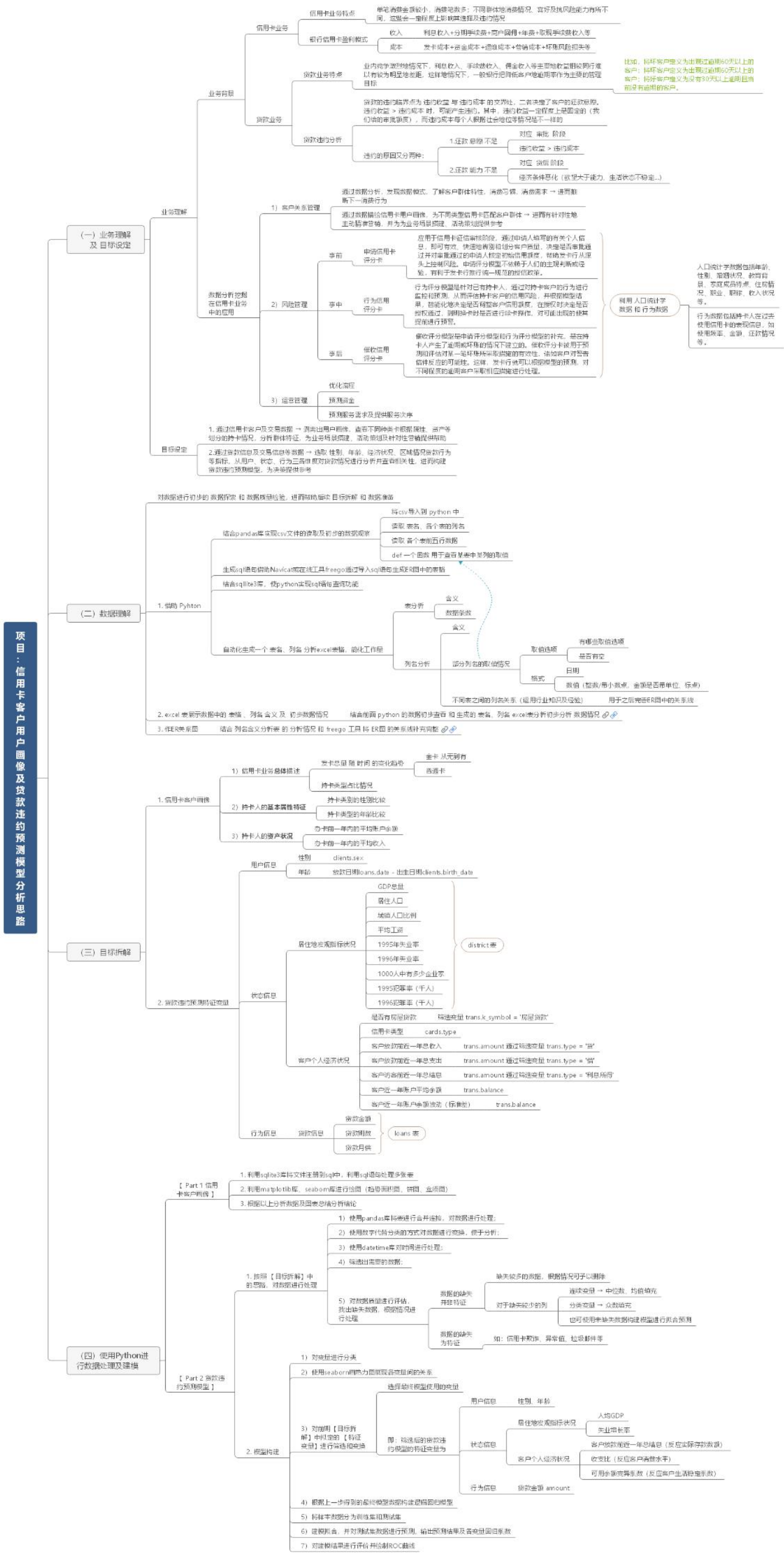
【以下为本项目分析思维导图：】

项目工具：Pycharm + Navicat 或 在线工具 freego



项目：信用卡客户用户画像及贷款违约预测模型分析思路
（一）业务理解及目标设定
业务理解
业务背景
信用卡业务
信用卡业务特点
单笔消费金额较小，消费笔数多；不同群体地消费情况、喜好及抗风险能力有所不同，这些会一定程度上影响其选择及违约情况
银行信用卡盈利模式
收入
利息收入+分期手续费+商户回佣+年费+取现手续费收入等
成本
发卡成本+资金成本+运维成本+营销成本+坏账风险损失等
贷款业务
贷款业务特点
业内竞争激烈地情况下，利息收入、手续费收入、佣金收入等主要地收益相较同行难以有较为明显地差距，这样地情况下，一般银行把降低客户地逾期率作为主要的管理目标
比如，将坏客户定义为出现过逾期60天以上的客户；将坏客户定义为出现过逾期60天以上的客户；将好客户定义为没有30天以上逾期且当前没有逾期的客户。
贷款违约分析
贷款的违约临界点为 违约收益 与 违约成本 的交界处，二者决定了客户的还款意愿。违约收益 > 违约成本 时，可能产生违约。其中，违约收益一定程度上是固定的（我们给的审批额度），而违约成本每个人根据社会地位等情况是不一样的
违约的原因又分两种：
1.还款 意愿 不足
对应 审批 阶段
违约收益 > 违约成本
2.还款 能力 不足
对应 贷后 阶段
经济条件恶化（欲望大于能力、生活状态不稳定...）
数据分析挖掘在信用卡业务中的应用
1）客户关系管理
通过数据分析，发现数据模式，了解客户群体特性，消费习惯、消费需求 → 进而推断下一消费行为
通过数据描绘信用卡用户画像，为不同类型信用卡匹配客户群体 → 进而有针对性地主动精准营销，并为为业务场景搭建、活动策划提供参考
2）风险管理
事前
申请信用卡评分卡
应用于信用卡征信审核阶段，通过申请人填写的有关个人信息，即可有效、快速地辨别和划分客户质量，决定是否审批通过并对审批通过的申请人核定初始信用额度，帮助发卡行从源头上控制风险。申请评分模型不依赖于人们的主观判断或经验，有利于发卡行推行统一规范的授信政策。
事中
行为信用评分卡
行为评分模型是针对已有持卡人，通过对持卡客户的行为进行监控和预测，从而评估持卡客户的信用风险，并根据模型结果，智能化地决定是否调整客户信用额度，在授权时决定是否授权通过，到期换卡时是否进行续卡操作，对可能出现的使其提前进行预警。
事后
催收信用评分卡
催收评分模型是申请评分模型和行为评分模型的补充，是在持卡人产生了逾期或坏账的情况下建立的。催收评分卡被用于预测和评估对某一笔坏账所采取措施的有效性，诸如客户对警告信件反应的可能性。这样，发卡行就可以根据模型的预测，对不同程度的逾期客户采取相应措施进行处理。
利用 人口统计学 数据 和 行为数据
人口统计学数据包括年龄、性别、婚姻状况、教育背景、家庭成员特点、住房情况、职业、职称、收入状况等。
行为数据包括持卡人在过去使用信用卡的表现信息，如使用频率、金额、还款情况等。
3）运营管理
优化流程
预测资金
预测服务需求及提供服务次序
目标设定
1. 通过信用卡客户及交易数据 → 刻画出用户画像，查看不同种类卡根据属性、资产等划分的持卡情况，分析群体特征，为业务场景搭建、活动策划及针对性营销提供帮助
2.通过贷款信息及交易信息等数据 → 选取 性别、年龄、经济状况、区域情况贷款行为等指标，从用户、状态、行为三各维度对贷款情况进行分析并查看相关性，进而构建贷款违约预测模型，为决策提供参考
（二）数据理解
对数据进行初步的 数据探索 和 数据质量检验，进而帮助后续 目标拆解 和 数据准备
1. 借助 Pyhton
结合pandas库实现csv文件的读取及初步的数据观察
将csv导入到 python 中
读取 表名、各个表的列名
读取 各个表前五行数据
def 一个函数 用于查看某表中某列的取值
生成sql语句借助Navicat或在线工具freego通过导入sql语句生成ER图中的表格
结合sqllite3库，使python实现sql语句查询功能
自动化生成一个 表名、列名 分析excel表格，简化工作量
表分析
含义
数据条数
列名分析
含义
部分列名的取值情况
取值选项
有哪些取值选项
是否有空
格式
日期
数值（整数/带小数点，金额是否带单位、标点）
不同表之间的列名关系（运用行业知识及经验）
用于之后完善ER图中的关系线
2. excel 表展示数据中的 表格、列名 含义 及 初步数据情况
结合前面 python 的数据初步查看 和 生成的 表名、列名 excel表分析初步分析 数据情况
3. 作ER关系图
结合 列名含义分析表 的 分析情况 和 freego 工具 将 ER图 的关系线补充完整
（三）目标拆解
1. 信用卡客户画像
1）信用卡业务总体描述
发卡总量 随 时间 的变化趋势
金卡 从无到有
普通卡
持卡类型占比情况
2）持卡人的基本属性特征
持卡类别的性别比较
持卡类型的年龄比较
3）持卡人的资产状况
办卡前一年内的平均账户余额
办卡前一年内的平均收入
2. 贷款违约预测特征变量
用户信息
性别
clients.sex
年龄
放款日期loans.date - 出生日期clients.birth_date
状态信息
居住地宏观指标状况
GDP总量
居住人口
城镇人口比例
平均工资
1995年失业率
1996年失业率
1000人中有多少企业家
1995犯罪率（千人）
1996犯罪率（千人）
district表
客户个人经济状况
是否有房屋贷款
筛选变量 trans.k_symbol = '房屋贷款'
信用卡类型
cards.type
客户放款前近一年总收入
trans.amount 通过筛选变量 trans.type = '贷'
客户放款前近一年总支出
trans.amount 通过筛选变量 trans.type = '借'
客户访客前近一年总结息
trans.amount 通过筛选变量 trans.type = '利息所得'
客户近一年账户平均余额
trans.balance
客户近一年账户余额波动（标准差）
trans.balance
行为信息
贷款信息
贷款金额
贷款期数
贷款月供
loans 表
（四）使用Python进行数据处理及建模
【 Part 1 信用卡客户画像 】
1. 利用sqlite3库将文件注册到sql中，利用sql语句处理多张表
2. 利用matplotlib库、seaborn库进行绘图（趋势面积图、饼图、盒须图）
3. 根据以上分析数据及图表总结分析结论
【 Part 2 贷款违约预测模型 】
1. 按照 【目标拆解】中的思路，对数据进行处理
1）使用pandas库将表进行合并连接，对数据进行处理；
2）使用数字代替分类的方式对数据进行变换，便于分析；
3）使用datetime库对时间进行处理；
4）筛选出需要的数据；
5）对数据质量进行评估，找出缺失数据，根据情况进行处理
数据的缺失并非特征
缺失较多的数据，根据情况可予以删除
对于缺失较少的列
连续变量 → 中位数、均值填充
分类变量 → 众数填充
也可使用未缺失数据构建模型进行拟合预测
数据的缺失为特征
如：信用卡欺诈、异常值、垃圾邮件等
2. 模型构建
1）对变量进行分类
2）使用seaborn画热力图展现各变量间的关系
3）对前期【目标拆解】中拟定的【特征变量】进行筛选和变换
选择最终模型使用的变量
即：筛选后的贷款违约模型的特征变量为
用户信息
性别、年龄
状态信息
居住地宏观指标状况
人均GDP
失业增长率
客户个人经济状况
客户放款前近一年总结息（反应实际存款数额）
收支比（反应客户消费水平）
可用余额变异系数（反应客户生活稳定系数）
行为信息
贷款金额 amount
4）根据上一步得到的最终模型数据构建逻辑回归模型
5）将样本数据分为训练集和测试集
6）建模拟合，并对测试集数据进行预测，输出预测结果及各变量回归系数
7）对建模结果进行评价并绘制ROC曲线

接下来进行代码演示

【信用卡客户用户画像及贷款违约预测模型】

--【数据理解 】--

(一) 导入数据 并进行初步的数据观察

1.改变工作目录到 数据集所在文件夹

In [ ]:

```
# -*- coding:utf-8 -*-

import  pandas as pd    # 用于 数据清洗 和 数据整理
import os   # 工作路径 及 文件夹操作
import datetime

pd.set_option('display.max_columns',None)

# 分别设置好 数据集路径 和 之后保存生成的文件 的路径

data_path = r'D:\Data_Sets_of_Analysis\Project_1_Analysis_of_Credit_Card_User_Portrait_and_Personal_Loan_Overdue\Data_Set_of_Credit_Card_User_Portrait_and_Personal_Loan_Overdue'
save_file_path = r'D:\Data_Sets_of_Analysis\Project_1_Analysis_of_Credit_Card_User_Portrait_and_Personal_Loan_Overdue'
os.chdir(data_path)
load_file = os.listdir()

print('\n' + '-'*25 + '打印目录列表' + '-'*25)    # 提示性分割线，方便阅读
print(load_file)     # 打印目录列表
print('\n'*2 + '='*100)   # 打印模块分割线，方便阅读
```

2.利用 pandas 读取 csv 文件

In [2]:

```
'''
【 #%% 】 打开 View → Scientific Model 后点击其左边的绿色剪头 可分块执行
'''
```

```
# # 因文件较多，故采用 for 循环结合 locals 函数动态生成多个变量

table_name = []
columns_name_express = []
columns_name = []
sql_statement = []
dataframe_table_columns_name = pd.DataFrame(columns=['表名','列名'])
   # 生成一个空的 DataFrame，其列名为'表名'和'列名'

for i in load_file:

    # 仅读取后缀为 csv 的表
    if i.split('.')[1] == 'csv':

        # locals()方法动态生成多个变量
        locals()[i.split('.')[0]] = pd.read_csv(i,encoding='gbk',erro
r_bad_lines=False,low_memory=False)

        # 将 每一个表的名字 都添加到 table_name 列表中
        table_name.append(i.split('.')[0])

        # 将 每一个表的名字 连同 每一列的名字都添加到 columns_name_ex
press 和 columns_name 列表中
        columns_name.append(list(locals()[i.split('.')[0]]))
        columns_name_express.append([i.split('.')[0] + '表的列名如下:
' + str(list(locals()[i.split('.')[0]])) + '\n' + '-'*125])
        '''
        i.split('.')[0] → 每一个表的名字
        str(list(locals()[i.split('.')[0]])) → 1.locals[]中填入表示
表名 的 i.split('.')[0]，表示选中该 locals 中的该表
                                                2.list(DataFrame)表示
DataFrame 的列名，故 list(locals()[i.split('.')[0]])
                                                    表示的是 取 表名 为
i.split('.')[0] 的 表 的 列名
                                                3.最后因为要连接字符串，
所以用 str()函数将以上进行转换
        '\n'  → 表示换行
        '-'*125 → 将字符串- 乘以 125 表示打印 - 125 次  作用为画一个
分割线，方便观察，不易串行
        最后将以上字段转换为列表，用 append 添加到 columns_name 中
        '''

        # 为方便用在线工具画 ER 图，在此顺便生成 sql 语句
        tn_str =''
```

```
        for j in range(len(list(locals()[i.split('.')[0]]))):
            tn_str += '\n' + list(locals()[i.split('.')[0]])[j] + ' '
 + 'TEXT' + ','
        sql_statement.append('CREATE TABLE' + ' ' + i.split('.')[0] +
 '\n' + '(' + tn_str.strip(',') + '\n' + ');')
```

```
        # 生成一个 表格、列名透视表 并输出 excel 文件 方便作 列名分析
        dataframe_table_columns_name = dataframe_table_columns_name.a
ppend(pd.DataFrame({'表名':i.split('.')[0],'列名':columns_name[-1]}))
```

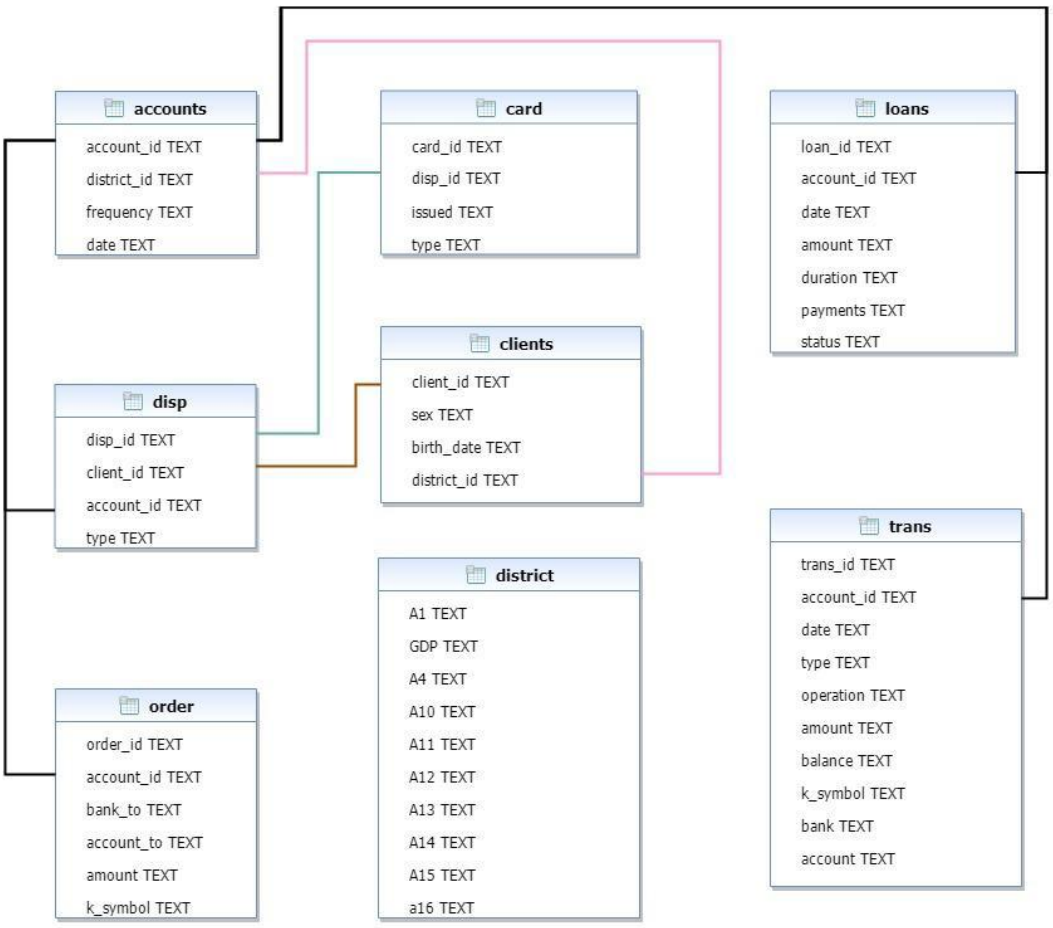
3.生成 sql 语句

In [ ]:

```
# 生成 sql 语句，复制后进入 freego 通过导入 MySQL DDL 自动生成 ER 表格，再
添加关系线即可生成 ER 图
```

```
# # 注释：列属性在此统一设置为【 TEXT 】仅为方便观察使用
```

```
print('在线 MySQL 语句绘制 ER 图链接：https://www.freedgo.com/erd-index.
html,复制以下生成的 sql 语句')
for i in sql_statement:
    print(i)
```



4.生成一个 表格、列名透视表 并输出 excel 文件 方便作 列名分析

In [ ]:

```
dataframe_table_columns_name['列名含义分析'] = 0  # 为作 pivot 图，增加全为 0 的数值列，之后再用 replace()，将 0 全部替换为空

warning = '▲ 注意：.to_excel 重复执行会覆盖原有表，故在正式编辑 excel 时应注意另存为新表，或创建一个副本'    # 增加一列提示，方便表格使用
pivot_table_column_name = pd.pivot_table(dataframe_table_columns_name, index=['表名','列名']).replace(0,'').append(pd.DataFrame(columns=[warning]))

print('\n'*2 + '='*100)  # 打印模块分割线，方便阅读

# 将以上得到的表格保存到路径为最开始已经设定好的【 save_file_path 】中，表格名称为【 列名含义分析表（请另存）.xls 】
try:
```

```
    pivot_table_column_name.to_excel(save_file_path +r'\列名含义分析表（请另存）.xls')
    print('表格已保存完毕')
except Exception:
    print('错误，‘列名含义分析表（请另存）.xls’未能成功保存，请检查报错原因(如：重复执行时，原先文件打开未关闭，则后一次执行代码时，无法覆盖原文件报错)')
```

表名	表分析	列名	列名含义分析	▲ 注意：.to_excel重复执行会覆盖原有表，故在正式编辑excel时应注意另存为新表，或创建一个副本
accounts 账户表 4500条	每条记录描述一个账户的静态信息	account_id	账户号（主键）	
		date	开户日期	2000-01-01 格式
		district_id	开户分行地区号	
		frequency	结算频度	分三种：交易之后马上、周结、月结
card 信用卡 892条	每条记录描述了一个账户上的信用卡信息	card_id	信用卡id（主键）	
		disp_id	账户权限号	
		issued	发卡日期	
		type	卡类型	普通卡、金卡、青年卡
clients 客户信息表 5369条	每条记录描述了一个客户的特征信息	birth_date	出生日期	2000-01-01 格式
		client_id	客户号（主键）	
		district_id	地区号（客户所属地区）	77个取值（数字）
		sex	性别	
disp 权限分配表 5369条	每条记录描述了客户和账户之间的关系，以及操作权限	account_id	账户号	
		client_id	顾客号	
		disp_id	权限设置号（主键）	
		type	权限类型	所有者、用户
district 人口地区统计表 77条数据（A15缺1条）	每条记录描述了一个地区的人口统计学信息	A1	同 district_id 地区号(主键)	
		A10	城镇人口比例	
		A11	平均工资	
		A12	1995年失业率	
		A13	1996年失业率	
		A14	1000人中有多少企业家	
		A15	1995犯罪率（千人）	
		A4	居住人口	
		GDP	GDP总量	
		a16	1996犯罪率（千人）	
loans 贷款表 682条数据	每条记录代表某个账户上的一条贷款信息	account_id	账户号	
		amount	贷款金额	类型：整数，无逗号分隔
		date	发放贷款日期	2000-01-01 格式
		duration	贷款期限	月份数：12、24、36、48、60
		loan_id	贷款号（主键）	
		payments	每月归还额	
		status	还款状态	A代表合同终止，没问题；B代表合同终止，贷款没有支付；C代表合同处于执行期，至今正常；D代表合同处于执行期，欠债状态。
order 支付命令表 6471条数据（其中k_symbol有NAN值）	每条记录代表描述了一个支付命令	account_id	发起订单的账户号	
		account_to	收款客户号	
		amount	金额	保留1位小数，无逗号分隔
		bank_to	收款银行	每家银行用两个字母来代表
		k_symbol	支付方式	保险支付、日常支出、租赁支付、贷款偿还、NAN
		order_id	订单号（主键）	
trans 交易表 1056320条数据（operation、k_symbol、bank、account存在NAN）	每条记录代表每个账户上的一条记录	account	对方账户号	
		account_id	发起订单的账户号	
		amount	金额	带单位$和逗号分隔
		balance	账户余额	带单位$和逗号分隔
		bank	对方银行	每家银行用两个字母来代表
		date	交易日期	
		k_symbol	交易特征	保险费、养老金、利息所得、房屋贷款、支付贷款、支票
		operation	交易类型	从他行收款、信用卡借方、信贷资金、汇款到另一家银行、现金
		trans_id	交易序号（主键）	
		type	借贷类型	借、贷

5. 列出刚刚读取的所有表名

In [ ]:

```
print('\n'*2 + '='*100)   # 打印模块分割线，方便阅读
print('所有表格的名称如下：' + str(table_name))
```

6.将每个表的列名（column）分别打印出来

In [ ]:

```
print('\n'*2 + '='*100)  # 打印模块分割线，方便阅读
for i in range(len(columns_name_express)):
    print(columns_name_express[i][0])
```

7.将每个表格的前 5 行打印出来，初步观察数据

In [ ]:

```
print('\n'*2 + '='*100)  # 打印模块分割线，方便阅读
for tn in load_file:
    col_num = len(list(locals()[tn.split('.')[0]]))
    pd.set_option('display.max_columns', col_num)    # 设置显示最大列数为 表的列数
    print('\n' + '-'*25 + '以下为%s 表'% tn.split('.')[0] + '-'*25)
    print(locals()[tn.split('.')[0]].head())     # 默认打印前五行
    print(locals()[tn.split('.')[0]].count())    # 打印各列数据数
```

8.创建一个函数用于 查询可选取值、空值

In [ ]:

```
# 创建一个函数用于 有选择性地 查询 某个表 中 某个列 的 可选取值（通过该函数还能查看是否有空值）
# # 1）因 def 中无法直接引用 local，故将所有的表格保存到一个字典中，键为 表名，值 为 表

dict_of_tables = dict()
for i in load_file:
    dict_of_tables[i.split('.')[0]] = locals()[i.split('.')[0]]     # 使用 for 循环将每一个表都增加到 dict_of_tables 中

# # 2）进行函数定义

def check_distinct_column_value():

    while True:
        t_name = input('请输入 表名 : ')
        c_name = input('请输入 列名 : ')

        try:
            import sqlite3
            con = sqlite3.connect(':memory:')
            dict_of_tables[t_name].to_sql(t_name,con)
```

```
            c_value = pd.read_sql_query('select distinct %s from %s'
 % (c_name, t_name), con)
            print(c_value)
            break
        except Exception:
            print('发生错误,请检查所输入的 表名 和 列名 及其对应关系
是否正确，并重新输入')

# # 3）选择是否调用该查询函数

while True:
    answer = input('是否调用“列名取值查看”函数？回答 Y 或 N：')
    if answer == 'Y':
        check_distinct_column_value()    #函数调用
    elif answer == 'N':
        break
    else:
        print('输入有误，请重新输入回答，仅可回答 Y 或 N')
```

-- 【 Part 1 信用卡用户画像 】 --

(二) 数据清洗及绘图 之 信用卡用户画像

1.将导入的文件注册到 sql 里

In [ ]:

```
'''
通过 信用卡客户画像 的 目标拆解 和 ER 图，需要用到 card、clients、disp
(连接关系用)、trans
根据之前的数据查看，这三个表中均无缺失的情况
'''

import sqlite3
con = sqlite3.connect(':memory:')
locals()['card'].to_sql('card',con)
locals()['clients'].to_sql('clients',con)
locals()['disp'].to_sql('disp',con)
locals()['trans'].to_sql('trans',con)
```

2.写 sql 语句通过 disp 表，将 card 和 client 两个表连接起来

In [ ]:

```
sql_card_client = '''
SELECT cd.*,ctdp.birth_date,ctdp.district_id,ctdp.sex
```

```
FROM card AS cd JOIN (
SELECT ct.birth_date,ct.district_id,ct.sex,dp.disp_id FROM clients AS
 ct
JOIN disp AS dp ON ct.client_id = dp.client_id WHERE dp.type == "所有
者")AS ctdp
ON cd.disp_id = ctdp.disp_id
'''

card_client = pd.read_sql_query(sql_card_client,con)
print(card_client)
```

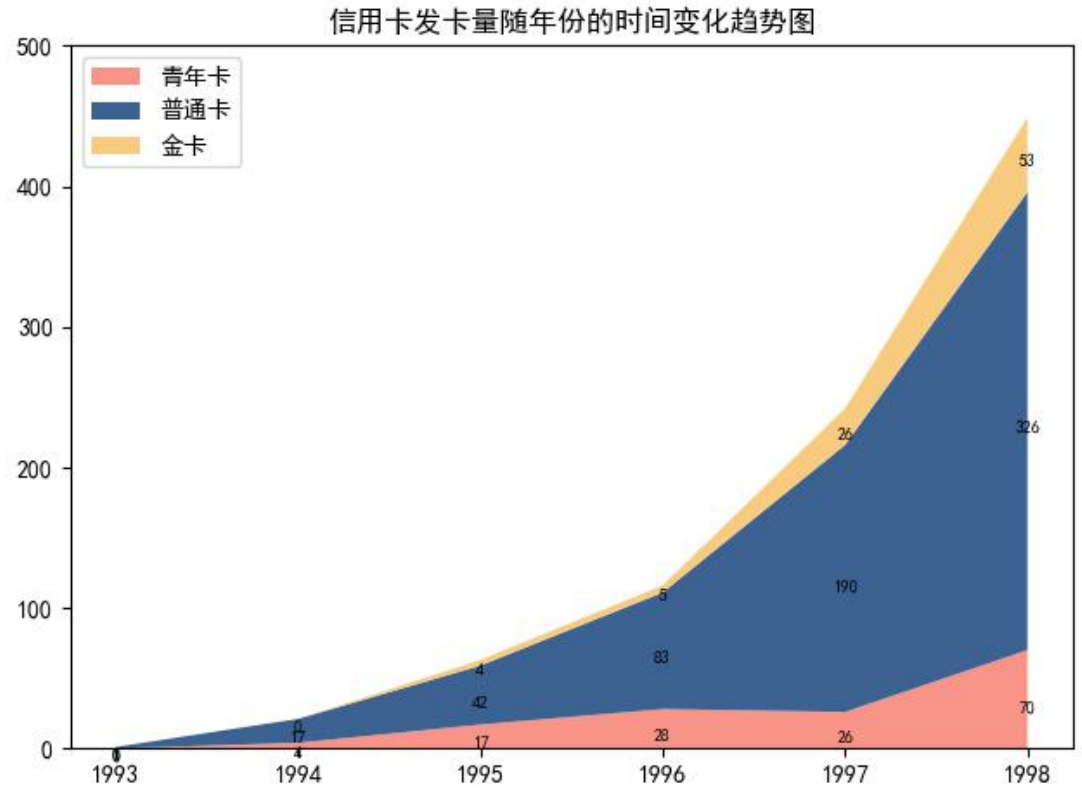
3.作图 → 信用卡业务总体描述

1) 发卡总量 随 时间 的变化趋势

In [ ]:

```
# # 因作图可能涉及到中文，在此先设置字体
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei']    # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False      # 解决保存图像负号'-'
显示为方块的问题



# # # 探究信用卡总量 不同年份 的总量变化
import datetime
import matplotlib.pyplot as plt

# # # 新增一列‘issue_year’，提取出年份
card_client['issue_year'] = pd.to_datetime(card_client['issued']).map
(lambda x:x.year)

# # # 创建一个交叉表，显示不同类别卡，不同年份的发行数量
cross_tab = pd.crosstab(card_client.issue_year,card_client.type)
print(cross_tab)

# # # 画趋势面积图
labels = ['青年卡','普通卡','金卡']

y1 = cross_tab.loc[:,'青年卡'].astype('int')  # 将'青年卡'这列的每行
的字符都转换成 int
y2 = cross_tab.loc[:,'普通卡'].astype('int')
y3 = cross_tab.loc[:,'金卡'].astype('int')
x = cross_tab.index  # 将 index 列，也就是 issue_year 转换成 int
```

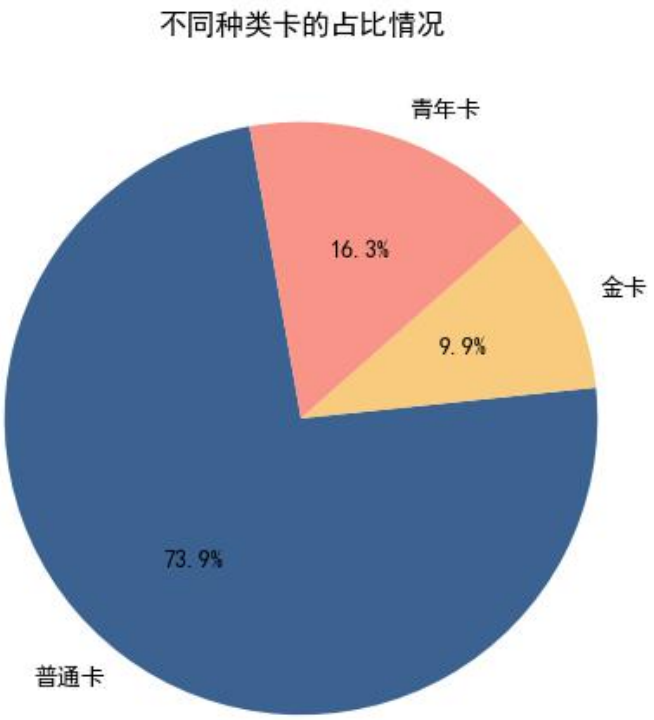
```
plt.stackplot(x,y1,y2,y3,labels=labels,colors=['#f89588','#3b6291','#f8cb7f'])
plt.title('信用卡发卡量随年份的时间变化趋势图')
plt.legend(loc = 'upper left')  # 设置图例位置在左上角
plt.ylim(0,500)      # 设置 y 轴刻度最大值

# # # 设置 数字标签，表明对应的发卡量
for a,b in zip(x,y1):
    plt.text(a,0.5*b-10,b,ha='center', va= 'bottom',fontsize=7)

for a,b in zip(x,y2):
    plt.text(a,0.5*b+list(y1)[list(x).index(a)]-10,b,ha='center', va= 'bottom',fontsize=7)

for a,b in zip(x,y3):
    plt.text(a,0.5*b+list(y1)[list(x).index(a)]+list(y2)[list(x).index(a)]-10,b,ha='center', va= 'bottom',fontsize=7)

plt.show()
'''
设置 数字标签，格式示例：
    for a,b in zip(x,y):
        plt.text(a, b+0.05, '%.0f' % b, ha='center', va= 'bottom',fontsize=7)
    a：数字标签的横轴坐标
    b+0.05：数字标签的纵轴坐标
    '%.0f' % b：格式化的数字标签(保留一位小数)
    ha='center'：horizontalalignment（水平对齐）
    va= 'bottom'：verticalalignment（垂直对齐）的方式
    fontsize：文字大小
本案例中的实例说明：
    因本例为堆叠图，故对数字标签的纵轴坐标相应做了值叠加
    y1 对应的数字标签的纵坐标放中间，并下移 10（下移 10 是为了看起来更美观）
    y2 对应的数字标签的纵坐标，要使其同样放中间，则需用 y2 对应的 b 值乘以 0.5 再加上 y1 对应的 b 值，再下移 10
    y3 对应的数字标签的纵坐标，要使其同样放中间，则需用 y3 对应的 b 值乘以 0.5 再加上 y1 和 y2 对应的 b 值，再下移 10

'''
```



2) 不同类型卡总发行量占比情况（饼图）

In [ ]:

```
cross_tab1 = cross_tab
cross_tab1.loc['Sum'] = 0
for i in list(cross_tab):
    cross_tab1.loc['Sum'][i] = sum(cross_tab1[i])
print(cross_tab1)
plt.pie(cross_tab1.loc['Sum'],labels=list(cross_tab1),autopct='%1.1
f%%',startangle=100,colors=['#3b6291','#f8cb7f','#f89588'])
plt.title('不同种类卡的占比情况')
plt.show()
```



补充 stack2dim 包的代码

- ▲ ▲ 【 这里补充下接下来所引用的 stack2dim 包的代码：

In [ ]:

```
# 如遇中文显示问题可加入以下代码
from pylab import mpl

mpl.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题


def stack2dim(raw, i, j, rotation=0, location='upper right'):
    '''
    此函数是为了画两个维度标准化的堆积柱状图
    raw 为 pandas 的 DataFrame 数据框
    i、j 为两个分类变量的变量名称，要求带引号，比如"school"
    rotation：水平标签旋转角度，默认水平方向，如标签过长，可设置一定角度，比如设置 rotation = 40
    location：分类标签的位置，如果被主体图形挡住，可更改为'upper left'
    '''
```

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

    data_raw = pd.crosstab(raw[i], raw[j])
    data = data_raw.div(data_raw.sum(1), axis=0)  # 交叉表转换成比率，为得到标准化堆积柱状图

    # 计算 x 坐标，及 bar 宽度
    createVar = locals()
    x = [0]  # 每个 bar 的中心 x 轴坐标
    width = []  # bar 的宽度
    k = 0
    for n in range(len(data)):
        # 根据频数计算每一列 bar 的宽度
        createVar['width' + str(n)] = list(data_raw.sum(axis=1))[n] / sum(data_raw.sum(axis=1))
        width.append(createVar['width' + str(n)])
        if n == 0:
            continue
        else:
            k += createVar['width' + str(n - 1)] / 2 + createVar['width' + str(n)] / 2 + 0.05
            x.append(k)

    # 以下是通过频率交叉表矩阵生成一串对应堆积图每一块位置数据的数组，再把数组转化为矩阵
    y_mat = []
    n = 0
    y_level = len(data.columns)
    for p in range(data.shape[0]):
        for q in range(data.shape[1]):
            n += 1
            y_mat.append(data.iloc[p, q])
            if n == data.shape[0] * data.shape[1]:
                break
            elif n % y_level != 0:
                y_mat.extend([0] * (len(data) - 1))
            elif n % y_level == 0:
                y_mat.extend([0] * len(data))

    y_mat = np.array(y_mat).reshape(-1, len(data))
```

```
    y_mat = pd.DataFrame(y_mat)  # bar 图中的 y 变量矩阵，每一行是一个 y 变量

    # 通过 x，y_mat 中的每一行 y，依次绘制每一块堆积图中的每一块图

    from matplotlib import cm
    cm_subsection = [level for level in range(y_level)]
    colors = [cm.Pastel1(color) for color in cm_subsection]

    bottom = [0] * y_mat.shape[1]
    createVar = locals()
    for row in range(len(y_mat)):
        createVar['a' + str(row)] = y_mat.iloc[row, :]
        color = colors[row % y_level]

        if row % y_level == 0:
            bottom = bottom = [0] * y_mat.shape[1]
            if math.floor(row / y_level) == 0:
                label = data.columns.name + ': ' + str(data.columns[row])
                plt.bar(x, createVar['a' + str(row)],
                        width=width[math.floor(row / y_level)], label=label, color=color)
            else:
                plt.bar(x, createVar['a' + str(row)],
                        width=width[math.floor(row / y_level)], color=color)
        else:
            if math.floor(row / y_level) == 0:
                label = data.columns.name + ': ' + str(data.columns[row])
                plt.bar(x, createVar['a' + str(row)], bottom=bottom,
                        width=width[math.floor(row / y_level)], label=label, color=color)
            else:
                plt.bar(x, createVar['a' + str(row)], bottom=bottom,
                        width=width[math.floor(row / y_level)], color=color)

        bottom += createVar['a' + str(row)]

    plt.title(j + ' vs ' + i)
    group_labels = [str(name) for name in data.index]
    plt.xticks(x, group_labels, rotation=rotation)
```

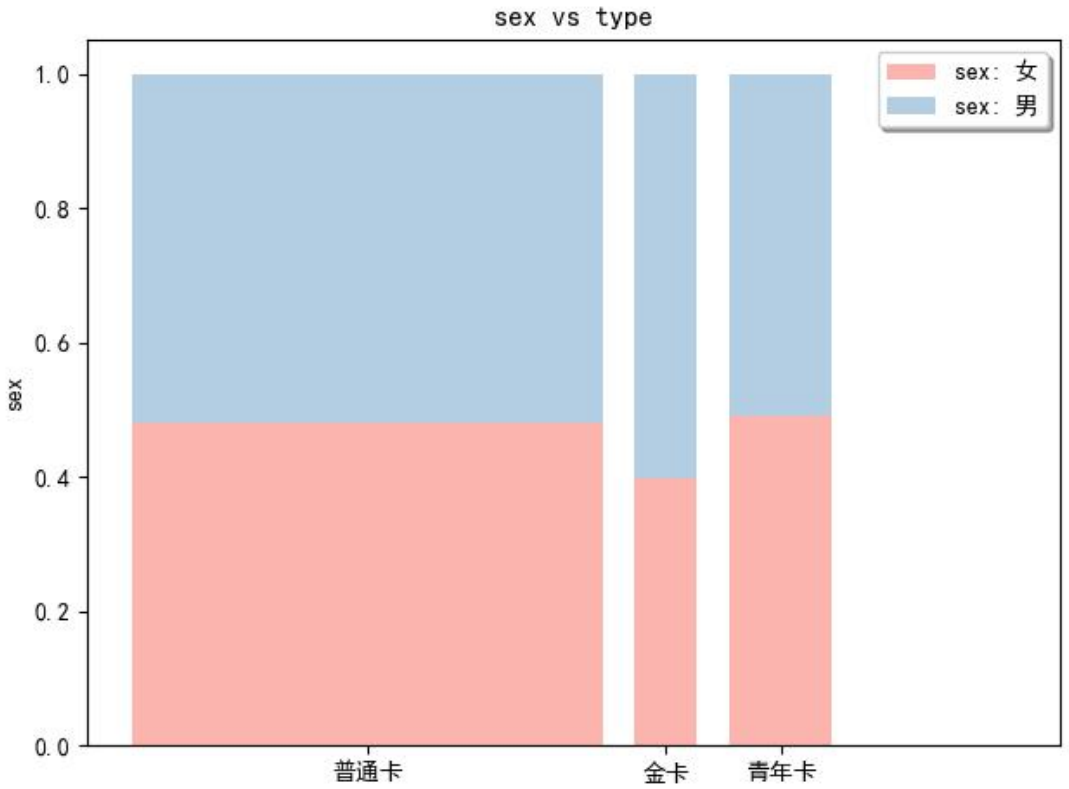
```
    plt.ylabel(j)
    plt.legend(shadow=True, loc=location)
    plt.show()
```

- 补充结束 】 ▲ ▲
  ## 4.基本属性特征

1) 不同卡类型的性别比较堆积图

In [ ]:

```
from stack2dim import *
stack2dim(card_client,'type','sex')
```



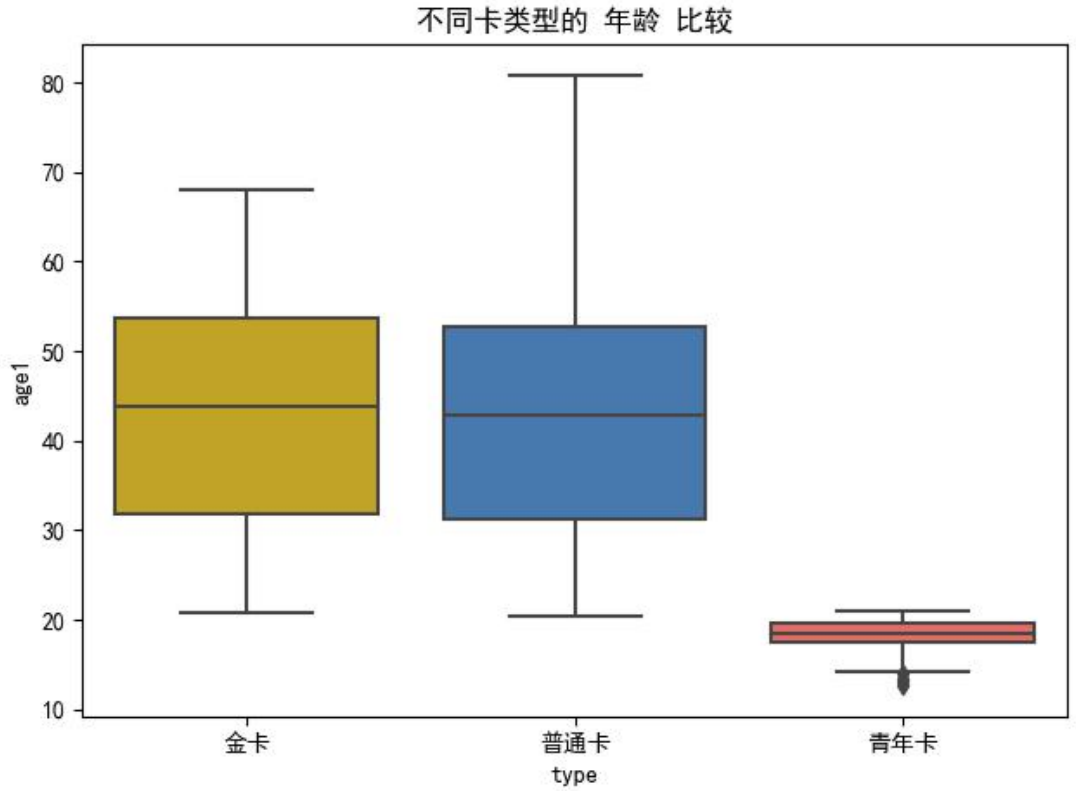
2) 不同卡类型的年龄比较

In [ ]:

```
import seaborn as sns
import time
card_client['age']=(pd.to_datetime(card_client['issued'])-pd.to_datet
ime(card_client['birth_date']))
```

```
card_client['age1']=card_client['age'].map(lambda x:x.days/365)
ax_age = sns.boxplot(x = 'type', y = 'age1', data = card_client,palet
te=sns.xkcd_palette(['gold','windows blue','coral']))
ax_age.set_title('不同卡类型的 年龄 比较')
```

```
plt.show()
```

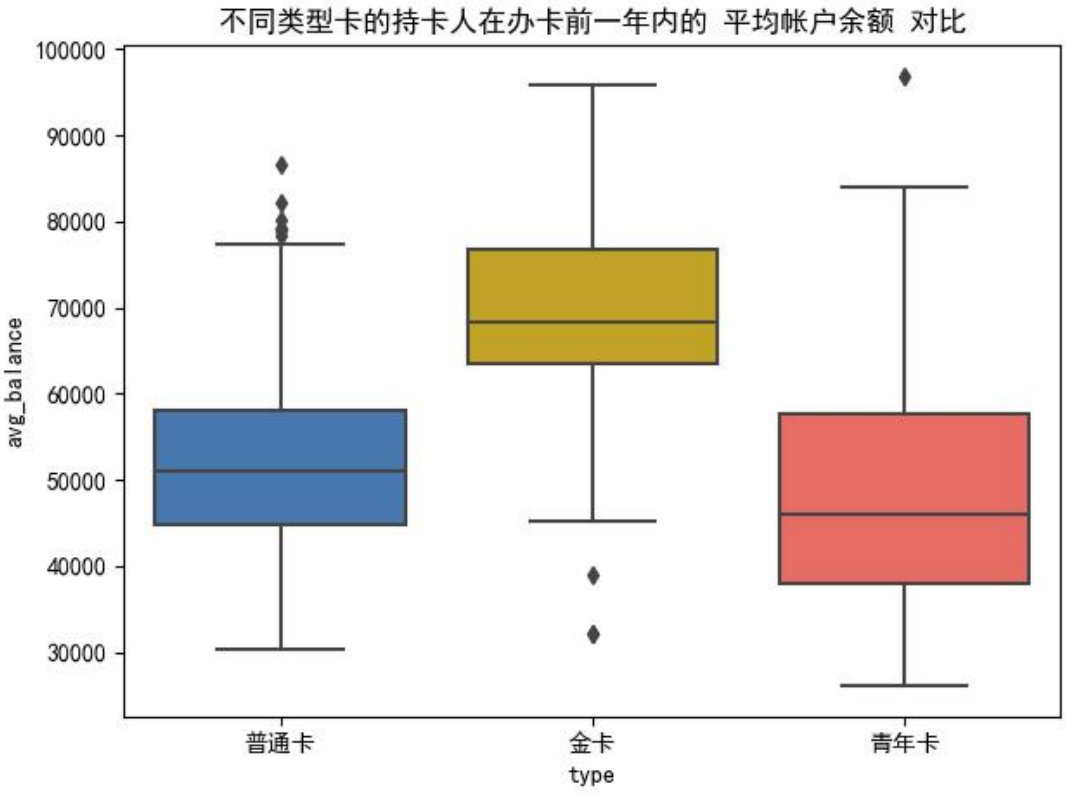


3) 不同类型卡的持卡人在办卡前一年内的平均帐户余额对比

In [ ]:

```
sql_card_client_trans = '''
select a.card_id,a.issued,a.type,c.type as t_type,c.amount,c.balance,
c.date as t_date
  from card as a
  left join disp as b on a.disp_id=b.disp_id
  left join trans as c on b.account_id=c.account_id
  where b.type="所有者"
  order by a.card_id,c.date
'''
card_client_trans = pd.read_sql_query(sql_card_client_trans,con)
# print(card_client_trans.head())

# # 标准化日期
card_client_trans['issued']=pd.to_datetime(card_client_trans['issued
'])
card_client_trans['t_date']=pd.to_datetime(card_client_trans['t_date
'])
print(card_client_trans)

# # 对帐户余额进行清洗：去掉金额单位和逗号分隔，便于计算
```

```
card_client_trans['balance_1'] = card_client_trans['balance'].map(lam
bda x:int(x.strip('$').replace(',','')))
print(card_client_trans)

# # 筛选出开卡前一年的数据
card_client_trans_1 = card_client_trans[card_client_trans.issued > ca
rd_client_trans.t_date][
    card_client_trans.t_date >= card_client_trans.issued-datetime.tim
edelta(days=365)
]
print(card_client_trans_1)

# #  分组计算余额均值
card_client_trans_1['avg_balance'] = card_client_trans_1.groupby('car
d_id')['balance_1'].mean()
card_client_trans_2 = card_client_trans_1.groupby(['type','card_id'])
['balance_1'].agg([('avg_balance','mean')])
# print(card_client_trans_1)
# print(card_client_trans_2)
card_client_trans_2.to_sql('card_client_trans_2',con)
card_client_trans_3 = card_client_trans_2.reset_index()
card_client_trans_3 = pd.read_sql('select * from card_client_trans_2',
 con)

colors = ['windows blue','gold','coral']
ax_balance = sns.boxplot(x='type',y='avg_balance',data=card_client_tr
ans_3,palette=sns.xkcd_palette(colors))
ax_balance.set_title('不同类型卡的持卡人在办卡前一年内的 平均帐户余额
 对比')
plt.show()
```



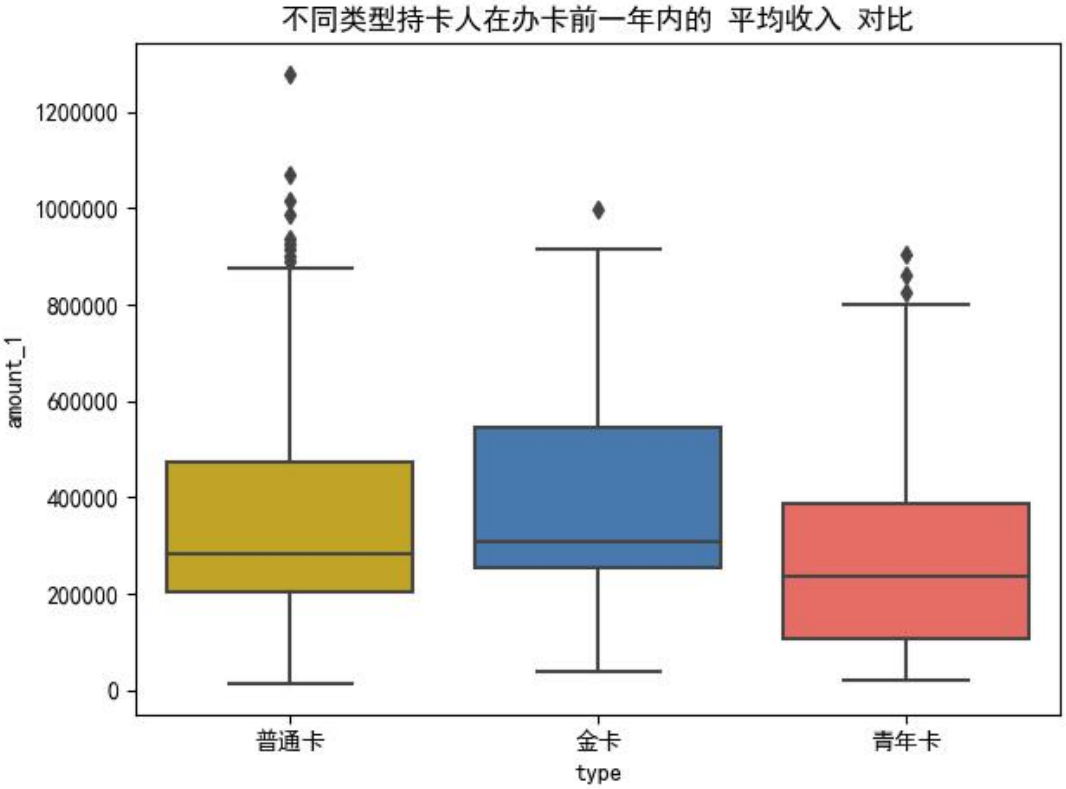
4) 不同类型持卡人在办卡前一年内的平均收入和平均支出对比

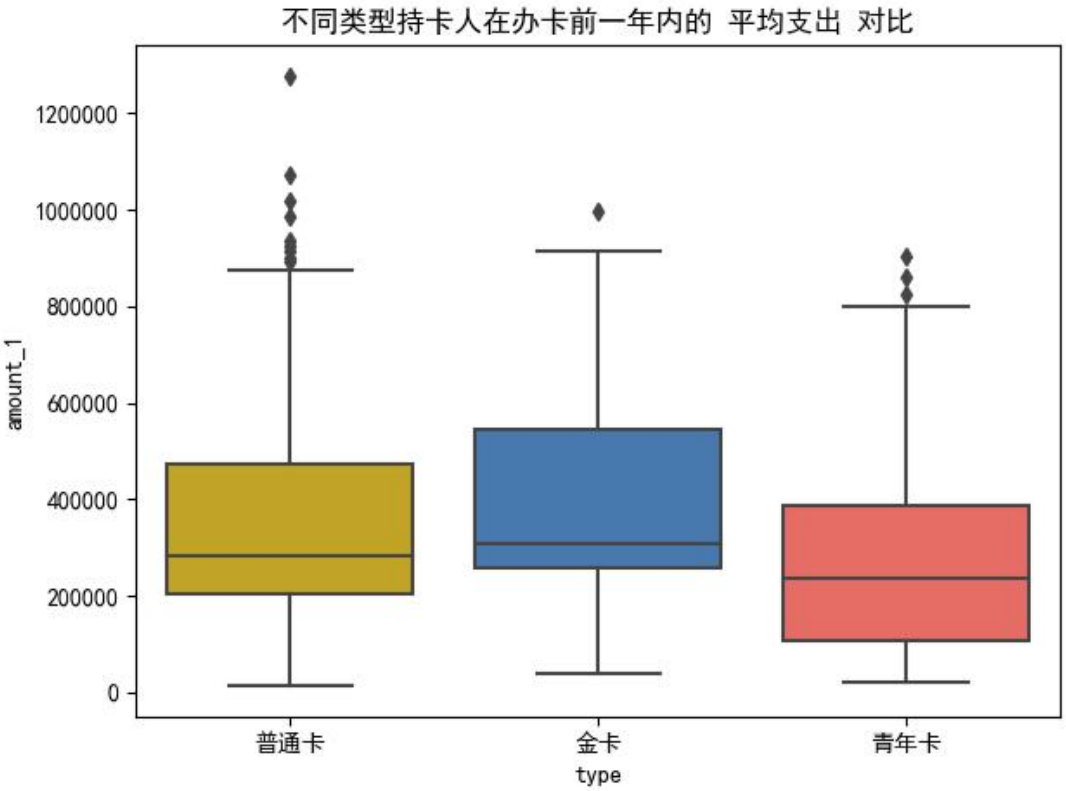
In [ ]:

```
# #

# # # 先将 借、贷 转换成 更易理解的 out、income
type_dict = {'借':'out','贷':'income'}
card_client_trans_1['type_1'] = card_client_trans_1.t_type.map(type_d
ict)

# # # 将 amount 金额列的 金额单位 和 逗号分隔 去掉
card_client_trans_1['amount_1'] = card_client_trans_1['amount'].apply
(lambda x:int(x.strip('$').replace(',','')))

card_client_trans_4 = card_client_trans_1.groupby(['type','card_id','
type_1'])[['amount_1']].sum()
card_client_trans_4.head()
card_client_trans_4.to_sql('card_client_trans_4',con)

card_client_trans_5 = card_client_trans_4.reset_index()
card_client_trans_5.to_sql('card_client_trans_5',con)
card_client_trans_6 = pd.read_sql_query('select * from card_client_tr
ans_5 where type_1 = "income"',con)
```

```
ax_amount_income = sns.boxplot(x='type',y='amount_1',data=card_client
_trans_6,palette=sns.xkcd_palette(['gold','windows blue','coral']))
ax_amount_income.set_title('不同类型持卡人在办卡前一年内的 平均收入
对比')
plt.show()
```

```
card_client_trans_7 = pd.read_sql_query('select * from card_client_tr
ans_5 where type_1 = "out"',con)
ax_amount_out = sns.boxplot(x='type',y='amount_1',data=card_client_tr
ans_6,palette=sns.xkcd_palette(['gold','windows blue','coral']))
ax_amount_out.set_title('不同类型持卡人在办卡前一年内的 平均支出 对比
')
plt.show()
```





In [ ]:

```
'''
【信用卡客户画像总结分析】：
（一）总体趋势（近六年）：
    1.逐年发卡量：金卡、普通卡均呈逐年上升趋势，青年卡在1997年的发行量同比降低，但总体为上升趋势；
                逐年发行量占比排名为：普通卡 > 青年卡 > 金卡
    2.总发卡量：总体发卡量占比排名为：普通卡 > 青年卡 > 金卡,其中普通卡占比接近总发卡量的3/4
（二）基本属性特征
    1.不同卡类型的 性别 比较：
        普通卡和青年卡 男女性比例较为均衡，基本为1：1；金卡的男性持有者比例相较女性持有者明显更多
    2.不同卡类型的 年龄 比较：
        普通卡和金卡的持有者年龄主要集中在30~60岁之间;而青年卡则普遍集中在25岁以内，卡类型设计与目标对象相符
    3.不同类型卡的持卡人在办卡前一年内的 平均帐户余额 对比：
        金卡持有者的办卡前一年的 平均余额 是要显著高于 普通卡 和 青年卡 的，卡类型设计与目标对象相符
    4.不同类型持卡人在办卡前一年内的 平均收入和平均支出 对比：
        三种类型的 平均收入、平均支出 排序均符合：金卡 > 普通卡 > 青年卡，金卡的持有人群为收入较高的群体，
        同样其支出情况也相应高于普通持卡人群，而青年卡，由于其持卡人群多为年龄层较小的人群，收入支出均较低，
```

```
        卡类型设计与目标对象情况相符
'''
```

-- 【 Part 2 贷款违约预测模型 】 --

(三) 数据清洗 之 贷款违约预测模型

'''
1.时间点的选择：选取放款时间点之前的一年，观察用户是否有逾期行为
2.loans 表中的贷款状态说明：A 代表合同终止，没问题；B 代表合同终止，贷款没有支付；
C 代表合同处于执行期，至今正常；D 代表合同处于执行期，欠债状态。
A 贷款正常还款，B、D 有问题，C 待定
'''

1.用户信息 → 将 性别、年龄、信用卡信息 添加到 loans 表中

In [ ]:

```
# # 1)在 loans 表中增加一列，用数字来代替贷款状态，方便后续分析

loan_status = {'B':1,'D':1,'A':0,'C':2}
locals()['loans']['loan_status'] = locals()['loans']['status'].map(lo
an_status)
print(locals()['loans'])

# # 2) 进行列添加
'''
通过 disp 表连接 loans 表和 clients 表
'''
data_1 = pd.merge(locals()['loans'],locals()['disp'],on='account_id',
how='left')
data_2 = pd.merge(data_1,locals()['clients'],on='client_id',how='left
')
data_3 = data_2[data_2.type == '所有者']


# # 增加年龄列
# # # 首先将字符串转换为 datetime 时间序列，再计算出年龄
data_3['age_temp'] = (pd.to_datetime(data_3['date'])-pd.to_datetime(d
ata_3['birth_date']))
data_3['age'] = data_3['age_temp'].map(lambda x:round(x.days/365,0))


# # 连接 card 表
```

```
data_4 = pd.merge(data_3,locals()['card'],on='disp_id',how='left')
```

2.状态信息 → 将 相关列添加

In [ ]:

```
# # 1)添加地区状态信息
data_5 = pd.merge(data_4,locals()['district'],left_on='district_id',r
ight_on='A1',how='left')

# # # 将需要的列筛选出来
trans_clients_district = data_5[['account_id','amount','duration','pa
yments','loan_status','type_y','sex',
                                 'age', 'district_id','GDP','A4','A10
','A11','A12','A13','A14','A15','a16']]


# # 2) 客户个人经济状况信息
trans_loans = pd.merge(locals()['trans'],locals()['loans'],on='accoun
t_id')
print(trans_loans.head())
print(list(trans_loans))
'''
合并后出现的列名后带_x 和_y 是因为合并的两张表中有有相同的列名，为示区
分加上的后缀；
_x 表示的是合并前'trans'表中的列，_y 表示的是合并前'loans'表中的列
'''
```

3.筛选数据

In [ ]:

```
# # 筛选出贷款前一年的交易数据
trans_loans['date_x'] = pd.to_datetime(trans_loans['date_x'])
trans_loans['date_y'] = pd.to_datetime(trans_loans['date_y'])

# # 将 amount_x 和 balance 由字符串类型，去掉$符号和逗号分隔，转化为数
值类型
trans_loans['amount_x'] = trans_loans['amount_x'].apply(lambda x:int
(x.strip('$').replace(',','')))
trans_loans['balance'] = trans_loans['balance'].apply(lambda x:int(x.
strip('$').replace(',','')))

# # 筛选出放款日期按 1 年内至前一天的交易记录
trans_loans = trans_loans[(trans_loans['date_x']<trans_loans['date_y
'])&((trans_loans['date_x']+datetime.timedelta(days=365))>trans_loans
['date_y'])]
```

```
# # 筛选用户前一年内结息总额
trans_loans_1 = trans_loans[trans_loans['k_symbol']=='利息所得']
trans_loans_1 = pd.DataFrame(trans_loans_1.groupby('account_id')['amo
unt_x'].sum())
trans_loans_1.columns = ['interest']

# # 筛选在本行是否由养老金和房屋贷款
trans_loans_2 = trans_loans[(trans_loans['k_symbol']=='养老金')|(tran
s_loans['k_symbol']=='房屋贷款')]

# # 标记是否在本行有房屋贷款
trans_loans_2 = pd.DataFrame(trans_loans_2.groupby('account_id')['k_s
ymbol'].count())
trans_loans_2['house_loan'] = '1'
del trans_loans_2['k_symbol']

# # 筛选客户一年内收入和支出（总和）
print(trans_loans_2.head())
trans_loans_3 = pd.DataFrame(trans_loans.pivot_table(values='amount_x
',index='account_id',columns='type'))
trans_loans_3.columns = ['out','income']

# # 筛选客户一年内余额的均值和标准差
trans_loans_4 = pd.DataFrame(trans_loans.groupby('account_id')['balan
ce'].agg(['mean','std']))
trans_loans_4.columns = ['balance_mean','balance_std']

# # 合并数据
data_temp = pd.merge(trans_loans_1,trans_loans_2,how='left',left_inde
x=True,right_index=True)
data_temp = pd.merge(data_temp,trans_loans_3,left_index=True,right_in
dex=True)
data_temp = pd.merge(data_temp,trans_loans_4,left_index=True,right_in
dex=True)
print(len(data_temp)) # 查看数据条数是否与贷款表条数一致
data_model = pd.merge(trans_clients_district,data_temp,left_on='accou
nt_id',right_index=True)
print(data_model)
```

(四) 模型构建

1. 数据清洗 及 变量选择

1) 查看数据缺失情况

In [ ]:

```
print(data_model.isnull().sum()/len(data_model))
'''
分析缺失情况、原因及处理：
    总共有 列存在缺失信息
    type_y：缺失信息接近 75%，信息缺失过多，删除列；
    A12 和 A15:A12 1995 年失业率 和 A15 1995 犯罪率(千人) 有极小部分数据缺失，因为是连续数据，使用中位数填充；
    house_loan：是否有房屋贷款，缺失值为没有房屋贷款，填充字符 '0'
'''
```

2) 数据处理

In [ ]:

```
del data_model['type_y']
data_model['A12'].fillna(data_model['A12'].median(),inplace=True)
data_model['A15'].fillna(data_model['A15'].median(),inplace=True)
data_model['house_loan'].fillna('0',inplace=True)
```

3) 对变量进行分类

In [ ]:

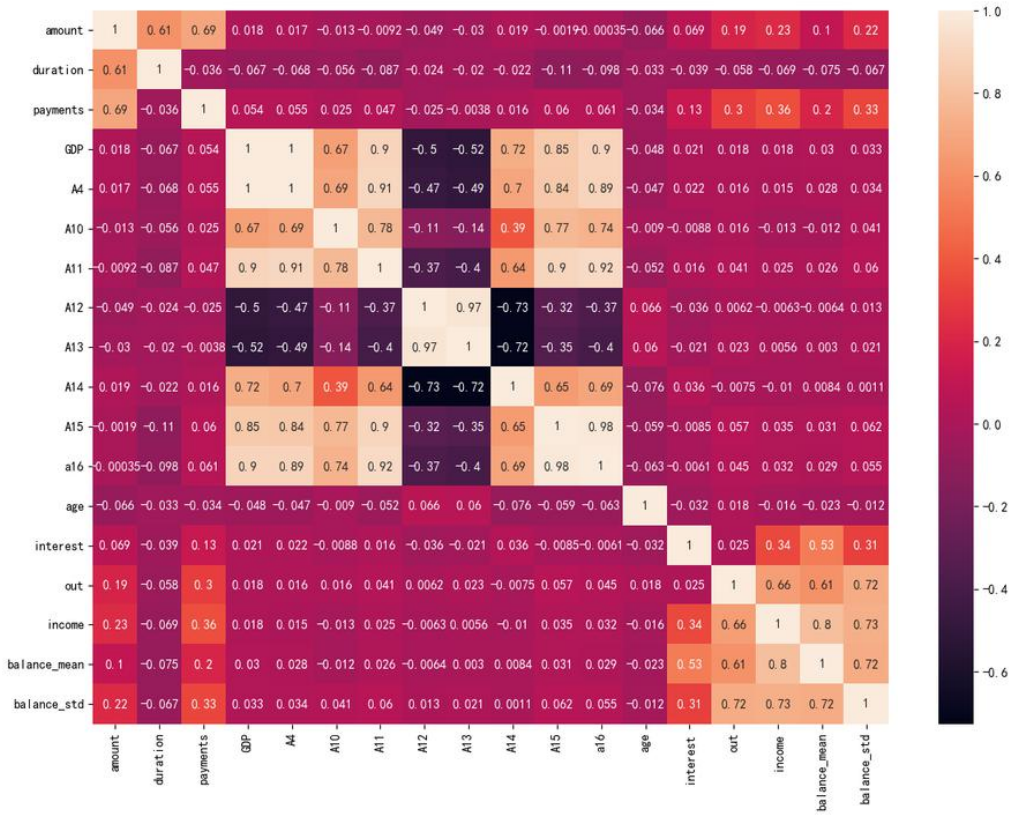
```
# # # 〈1〉因变量
y = 'loan_status'

# # # 〈2〉连续变量
var_c =  ['amount', 'duration', 'payments', 'GDP',
       'A4', 'A10', 'A11', 'A12', 'A13', 'A14', 'A15', 'a16', 'age',
       'interest', 'out', 'income', 'balance_mean',
       'balance_std']

# # # 〈3〉分类变量
var_d = ['sex','house_loan']

# # # # 对 sex 和 house_loan 两个分类变量二值化，方便分析
data_model['sex_kind'] = data_model['sex'].map({'男':1,'女':0})
data_model['house_loan_kind'] = data_model['house_loan'].map({'1':1,'0':0})
```

4) 使用 热力图 查看个变量间的关系

In [ ]:

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns

corr = data_model[var_c+var_d].corr()
plt.figure(figsize=(12,9))  # 指定宽和高（单位：英寸）
sns.heatmap(corr,vmax=1,annot=True) # vmax 设置热力图颜色取值的最大值；annot 设置是否显示格子数字
plt.show()
```



5) 选择最终模型使用的变量

In [ ]:

```
'''
从热力图观察可知：
    贷款信息、居住地信息、经济状况信息内各变量具有高相关性，对变量进行筛选及转换
    1. 贷款信息中：保留 amount
    2. 居住地信息：
                1）采用人均 GDP，即对变量进行转换；
                2）采用失业增长率
    3. 经济状况信息：
                1）客户放款前近一年总结息（反应实际存款数额）
                2）收支比（反应客户消费水平）
                3）可用余额变异系数（反应客户生活状态稳定系数）
'''
```

```
data_model['GDP_per'] = data_model['GDP']/data_model['A4']  # 人均GDP人民生活水平的一个标准
data_model['unemployment'] = data_model['A13']/data_model['A12']    # 失业增长率一定程度上反应经济增长率
data_model['out/in'] =  data_model['out']/data_model['income']  # 消费占收入比重，一定程度反应客户消费水平
data_model['balance_a']  =  data_model['balance_std']/data_model['balance_mean']    # 可用余额变异系数
var = ['account_id','sex_kind','age','amount','GDP_per','unemployment','out/in','balance_a']
# print(data_model)
# print(list(data_model))
```

2. 逻辑回归构建

In [ ]:

```
data_model = data_model[var+[y]]
for_predict = data_model[data_model[y]==2]  # loan_status 为2表示状态C，即：待定
data_model = data_model[data_model[y]!=2]

# # 定义自变量和因变量
import numpy as np
X = data_model[var]
Y = data_model[y]

# # 将样本数据建立训练集和测试集，测试集取20%的数据
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 = 1234)
```

3. 建模(使用逻辑回归 L1 正则化参数)

In [ ]:

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(penalty='l1',solver='liblinear')
var_temp = ['sex_kind','age','amount','GDP_per','unemployment','out/in','balance_a']
x_train_temp = x_train[var_temp]
clf = LR.fit(x_train_temp,y_train) # 拟合

x_test_temp = x_test[var_temp]
y_pred = clf.predict(x_test_temp)    # 预测测试集数据
```

```
test_result = pd.DataFrame({'account_id':x_test['account_id'],'y_pred
ict':clf.predict(x_test_temp)})
new_test_result = test_result.reset_index(drop=True)
print(test_result)  # 输出测试集中 account_id 对应的贷款状态预测
print(new_test_result)  # 输出测试集中 account_id 对应的贷款状态预测


print(clf.coef_)     #查看各变量的回归系数
```

4. 建模结果评价

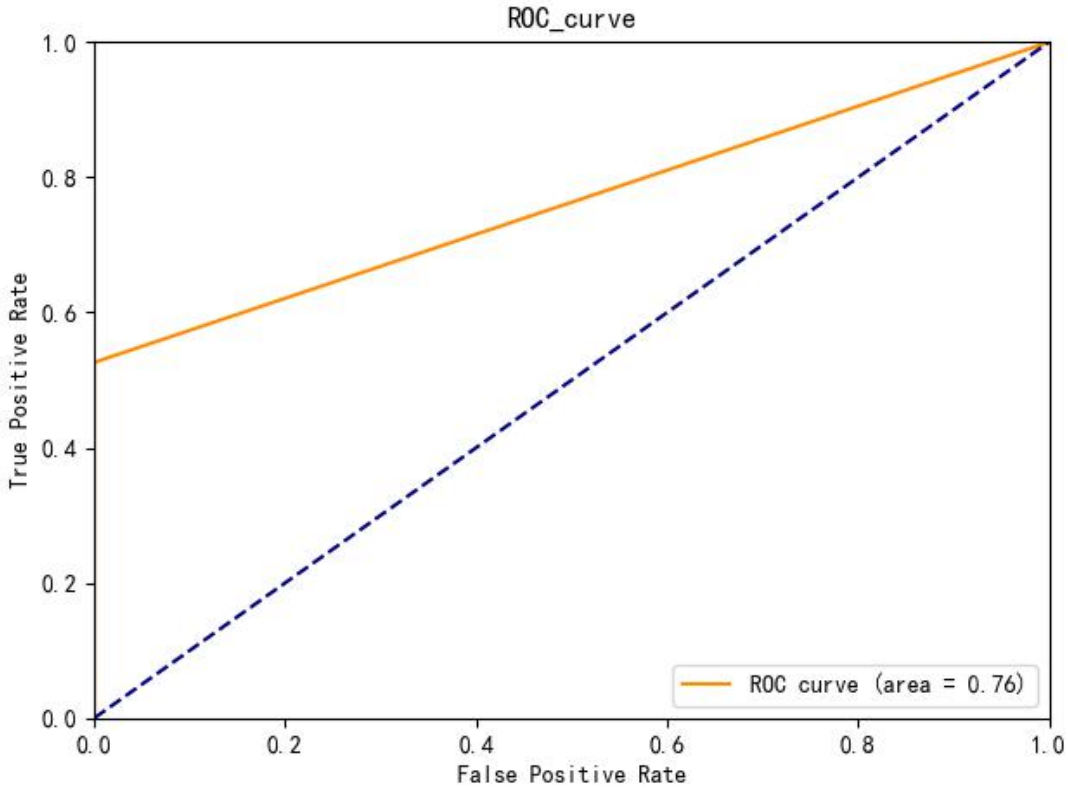
In [ ]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
'''
模型的精确率 0.87，召回率 0.84，f1_score 为 0.82
'''
```

5. 绘制 ROC 曲线

In [ ]:

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',label='ROC curve (area = %0.2f)
' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC_curve')
plt.legend(loc="lower right")
plt.show()
```



本案例来源：https://www.kesci.com/home/project/5ed9c13fb772f5002d6dc07c