac{offset}{n}\tag{7}$$

对于评分卡的分值，我们可以这样计算：

$$\begin{aligned}score&=\log(odds)*factor+offset\\&=(\sum_{i=1}^{n}(woe_i*\beta_i)+a)*factor+offset\\&=(\sum_{i=1}^{n}(woe_i*\beta_i+\frac{a}{n}))*factor+offset\\&=\sum_{i=1}^{n}((woe_i*\beta_i+\frac{a}{n})*factor+\frac{offset}{n})\end{aligned}\tag{8}$$

评分标准

依据以上论文资料得到：

a=log (p_good/P_bad)

Score = offset + factor * log(odds)

在建立标准评分卡之前，需要选取几个评分卡参数：基础分值、PDO（比率翻倍的分值）和好坏比。 这里，采用600分为基础分值，PDO为20 （每高20分好坏比翻一倍），好坏比取20。

```
coe = model.coef_[0]
```

```
import math
#取600分为基础分值，PDO为20（每高20分好坏比翻一倍），好坏比取20。
p = 20 / math.log(2)
q = 600 - 20 * math.log(20) / math.log(2)
baseScore = round(q + p * coe[0], 0)
baseScore
```

个人总评分=基础分+各部分得分

评分标准 - 部分评分

计算各变量部分的分数。各部分得分函数：

```
#计算分数函数
def get_score(coe, woe, factor):
    scores=[]
    for w in woe:
        score=round(coe * w * factor, 0)
        scores.append(score)
    return scores
```

评分标准 - 部分评分

RevolvingUtilizationOfUnsecuredLines变量的分数：

```
x1 = get_score(coe[0], woex1, p)
dfx1['score'] = Series(x1)
dfx1
```

	min	max	sum	total	rate	woe	goodattribute	badattribute	score
0	0.000000	0.031125	35659	36339	0.981287	1.322345	0.262879	0.070060	24.0
1	0.031128	0.158089	35590	36338	0.979415	1.225098	0.262370	0.077066	22.0
2	0.158100	0.558255	34499	36338	0.949392	0.294389	0.254327	0.189470	5.0
3	0.558278	50708.000000	29900	36339	0.822807	-1.101834	0.220423	0.663404	-20.0

评分标准 - 部分评分

Age变量的分数：

```
x2 = get_score(coe[1], woex2, p)
dfx2['score'] = Series(x2)
dfx2
```

	min	max	sum	total	rate	woe	goodattribute	badattribute	score
0	21	30	8913	10054	0.886513	-0.581713	0.065707	0.117556	-9.0
1	31	34	7469	8349	0.894598	-0.498725	0.055062	0.090666	-7.0
2	35	38	8691	9617	0.903712	-0.398150	0.064070	0.095405	-6.0
3	39	41	8271	9095	0.909401	-0.330979	0.060974	0.084896	-5.0
4	42	43	5675	6224	0.911793	-0.301592	0.041836	0.056563	-4.0
5	44	46	9543	10381	0.919276	-0.204774	0.070351	0.086338	-3.0
6	47	48	6816	7399	0.921206	-0.178478	0.050248	0.060066	-3.0
7	49	50	6890	7471	0.922233	-0.164243	0.050793	0.059860	-2.0
8	51	53	9850	10639	0.925839	-0.112859	0.072614	0.081290	-2.0
9	54	55	6360	6810	0.933921	0.011217	0.046886	0.046363	0.0
10	56	58	9614	10151	0.947099	0.247659	0.070875	0.055327	4.0
11	59	61	9290	9765	0.951357	0.336060	0.068486	0.048939	5.0
12	62	63	6792	7081	0.959187	0.519755	0.050071	0.029775	8.0
13	64	66	7602	7856	0.967668	0.761513	0.056042	0.026169	11.0
14	67	70	7927	8133	0.974671	1.012835	0.058438	0.021224	15.0
15	71	76	8039	8247	0.974779	1.017203	0.059264	0.021430	15.0
16	77	107	7906	8082	0.978223	1.167574	0.058283	0.018133	17.0

评分标准 - 部分评分

NumberOfTime30-59DaysPastDueNotWorse变量的分数：

```
x3 = get_score(coe[2], woex3, p)
index = list(range(len(dfx3.index)))
dfx3.index = index
dfx3['score'] = Series(x3)
dfx3
```

	min	max	sum	total	rate	woe	goodattribute	badattribute	score
0	0	0	117077	122020	0.959490	0.527540	0.863094	0.509273	16.0
1	1	1	13381	15744	0.849911	-0.903415	0.098645	0.243458	-27.0
2	2	3	4467	6279	0.711419	-1.735033	0.032931	0.186689	-52.0
3	4	5	606	1075	0.563721	-2.381042	0.004467	0.048321	-71.0
4	6	13	117	236	0.495763	-2.654269	0.000863	0.012260	-79.0

评分标准 - 部分评分

NumberOfTimes90DaysLate变量的分数：

```
x7 = get_score(coe[3], woex7, p)
index = list(range(len(dfx7.index)))
dfx7.index = index
dfx7['score'] = Series(x7)
dfx7
```

	min	max	sum	total	rate	woe	goodattribute	badattribute	score
0	0	0	131008	137449	0.953139	0.375256	0.965794	0.663610	19.0
1	1	1	3396	5130	0.661988	-1.965152	0.025035	0.178652	-97.0
2	2	3	1041	2178	0.477961	-2.725530	0.007674	0.117144	-135.0
3	4	5	142	417	0.340528	-3.298263	0.001047	0.028333	-163.0
4	6	17	61	180	0.338889	-3.305569	0.000450	0.012260	-164.0

评分标准 - 部分评分

NumberOfTime60-89DaysPastDueNotWorse变量的分数：

```
x9 = get_score(coe[4], woex9, p)
index = list(range(len(dfx9.index)))
dfx9.index = index
dfx9['score'] = Series(x9)
dfx9
```

	min	max	sum	total	rate	woe	goodattribute	badattribute	score
0	0	0	130993	138127	0.948352	0.272953	0.965683	0.735009	8.0
1	1	1	3905	5647	0.691518	-1.830095	0.028788	0.179477	-56.0
2	2	3	688	1415	0.486219	-2.692457	0.005072	0.074902	-82.0
3	4	11	62	165	0.375758	-3.144914	0.000457	0.010612	-96.0

自动评分系统

为了方便使用，需要构建一个自动评分的系统。根据变量来计算分数，实现如下：

```
#根据变量计算分数
def compute_score(series, cut, score):
    list = []
    i = 0
    while i < len(series):
        value = series[i]
        j = len(cut) - 2
        m = len(cut) - 2
        while j >= 0:
            if value >= cut[j]:
                j = -1
            else:
                j -= 1
                m -= 1
        list.append(score[m])
        i += 1
    return list
```

自动评分系统

对预留的测试集的计算结果：

```
test1 = pd.read_csv('data/scoring/TestData.csv')
test1['BaseScore']=Series(np.zeros(len(test1)))+baseScore
test1['x1'] = Series(compute_score(test1['RevolvingUtilizationOfUnsecuredLines'], cutx1, x1))
test1['x2'] = Series(compute_score(test1['age'], cutx2, x2))
test1['x3'] = Series(compute_score(test1['NumberOfTime30-59DaysPastDueNotWorse'], cutx3, x3))
test1['x7'] = Series(compute_score(test1['NumberOfTimes90DaysLate'], cutx7, x7))
test1['x9'] = Series(compute_score(test1['NumberOfTime60-89DaysPastDueNotWorse'], cutx9, x9))
test1['Score'] = test1['x1'] + test1['x2'] + test1['x3'] + test1['x7'] +test1['x9']  + baseScore
test1.to_csv('data/scoring/ScoreData.csv', index=False)
test1.head()
```

90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents	BaseScore	x1	x2	x3	x7	x9	Score
1	1	0	2.0	532.0	-20.0	-4.0	-71.0	-135.0	-56.0	246.0
0	3	0	1.0	532.0	22.0	5.0	-27.0	-97.0	-56.0	379.0
0	2	0	3.0	532.0	5.0	-3.0	-27.0	-97.0	-56.0	354.0
0	2	0	0.0	532.0	24.0	11.0	-27.0	-97.0	-56.0	387.0
0	1	0	0.0	532.0	24.0	-5.0	-27.0	-97.0	-56.0	371.0

2019
Python开发者日
让开发者紧跟技术潮流

字段对照表

SeriousDlqin2yrs	Y
RevolvingUtilizationOfUnsecuredLines	X1
age	X2
NumberOfTime30-59DaysPastDueNotWorse	X3
DebtRatio	X4
MonthlyIncome	X5
NumberOfOpenCreditLinesAndLoans	X6
NumberOfTimes90DaysLate	X7
NumberRealEstateLoansOrLines	X8
NumberOfTime60-89DaysPastDueNotWorse	X9
NumberOfDependents	X10