|  |  |
| --- | --- |
| 队伍编号 | MCB2103075 |
| 赛道 | A |

二手车估价问题探讨

摘要

随着我国的机动车数量不断增长，人均保有量业随之增加，机动车以“二手车”的形式在流通环节，包括二手车收车、二手车拍卖、二手车零售、二手车置换等环节的流通需求越累越大。本文基于特征处理对影响二手车销量的数据进行分析，并通过皮尔逊系数法计算各种特征对二手车销售周期的影响因素的关键影响。

对于二手车零售交易价格预测分析，首先对数据进行处理，对数据中存在的缺失值、异常值进行处理，然后进行特征构造并处理匿名特征，对部分数值型数据进行分桶，最后采用皮尔逊相关系数对特征进行选择处理，之后进行数据标准化、并训练模型，采用集成学习预测二手车的零售交易价格。同时根据问题一中提供的模型评测标准进行构建评估函数，并进行预测与评估结果。

对于成交周期关键因素分析，主要任务是对车辆成交周期的分析，需要挖掘影响车辆车轿周期的关键因素，并进一步说明采取什么有效手段，可以加快门店在库车辆的销售速度。针对此问题，本次研究首先对附件四所给数据进行分析与研究，之后结合附件一车辆相关信息对车辆的销售周期进行进一步的分析，在之后，考虑到问题的针对性，采用皮尔逊相关系数对销售周期的影响因素进行更进一步的分析。

对于对于其他可探究问题分析，我们进行了文献的搜集整理，并根据给出的样本数据，进行分析，分析二手车的销量，价格以及销售周期等因素去提升二手车的流通销售分析，便于二手车的零售市场。

对于对于其他可探究问题分析，我们进行了文献的搜集整理，并根据给出的样本数据，进行分析，分析二手车的销量，价格以及销售周期等因素去提升二手车的流通销售分析，便于二手车的零售市场。

对于结合库存管理的门店经营模型构建分析，在本次模型架构中，本次研究通过架构模型对是否调价、什么时候调价、调价幅度进行预测，然后通过这些信息因素对门店销售利润进行预测分析。对于结合库存管理的门店经营模型构建分析方案二，针对这样一个优化问题，即在众多方案或参数值中寻找最优方案或参数值，以使得某个或多个功能指标达到最优，使得系统的性能指标达到最大值。欲采取智能优化算法中的遗传算法给出优化方案。将抽象数学模型转化为遗传算法中的优化问题进行模型构建。

关键词：特征处理，皮尔逊系数法，集成学习，遗传算法，最优化，K近邻算法

目录

[一、问题重述 1](#_Toc98168394)

[1.1 问题的背景 1](#_Toc98168395)

[1.2 问题的提出 1](#_Toc98168396)

[二、问题分析 2](#_Toc98168397)

[2.1 二手车零售交易价格预测分析 2](#_Toc98168398)

[2.2 成交周期关键因素分析 2](#_Toc98168399)

[2.3 其他可探究问题分析 3](#_Toc98168400)

[2.4 成交周期预估建模分析 3](#_Toc98168401)

[2.5 结合库存管理的门店经营模型构建分析 3](#_Toc98168402)

[三、符号说明 3](#_Toc98168403)

[四、基于二手车零售交易价格预测分析 4](#_Toc98168404)

[4.1 数据处理 4](#_Toc98168405)

[4.2 模型架构 12](#_Toc98168406)

[4.3模型检验 16](#_Toc98168407)

[4.4模型评价 17](#_Toc98168408)

[五、基于成交周期关键因素的研究分析 17](#_Toc98168409)

[5.1 影响因素初步探索 17](#_Toc98168410)

[5.2 结合附件一的数据分析 21](#_Toc98168414)

[5.3 影响因素进一步挖掘 27](#_Toc98168418)

[5.4 采用关键因素加快在库车辆的销售速度 31](#_Toc98168423)

[六、其他可探究问题分析 31](#_Toc98168425)

[6.1 淡旺季之于二手车买卖的影响探讨 32](#_Toc98168426)

[6.2 处置积压库存车辆策略分析 32](#_Toc98168427)

[6.3 价格差带来的利润问题 33](#_Toc98168428)

[七、成交周期预估建模分析 33](#_Toc98168429)

[7.1 回归预测成交周期 33](#_Toc98168430)

[7.2特征工程 46](#_Toc98168437)

[7.3模型的构造(lightGBM) 49](#_Toc98168443)

[7.4 模型评估 49](#_Toc98168444)

[7.5模型融合，简单加权融合 50](#_Toc98168445)

[7.6第二种思路 50](#_Toc98168446)

[八、 结合库存管理的门店经营模型构建分析 52](#_Toc98168447)

[8.1 方案一：K近邻模型预测数据分析 52](#_Toc98168448)

[8.2 方案二：遗传算法建模解题分析 53](#_Toc98168454)

[参考文献 58](#_Toc98168461)

[附录 58](#_Toc98168462)

[附录 1：二手车零售交易价格预测分析代码 59](#_Toc98168463)

[附录 2：基于成交周期因素分析代码 70](#_Toc98168464)

[附录3：成交周期预估分析代码 120](#_Toc98168465)

[附录4：结合库存管理的门店经营模型构建分析代码 137](#_Toc98168469)

# 一、问题重述

## 1.1 问题的背景

随着我国的机动车数量不断增长，人均保有量也随之增加，机动车以“二手车”形式在流通环节，包括二手车收车、二手车拍卖、二手车零售、二手车置换等环节的流通需求越来越大。二手车作为一种特殊的“电商商品”，因为其“一车一况”的特性比一般电商商品的交易要复杂得多，究其原因是二手车价格难于准确估计和设定，不但受到车本身基础配置，如品牌、车系、动力等的影响，还受到车况如行驶里程、车身受损和维修情况等的影响，甚至新车价格的变化也会对二手车价格[1]带来作用。目前国家并没有出台一个评判二手车资产价值的标准。

一些二手车交易平台和二手车第三方估价平台都从自身的角度建立了一系列估价方法用于评估二手车资产的价值。

在一个典型的二手车零售场景，二手车一般通过互联网等线上渠道获取用户线索，线下实体门店对外展销和售卖，俗称 O2O 门店模式。门店通过“买手” 从个人或其他渠道收购二手车，然后由门店定价师定价销售，二手车商品和其他商品一样，如果定价太高滞销也会打折促销，甚至直接以较低的价格打包批发，直至商品最终卖出[2]。

面对二手车市场的供求要求[3]，对二手车数据进行深入的分析，分析相关数据判断二手车的销量，并进行数据分析与建模分析车辆交易周期的影响因素等，解决二手车交易平台在交易中遇到的问题。

## 1.2 问题的提出

基于针对初赛问题所做的分析与处理，再根据附件所给的数据，通过建立特征处理与数学建模帮助二手车交易平台进行更好的交易：

1.二手车零售交易价格预测：基于问题相关数据，构建预测二手车交易价格的训练模型和测试模型，并对附件 2 中的“估价验证数据”进行预测。

2.成交周期关键因素分析：对车辆的成交周期进行分析，挖掘车辆成交周期的关键因素。并分析如果需要加快门店在库车辆的销售速度，可以结合这些关键因素采取哪些手段，并进一步说明这些手段的使用条件和预期效果。

3.其他可探究问题分析：分析二手车交易过程中，有哪些问题值得研究，可以帮助二手车交易平台进行更好的交易。

4.成交周期预估建模：在成交周期关键因素分析的基础上，利用附件4构建“门店交易训练数据”构建交易周期预测模型，进而对附件5“门店交易验证数据”进行预测。

5.结合库存管理的门店经营模型构建：在门店经营过程中对库存进行有效管理，同时在保障门店经营目标的情况下，通过适时对在库车辆进行调价使得门店销售利润最大化。

# 二、问题分析

## 2.1 二手车零售交易价格预测分析

问题一是根据以往的数据预测二手车的价格，属于有监督学习里的回归问题，首先查看 36 列变量信息字段的描述，对于时间类型的数据读入数据时转换成时间类型格式，然后查看数据的缺失情况，对缺失严重的特征如 anonymousFeature7 和 anonymousFeature15 的缺失值以达到 50%以上，对这类特征进行删除，并对缺失值不大的特征进行填充如 'carCode', 'modelyear', 'gearbox'，由于它们都是分类特征采用众数进行填充，对时间基础周期特征(年月日特征拆解)，同时构建时间差特征如：'tradeTime'-'registerDate' (汽车的使用时间)，对定类数据进行 Frequency 编码，对部分数值型变量进行数据分桶，根据部分特征明显的匿名变量构造新特征，比如 anonymousFeature12 （4220\*1740\*1625）很有可能就是汽车的体积，因此可以构造长宽高三个新的特征，接下来就可以进行特征的选择了，可以使用基于皮尔逊相关系数进行特征选择，然后进行降维可以选择 PCA 主成分分析，然后选择集成学习随机森林训练测试集，最后可以使用 XGboost 优化一下模型。

## 2.2 成交周期关键因素分析

问题二需要结合附件 4“门店交易训练数据”对车辆的成交周期进行分析，挖掘影响车辆成交周期的关键因素。关键因素即一定程度上起决定性作用的影响因素，本报告欲先根据附件 4 对数据进行初步分析，即初步判断哪些因素可能成为关键因素，附件 4 中的价格因素很明显将会成为影响成交周期的关键因素之一，故而在本报告中对于问题二的初步分析的重点就是价格因素，在价格数据记录上做文章进行分析。出于更加全面地考虑，价格差、成交时间、价格变更次数等也将成为初步分析的重点内容。鉴于附件 4 的数据特征过少，不足以说明问题，故将附件 4 同附件 1 进行信息整合，得到较为全面的特征集合，通过进一步挖掘，筛选得出关键因素。为了使得本报告中对于成交周期的影响因素挖掘具有充分说明性，进一步构建模型，即通过特征选择、皮尔逊相关系数计算分析、构建模型，根据现有数据特征对成交周期进行预测，在预测的过程中，不断变更输入的特征子集，观察得分变化。变化较大，即影响较大，从而得出影响成交周期[4]的关键因素。

## 2.3 其他可探究问题分析

问题三需要依据给出的样本数据集，给出值得研究的问题，即进行问题拓展，并在问题拓展分析之后给出思路。针对现有数据集，查阅、搜集、整理有关二手车市场[5]的资料，关注二手车买卖中亟待解决的问题，从而提出问题，并试图在本报告中给出思路，予以参考。在查阅大量资料后，本报告欲就淡旺季、车库存货积压[6][7]等方面提出问题并给出思路。

## 2.4 成交周期预估建模分析

成交周期=成交时间-上架时间，先把数据转化成时间类型，计算出成交周期，然后看特征的分布情况，去除掉训练集和测试集中数据分布不同的数据，然后检测异常值，对异常值进行处理，然后查看预测值的数据分布情况，对不符合正太分布的进行Box-Cox转换，最后对数据值数据进行皮尔逊相关性的检测，筛选出合适的特征，最够使用lightGBM模型进行预测，进行10折交叉验证后，使用XGboost和lightGBM进行简单的加权融合增强模型的鲁棒性。

## 2.5 结合库存管理的门店经营模型构建分析

在本次的模型架构的过程中，通过对附件1与附件4数据的综合处理，结合二手车的销售信息，对二手车是否进行调价，什么时候进行调价以及调价幅度进行预测，之后再根据以预测的信息对销售周期进行预测分析，以判断如何更好地保障门店的经营目标。

对于本题，本次还采用了方案二，即首先由题得到抽象数学f(t,△pi)，进而将其转化为优化问题进行模型构建，即用标准遗传算法求函数f(t,△pi)的最大值，经过初始化种群数目、染色体二进制编码长度、最大进化代数、交叉概率、变异概率；产生初始种群，计算个体适应度值，进行选择操作、基于概率的交叉和变异操作，产生新的种群，并把历代的最优个体保留在新种群中等操作不断迭代求得优化值。

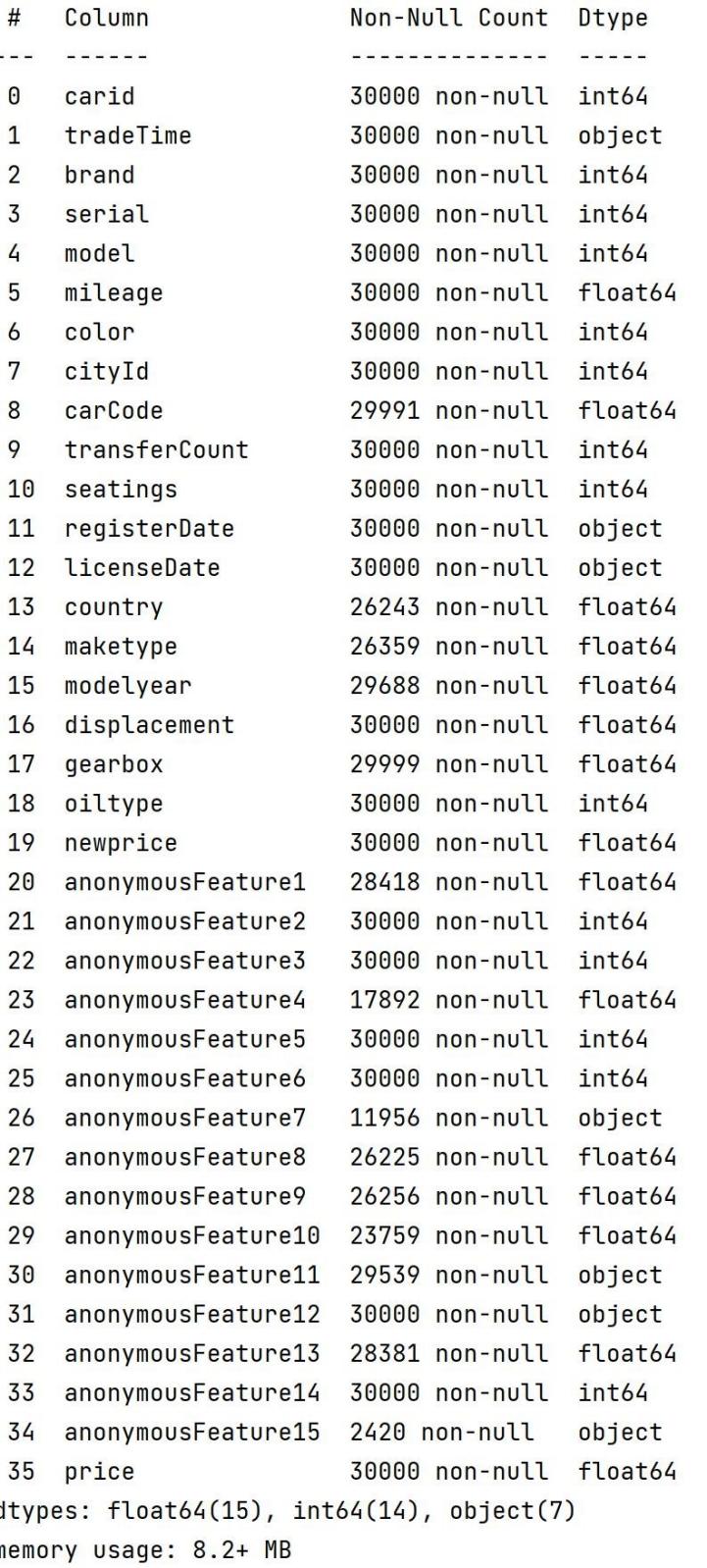
# 三、符号说明

|  |  |
| --- | --- |
| Features | Description |
| updateTimeX(x∈（1,12）) | 第x次更新价格时间 |
| updatePriceX(x∈（1,12）) | 第x次更新对应的价格 |
| updatenum | 每条记录对应更新价格次数 |
| withdrawPeriod | 计算出的成交周期 |
| withdraw\_month | 成交月份 |
| dif\_price | 成交差价 |
| anonymousFeature13\_year | 匿名特征13转化成时间后的年份 |
| anonymousFeature13\_month | 匿名特征13转化成时间后的月份 |
| anonymousFeature12\_length | 车辆长度 |
| anonymousFeature12\_width | 车辆宽度 |
| anonymousFeature12\_height | 车辆高度 |
| old\_year | 汽车的使用时间 |
| old\_year\_1 | 汽车的注册时长 |
| mileage\_bin | 对mileage进行分箱后的区间段号 |
| decreasing\_count | 降价的次数 |

# 四、基于二手车零售交易价格预测分析

## 4.1 数据处理

1. 给匿名变量命名，从 anonymousFeature1 到 anonymousFeature15
2. 查看变量数据类型

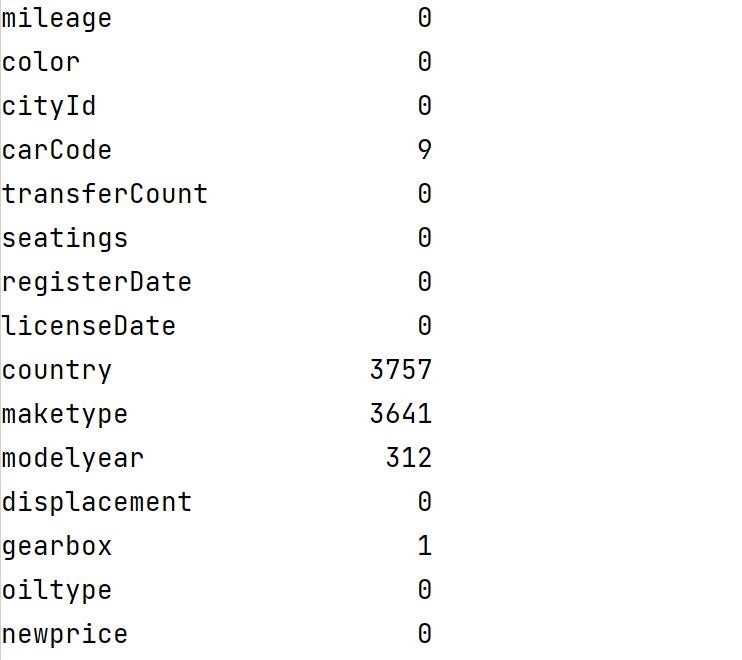


1. 前五列'carid', 'tradeTime', 'brand', 'serial', 'model'都是 id 和时间，不能做中心化操作，而且通过 data.iloc[:, :5].isnull().sum()



发现没有缺失值，因此不需要处理。

其他列缺失值的情况



可以看出除了 15 个匿名特征外，carCode 有 9 个缺失值，country 有 3757 个缺失值 maketype 有 3641 个缺失值，modelyear 有 312 个缺失值。gearbox 有一个缺失值。

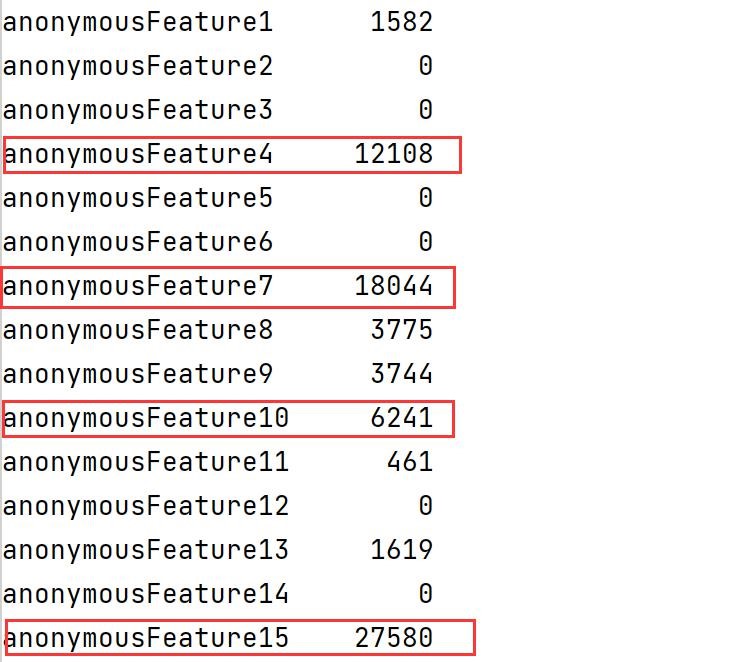
对于缺失值较少的列，如 carCode 和 gearbox 直接进行删除操作。 country ,maketype 和 modelyear 由于缺失值较多，根据数据类型，定类数据选择使用众数填充比较适合。

data\_1['country'].unique() 可以发现



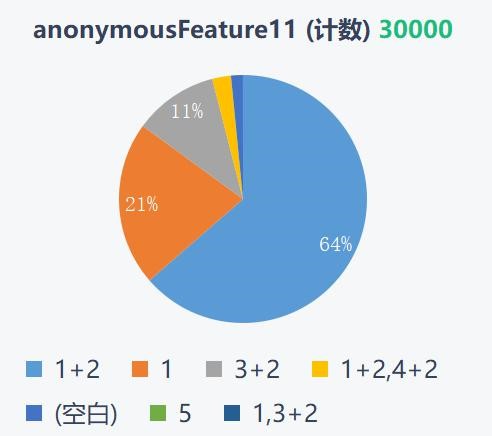
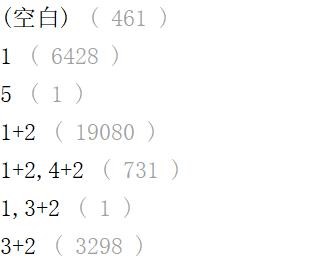
这个0导致数据量纲较大，会对后期模型的性能产生影响，因此结合所给的信息将其改为779410

4. 对 15 个匿名变量的分析



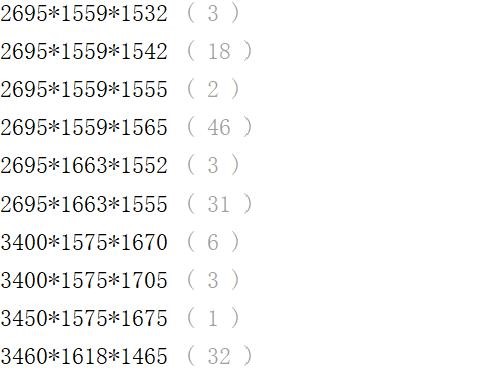
上图圈出的特征由于数据缺失严重直接删除这些特征

①对 anonymousFeature11



根据显示的信息猜测此特征是某种物品的个数，预测应该为字符串中所有数字之和，使用正则匹配提取所有数值并求和作为新的 anonymousFeature11。

②对 anonymousFeature12



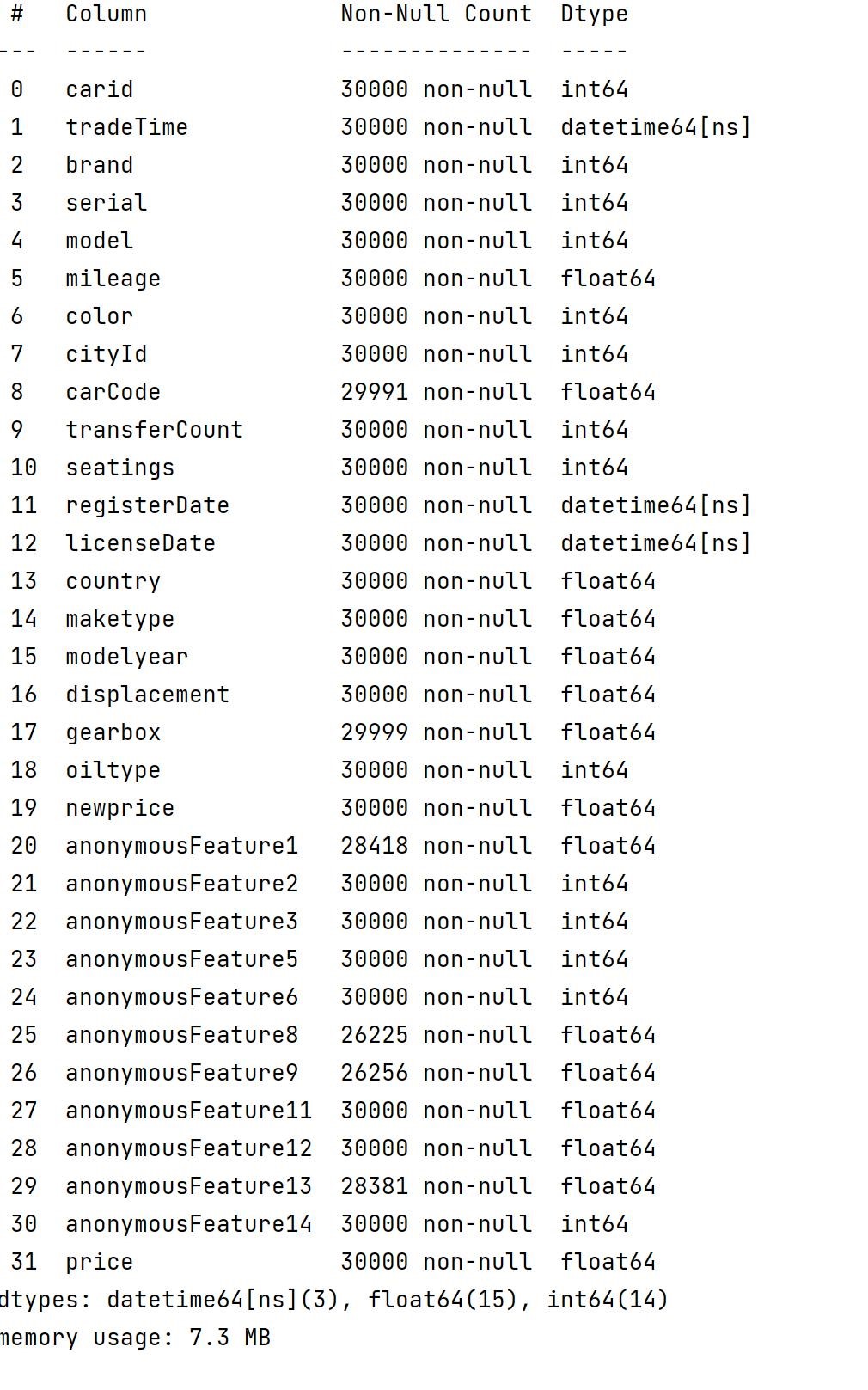
根据其值的表现形式估计其为汽车的体积，所以其值应为三个数的乘积，使用\*切分数据并求出乘积做为新的anonymousFeature12，并用切分得到的三个数据 列 构 建 三 个 新 的 特 征 anonymousFeature12\_length ， anonymousFeature12\_width，anonymousFeature12\_height。

③对 anonymousFeature13



把 anonymousFeature13 转换成 字 符串 进行 切 片， 取 前四 位为 anonymousFeature13\_year，其他为 anonymousFeature13\_month。

经过以上处理，数据已经全部变成数值型特征和时间类型的特征



1. 处理定类特征

对 color，carCode，country，modelyear 这些类别行的特征，因为是定类数据，数据之间本来应该是没有相对大小的，但是转为数值后比如 1，2，3 会有相对大小，会对树模型产生不好的影响。因此本组选择使用 Frequency 编码对定类数据进行转换，Frequency 编码通过计算特征变量中每个值的出现次数来表示该特征的信息。

1. 再用时间类型的特征构建新特征 ①基础周期特征的拆除（年月日特征拆解）

data['tradeTime\_year']=data['tradeTime'].dt.year data['tradeTime\_month']=data['tradeTime'].dt.month data['tradeTime\_day']=data['tradeTime'].dt.day data['registerDate\_year']=data['registerDate'].dt.year data['registerDate\_month']=data['registerDate'].dt.month data['registerDate\_day']=data['registerDate'].dt.month

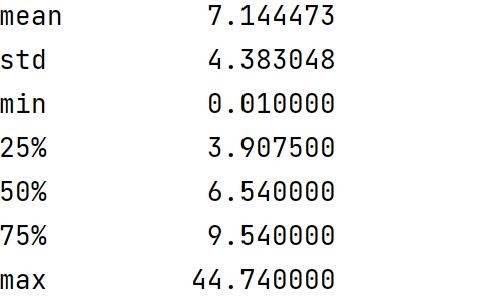
②时间差

‘old\_year’='tradeTime'-'registerDate' (汽车的使用时间)

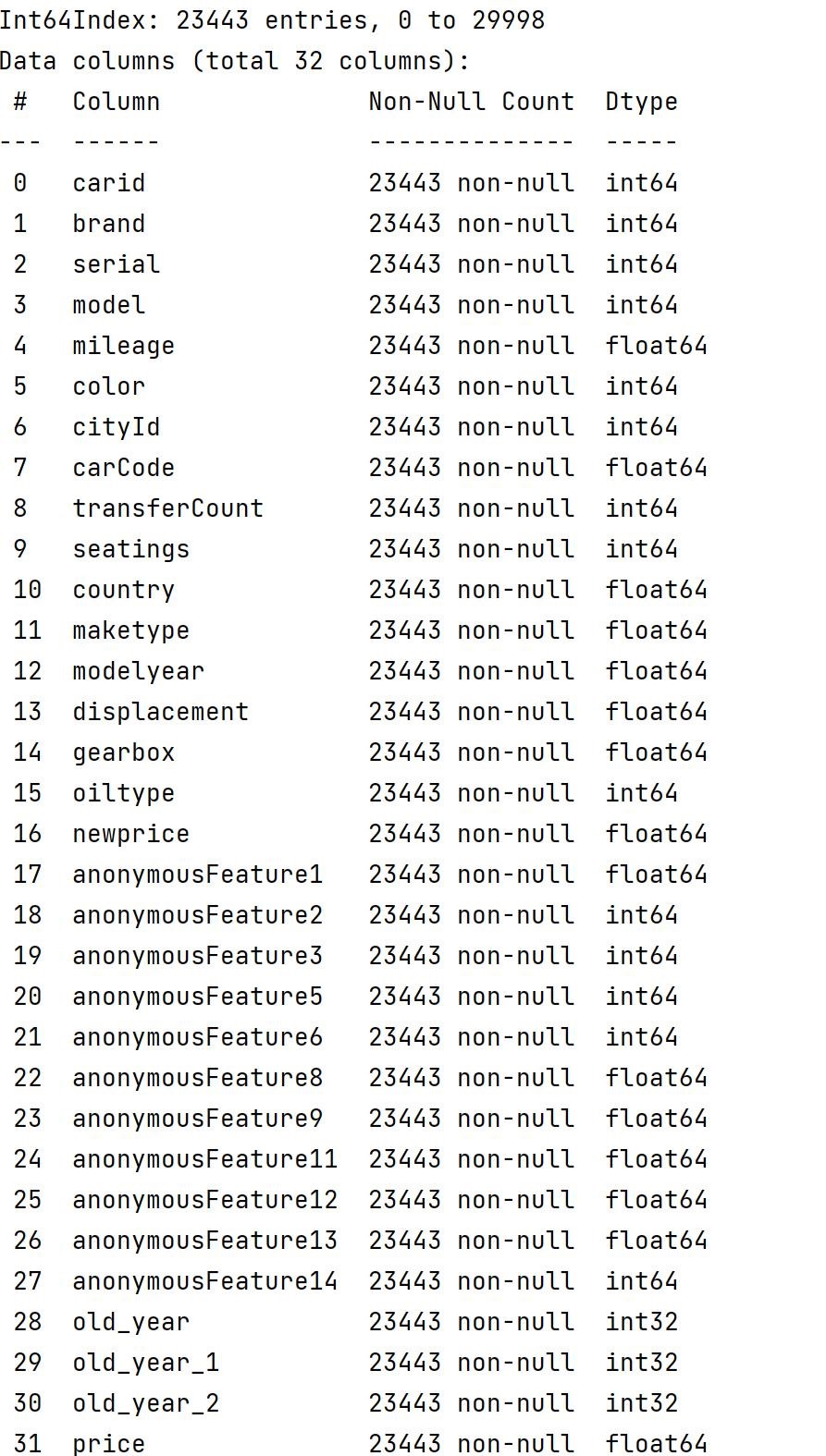
‘old\_year\_1’='tradeTime'-'licenseDate'

经过以上处理后，特征以全部转换成数据值

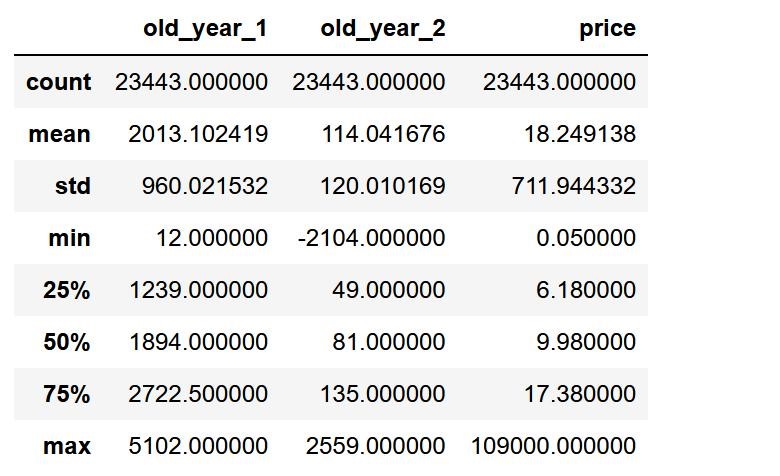
1. 数值特征[8]的处理数据分桶，对 mileage 特征进行数据分桶,mileage 的均值为 7.14，其余统计量如下



因此分组区间为 (0,1],(1,4],(4,7.15],(7.15,10],(10,50]



1. 清除数据中的异常值首先查看数据的分布，找出异常值 data\_2.describe()



可以明显的看出，price 有异常值，接下来用箱线图检测异常值，并构造函数处理异常值

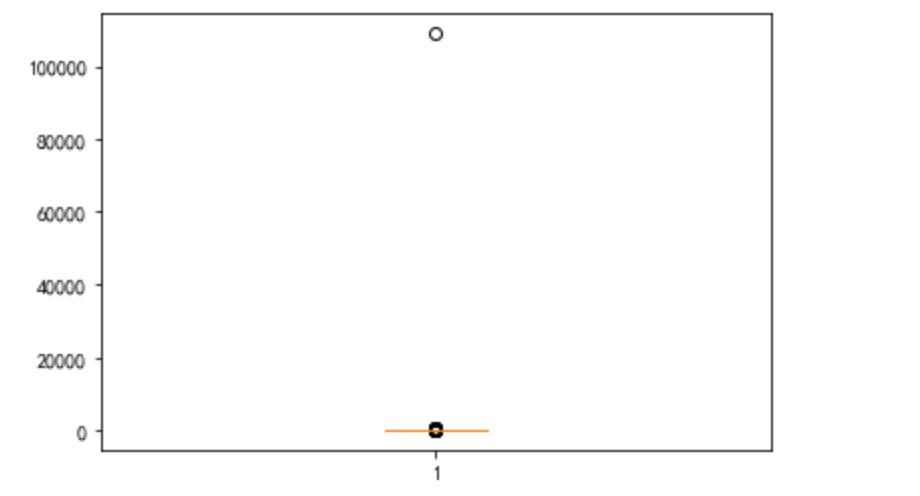


图 1 Price 的箱线图

从图中我们也可以看出，price 明显存在异常值。

## 4.2 模型架构

1.模型假设

（1）假设销售价格不会超过规定的阈值

（2）假设销售的谈判技巧相同或没有显著差异

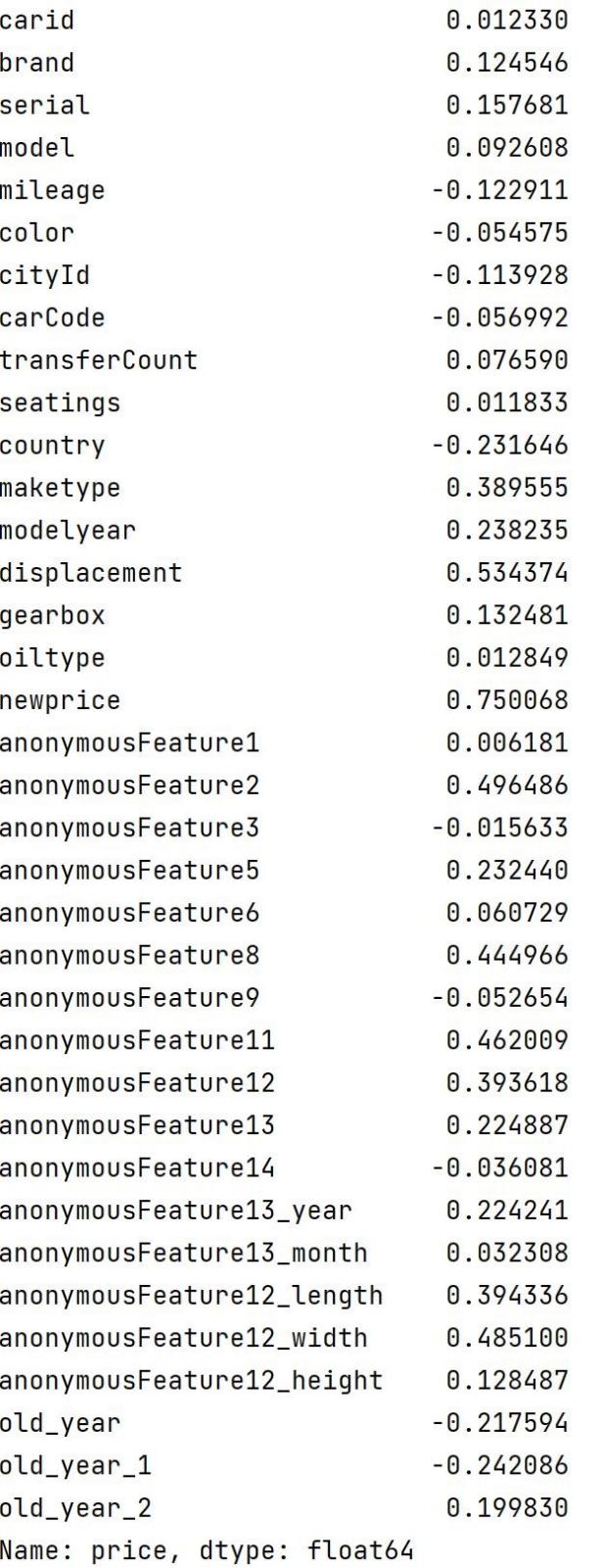
（3）假设这些销售记录都来自同一个平台或者平台间的评判标准类似

（4）假设车身受损和维修情况相同

2.从这些特征中选择出最合适的特征（特征选择），为模型构建做准备。

皮尔逊相关系数的特征选择[9]





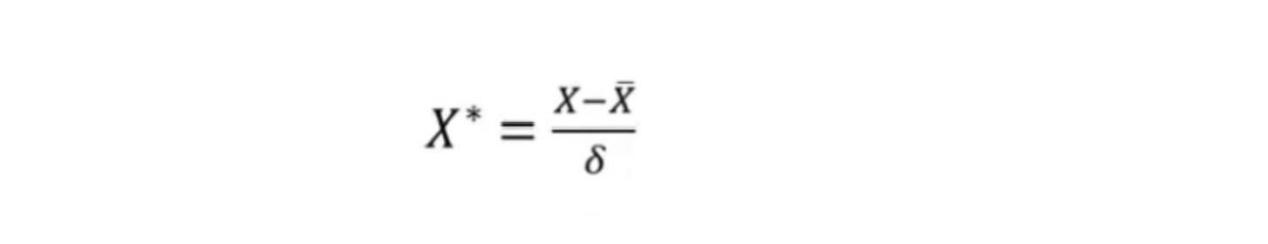
3.进行数据降维（PCA 主成分分析）

主成分分析(PCA,principal components analysis)将高维数据投影至低维空间，低维空间中的新特征叫主成分/超级列[10]。只需要几个主成分就可以准确解释整个数据集。

1. 创建数据集的协方差矩阵;
2. 计算协方差矩阵的特征值;
3. 保留前 n 个特征值&特征向量(按特征值降序排列);
4. 用保留的特征向量转换新的数据点。

4.数据标准化数据标准化是而在多指标评价体系中，由于各评价指标的性质不同，通常具有不同的量纲和数量级。当各指标间的水平相差很大时，如果直接用原始指标值进行分析，就会突出数值较高的指标在综合分析中的作用，相对削弱数值水平较低指标的作用。因此，为了保证结果的可靠性，需要对原始指标数据进行标准化处理。这里采用的标准差标准化

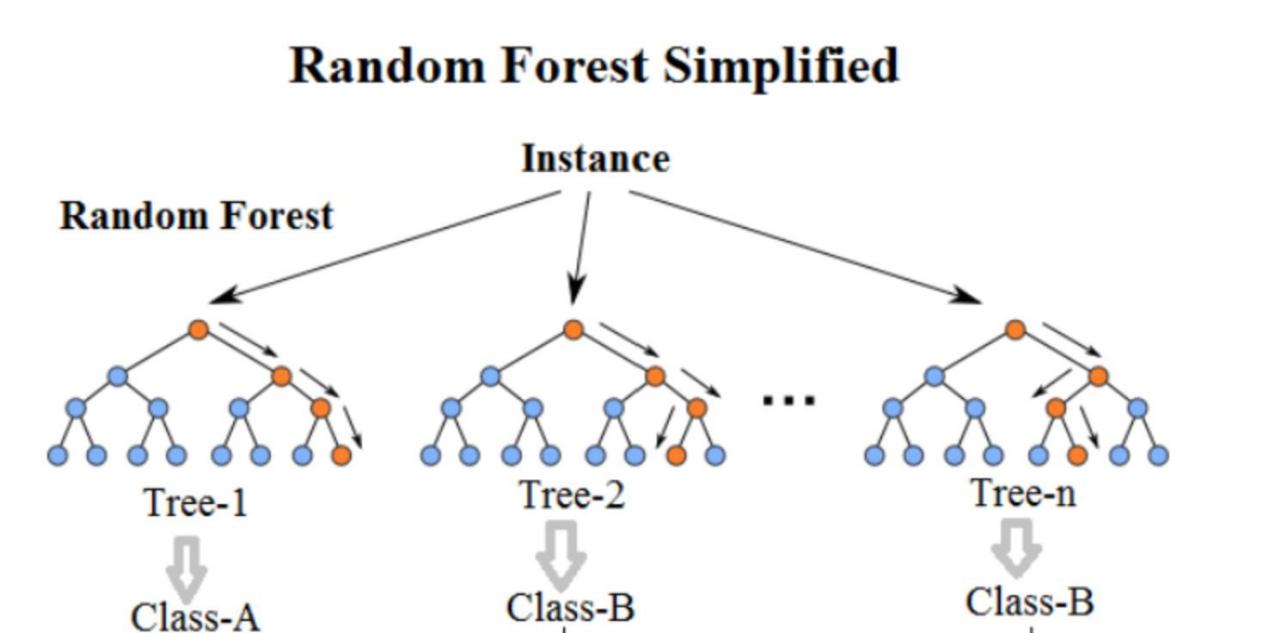
标准差标准化也叫零均值标准化或分数标准化,是当前使用最广泛的数据标准化方法。经过该方法处理的数据均值为 0 ,标准差为 1 , 转化公式如下。



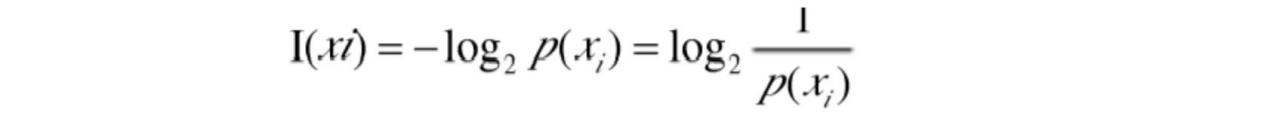
其中 X 为原始数据的均值, δ为原始数据的标准差。标准差标准化后的值区间不局限于[0,1] ,并且存在负值。同时也不难发现,标准差标准化和离差标准化- -样不会改变数据的分布情况。

5.训练模型

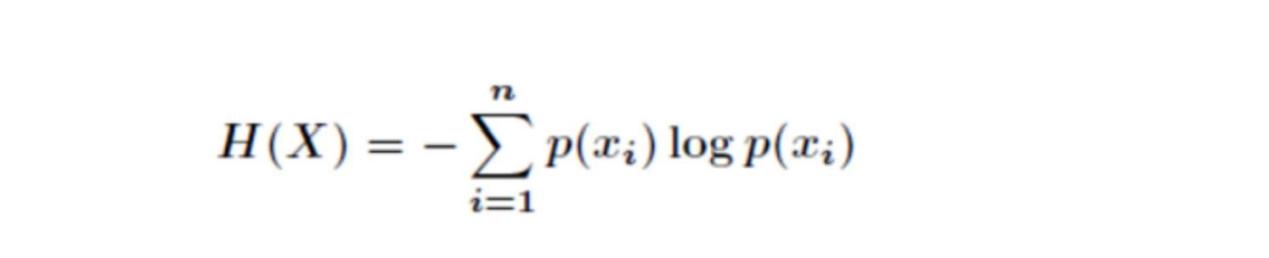
综合考虑各方面因素后选择集成学习里的随机森林进行模型的训练，随机森林就是使用多个决策树算法，创建弱评估器，其基本原理如下



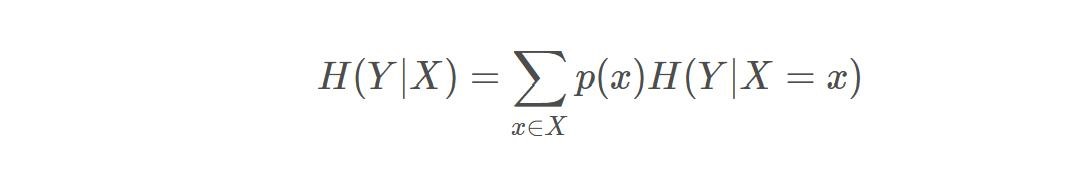
而决策数进行特征选择和划分的时候主要依据信息增益，这里要引入部分信息论的知识，信息的计算公式如下



而熵[11]就是信息的期望，其计算公式如下



然后还需要计算一下条件熵（条件熵 H（Y|X）表示在已知随机变量 X 的条件下随机变量 Y 的不确定性）



最后需要计算的就是要求的信息增益信息增益是：信息熵-条件熵。

而决策数在进行特征选择的时候就是根据信息增益如果IG（信息增益大）的话那么这个特征对于分类来说很关键，决策树就是这样来找特征的。随机森林就是所有决策树平均值[12]。

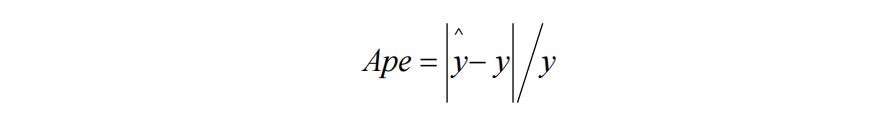
6.进行模型的预测

使用训练好的模型将测试集的数据放入进行预测

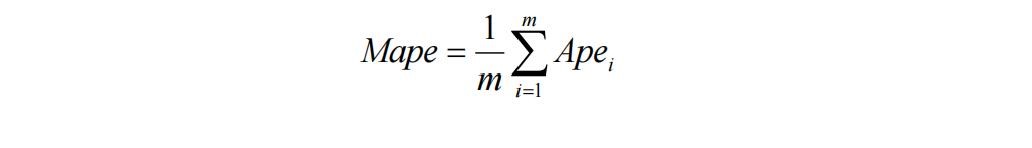
7.得出预测结果后利用评估函数进行评价，评判模型好坏评估函数

0**.**2**\*(**1*Mape***)**0**.**8**\*** *Accuracy*5

*Ape*(相对误差)：



Mape(平均相对误差)：

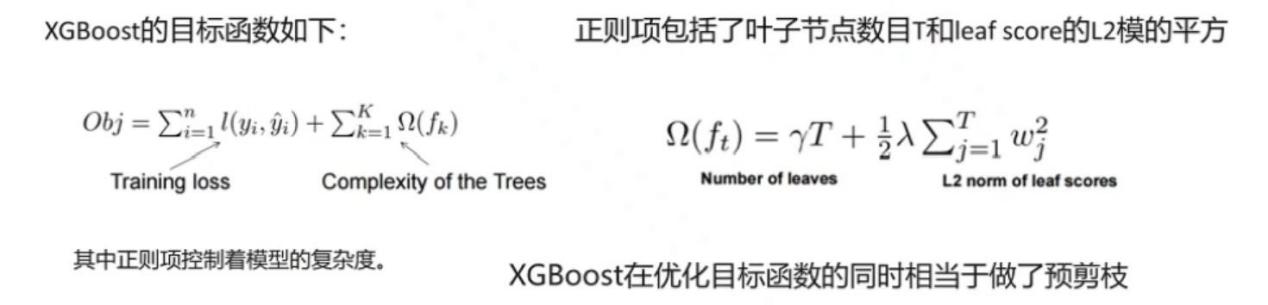


其中，真实值*y* **(***y*1**,** *y*2**,..,** *ym***)**，模型预测值为*y***(***y*1**,** *y*2 ,**..,** *ym***)**；

*Accuracy5*(5%误差准确率):

*Accuracy*5 =*count***(***Ape* 0**.**05)/*count***(***total***)**

8.最后使用 XGboost 优化一下模型



XGBoost 的优势:

（1）正则化可有效防止模型过拟合

（2）并行处理

XGBoost 并行处理数据的速度要比普通提升树算法更快.

（3）自动处理缺失数据

不做切分点，但会把缺失值放进左右子树看效果

（4）剪枝策略

普通的提升采用的是贪心算法，只有在不再有增益时才会停止分裂

（5）可以在目前树模型上继续迭代

XGBoost 可以在训练中使用不同的迭代策略[13]。

## 4.3 模型检验

将训练集拆分后根据预测值和实际的值绘制散点图进行观察

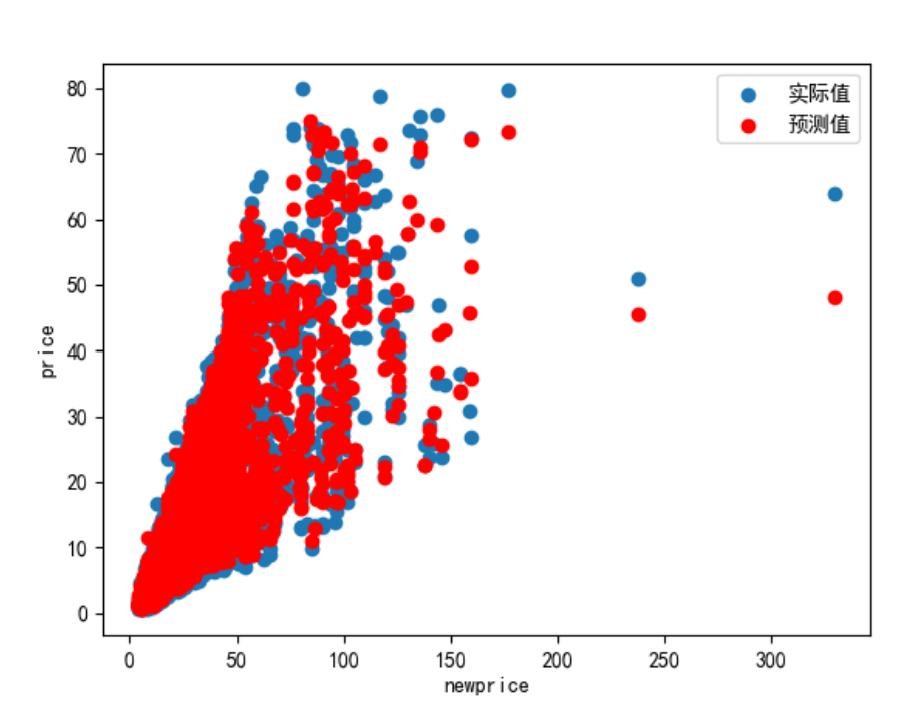


图 2 预测值和实际的值绘制散点图

## 4.4 模型评价

该基于随机森林模型，随机森林是一个包含多个决策树的分类器。

1. 优点：

（1）针对较大数据集，有着较好的预测效果。

（2）训练完模型之后可以直接得出关键特征，即最具影响力的一些特征集合。

（3）在创建随机森林的时候，对 generlization error 使用的是无偏估计，因此模型泛化能力强。

（4）因为树与树之间是相互独立的，故训练速度相对较快。

（5）实现较为容易，预测过程也并不复杂。

2. 缺点：

（1）当进行回归时，若超越训练集数据范围的预测，这可能导致出现过度拟合的情况。

（2）无法进入模型内部，只能通过调参和随机种子得到更好的效果。

# 五、基于成交周期关键因素的研究分析

## 5.1 影响因素初步探索

在此过程中，考虑到文件对附件四的要求，首要对附件四相关数据进行处理分析，并对影响因素进行分析处理。

### 5.1.1 数据处理

（1）说明：对影响因素的初步探索主要是针对附件 4 进行数据探索与分析。

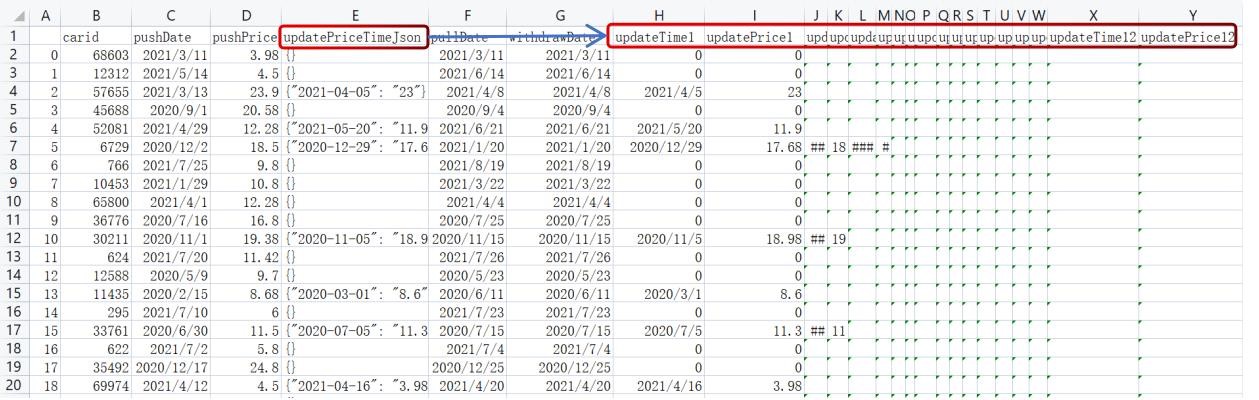
（2）处理思路与方法：

①提取每次变更价格的时间和此次对应价格

经过统计，变更价格存在 0、1、2、3、4、5、6、7、8、12 的频次数量，故首先对每一次的更新做信息提取，由“updatePriceTimeJson”属性列提取出每一次的更新时间和对应价格，即由“updatePriceTimeJson”属性列提取出

“updateTime1”、“updatePrice1”.....“updateTime12”、“updatePrice12”。

截取提取出的部分信息如下图所示：



②获取每一次记录对应的变更价格次数

仍然由“updataPriceTimeJson”属性列统计得到附件四中每一条记录所对应车辆的价格变更次数（更新次数）。更新次数统计如下图：

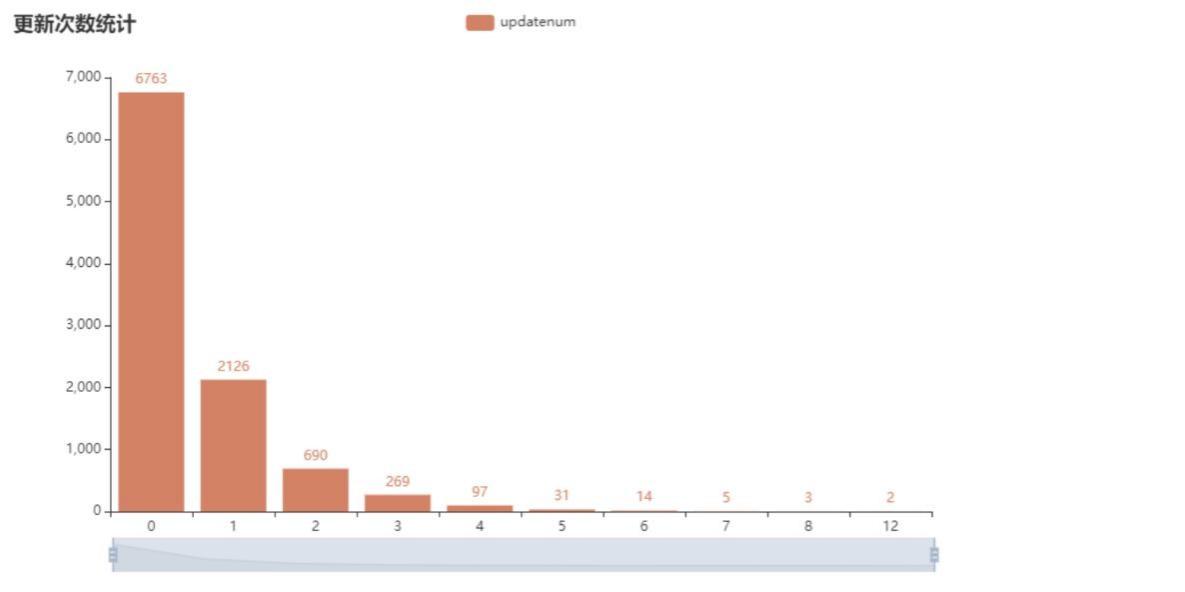


图 3 车辆价格更新次数统计图

由图可以看出，绝大多数二手车并不存在对价格的调整。相对来讲，调整价格的二手车占据少数（调整价格的原因在报告下文有分析，此处不赘述），即使调整价格，也大多为 1、2、3 次，调整价格超过三次的车辆占据极少数。

③计算得到成交周期针对成交车辆和未成交车辆分别进行计算：



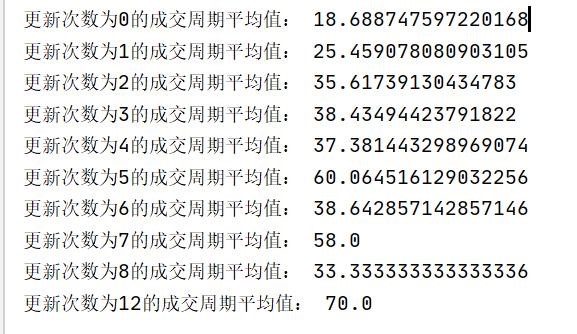
④计算成交差价（成交差价 = 成交价格 - 上架价格）

考虑到成交差价对成交周期的影响，由成交价格（即调整到最新的价格）减去该车辆上架时的价格得到成交差价，便于下一步的分析。

### 5.1.2 价格变更次数对成交周期的影响分析

（1）价格变更次数之于成交周期的影响

考虑到买家在购入二手车时，如果不急于用车，会持观望态度，即等待降价。而卖家可能因已上架车辆迟迟未卖出进行降价或者因市场需求原因对已上架车辆进行升价。由此探索价格变动频次对成交周期的影响。结果如下：



由计算结果可以看到，价格更新次数之于成交周期的大致趋势：价格更新频次越频繁，成交周期越长。若当前市场该车需求不大且消费者不急于购入二手车，就会持有观望态度，等待车辆降价，以达到最省钱的目的。站在卖家的角度，如果车辆长期未卖出，会导致车库中的车辆积压的情况，故被迫削减利益，更新降价。

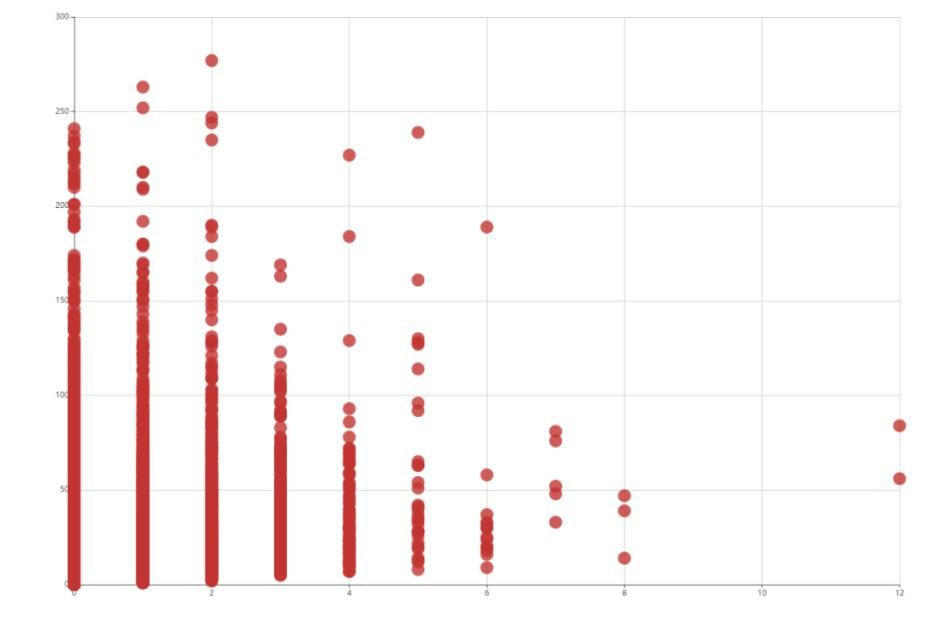
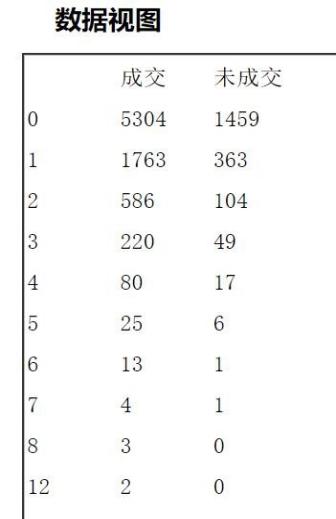


图 4 价格更新趋势

（2）价格变更次数之于车辆成交与否的影响频繁变更价格对于车辆能否成交存在着一定的关系，统计结果如下：



图 5 价格变更与车辆成交



由统计结果可以看出，无论是成交车辆还是未成交车辆，未变更价格的车辆都占据大多数。由数据视图中的数据可以看到，坐实了上文分析的情况，即当车辆长期积压时，卖家通过频繁变更价格降价，此时买家就会看准时机，出价购车。

又根据成交车辆和未成交车辆的成交周期均值，相差无几。

### 5.1.3 上架价格对成交周期的影响分析

（1）二手车价格分布情况

首先通过散点图得到二手车价格的分布情况，由图可以看到，符合对二手车价格的预期，相对来讲价格较低的占据多数。

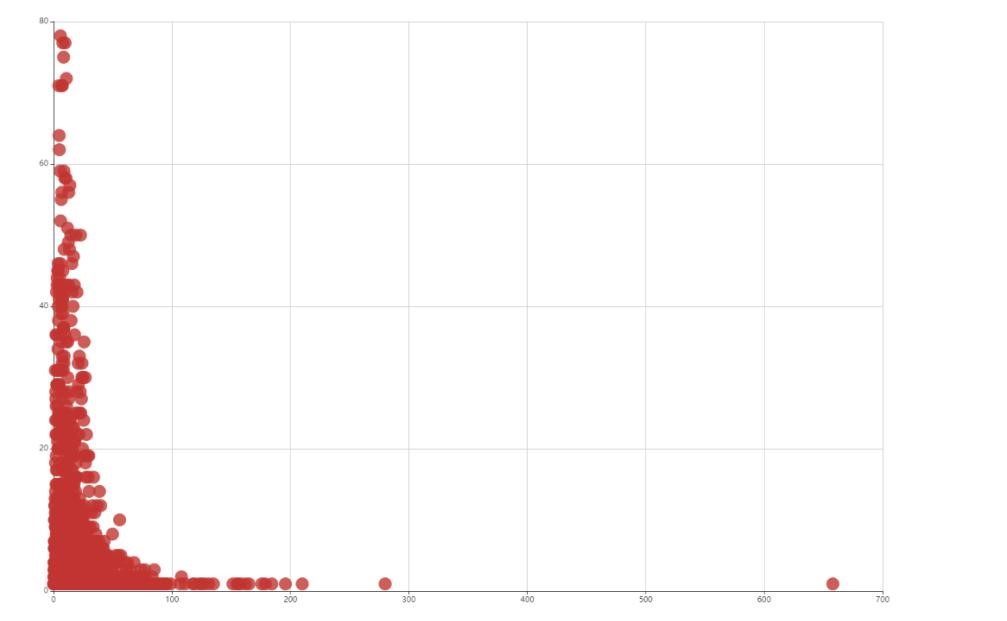
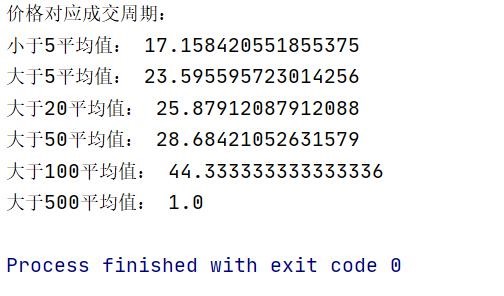


图 6 二手车价格分布情况

（2）上架价格对成交周期的影响分析



由统计结果，可以看到车辆价格越低，成交周期越短。结合之前分析的价格变更次数可知，车辆价格越低越容易卖出去，亦即成交周期相对较短。

## 5.2 结合附件一的数据分析

在对任务二车辆的交易周期的分析过程中，本次研究考虑到实际销售过程中存在的因为车的自身特征，如品牌、车系、动力以及车况等因素对二手车的销售周期[14]的影响，综合附件一与附件四的数据信息对影响车辆交易周期的关键因素进行分析挖掘。

### 5.2.1 数据分析

在本次研究中，主要研究对象为商品的销售周期，而在文件中对销售周期的计算为出售日期-上架日期。其次考虑到日期的变化性，我们对车辆的销售周期进行分组分析，以便进一步的实现数据的观察与处理。对销售周期进行分组时，我们分为销售周期天数为 0， 1， 2， 3， 4， 5-7， 8-17， 18-50， 51-91， 92-237，这样可以帮助我们更好的整理数据之间的关系，并进行数据分析。

### 5.2.2 分析车辆相关信息对销售周期天数的影响

在本次研究中，通过对车辆的相关信息进行销售周期天数的研究分析[15]，对于车辆的本身信息，主要从车辆品牌、颜色、燃油类型等方面进行了分析

首先，在进行车辆相关信息对销售周期天数的影响时，我们首先分析未修改的价格的车辆信息，通过绘制不同的销售周期（天）的散点图进行了数据的分析于观察，从图中可以看出明显销售周期短的车辆的销售情况好，对于出售的二手车而言，在较短周期天数内出售的车辆较多。

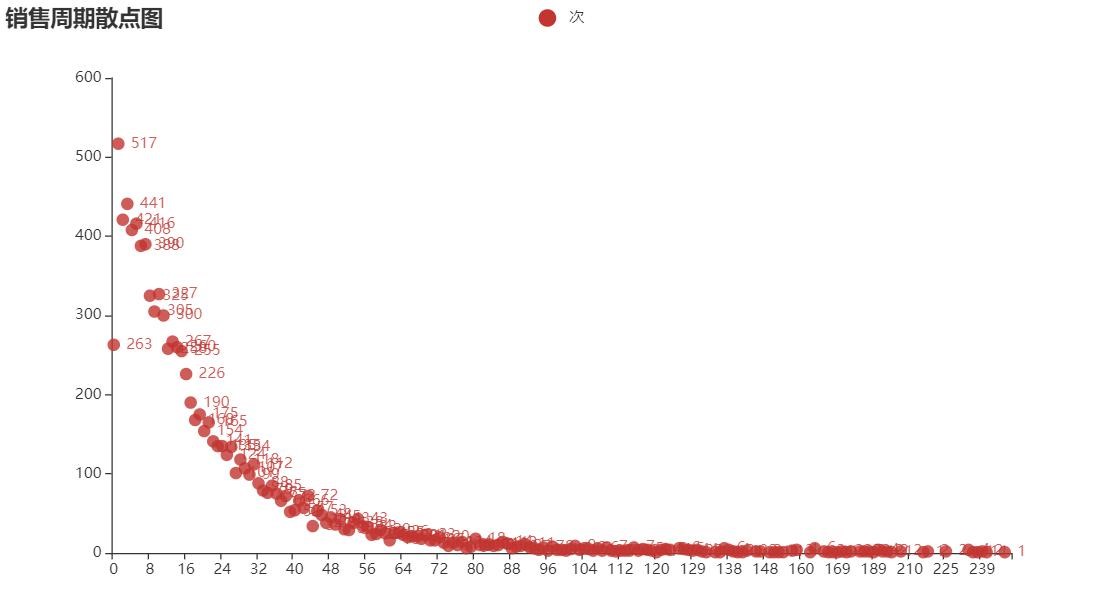


图 7 车辆的销售周期

其次，我们分析了品牌 id 对销售周期的影响，通过观察每个品牌的销售量以及每个天数的销售周期，并对前十数据进行绘图分析，通过每个品牌的销售量分析每个品牌的受欢迎程度；由于品牌量过多，无法一一比较个品牌的销售周期，暂取销售量最多和销售量最少的十个品牌进行销售周期天数的分析于整理，进行观察品牌对销售量的影响。

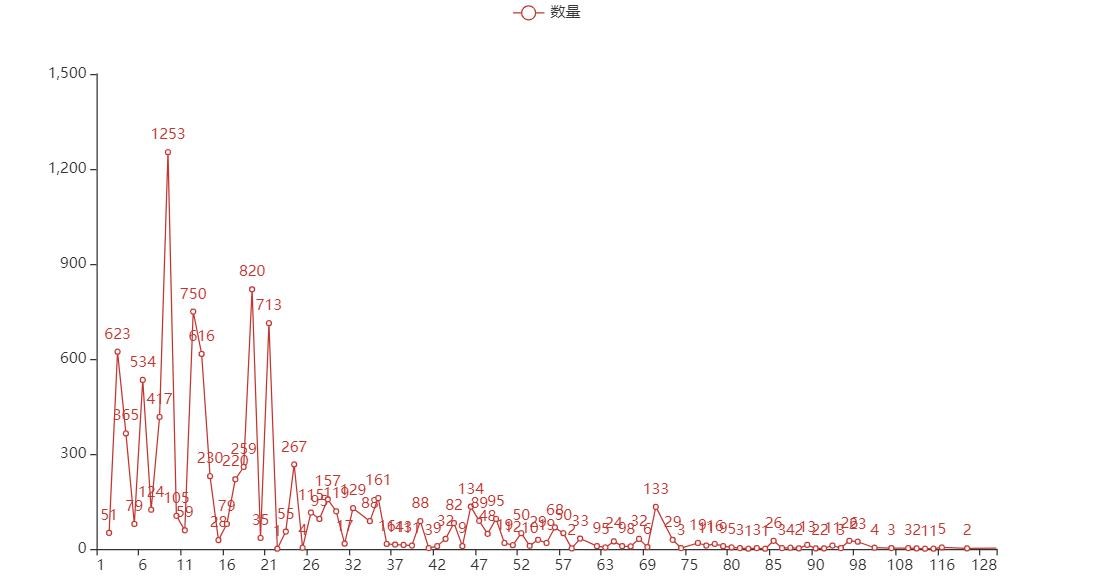


图 8 各品牌的二手车销售量

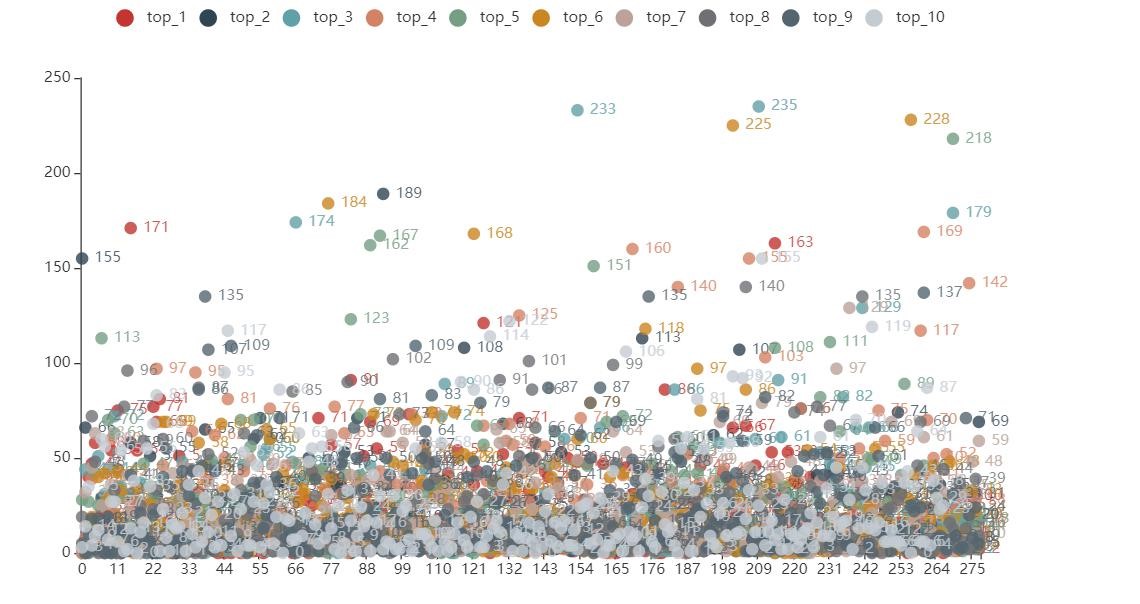


图 9 品牌销售量前十的销售天数分析

从图中我们观察到品牌对销售量的影响居于显著作用，好的品牌可以助推二手车的销售量，然而对于不同车辆的销售周期而言，好的品牌夜有销售周期天数较长的存在，故而品牌对二手车的销售周期天数无显著影响。

之后，在对车系、车型 id、里程、车辆所在城市 id、排量等种类较多的情况进行了如品牌的取销量前十与后十的方式进行了分析。实验过程中，对该类数据的分析如下图所示：

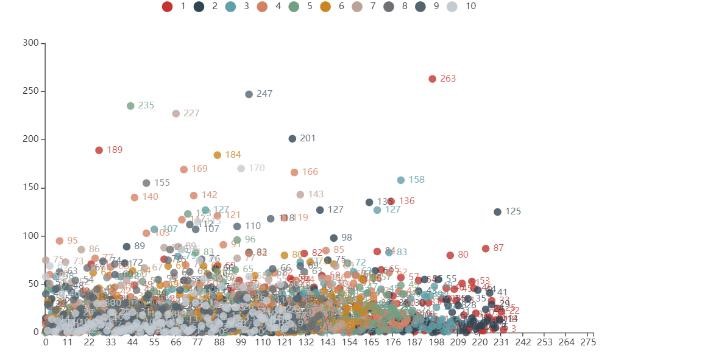
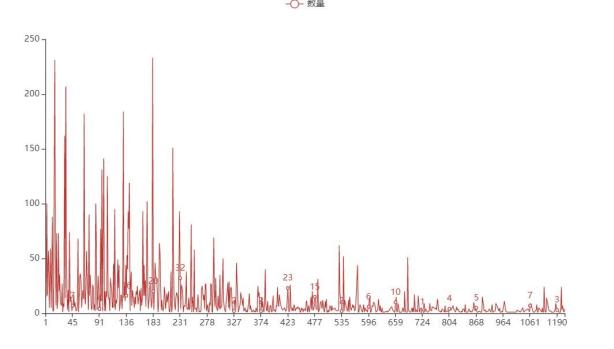


图 10 车系销量分布情况 图 11 车系销量前十的销售周期天数情况

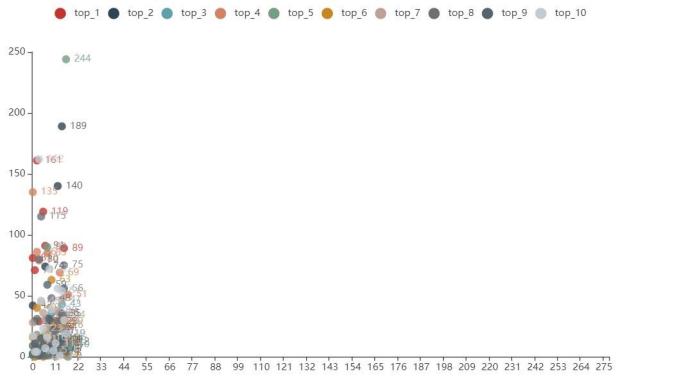
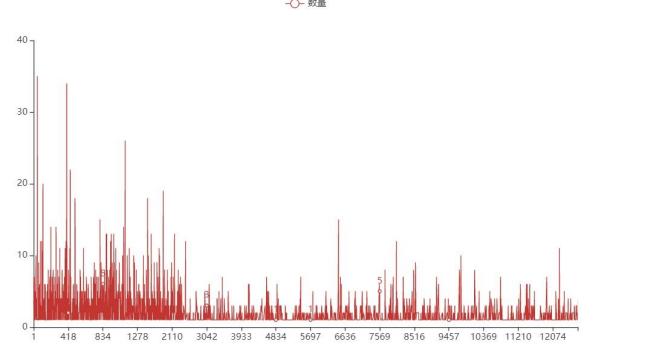


图 12 车型 id 的销售周期天数基本情况 图 13 里程的前十的销售周期前十的基本情况

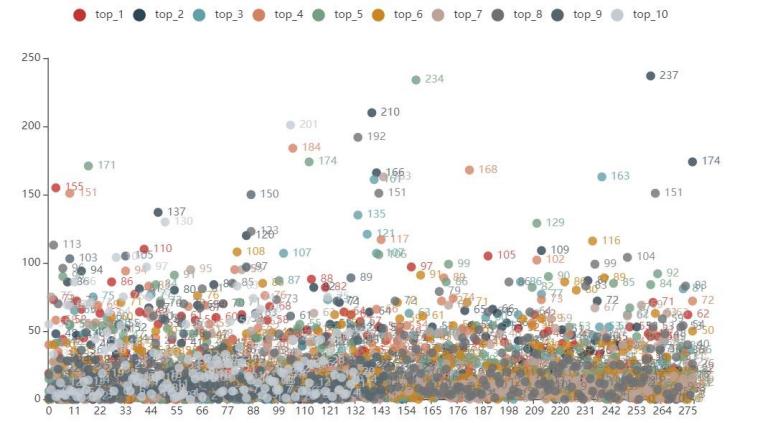


图 14 车辆所在城市 id 销量分布图 图 15 排量销量前十对销售周期天数的影响分布图

对于车辆颜色、过户次数、燃油类型这种种类较少的则采用整体分析，观察其不同的种类对销售周期天数的影响

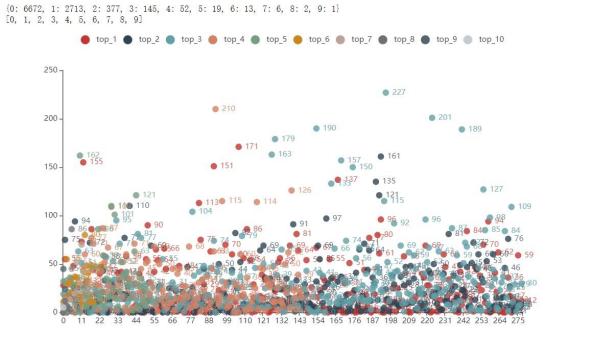
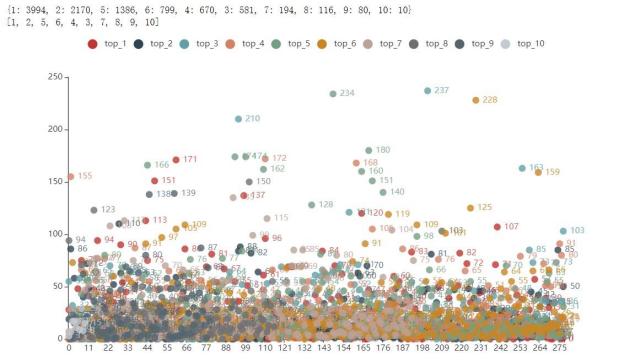


图 16 车辆颜色对销售周期天数的 图 17 过户次数的销售周期天数分布影响 图

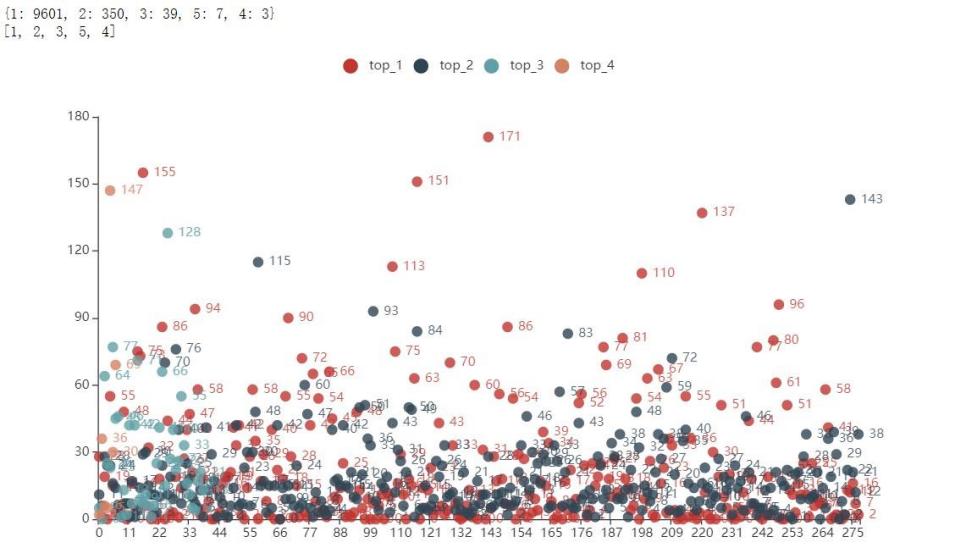


图 18 燃油类型对车辆销售周期天数的影响

### 5.2.3 对销售周期进行分区间进行统计整理

通过对销售周期进行分区间整理，通过对销售周期进行分组，可以更好的实现对销售周期的分析。对销售周期进行分组时，我们分为销售周期天数为 0， 1，

2， 3， 4， 5-7， 8-17， 18-50， 51-91， 92-237，这样可以帮助我们更好的整理数据之间的关系，并进行数据分析。在进行区间分析的时候，我们参考了 2.2.1 的处理方式，并进行了简略版的处理与分析。

首先，通过对销售周期区间的大致数量分析，让我们更好的了解数据图谱的分析。

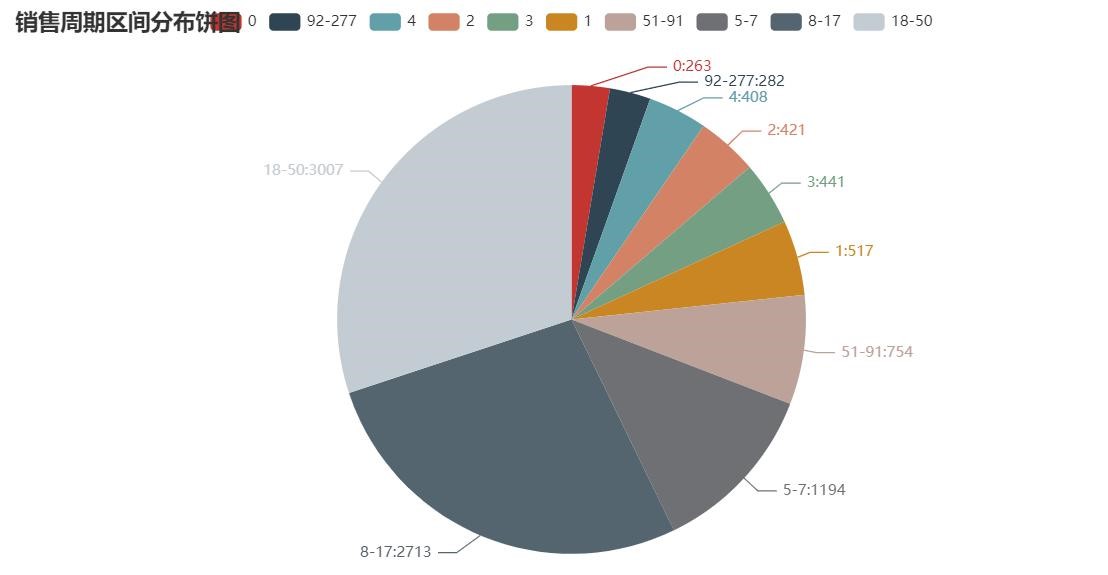


图 19 销售周期区间分布情况

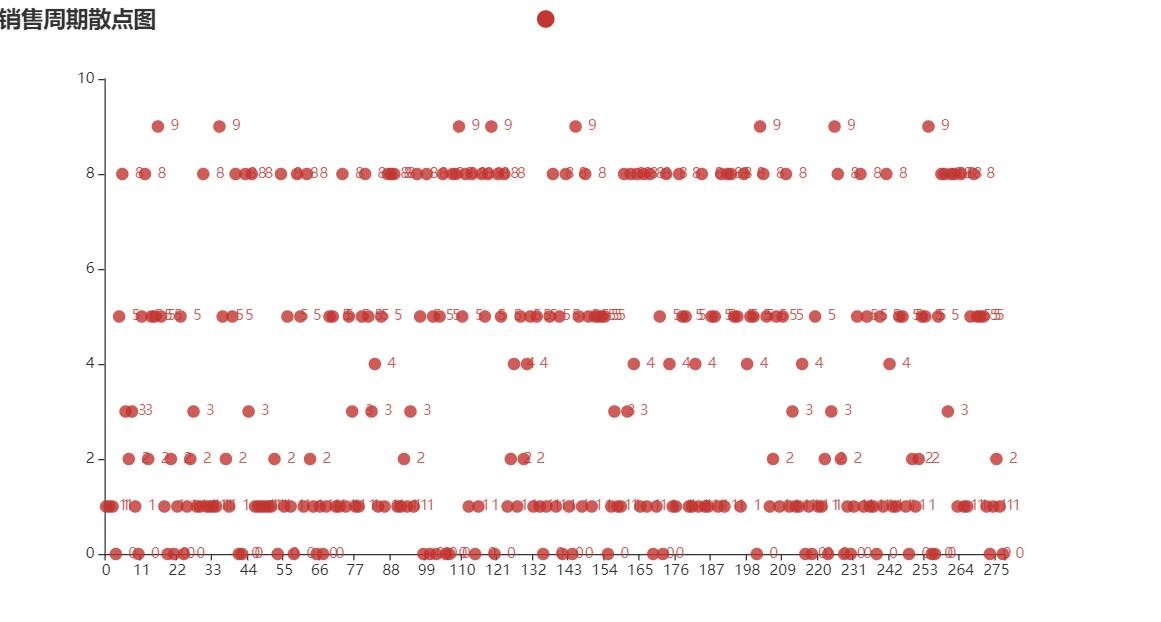


图 20 整体的销售周期区间的分布情况

其次，我们参考了品牌对销售周期的影响，绘制销售量前十的品牌的销售周期区间的变化

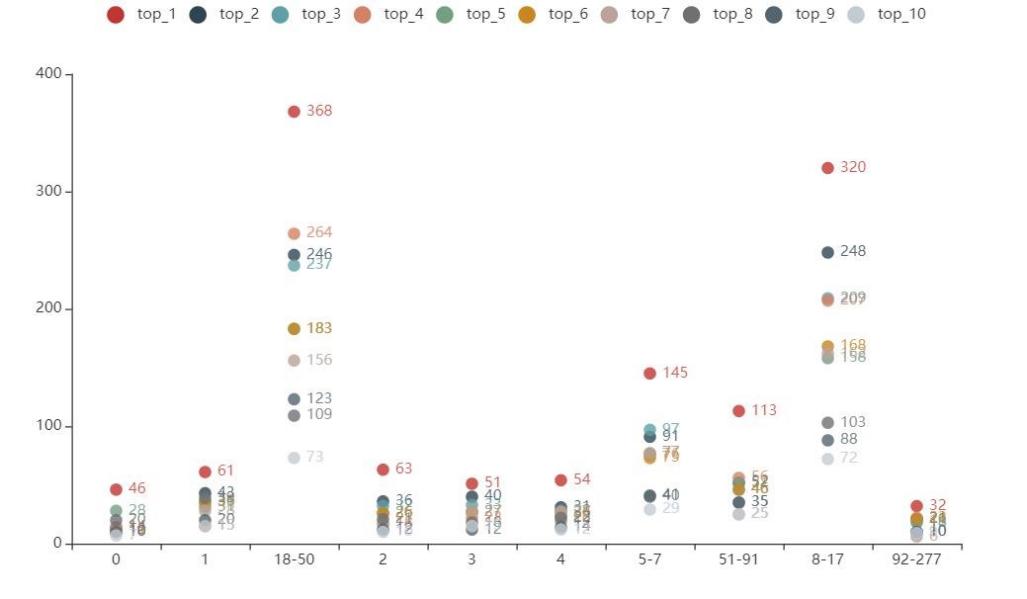


图 21 品牌销量前十的销售区间的数量分布从图中，可以看出，18-50 与 8-17 的区间中销售量较高。

之后，我们还进行了车辆颜色、过户次数、载客人数、燃油类型等数据分类清新的数据进行可观察，依次查看对二手车销量情况的分析。

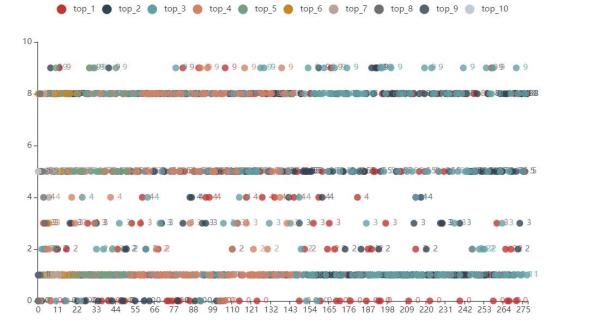
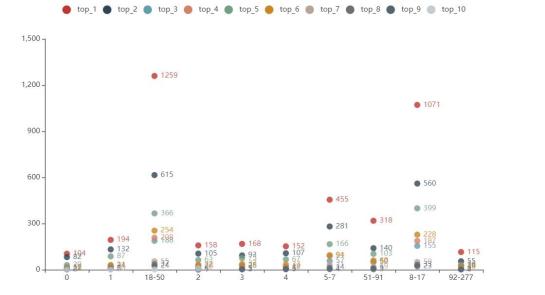


图 22 车辆颜色相应区间数量分 图 23 过户次数对应的整体的销售析 周期区间分布情况

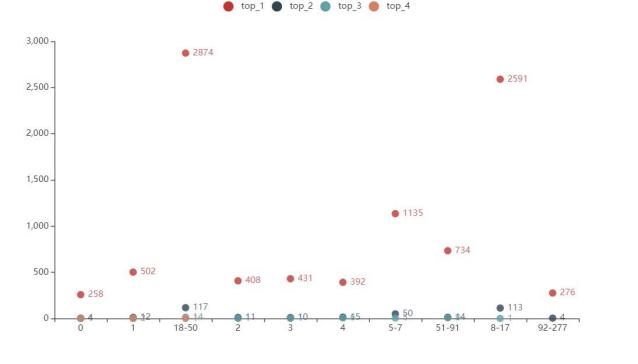


图 24 载客人数对应的整体的销 图 25 燃油类型相应区间数量分析售周期区间分布情况

从图中可以看出，车辆颜色、过户次数、载客人数、燃油类型这些数据对二手车的销量周期存在影响，但影响性不大，相应的情况回造成二手车的销量改变，但无法对二手车的整体销量周期造成影响。

## 5.3 影响因素进一步挖掘

### 5.3.1 方法与思路说明

（1）分析思路：利用特征排序（特征选择）[16]对成交周期进行分析与模型构

建，目的在于减少特征数量、降维，使得模型的泛化能力更强，减少过拟合。除此之外，考虑到特征值的实际含义，还可以增强对特征和特征值之间的理解。

（2）分析方法：借助单变量特征选择中的 Pearson 相关系数进行分析，单变量特征选择能够对每一个特征进行测试，处理特征和响应变量之间的关系，根据得分舍弃掉不好的特征，从而衡量变量之间的线性关系。该方法通常对于理解数据具有较好的效果[17]。

### 5.3.2 基于皮尔逊相关系数的模型架构

（1）数据处理：将附件 4 和附件 1 进行合并，即通过数据整合得到全面特征

与对应特征值。

（2）Pearson相关系数计算

皮尔森相关系数（Pearson correlation coefficient）也称皮尔森积矩相关系数 (Pearson product-moment correlation coefficient) ，是一种线性相关系数。皮尔森相关系数是用来反映两个变量线性相关程度的统计量。借助 pandas 进行相关系数的计算，主要代码如下：

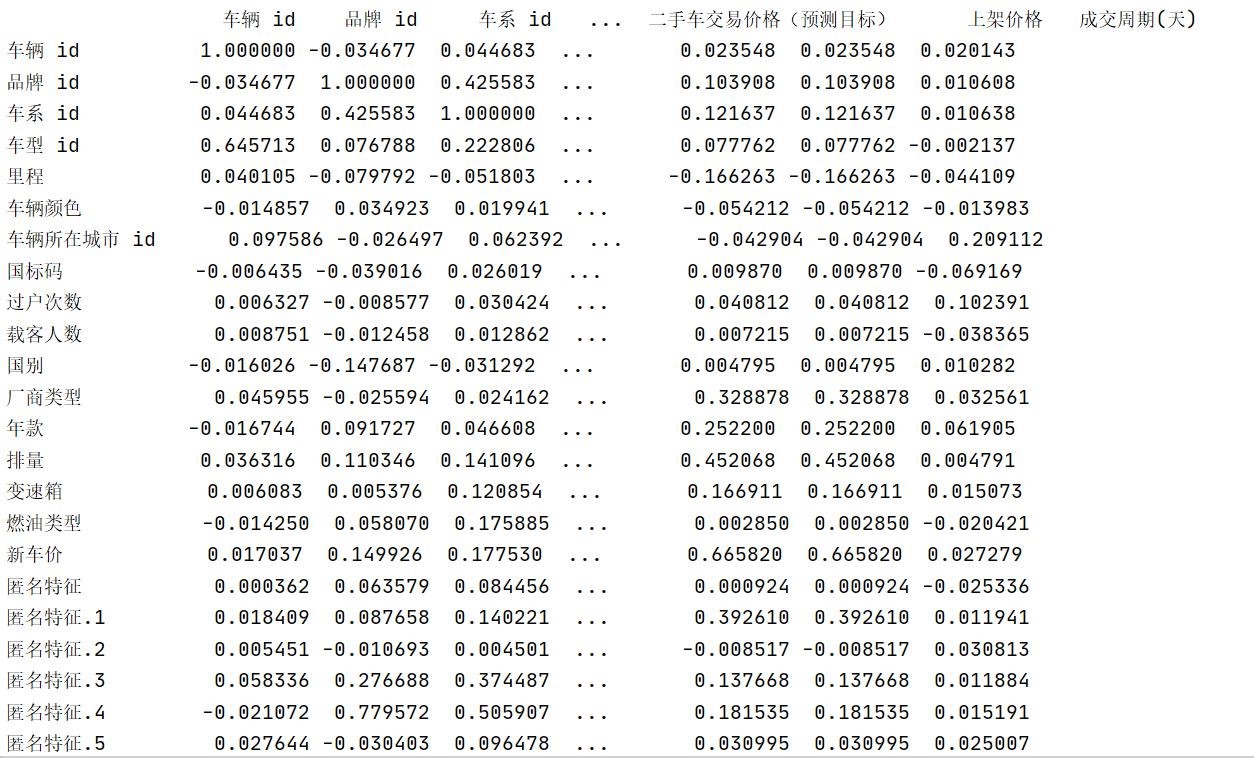
data = pd.read\_csv(r"C:\Users\Lenovo\Desktop\总.csv") df = pd.DataFrame(data)

print(df.corr())

#print(df.corr('spearman'))

#print(df.corr('kendall'))

通过计算得到的部分结果如下所示：



根据以上得分舍弃掉不好的特征，从而衡量变量之间的线性关系，为进一步分析提供有力依据。其中，对交易周期影响因素从大到小的影响为：

|  |  |
| --- | --- |
| 影响因素 | 相关系数 |
| 成交周期(天) | 1.0 |
| 车辆所在城市 id | 0.23534563229055183 |
| 过户次数 | 0.10216949426423909 |
| 日期差 | 0.08962993102729372 |
| 上架价格 | 0.0707070365524042 |
| 二手车交易价格（预测目标） | 0.06494542929683957 |
| 国标码 | 0.06431636303629248 |
| 年款 | 0.060578636444364176 |
| 厂商类型 | 0.0487196933921786 |
| 匿名特征.12 | 0.04655001366089112 |
| 里程 | 0.03478575214638492 |
| 匿名特征.7 | 0.028141715098782813 |
| 匿名特征.13 | 0.027435873114480064 |
| 匿名特征.9 | 0.026132094159409053 |
| 载客人数 | 0.024889749547639145 |
| 新车价 | 0.023500389301353265 |
| 匿名特征.2 | 0.022423140276252595 |
| 车系 id | 0.022367014670937595 |
| 价格差 | 0.02167497669814242 |
| 匿名特征.5 | 0.021066990055458513 |
| 匿名特征 | 0.020783016045806048 |
| 车辆 id | 0.018827544991527216 |
| 匿名特征.1 | 0.015929103972915033 |
| 品牌 id | 0.014789694600914423 |
| 匿名特征.4 | 0.014051087602147447 |
| 排量 | 0.009639581177829268 |
| 变速箱 | 0.009395586940910267 |
| 车辆颜色 | 0.007567743214531442 |
| 国别 | 0.004328849560166613 |
| 匿名特征.8 | 0.003765964452441954 |
| 燃油类型 | 0.002944137358971221 |
| 匿名特征.3 | 0.0015831774038671997 |
| 车型 id | 0.0005102795570690364 |

表 1 相关系数分析表

（3）Pearson相关系数绘图展示与分析

根据以上计算，绘制相关系数图如下所示：

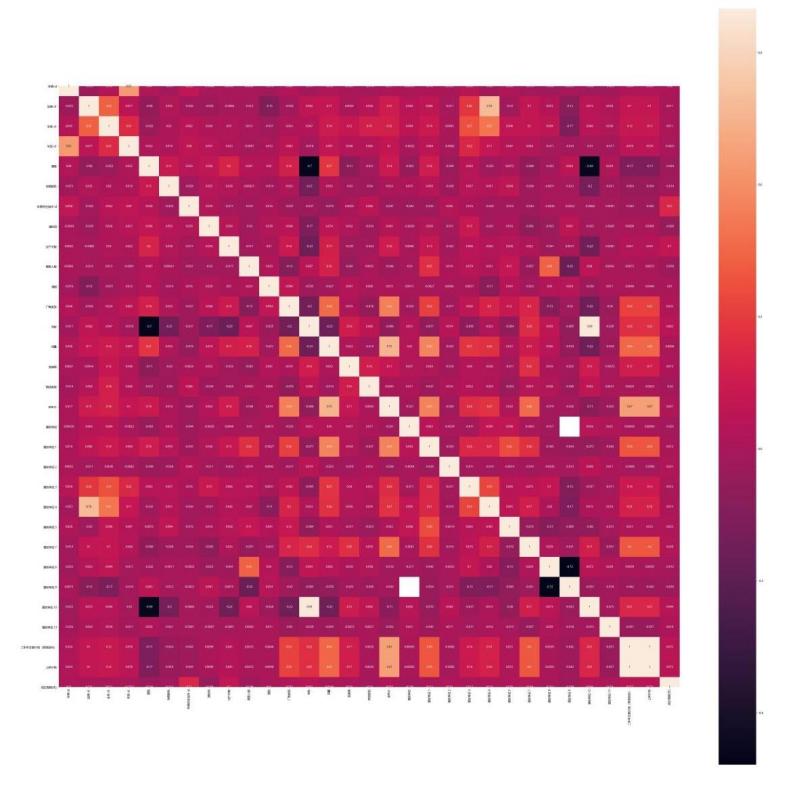


图 26 销售周期相关系数图依据相关系数图可以看出，成交价格仍然是影响成交周期强有力的因素。再进一步分析过程中，注意到价格和日期对成交周期的影响因素较大，其中价格对交易周期的相关系数为 0.0707070365524042，在对日期进行分析时，考虑到日期的不同，进行日期差进行计算，其中日期差 = 上架时间 – 注册日期。对日期差进行相关系数分析，得到的相关系数为 0.0896299310272937。

对比绘图展示，如下图：



图 27 交易周期与日期差

从图中可以看出，日期差较小的，其交易周期也较短，可以更好的进行二手车的出售。

### 5.3.3 模型检验

将预测值改为周期，进行回归预测



得分最高的时候选择的特征就是对周期影响因素最大的特征

### 5.3.4 模型评价

采用与问题一相同的模型，基于 xgboost 可解决过拟合、能够减少计算的优点，之后利用 xgboost 进行模型优化。进一步提升模型的性能。

## 5.4 采用关键因素加快在库车辆的销售速度

需要加快门店在库车辆的销售速度，本报告欲结合以上分析得出的关键因素采取对应的手段，并说明这些手段的适用条件和预期效果。

由问题二的分析、挖掘、皮尔逊相关系数计算等措施，得出影响二手车成交周期的最大关键因素为价格。因而上架的二手车如何定价显然成为了关键问题。

故而基于价格进行在库车辆销售速度加快策略分析：

1. 由于二手车的定价要依托于新车市场的行情与售价，故而不能以卖家期望盈利值去定价，应当注意行市变化。注意该种车辆的新车价格，由该价格确定合适且能够最大限度满足盈利预期的二手车定价，自然会有消费者在权衡利弊之下倾向于二手车。
2. 依据二手车上架后买家的反馈，要及时调整定价，常常更新以达到盈利最大化。对于迟迟未卖出的二手车而言，降低价格时候采取多次少量的手段，即不能一次性降到利润最小值对应的最低价格，而是以一定价格步长一次次逼近最低价格。这样不至于利益直接跌到最小。

# 六、其他可探究问题分析

依据所给样本数据集，下文给出本报告认为值得研究的问题与相关思路。

## 6.1 淡旺季之于二手车买卖的影响探讨

经统计发现二手车买卖存在淡旺季，卖家一般都会选择在年底清货，避免下一年该车贬值。由于二手车价值是按年来折算的，12 月和 1 月即使仅仅相差一个月，虽然就实际性能来讲，相差不大，但对于车龄来说，的确相差一年。虽然依据现有数据并无 12 月销售情况，但可以推测出 12 月很大程度上可以算是二手车买卖的旺季。另外一方面，到了年底，买车的也会增多，对二手车的需求会增大，这也是是导致年底成为旺季的原因之一。除此之外，夏季（暑假）也可以看做是二手车买卖旺季，二手车价格最低的时候，应该是在最炎热的 7 月下旬左右，这个时候因为天气炎热，很少有人去买二手车，车商为了促进交易一般会给予一定的优惠，所以这个时候买二手车是最划算的，想购买二手车的也会选择这个时候。

根据现有数据，得到统计结果如下：



依据以上情况分析，淡旺季之于二手车买卖影响不可小觑。

由以上分析，本报告给出针对淡旺季情况的增加盈利的思路：由历史记录数据不仅要得出二手车全行业的淡旺季，而且要根据不同车型等车辆特征得出对应的“基于不同车辆特征本身的淡旺季”。从而在旺季增加该种车辆的库存，达到盈利的目的。淡季可以采取降价促销、优惠活动等措施清理库存，保证不亏本。

## 6.2 处置积压库存车辆策略分析

在二手车销售的过程中，必须保证库存的新鲜度，理论上说成交周期越短越好。二手车受到新车促销降价影响很大，有很强的季节性风险，有明显的淡旺季之分。需要在合适的时间压缩库存，处置积压车辆。也就是说，二手车卖家并不是全年盈利，而是旺季盈利，淡季保本（保证不亏本）。

由以上分析，本报告给出针对处置积压库存车辆以增加盈利的思路：针对积压车辆带来的问题，可以采取旺季提高利润百分点，以达到弥补淡季亏本的目的。

## 6.3 价格差带来的利润问题

在二手车买卖的过程中，鉴于市场或卖家需求、该种车辆存货积压等问题，存在更新价格的情况。需求增大时，抬高价格以达到盈利目的，但是当该车辆长期卖不出去时，需要降价处理。那么降价的“度”是本报告要探讨的。在急于处理车辆的同时，不可以一味以清货为目的，还要考虑到是否盈利的问题。故而变更价格时，针对降价的车辆，要衡量利润，即在车辆上架时候就可以根据以往销售记录确定该车辆最低价格，达到利益最大化的效果。即对于根据二手车的新旧程度，控制二手车的价格差，不可一味只顾赚钱，不管车辆的价格空间。

由以上分析，本报告给出价格差带来的利润问题的解决思路：在二手车的销售过程中，考虑到整体的销售情况，为保证车辆的合理销售，应当在车辆上架之前依据二手车卖家的盈利预期计算该批车辆利润，确定在变更价格的过程中的最低车价。

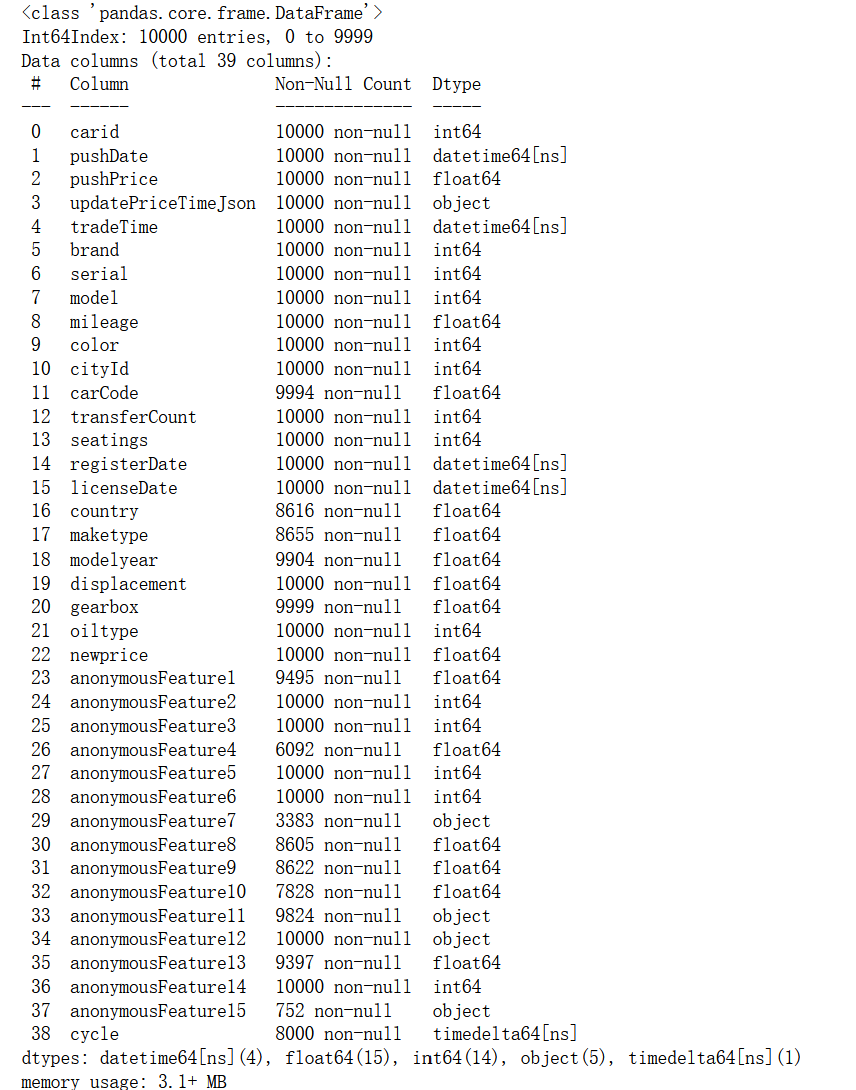
# 七、成交周期预估建模分析

## 7.1 回归预测成交周期

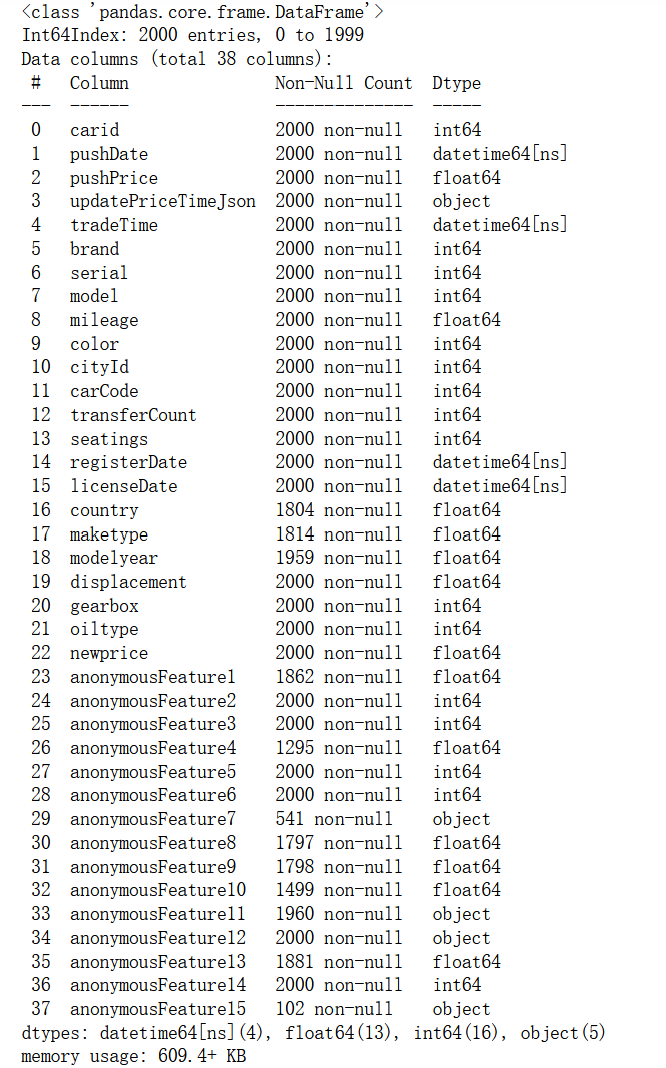
### 7.1.1 数据分析

根据题意，附件1和附件4合并作为训练集，附件5和附件2合并作为测试集根据所给的变量信息，初步判断，tradeTime, registerDate, licenseDate, pushDate，pullDate，withdrawDate为时间类型，因此在读入数据的时就直接通过设置parse\_dates参数转化成时间类型（datetime64[ns] ）

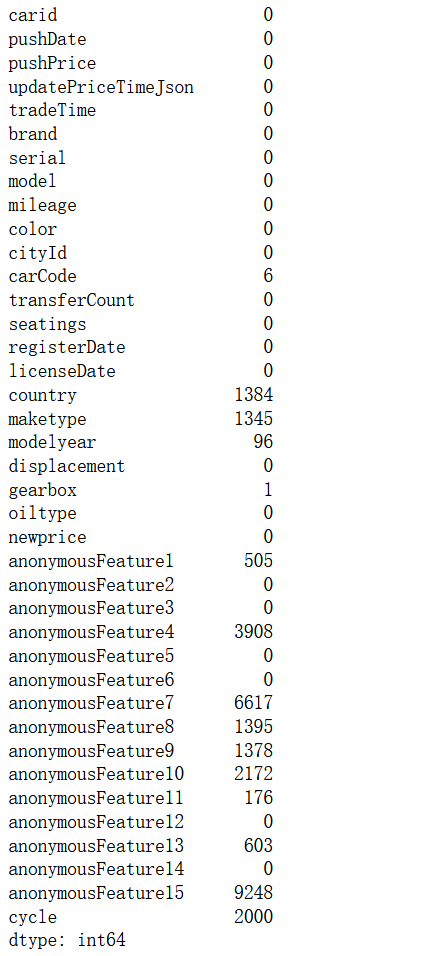
查看训练集的数据情况



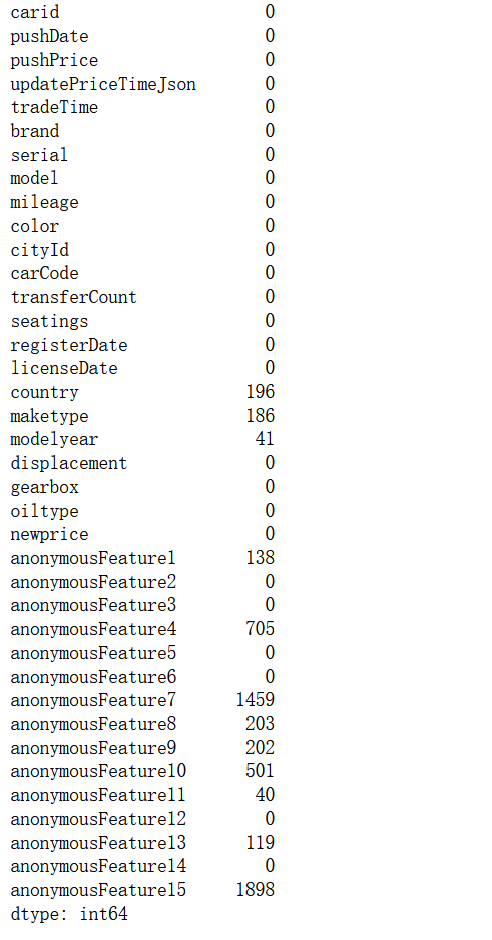
测试集的数据情况



训练集的缺失值



测试集的缺失值



画图显示缺失情况

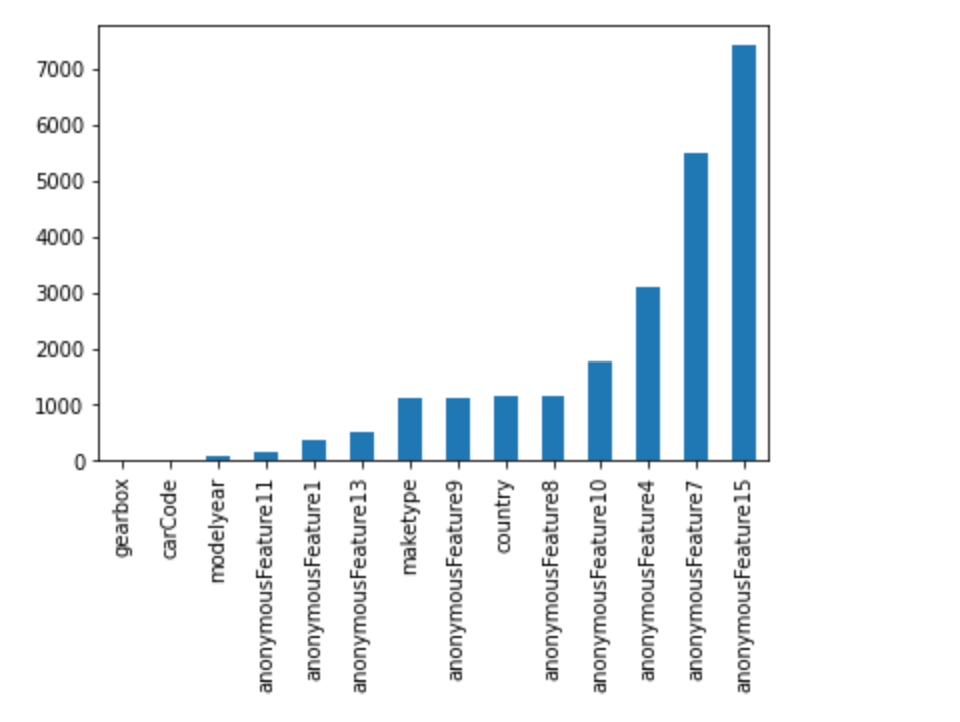


图28 训练集的缺失情况

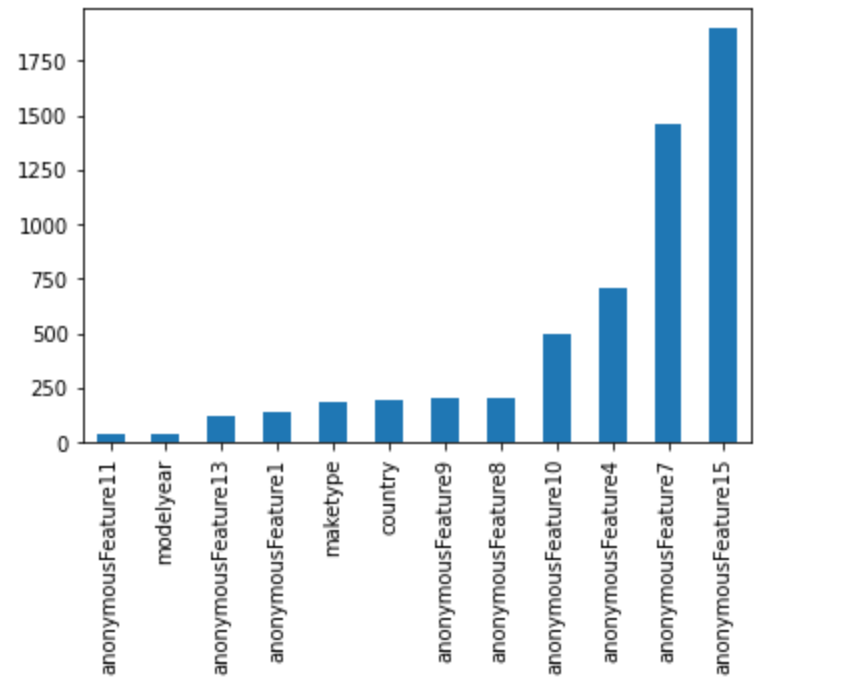


图29 测试集的缺失情况

无效矩阵的数据密集显示：

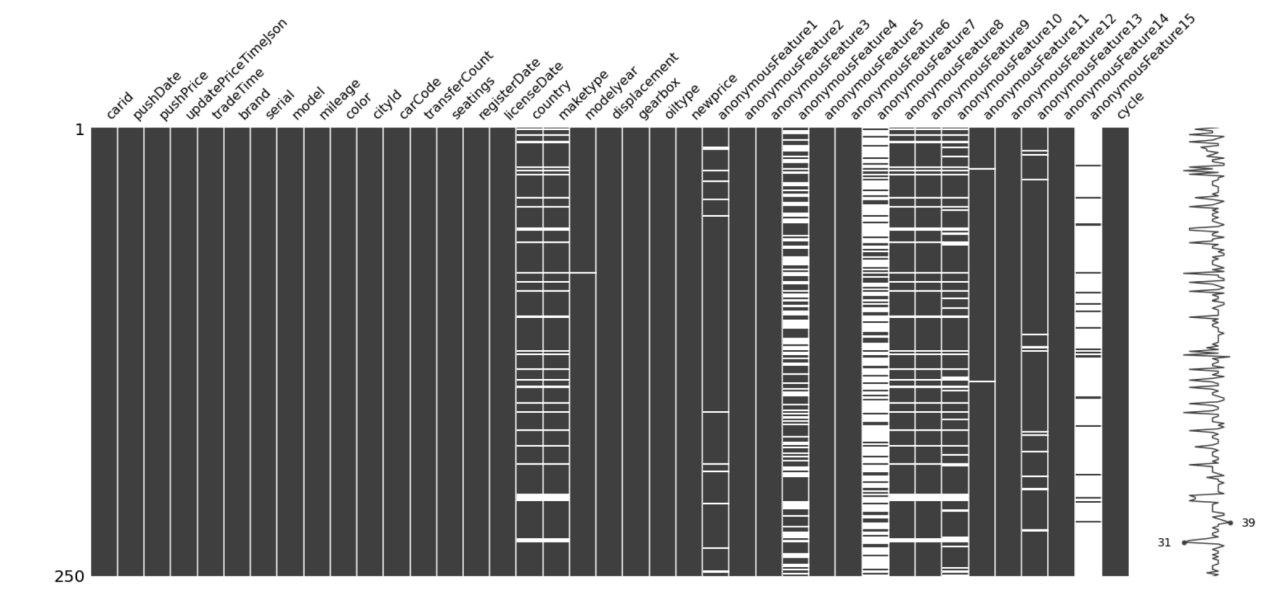


图30 训练集

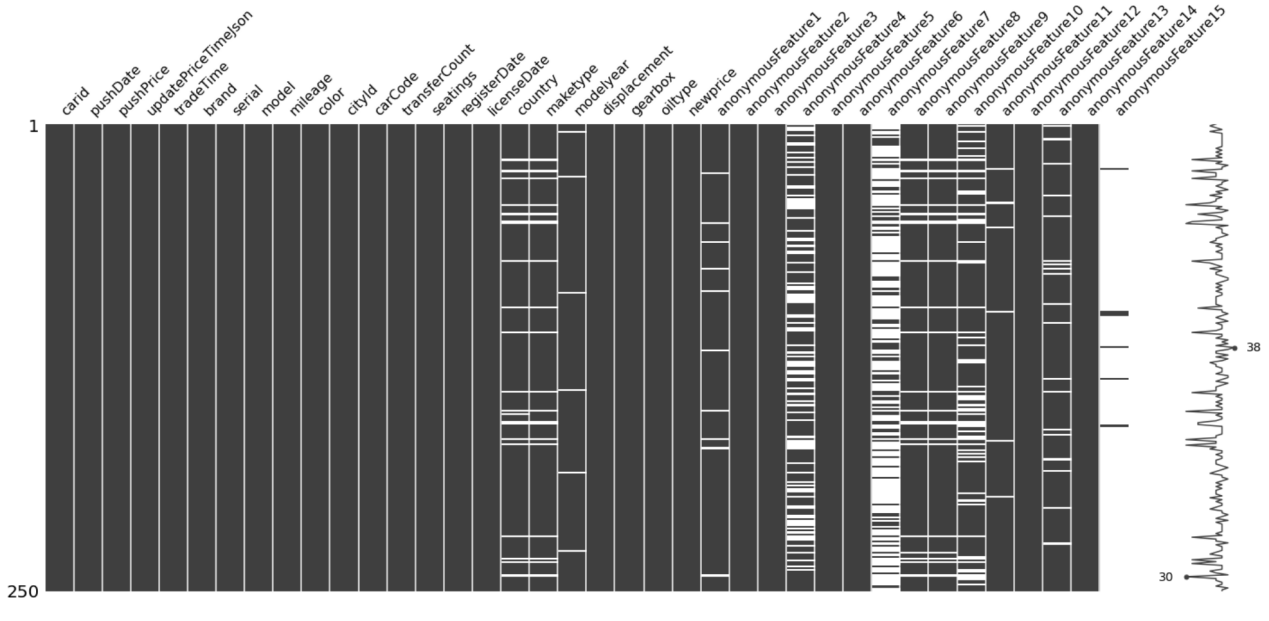


图31 测试集

列的无效的简单[可视化](https://so.csdn.net/so/search?q=%E5%8F%AF%E8%A7%86%E5%8C%96&spm=1001.2101.3001.7020)

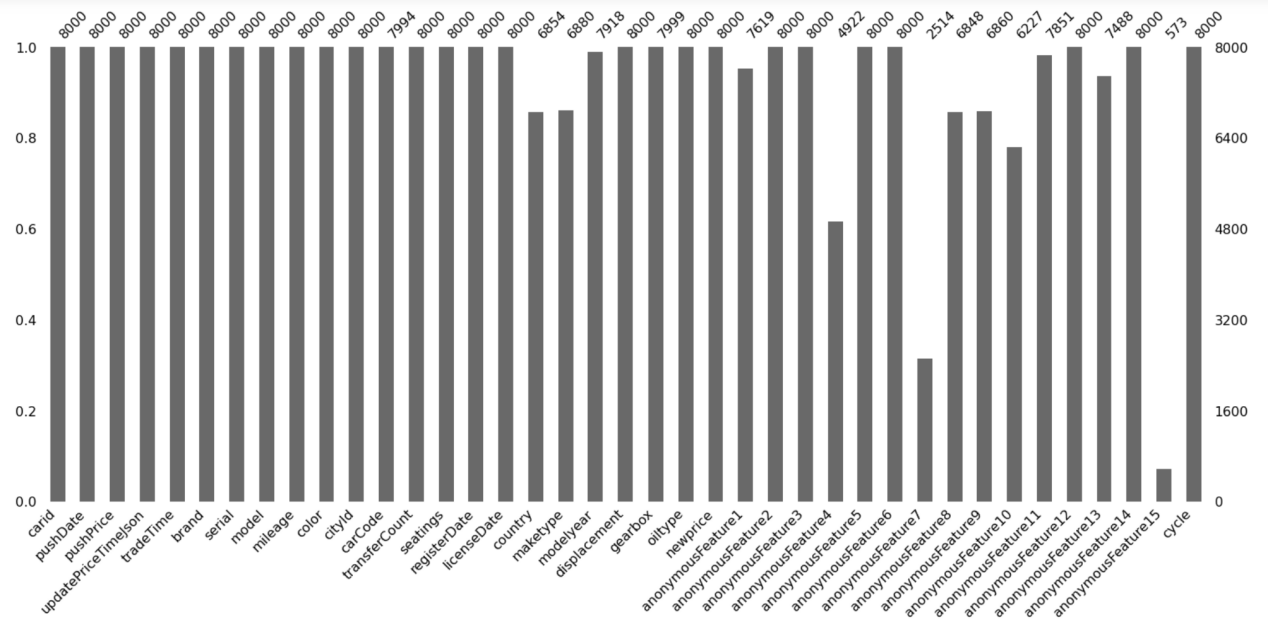


图32 训练集

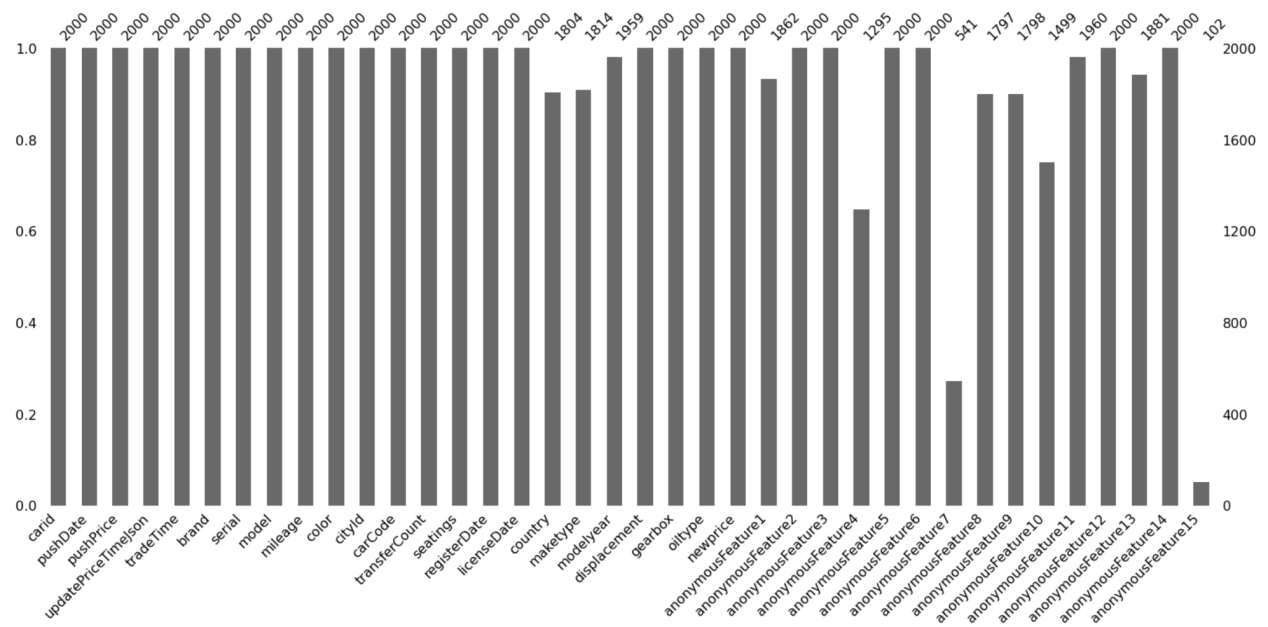


图33 测试集

删除nan存在的过多的特征('anonymousFeature4','anonymousFeature7','anonymousFeature15')，

对于'anonymousFeature1','anonymousFeature8','anonymousFeature9','anonymousFeature10'

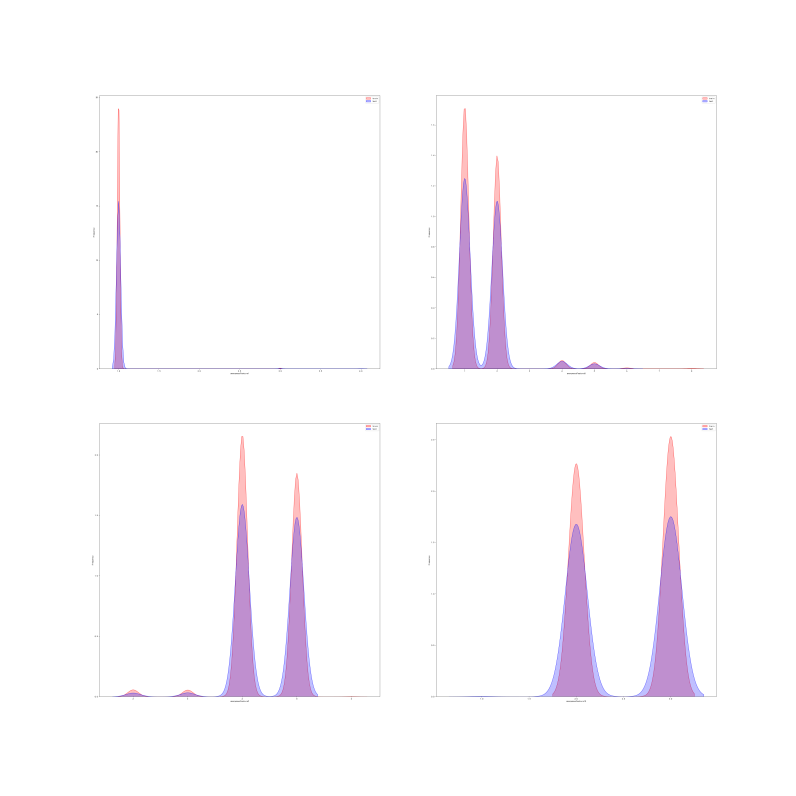


图34匿名特征1，8，9，10在测试集和训练集上的kde曲线

训练集和测试集中的分布相同，因此不做处理，其余缺失值较少的使用lgb树模型可以直接空缺，让树自己去优化。

### 7.1.2 删除无用特征

发现anonymousFeature1的严重倾斜，一般不会对预测有什么帮助，故删除

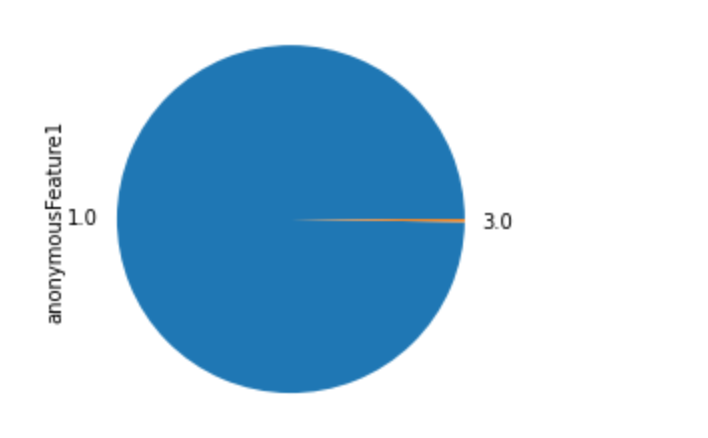
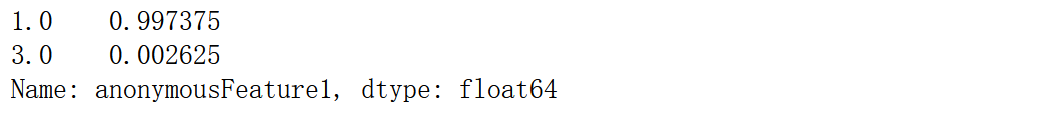
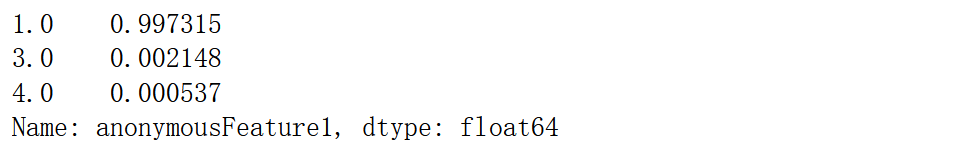


图35 训练集中anonymousFeature1的分布情况

anonymousFeature1训练集中的分布占比



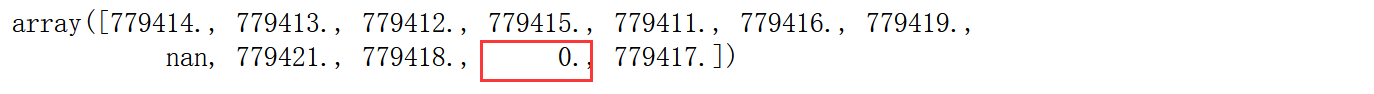
anonymousFeature1测试集中的分布占比



都出现了严重的倾斜，因此考虑直接删除此特征

### 7.1.3 删除异常

country里的0根据分析应改为779410



### 7.1.4 查看cycle的分布情况

成交周期=成交时间-上架时间

cycle=withdrawDate-pushDate

查看cycle的整体分布情况

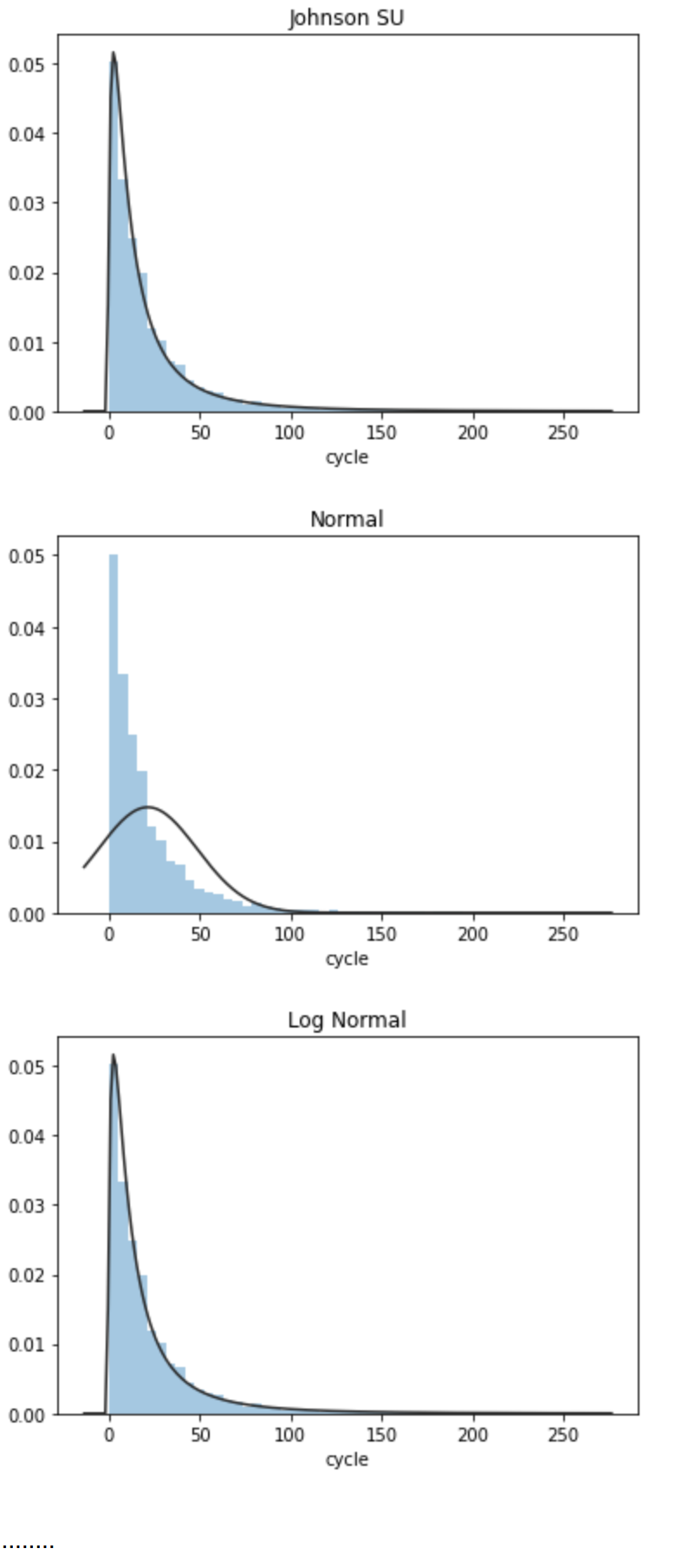


图36 cycle的整体分布情况

发现cycle不服从正态分布，所以在进行回归之前，它必须进行转换。显然对数变换做得很好

查看预测值的频次分布

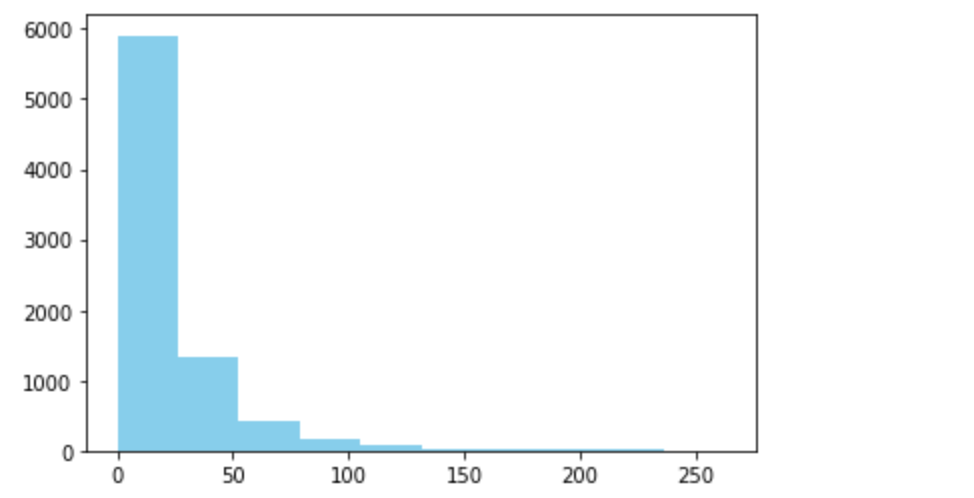


图37 预测值的频次分布

查看频数, 大于100得值极少，其实这里也可以把这些当作特殊得值（异常值）直接用填充或者删除。

进行log变换之后的分布较均匀

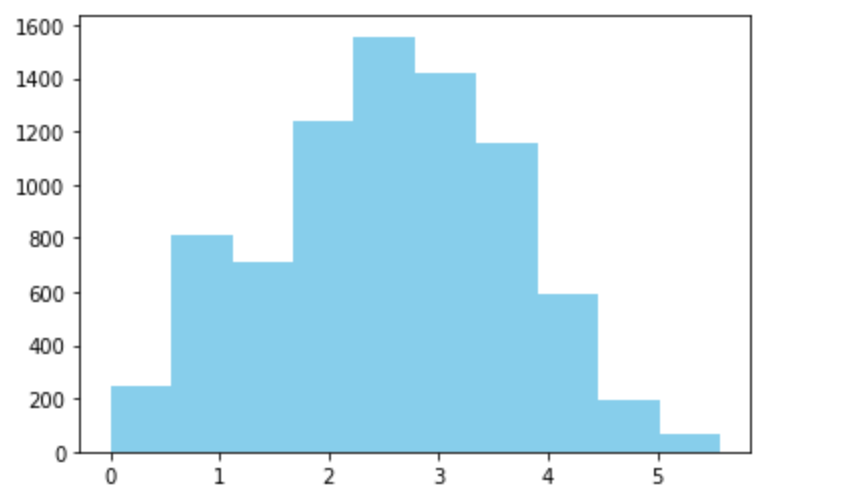


图38 log变换后的频次分布

### 7.1.5 数值特征的分布

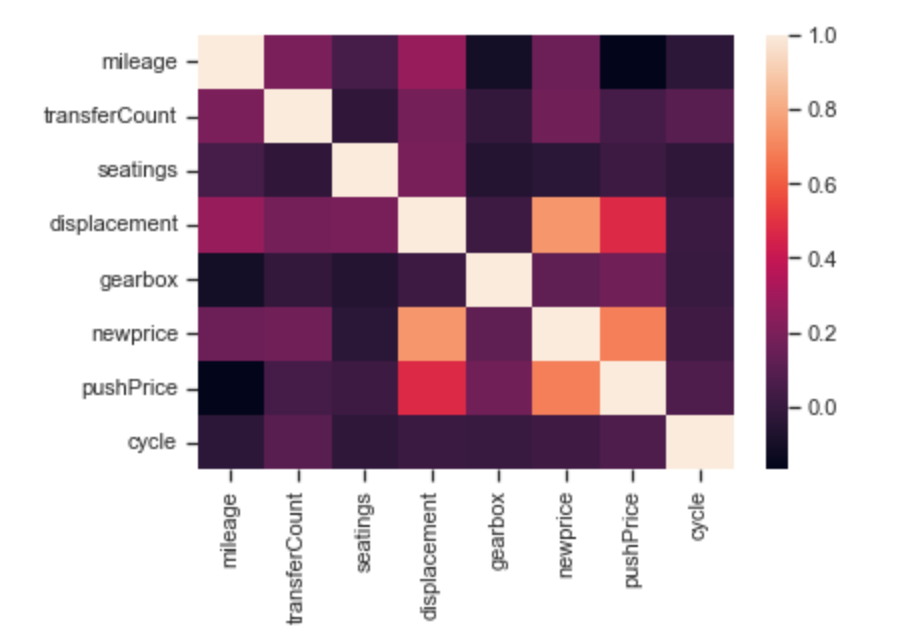


图39 数值特征分布

从图中可以显然看出可以看出pushPrice和transferCount对cycle的影响较高

### 7.1.6 类别特征与cycle(成交周期)的关系

可以通过小提琴图来观测类别变量在不同类别时，cycle的分布情况

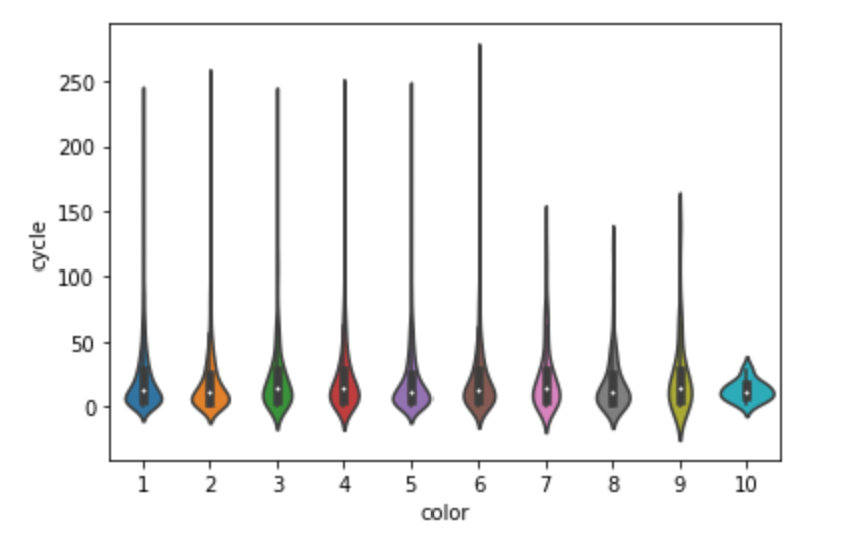


图40 cycle的分布情况

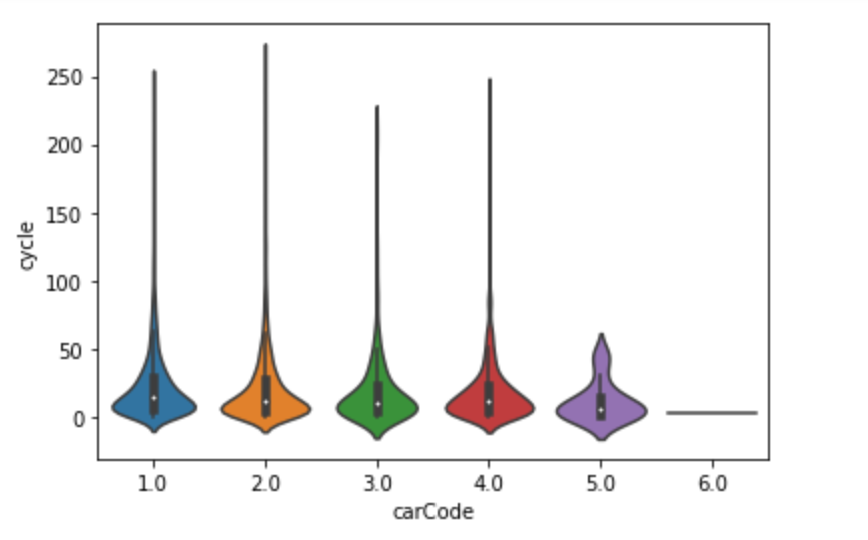


图41 cycle的分布情况

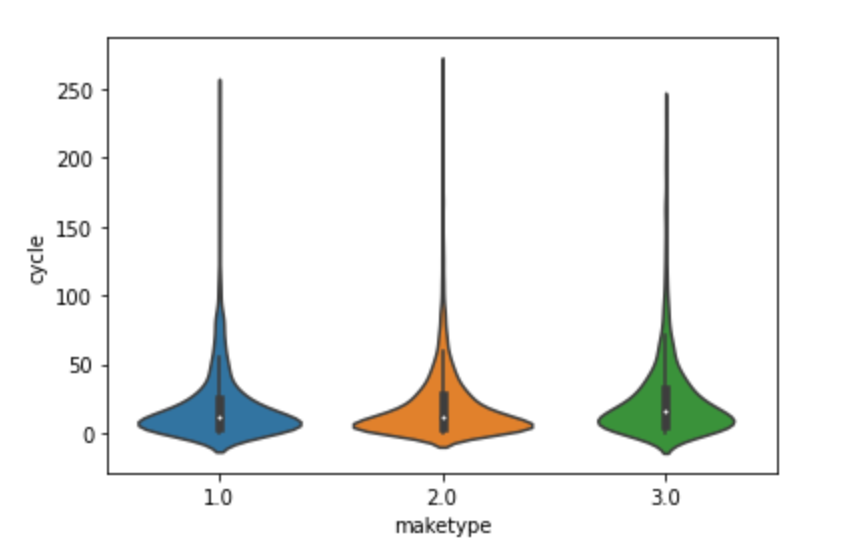


图42 cycle的分布情况

## 7.2特征工程

### 7.2.1 对定量数据检测异常值

'mileage','transferCount','seatings','displacement','gearbox','newprice','pushPrice'

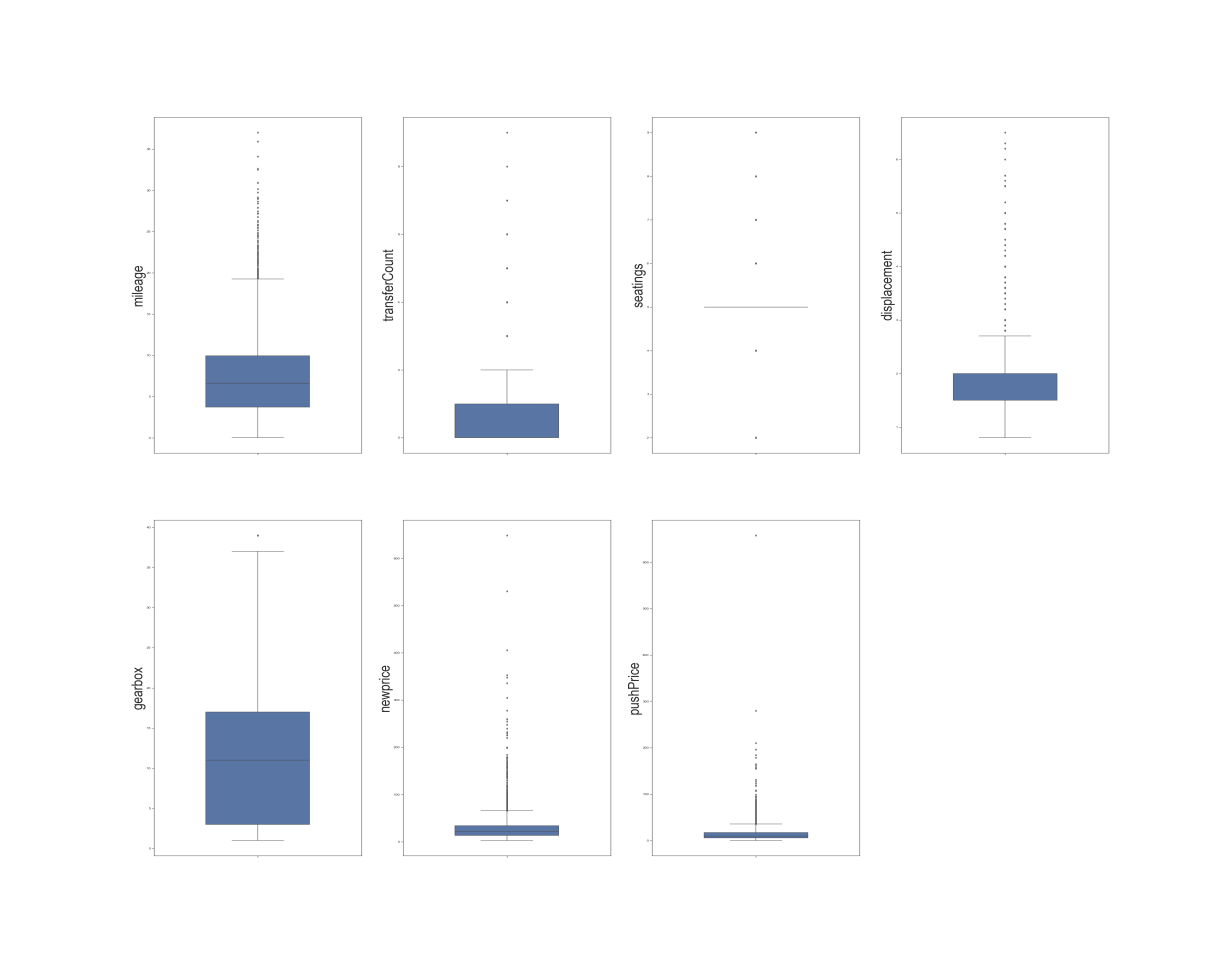


图43 异常值检测

可以明显的看出gearbox有较少异常值，而其他的六个特征异常值较多

### 7.2.2 构造新的特征

|  |  |  |  |
| --- | --- | --- | --- |
| pushPrice | updatePriceTimeJson | decreasing\_count | last\_updatePrice |
| 3.98 | {} | 0 | 3.98 |
| 4.5 | {} | 0 | 4.5 |
| 23.9 | {"2021-04-05": "23"} | 1 | 23 |
| 20.5798 | {} | 0 | 20.5798 |
| 12.28 | {"2021-05-20": "11.9"} | 1 | 11.9 |
| 18.5 | {"2020-12-29": "17.68", "2020-12-30": "17.8799", "2021-01-15": "16.8799"} | 3 | 16.8799 |

表2 特征分布

decreasing\_count:总共降价了几次

last\_updatePrice:最终的价格

### 7.2.3 构造的其他特征

对anonymousFeature11，anonymousFeature12，anonymousFeature13，old\_year，old\_year\_1的处理与上文相同，这里不再进行累述

### 7.2.4 基础周期特征的拆除

基础周期特征的拆除（年月日特征拆解）

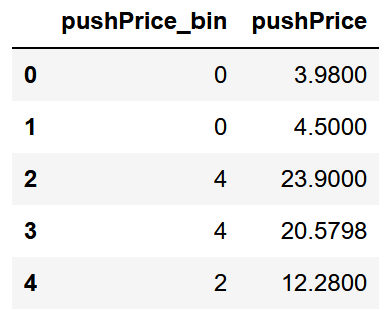
df\_train['pushDate\_year'] = df\_train['pushDate'].dt.year

df\_train['pushDate\_month'] = df\_train['pushDate'].dt.month

df\_train['pushDate\_day'] = df\_train['pushDate'].dt.day

### 7.2.5 对连续数据进行分箱处理

数据分箱-pushPrice



1. 离散后稀疏向量内积乘法运算速度更快，计算结果也方便存储，容易扩展；

2. 离散后的特征对异常值更具鲁棒性， pushPrice>35 为 6，对于pushPrice为 200 的也不会对模型造成很大的干扰；

3. LR 属于广义线性模型，表达能力有限，经过离散化后，每个变量有单独的权重，这相当于引入了非线性，能够提升模型的表达能力，加大拟合；

4. 离散后特征可以进行特征交叉，提升表达能力，由 M+N 个变量编程 M\*N 个变量，进一步引入非线形，提升了表达能力；

5. 特征离散后模型更稳定，车辆的价格区间，不会因为价格的轻微变化就变化

## 7.3模型的构造(lightGBM)

基于决策树算法的快速、分布式、高性能梯度提升（GBT、GBDT、GBRT、GBM 或 MART）框架。

模型的优点：

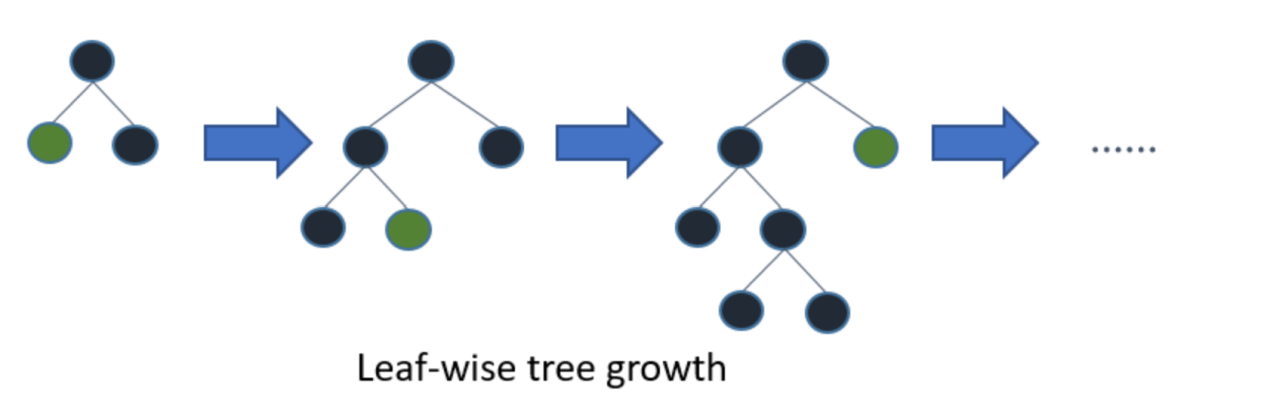
1更快的训练速度和更高的效率。

2降低内存使用率。 更好的准确性。

3支持并行、分布式和 GPU 学习。

4能够处理大规模数据

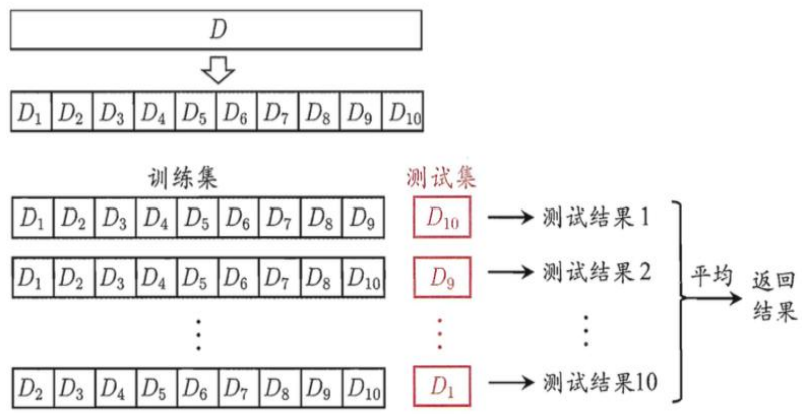
LightGBM 逐叶生长树（最佳优先）[7]。它将选择具有最大增益的叶子来生长。保持#leaf 固定，逐叶算法往往比逐层算法实现更低的损失

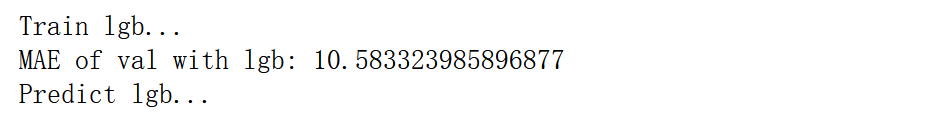


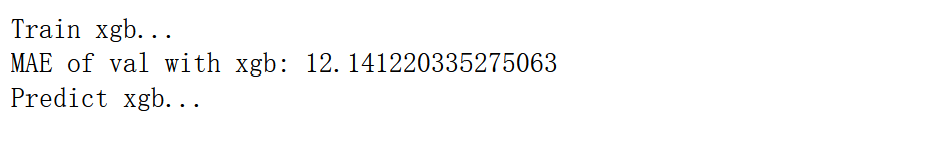
XGboost模型在上文已经介绍过了，这里不再进行累述

## 7.4 模型评估

模型验证，10折交叉验证

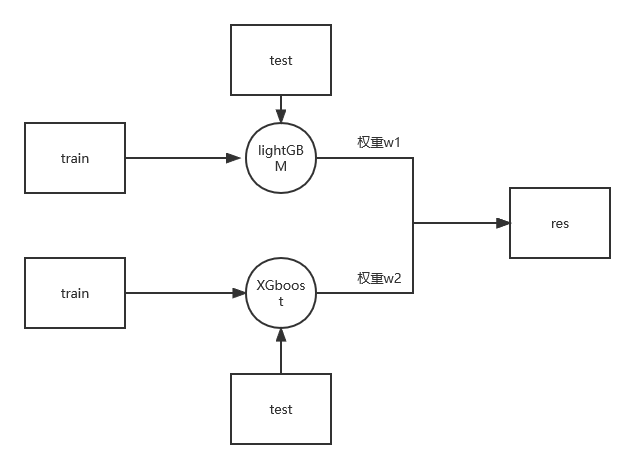




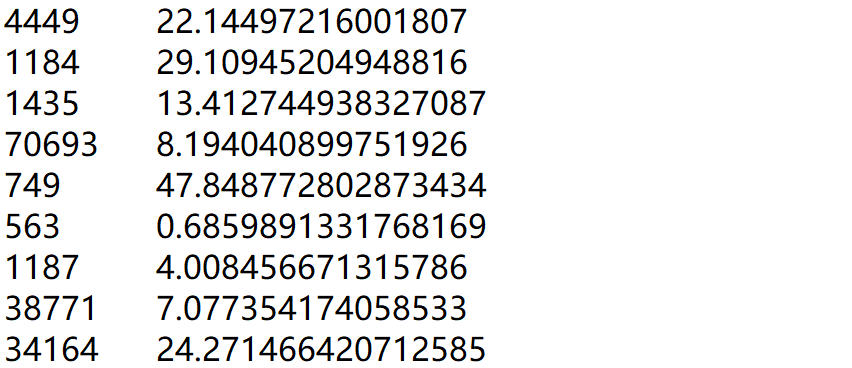


## 7.5 模型融合，简单加权融合

Res=(1-MAE\_lgb/(MAE\_xgb+MAE\_lgb))\*subA\_lgb+(1-MAE\_xgb/(MAE\_xgb+MAE\_lgb))\*subA\_xgb

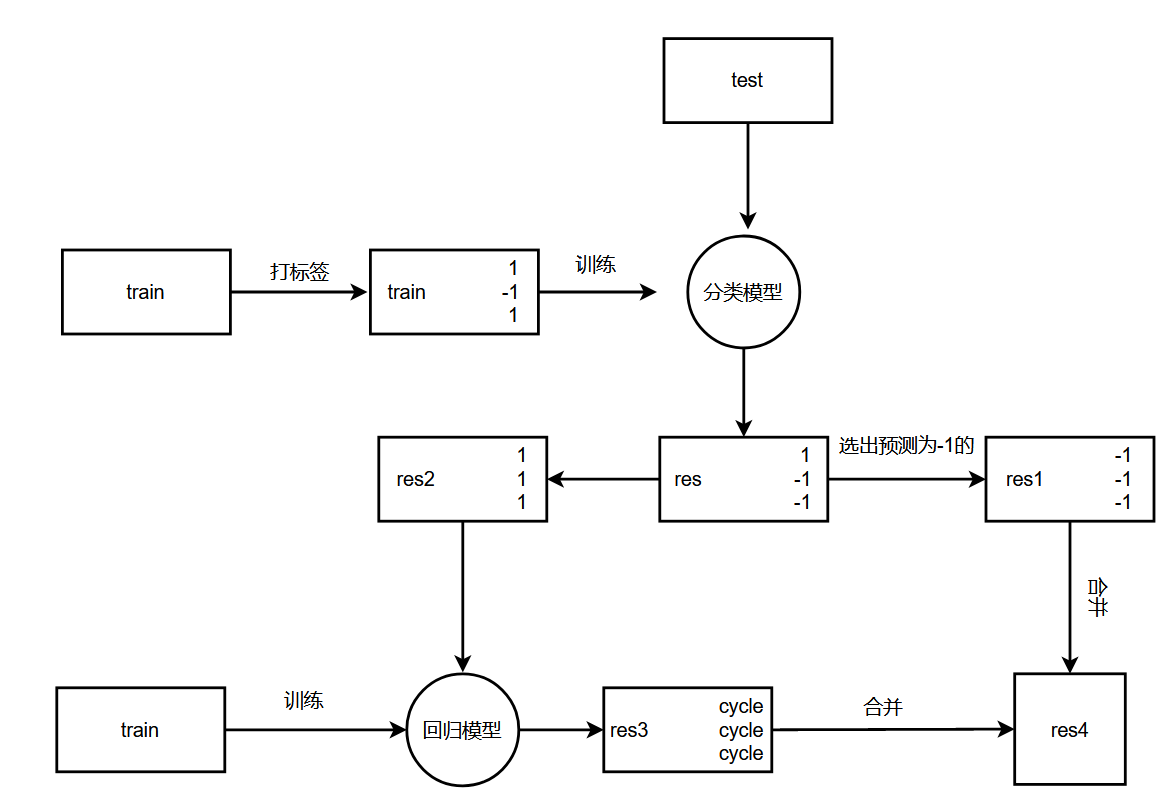


预测结果（部分）



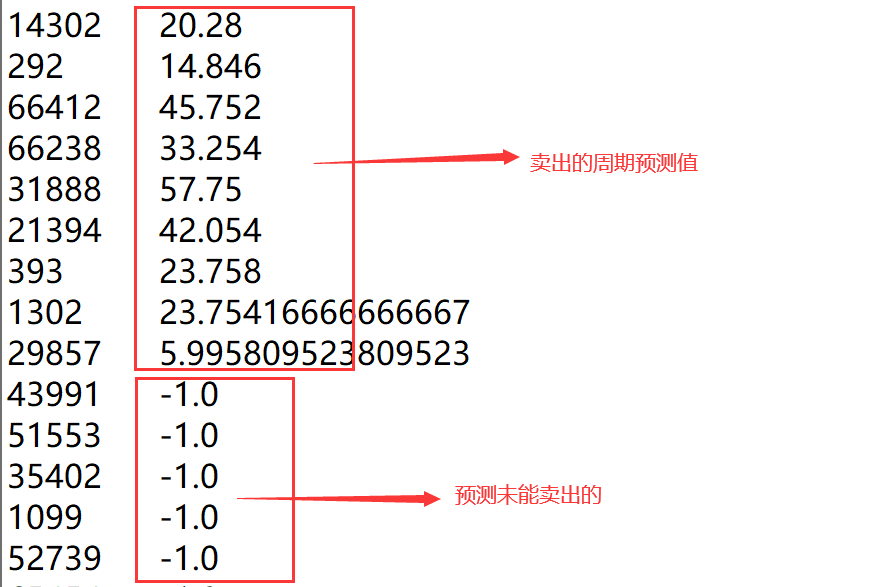
## 7.6 第二种思路

先建立一个分类模型，对卖出去和卖不出去（成交时间为空）的进行分类，将卖不出去的周期设置成-1，然后把能卖出的数据进行回归预测，最后将结果合并。



其余过程与方案一相同，只是增加了一个分类模型，分类模型使用的是随机森林。

预测结果如下：



# 八、 结合库存管理的门店经营模型构建分析

## 8.1 方案一：K近邻模型预测数据分析

### 8.1.1 问题分析

在复赛问题二中，需要对库存进行有效管理，以保障成本最小化的情况下，最大化门店的销售利润[18]。因此在本次研究中，考虑到门店的经营状况，以及具体的车辆信息未知，我们结合了初赛问题的附件1与附件4对车辆是否进行调价，什么时候调价以及调价幅度进行了预测，并根据预测信息对销售周期进行预测，最后根据销售信息得到每月的销售额信息。

### 8.1.2 模型假设

（1）在成本最小化的过程中，只考虑车辆资金占用成本和停车位占用成本，不考虑员工的人力成本等成本。

（2）营业额以月为单位进行评估。

### 8.1.3 模型介绍

在本次研究过程中，采用K近邻算法[19]进行模型架构，首先通过对辆是否进行调价，什么时候调价以及调价幅度等信息的预测调整，之后根据已有信息与预测得到的信息进行销售周期的预测。

### 8.1.4 模型算法

（1）算法思想：

对于任意的 n 为输入向量，其对应于特征空间一个点，输出为该特征向量所对应的类别标签或者预测值。

（2）算法思路：

如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。

（3）k近邻模型的三个基本要素：

k值的选择：k值的选择会对结果产生重大影响。较小的k值可以减少近似误差，但是会增加估计误差；较大的k值可以减小估计误差，但是会增加近似误差。一般而言，通常采用交叉验证法来选取最优的k值[20]。

距离度量：距离反映了特征空间中两个实例的相似程度。可以采用欧氏距离、曼哈顿距离等。

分类决策规则：往往采用多数表决。

### 8.1.5 模型评价

本次模型由于数据信息不完备，以及模型的K值对结果会产生巨大影响，结果存在一定的偏差，但模型的完备使得在一定信息的情况下，可以为门店的销售策略提供一定的参考价值，便于门店经营信息的分析。

## 8.2 方案二：遗传算法建模解题分析

### 8.2.1 问题分析

由问题二的提出可以将问题归纳为：在库存管理的过程中，既要考虑到在库车辆的情况，又要顾及到新收车辆的情况；除此之外，要对在库车辆（包括热销车辆和滞销车辆）适时进行调价，以保障门店的经营目标。问题梳理图如下：



图44 问题梳理

显然，该问题为一个优化问题，即在众多方案或参数值中寻找最优方案或参数值，以使得某个或多个功能指标达到最优，使得系统的性能指标达到最大值。由以上分析，本报告欲采取智能优化算法中的遗传算法[21]给出优化方案。

### 8.2.2 符号说明

|  |  |
| --- | --- |
| **character** | **Description** |
| T1 | 车辆上架时间 |
| T2 | 车辆成交时间/下架时间 |
| t | 车辆调价时间 |
| ai | 车辆停车位占用成本（该成本以天为单位计算，i指carid） |
| bi | 车辆进价（i指carid） |
| ci | 车辆定价（i指carid） |
| wi | 日资金占用成本率（详解见表格下方说明） |
| Z | 经营目标 |
| △pi | 车辆调价大小（i指carid） |

表3 符号说明

**※说明**：由于资金占用成本属于[企业资金](https://baike.baidu.com/item/%E4%BC%81%E4%B8%9A%E8%B5%84%E9%87%91)成本的一种。指企业因占用资金而向资金供应者支付的各种[资金占用费](https://baike.baidu.com/item/%E8%B5%84%E9%87%91%E5%8D%A0%E7%94%A8%E8%B4%B9/6589588)，如各种长[短期借款](https://baike.baidu.com/item/%E7%9F%AD%E6%9C%9F%E5%80%9F%E6%AC%BE/10493217)的利息、[长期债券](https://baike.baidu.com/item/%E9%95%BF%E6%9C%9F%E5%80%BA%E5%88%B8/10921180)的利息、[优先股](https://baike.baidu.com/item/%E4%BC%98%E5%85%88%E8%82%A1/648709)的[股息](https://baike.baidu.com/item/%E8%82%A1%E6%81%AF/273114)、[普通股](https://baike.baidu.com/item/%E6%99%AE%E9%80%9A%E8%82%A1/648529)的红利等。

鉴于资金占用成本具有经常性、定期性支付的特征，它与[资金占用](https://baike.baidu.com/item/%E8%B5%84%E9%87%91%E5%8D%A0%E7%94%A8/9903690)的期限成正比例关系，故该模型以天为最小单位评估月经营情况，引入wi作为日资金占用成本率。将月资金占用成本以占比的形式划分计算到每一天，以天为最小单位构建模型[22]。且由于车辆调价前后给门店带来的利润不同，故资金占用成本也是不同的。wi中的i指时期（w1为调价前的资金占有成本率，w2为调价后的资金占有成本率）。

### 8.2.3 模型假设

（1）在成本最小化的过程中，只考虑车辆资金占用成本和停车位占用成本，不考虑员工的人力成本等成本。

（2）经营目标以月为单位进行评估。

### 8.2.4 模型构建与求解

**（1）抽象数学模型描述**

由10.1的问题分析可知，该问题在保障经营目标的同时，要尽可能地实现成本最小化且销售利润最大化，即（销售额-成本）max 。由此抽象出一下数学模型：

(ci+△pi)-[(T2-T1)\*ai+(t-T1)\*bi\*w1+(T2-t)\*(bi+△pi)\*w2]>=Z

※说明：抽象函数f(t,△pi)=(ci+△pi)-[(T2-T1)\*ai+(t-T1)\*bi\*w1+(T2-t)\*(bi+△pi)\*w2]是指某辆车该月的经营利润，每辆车的经营利润加和即该门店该月的经营利润。而且在实际问题中，只有调价时间t和调价变化量△pi为未知的，其他参量在实际门店经营中都是已知的[23]。由该表达式以遗传算法构建模型可以达到寻找到合适的调价时间以及调价多少的目的，同时可以得到某车辆的月盈利，进一步加和累计得到门店月盈利，从而去评估是否达到经营目标。

**（2）构建门店经营模型**

将本报告上述抽象数学模型转化为遗传算法中的优化问题进行模型构建，即用标准遗传算法求函数f(t,△pi)的最大值，其中t的取值范围为[1.30]（这里将一个月统一认为是30天）。△pi的取值范围为[定价-进价-成本，自定义值]（即该范围的下界意味着降价的前提是不亏本，而该区间的上界则需要根据市场以及消费者的购车意愿情况确定最大升价空间）。

构建过程如下：  
①初始化种群数目、染色体二进制编码长度、最大进化代数、交叉概率、变异概率。

②产生初始种群，计算个体适应度值，进行选择操作、基于概率的交叉和变异操作，产生新的种群，并把历代的最优个体保留在新种群中，进行下一步遗传操作。（对于本问题来讲，就是初始化t、△pi ，计算优劣程度，选择淘汰的（t,△pi），经过一系列交叉变异操作产生新种群，亦即将每次迭代产生的最优解保留至新种群，淘汰劣质解，不断迭代，得到最优的（t,△pi））

③判断是否满足终止条件：若满足，则结束搜索过程，输出优化值；若不满足，则继续进行迭代优化。

（3）模型求解思路和算法步骤

①算法步骤：

由以上模型构建确定算法步骤如下：

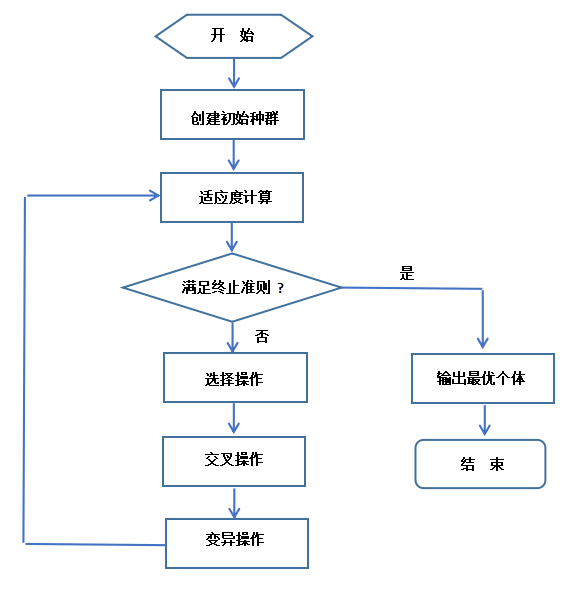


图45 算法步骤

②模型求解：

接下来给出模型求解过程部分函数（完整代码见附录）：

|  |  |
| --- | --- |
| 自定义函数 | 说明 |
| def F(x, y): | 模型抽象数学函数表达式定义函数 |
| def plot\_3d(ax): | 迭代过程数据可视化函数 |
| def get\_fitness(pop): | 适应度计算函数 |
| def translateDNA(pop): | 二进制编码函数 |
| def crossover\_and\_mutation(pop,CROSSOVER\_RATE=0.8): | 交叉操作函数 |
| def mutation(child, MUTATION\_RATE=0.003): | 变异操作函数 |
| def select(pop, fitness): | 选择操作函数 |
| def print\_info(pop): | 有关信息打印函数 |

表4 模型求解自定义函数说明

主函数如下：  
if \_\_name\_\_ == "\_\_main\_\_":  
 fig = plt.figure()  
 ax = Axes3D(fig)  
 plt.ion() plot\_3d(ax)  
 pop = np.random.randint(2, size=(POP\_SIZE, DNA\_SIZE \* 2))for \_ in range(N\_GENERATIONS): *# 迭代* x, y = translateDNA(pop)  
 if 'sca' in locals():  
 sca.remove()  
 sca = ax.scatter(x, y, F(x, y), c='black', marker='o')  
 plt.show()  
 plt.pause(0.1)  
 pop = np.array(crossover\_and\_mutation(pop, CROSSOVER\_RATE))  
 fitness = get\_fitness(pop)  
 pop = select(pop, fitness) *#生成新的种群* print\_info(pop)  
 plt.ioff()  
 plot\_3d(ax)

### 8.2.5 模型检验

由以上构建的模型，本报告现欲给出某辆车的月销售利润求解以及调价时间t和调价变化量△pi的寻找过程以及结果[24]。

现在假设T1 = 2，T2 = 30（即使得本车在本月卖出，避免沦为下个月的滞销车辆而导致利润减少），ai = 100，bi = 30000，ci = 50000，w1 = 1.5%，w2 = 2.0%。则f(t,△pi)=(50000+△pi)-[(30-2)\*100+(t-2)\*30000\*1.5%+(30-t)\*(30000+△pi)\*2.0%]

将函数编入程序中求得到寻找t和△pi的结果图如下：可以看到的是，在利润最大化的情况下寻找到了最优的调价时间t和△pi（图中x代表t，y代表△pi）。

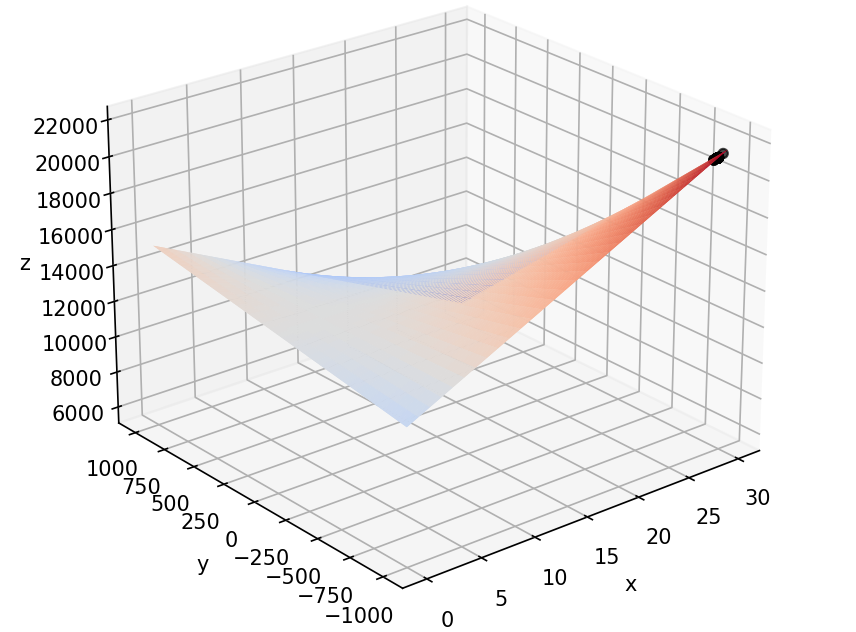


图46 模型检验

### 8.2.6 模型评价

模型优点：  
①与问题领域无关切快速随机的搜索能力。  
②搜索从群体出发，具有潜在的并行性，可以进行多个个体的同时比较。  
③搜索使用评价函数启发，过程简单。  
④使用概率机制进行迭代，具有随机性。  
⑤具有可扩展性，容易与其他算法结合。

模型缺点：

①从理论上，遗传算法能解决任意维函数的组合优化问题，但是在具体应用时将受到超大规模优化问题的限制，主要原因是遗传算法在进化搜索过程中，每代须维持一定的群体规模。群体规模小，包含的信息量就少，算法不能得到充分的发挥；群体规模太大，虽然包含的信息量多，但计算量的增加限制了算法的使用。

②遗传算法的另一个不足之处是易“早熟”：一方面，算法中的交叉运算使群体中的染色体具有局部的相似性，父代染色体信息交换量小使搜索停滞不前；另一方面，变异概率太小导致搜索不能转向其他解空间进行，也会引起算法“爬山”能力差。

# 参考文献

[1]冯秀荣, 王斌. 影响二手车价值的因子分析[J]. 商业研究, 2008(2):4.

[2]丁礼灯 , 席敏 . 我国二手车市场现状分析 [J]. 法制与社会：锐视版 ,

2008(24):2.

[3]蔡云 , 唐岚 , 谭金会 . 中国二手车市场的发展分析 [J]. 中国集体经济 ,

2009(10S):2.

[4]吴勃生, 郭殿臣. 我国二手车市场的发展状况及经营策略调整[J]. 中国科技信息, 2010(22):2.

[5]杨凯, 王军雷, 康凯. 国内二手车市场现状与展望[J]. 汽车工业研究, 2014.

[6]栾兆宇. 中国品牌二手车市场营销模式研究[D]. 吉林大学, 2011.

[7]何继志. 二手车市场发展的现状与前景展望[J]. 中国工程咨询, 2003(2):3.

[8]傅志中, 吴宇峰, 徐进,等. 一种基于特征处理的图像信息融合及超分辨率重建方法:, CN112598575A[P]. 2021.

[9]陈功平, 王红. 改进 Pearson 相关系数的个性化推荐算法[J]. 山东农业大学学报：自然科学版, 2016, 47(6):5.

[10]李靖华, 郭耀煌. 主成分分析用于多指标评价的方法研究——主成分评价[J]. 管理工程学报, 2002, 16(1):5.

[11]吴成茂. 基于加权香农熵的图像阈值法[J]. 计算机工程与应用, 2008,44(18):4.

[12]克劳德 E. 香农. 《通信的数学原理》. 《贝尔实验室技术杂志》.1948

[13] Chen T , Tong H , Benesty M . xgboost: Extreme Gradient Boosting[J].2016.S

[14]佚名. 二手车销售[J]. 汽车驾驶员, 2006(2):2.

[15]杨斌. 二手车销售状况与发展趋势研究[J]. 湖北农机化, 2019(10):1.

[16]邓貌. 模式识别中特征处理方法的应用研究[J]. 2009.

[17]于玲, 吴铁军. 集成学习:Boosting 算法综述[J]. 模式识别与人工智能, 2004,17(1):8.

[18] 张涛, 陈先, 谢美萍,等. 基于K近邻非参数回归的短时交通流预测方法[J]. 系统工程理论与实践, 2010(2):9.

[19] 杨帆, 林琛, 周绮凤,等. 基于随机森林的潜在k近邻算法及其在基因表达数据分类中的应用[J]. 系统工程理论与实践, 2012, 32(4):815-825.

[20] 王煜. 基于决策树和K最近邻算法的文本分类研究[D]. 天津大学, 2006.

[21] 李敏强, 徐博艺, 寇纪淞. 遗传算法与神经网络的结合[J]. 系统工程理论与实践, 1999, 19(2):65-69.

[22] 李敏强. 遗传算法的基本理论与应用[M]. 科学出版社, 2002.

[23] 陈国良, 王熙法, 庄镇泉,等. 遗传算法及其应用[M]. 人民邮电出版社, 1999.

[24] 张文修. 遗传算法的数学基础[J]. 西安交通大学出版社, 2000.

# 附录

使用软件名称：PyCharm，Jupyter Notebook

## 附录 1：二手车零售交易价格预测分析代码

##### 1.模型训练

import re import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestRegressor from sklearn.preprocessing import StandardScaler from category\_encoders.leave\_one\_out import LeaveOneOutEncoder from sklearn.model\_selection import cross\_val\_score from sklearn.decomposition import PCA import xgboost as xgb

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

"""

评估函数

"""

def estimate(y\_true=None,y\_pred=None): y\_true=np.array(list(y\_true)) y\_pred=np.array(list(y\_pred)) Ape=np.abs(y\_pred-y\_true)/y\_true

Mape=sum(Ape)/len(y\_true)

# Ape\_count=[np.nan if x <=0.05 else x for x in Ape] Ape\_count=len(np.where(Ape<=0.05)[0])/len(Ape) return 0.2\*(1-Mape)+0.8\*Ape\_count

"""

读入数据和数据清洗

"""

df = pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件 1：估价训练数据.txt',

parse\_dates=[1, 11, 12],sep='\t', header=None,

encoding='gbk')

# df = pd.read\_table('../data/附件 1：估价训练数据.txt', sep='\t', header=None, encoding='gbk') data = pd.DataFrame(data=df)

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color', 'cityId',

'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country',

'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1, 16): str\_ = 'anonymousFeature'+str(i) columns.append(str\_)

columns.append('price') data.columns = columns

data=data.drop(['country','anonymousFeature4','anonymousFeature7','anonymous

Feature10','anonymousFeature15'

,'maketype','anonymousFeature1','anonymousFeature8','anony

mousFeature9'],axis=1) # 删除缺失值严重的特征 data.info()

nn = ['carCode', 'modelyear', 'gearbox']

for i in nn:

x = int(data[i].mode()) data[i].fillna(x, inplace=True)

"""

特征构造

"""

data['tradeTime\_year']=data['tradeTime'].dt.year data['tradeTime\_month']=data['tradeTime'].dt.month data['tradeTime\_day']=data['tradeTime'].dt.day data['registerDate\_year']=data['registerDate'].dt.year data['registerDate\_month']=data['registerDate'].dt.month data['registerDate\_day']=data['registerDate'].dt.day

print(data['registerDate\_day'].value\_counts().plot(kind='bar',rot=30)) plt.show()

# 处理定类数据（Frequency 编码）

data['color'] = data['color'].map(data['color'].value\_counts())

data['carCode'] = data['carCode'].map(data['carCode'].value\_counts()) data['modelyear'] = data['modelyear'].map(data['modelyear'].value\_counts())

# 处理匿名特征 11 data\_1=data.dropna() #

data\_1.info()

def deal\_11(x):

return sum([float(x) for x in re.findall("\d",x)])

data\_1['anonymousFeature11']=data\_1['anonymousFeature11'].map(deal\_11) data\_1.info()

# 处理匿名特征 12 def deal\_12(x):

li=[float(x) for x in re.findall("\d+",x)] return li[0]\*li[1]\*li[2]

data\_1['anonymousFeature12\_length']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[0]))

data\_1['anonymousFeature12\_width']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[1]))

data\_1['anonymousFeature12\_height']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[2]))

data\_1['anonymousFeature12']=data\_1['anonymousFeature12'].map(deal\_12)

# 处理匿名特征 13 def deal\_13(x):

return x[:4], x[4:6]

data\_1['anonymousFeature13']=data\_1['anonymousFeature13'].astype('string') data\_1['anonymousFeature13\_year']=data\_1['anonymousFeature13'].map(deal\_1

3) # (2017, 09)

data\_1['anonymousFeature13\_month']=data\_1['anonymousFeature13\_year'].appl y(lambda x: int(x[1]))

data\_1['anonymousFeature13\_year']=data\_1['anonymousFeature13\_year'].apply(l ambda x: int(x[0]))

data\_1['anonymousFeature13']=data\_1['anonymousFeature13'].astype('float')

# 构建新特征

data\_1['old\_year']=data\_1['tradeTime']-data\_1['registerDate'] data\_1['old\_year']=data\_1['old\_year'].apply(lambda x:str(x).split(' ')[0]) data\_1['old\_year']=data\_1['old\_year'].astype(int)

data\_1['old\_year\_1']=data\_1['tradeTime']-data\_1['licenseDate'] data\_1['old\_year\_1']=data\_1['old\_year\_1'].apply(lambda x:str(x).split(' ')[0]) data\_1['old\_year\_1']=data\_1['old\_year\_1'].astype(int)

# 数据分桶

bin=[0, 1, 4, 7.15, 10, 50]

data\_1['mileage\_bin']=pd.cut(data\_1['mileage'],bins=bin,labels=False)

y = data\_1['price']

data\_2 = data\_1.drop(['price'],axis=1) print(data\_2.columns) data\_2['price'] = y

data\_2.info()

# data\_2=data\_2.dropna()

data\_2=data\_2.drop(['tradeTime','registerDate','licenseDate'],axis=1)

# 看看有没有异常值

data\_2.describe() data\_2=data\_2[data\_2['price']<80]

data\_2.columns data\_2.info()

"""

特征选择

"""

# 基于皮尔逊相关系数

pearson = data\_2.corr()

index = pearson['price'][:-1].abs() > 0.1

X = data\_2.iloc[:,:-1]

X\_subset = X.loc[:, index]

# X\_subset.columns

"""

降维

"""

# pca=PCA(n\_components=6)

# X\_new\_sk=pca.fit\_transform(X\_subset)

"""

模型训练

"""

# standardScaler=StandardScaler()

# X = standardScaler.fit\_transform(X\_subset)

X\_train, X\_test, y\_train, y\_test =

train\_test\_split(X\_subset,data\_2['price'].to\_numpy() , test\_size=0.2,random\_state=3)

random\_model =

RandomForestRegressor(n\_estimators=500,random\_state=33,n\_jobs=-1) random\_model.fit(X\_train,y\_train) y\_pred = random\_model.predict(X\_test) score = estimate(y\_true=y\_test,y\_pred=y\_pred)

"""

XGboost 优化模型

"""

xlf = xgb.XGBRegressor(max\_depth=50, # 差不多

learning\_rate=0.05, n\_estimators=300, min\_child\_weight=2, subsample=0.8, colsample\_bytree=0.8,

seed=0,

)

xlf.fit(X\_train, y\_train, eval\_metric='mae', eval\_set=[(X\_train, y\_train), (X\_test, y\_test)], early\_stopping\_rounds=20) preds = xlf.predict(X\_test)

score=estimate(y\_true=y\_test,y\_pred=preds) score

"""

画散点图观察模型的效果

"""

random\_model.score(X\_test,y\_test) random\_model.score(X\_train,y\_train)

np.array(y\_test)

plt.scatter(X\_test['newprice'], y\_test) # 样本实际分布 plt.scatter(X\_test['newprice'], y\_pred, color='red') # 绘制拟合曲线 plt.legend(['实际值','预测值']) plt.xlabel("newprice") plt.ylabel("price") plt.show()

##### 2.模型预测

import re import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestRegressor from sklearn.preprocessing import StandardScaler

from category\_encoders.leave\_one\_out import LeaveOneOutEncoder from sklearn.model\_selection import cross\_val\_score import xgboost as xgb

plt.rcParams['font.sans-serif'] = ['SimHei'] plt.rcParams['axes.unicode\_minus'] = False

def estimate(y\_true=None,y\_pred=None):

y\_true=np.array(list(y\_true)) y\_pred=np.array(list(y\_pred)) Ape=np.abs(y\_pred-y\_true)/y\_true

Mape=sum(Ape)/len(y\_true)

# Ape\_count=[np.nan if x <=0.05 else x for x in Ape] Ape\_count=len(np.where(Ape<=0.05)[0])/len(Ape) return 0.2\*(1-Mape)+0.8\*Ape\_count

def train(data\_0=None):

data=pd.DataFrame(data=data\_0)

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color',

'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country',

'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1, 16): str\_ = 'anonymousFeature' + str(i) columns.append(str\_)

columns.append('price') data.columns = columns

#

data=data.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature1

0','anonymousFeature15'],axis=1) # 删除缺失值严重的特征

data = data.drop(['country', 'anonymousFeature4', 'anonymousFeature7',

'anonymousFeature10', 'anonymousFeature15'

, 'maketype', 'anonymousFeature1', 'anonymousFeature8', 'anonymousFeature9'], axis=1) data.info()

nn = ['carCode', 'modelyear', 'gearbox']

for i in nn:

x = int(data[i].mode()) data[i].fillna(x, inplace=True)

"""

特征构造

"""

data['tradeTime\_year'] = data['tradeTime'].dt.year data['tradeTime\_month'] = data['tradeTime'].dt.month data['tradeTime\_day'] = data['tradeTime'].dt.day data['registerDate\_year'] = data['registerDate'].dt.year data['registerDate\_month'] = data['registerDate'].dt.month data['registerDate\_day'] = data['registerDate'].dt.month

print(data['registerDate\_day'].value\_counts().plot(kind='bar', rot=30)) plt.show()

# 处理定类数据（Frequency 编码）

data['color'] = data['color'].map(data['color'].value\_counts()) data['carCode'] = data['carCode'].map(data['carCode'].value\_counts()) # data['country'] = data['country'].map(data['country'].value\_counts()) data['modelyear'] = data['modelyear'].map(data['modelyear'].value\_counts())

# 处理匿名特征 11

data\_1 = data.dropna() #

data\_1.info()

len(list(data\_1['anonymousFeature11']))

def deal\_11(x):

return sum([float(x) for x in re.findall("\d", x)])

data\_1['anonymousFeature11'] =

data\_1['anonymousFeature11'].map(deal\_11) data\_1.info()

# 处理匿名特征 12 def deal\_12(x):

li = [float(x) for x in re.findall("\d+", x)] return li[0] \* li[1] \* li[2] data\_1['anonymousFeature12\_length'] = data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[0])) data\_1['anonymousFeature12\_width'] =

data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[1])) data\_1['anonymousFeature12\_height'] =

data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[2])) data\_1['anonymousFeature12'] =

data\_1['anonymousFeature12'].map(deal\_12)

# 处理匿名特征 13 def deal\_13(x):

return x[:4], x[4:6]

data\_1['anonymousFeature13'] =

data\_1['anonymousFeature13'].astype('string') data\_1['anonymousFeature13\_year'] =

data\_1['anonymousFeature13'].map(deal\_13) # (2017, 09) data\_1['anonymousFeature13\_month'] =

data\_1['anonymousFeature13\_year'].apply(lambda x: int(x[1])) data\_1['anonymousFeature13\_year'] =

data\_1['anonymousFeature13\_year'].apply(lambda x: int(x[0])) data\_1['anonymousFeature13'] =

data\_1['anonymousFeature13'].astype('float')

data\_1['old\_year'] = data\_1['tradeTime'] - data\_1['registerDate'] data\_1['old\_year'] = data\_1['old\_year'].apply(lambda x: str(x).split(' ')[0]) data\_1['old\_year'] = data\_1['old\_year'].astype(int)

data\_1['old\_year\_1'] = data\_1['tradeTime'] - data\_1['licenseDate'] data\_1['old\_year\_1'] = data\_1['old\_year\_1'].apply(lambda x: str(x).split(' ')[0]) data\_1['old\_year\_1'] = data\_1['old\_year\_1'].astype(int)

# 数据分桶

bin = [0, 1, 4, 7.15, 10, 50]

data\_1['mileage\_bin'] = pd.cut(data\_1['mileage'], bins=bin, labels=False)

y = data\_1['price']

data\_2 = data\_1.drop(['price'], axis=1) print(data\_2.columns) data\_2['price'] = y

data\_2.info()

# data\_2=data\_2.dropna()

data\_2 = data\_2.drop(['tradeTime', 'registerDate', 'licenseDate'], axis=1)

# 看看有没有异常值

data\_2.describe()

data\_2 = data\_2[data\_2['price'] < 80] data\_2.columns data\_2.info()

# data\_2.drop([22115],axis=0,inplace=True)

"""

特征选择

"""

# 基于皮尔逊相关系数

pearson = data\_2.corr()

index = pearson['price'][:-1].abs() > 0.1

X = data\_2.iloc[:, :-1]

X\_subset = X.loc[:, index]

# X\_subset.columns

"""

降维

"""

"""

模型训练

"""

# standardScaler = StandardScaler()

# X = standardScaler.fit\_transform(X\_subset.to\_numpy())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_subset, data\_2['price'].to\_numpy(), test\_size=0.2, random\_state=3)

# random\_model = RandomForestRegressor(n\_estimators=500, random\_state=33, n\_jobs=-1)

# random\_model.fit(X\_train, y\_train)

# y\_pred = random\_model.predict(X\_test) xlf = xgb.XGBRegressor(max\_depth=50, # 差不多

learning\_rate=0.05, n\_estimators=300, min\_child\_weight=2, subsample=0.8, colsample\_bytree=0.8,

seed=0,

)

xlf.fit(X\_train, y\_train, eval\_metric='mae', eval\_set=[(X\_train, y\_train), (X\_test,

y\_test)],

early\_stopping\_rounds=20) preds = xlf.predict(X\_test) return xlf,X\_subset.columns,preds,y\_test

def test(data\_0=None,model=None,feature=None):

data=pd.DataFrame(data=data\_0)

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color',

'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country',

'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1, 16): str\_ = 'anonymousFeature' + str(i) columns.append(str\_)

data.columns = columns

#

data=data.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature1

0','anonymousFeature15'],axis=1) # 删除缺失值严重的特征

data = data.drop(['country', 'anonymousFeature4', 'anonymousFeature7',

'anonymousFeature10', 'anonymousFeature15'

, 'maketype', 'anonymousFeature1', 'anonymousFeature8', 'anonymousFeature9'], axis=1)

nn = ['carCode', 'modelyear', 'gearbox','anonymousFeature13'] for i in nn:

x = int(data[i].mode()) data[i].fillna(x, inplace=True)

"""

特征构造

"""

data['tradeTime\_year'] = data['tradeTime'].dt.year data['tradeTime\_month'] = data['tradeTime'].dt.month data['tradeTime\_day'] = data['tradeTime'].dt.day data['registerDate\_year'] = data['registerDate'].dt.year data['registerDate\_month'] = data['registerDate'].dt.month data['registerDate\_day'] = data['registerDate'].dt.day

# 处理定类数据（Frequency 编码）

data['color'] = data['color'].map(data['color'].value\_counts())

data['carCode'] = data['carCode'].map(data['carCode'].value\_counts()) # data['country'] = data['country'].map(data['country'].value\_counts()) data['modelyear'] = data['modelyear'].map(data['modelyear'].value\_counts())

# 处理匿名特征 11

data\_1=data data\_1.info() data\_1['anonymousFeature11'].fillna('3',inplace=True) def deal\_11(x):

return sum([float(x) for x in re.findall("\d", x)])

data\_1['anonymousFeature11'] =

data\_1['anonymousFeature11'].map(deal\_11) data\_1.info()

# 处理匿名特征 12

data\_1['anonymousFeature12'].fillna('5015\*1874\*1455',inplace=True) def deal\_12(x):

li = [float(x) for x in re.findall("\d+", x)] return li[0] \* li[1] \* li[2]

data\_1['anonymousFeature12\_length'] =

data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[0])) data\_1['anonymousFeature12\_width'] =

data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[1])) data\_1['anonymousFeature12\_height'] =

data\_1['anonymousFeature12'].apply(lambda x: int(x.split('\*')[2])) data\_1['anonymousFeature12'] =

data\_1['anonymousFeature12'].map(deal\_12)

# 处理匿名特征 13 def deal\_13(x):

return x[:4], x[4:6]

data\_1['anonymousFeature13'] =

data\_1['anonymousFeature13'].astype('string') data\_1['anonymousFeature13\_year'] =

data\_1['anonymousFeature13'].map(deal\_13) # (2017, 09) data\_1['anonymousFeature13\_month'] =

data\_1['anonymousFeature13\_year'].apply(lambda x: int(x[1])) data\_1['anonymousFeature13\_year'] =

data\_1['anonymousFeature13\_year'].apply(lambda x: int(x[0])) data\_1['anonymousFeature13'] = data\_1['anonymousFeature13'].astype('float') data\_1['old\_year'] = data\_1['tradeTime'] - data\_1['registerDate'] data\_1['old\_year'] = data\_1['old\_year'].apply(lambda x: str(x).split(' ')[0]) data\_1['old\_year'] = data\_1['old\_year'].astype(int)

data\_1['old\_year\_1'] = data\_1['tradeTime'] - data\_1['licenseDate'] data\_1['old\_year\_1'] = data\_1['old\_year\_1'].apply(lambda x: str(x).split(' ')[0]) data\_1['old\_year\_1'] = data\_1['old\_year\_1'].astype(int)

# 数据分桶

bin = [0, 1, 4, 7.15, 10, 50]

data\_1['mileage\_bin'] = pd.cut(data\_1['mileage'], bins=bin, labels=False) data\_2 = data\_1.drop(['tradeTime', 'registerDate', 'licenseDate'], axis=1) X=data\_2[feature] y=model.predict(X) return X ,y

df = pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件 1：估价训练数据.txt',

parse\_dates=[1, 11, 12],sep='\t', header=None,

encoding='gbk')

df\_1 = pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件

2：估价验证数据.txt',

parse\_dates=[1, 11, 12],sep='\t', header=None,

encoding='gbk')

model,feature,y\_p,y\_t=train(data\_0=df) score=estimate(y\_true=y\_t,y\_pred=y\_p)

X, y\_=test(data\_0=df\_1,model=model,feature=feature) data\_=pd.DataFrame(df\_1.iloc[:,0])

data\_['price']=y\_ print(data\_)

data\_.to\_csv('../data/附件 3：估价模型结果.txt',sep='\t',index=False, header=None)

# data\_.to\_csv(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件 3：估价模

型结果.txt',sep='\t',index=False, header=None)

## 附录 2：基于成交周期因素分析代码

##### 2.1 结合附件四 2.1.1 初步分析

import pandas as pd from collections import Counter

data = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4.csv", encoding='utf-8')

#print(data) listT1= [] listP1= [] listT2= [] listP2= [] listT3= [] listP3= [] listT4= [] listP4= [] listT5= [] listP5= [] listT6= [] listP6= [] listT7= [] listP7= [] listT8= [] listP8= [] listT12= [] listP12= [] list\_len = [] for i in data.updatePriceTimeJson:

i = i.split('"') list\_len.append(len(i))

if len(i) == 5:#-------------------------------------1

#print(i) #print(len(i)) listT1.append(i[1])

listP1.append(i[3])

x = " "

#print(i) listT2.append(x) listP2.append(x) listT3.append(x) listP3.append(x) listT4.append(x) listP4.append(x) listT5.append(x) listP5.append(x) listT6.append(x) listP6.append(x) listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 9:#-------------------------------------2 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5])

listP2.append(i[7])

x = " "

#print(i) listT3.append(x) listP3.append(x) listT4.append(x) listP4.append(x) listT5.append(x) listP5.append(x) listT6.append(x) listP6.append(x) listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x) listP12.append(x)

#print(i[1])

#data.updateTime = data.updatePriceTimeJson.apply(lambda x:i[1])

#data.updatePrice = i[3]

elif len(i) == 13:#-------------------------------------3 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7])

listT3.append(i[9]) listP3.append(i[11])

x = " "

listT4.append(x) listP4.append(x) listT5.append(x) listP5.append(x) listT6.append(x) listP6.append(x) listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 17:#-------------------------------------4 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15])

elif len(i) == 21:#-------------------------------------4 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15])

x = " "

listP5.append(x) listT6.append(x) listP6.append(x) listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 25: # -------------------------------------5 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15]) listT5.append(i[17])

listP5.append(i[19])

x = " "

listT6.append(x) listP6.append(x) listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 29: # -------------------------------------6

listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15]) listT5.append(i[17]) listP5.append(i[19]) listT6.append(i[21])

listP6.append(i[23])

x = " "

listT7.append(x) listP7.append(x) listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 33: # -------------------------------------7

listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15]) listT5.append(i[17]) listP5.append(i[19]) listT6.append(i[21]) listP6.append(i[23]) listT7.append(i[25]) listP7.append(i[27])

x = " "

listT8.append(x) listP8.append(x) listT12.append(x)

listP12.append(x)

elif len(i) == 37: # -------------------------------------8

listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15]) listT5.append(i[17]) listP5.append(i[19]) listT6.append(i[21]) listP6.append(i[23]) listT7.append(i[25]) listP7.append(i[27]) listT8.append(i[29]) listP8.append(i[31])

x = " "

listT12.append(x) listP12.append(x)

else:#------------------------------------12 listT1.append(i[1]) listP1.append(i[3]) listT2.append(i[5]) listP2.append(i[7]) listT3.append(i[9]) listP3.append(i[11]) listT4.append(i[13]) listP4.append(i[15]) listT5.append(i[17]) listP5.append(i[19]) listT6.append(i[21]) listP6.append(i[23]) listT7.append(i[25]) listP7.append(i[27]) listT8.append(i[29]) listP8.append(i[31]) listT12.append(i[33]) listP12.append(i[35])

print(data)

#for i in list:

#data.updateTime = data.updatePriceTimeJson.apply(lambda x: i) listnum = [] for i in list\_len:

if i == 1: listnum.append(0)

elif i == 5: listnum.append(1)

elif i == 9: listnum.append(2)

elif i == 13: listnum.append(3)

elif i == 17: listnum.append(4)

elif i == 21: listnum.append(5)

elif i == 25: listnum.append(6)

elif i == 29: listnum.append(7)

elif i == 33: listnum.append(8)

elif i == 49: listnum.append(12)

data.insert(loc=6, column='updateTime1', value=listT1) data.insert(loc=7, column='updatePrice1', value=listP1) data.insert(loc=8, column='updateTime2', value=listT2) data.insert(loc=9, column='updatePrice2', value=listP2) data.insert(loc=10, column='updateTime3', value=listT3) data.insert(loc=11, column='updatePrice3', value=listP3) data.insert(loc=12, column='updatenum', value=listnum)

# print(data)

# print(Counter(list\_len)) # print(Counter(listnum))

data.to\_csv("C:\\Users\Lenovo\Desktop\\4\_integration.csv", encoding='utf-8')

#print(Counter(data.updateTime))

data1 = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4\_integration.csv",

encoding='utf-8')

from pyecharts import options as opts from pyecharts.charts import Bar from pyecharts.faker import Faker dict = dict(Counter(listnum)) print(dict.keys) listx= [0,1,2,3,4,5,6,7,8,12] listy = [6763,2126,690,269,97,31,14,5,3,2]

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_更新频次数统计

# c = (

# Bar()

# .add\_xaxis(listx)

# .add\_yaxis("updatenum", listy, color=Faker.rand\_color())

# .set\_global\_opts(

# title\_opts=opts.TitleOpts(title="更新次数统计"),

|  |  |  |
| --- | --- | --- |
| #  opts.DataZoomOpts(type\_="inside")],  # )  # .render("更新次数统计.html")  # ) sum = 0 for i in data1: | datazoom\_opts=[opts.DataZoomOpts(), | |
| data1.withdrawPeriod = | pd.to\_datetime(data1.pullDate) | - |

pd.to\_datetime(data1.pushDate) print(data1.withdrawPeriod) list\_withdrawPeriod = [] for i in data1.withdrawPeriod:

list\_withdrawPeriod.append(i) import pandas as pd data1.withdrawPeriod

=data1.withdrawPeriod.astype('timedelta64[D]').astype(float) print(data1.withdrawPeriod)

list\_withdrawPeriod = list(data1.withdrawPeriod)

print(list\_withdrawPeriod) print(data1) data1.insert(loc=14, column='withdrawPeriod', value=list\_withdrawPeriod) #data['L'] = pd.to\_datetime(data['LOAD\_TIME'])-pd.to\_datetime(data['FFP\_DATE']) data1.to\_csv("C:\\Users\Lenovo\Desktop\\4\_hhh.csv", encoding='utf-8') data2 = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4\_hhh.csv", encoding='utf-8')

print(" 更新次数为 0 的成交周期平均值： ",data2[data2.updatenum ==

0].withdrawPeriod.mean())

print(" 更新次数为 1 的成交周期平均值： ",data2[data2.updatenum ==

1].withdrawPeriod.mean())

print(" 更新次数为 2 的成交周期平均值： ",data2[data2.updatenum ==

2].withdrawPeriod.mean())

print(" 更新次数为 3 的成交周期平均值： ",data2[data2.updatenum ==

3].withdrawPeriod.mean())

print(" 更新次数为 4 的成交周期平均值： ",data2[data2.updatenum ==

4].withdrawPeriod.mean())

print(" 更新次数为 5 的成交周期平均值： ",data2[data2.updatenum ==

5].withdrawPeriod.mean())

print(" 更新次数为 6 的成交周期平均值： ",data2[data2.updatenum ==

6].withdrawPeriod.mean())

print(" 更新次数为 7 的成交周期平均值： ",data2[data2.updatenum ==

7].withdrawPeriod.mean())

print(" 更新次数为 8 的成交周期平均值： ",data2[data2.updatenum ==

8].withdrawPeriod.mean())

print(" 更新次数为 12 的成交周期平均值： ",data2[data2.updatenum ==

12].withdrawPeriod.mean())

###### 2.1.2 数据可视化

import pandas as pd from collections import Counter

data2 = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4\_hhh.csv", encoding='utf-8') print(sorted(list(Counter(data2.pushPrice))))

print("价格对应平均成交周期：")

print("小于 5 平均值：",data2[data2.pushPrice < 5].withdrawPeriod.mean()) print("大于 5 平均值：",data2[data2.pushPrice > 5].withdrawPeriod.mean()) print("大于 20 平均值：",data2[data2.pushPrice > 20].withdrawPeriod.mean()) print("大于 50 平均值：",data2[data2.pushPrice > 50].withdrawPeriod.mean()) print("大于 100 平均值：",data2[data2.pushPrice > 100].withdrawPeriod.mean())

print("大于 500 平均值：",data2[data2.pushPrice > 500].withdrawPeriod.mean()) print(dict(Counter(data2.pushPrice)))

a = dict(Counter(data2.pushPrice)) import pyecharts.options as opts

x\_data = list(a.keys()) y\_data = list(a.values()) from pyecharts.charts import Scatter

(

Scatter(init\_opts=opts.InitOpts(width="1600px", height="1000px"))

.add\_xaxis(xaxis\_data=x\_data)

.add\_yaxis( series\_name="", y\_axis=y\_data, symbol\_size=20,

label\_opts=opts.LabelOpts(is\_show=False),

)

.set\_series\_opts() .set\_global\_opts(

xaxis\_opts=opts.AxisOpts( type\_="value", splitline\_opts=opts.SplitLineOpts(is\_show=True)

),

yaxis\_opts=opts.AxisOpts( type\_="value", axistick\_opts=opts.AxisTickOpts(is\_show=True), splitline\_opts=opts.SplitLineOpts(is\_show=True),

),

tooltip\_opts=opts.TooltipOpts(is\_show=False),

)

.render("价格—价格出现次数次数散点图.html")

)

import pandas as pd from collections import Counter

data2 = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4\_hhh.csv", encoding='utf-8') x1 = data2[data2.withdrawDate == "0"].updatenum

print(" 未 成 交 车 辆 的 更 新 频 次 统 计 ： ",data2[data2.withdrawDate ==

"0"].updatenum.count())

print(x1)

print("未成交车辆的更新频次统计的 Counter 的结果：",Counter(x1))

#data[data.isevening==1].Dept.count() x2 = data2[data2.withdrawDate != "0"].updatenum print(" 成 交 车 辆 的 更 新 频 次 统 计 ： ",data2[data2.withdrawDate !=

"0"].updatenum.count())

print(x2)

print("成交车辆的更新频次统计的 Counter 的结果：",Counter(x2))

#data[data.isevening==1].Dept.count() from pyecharts import options as opts from pyecharts.charts import Bar from pyecharts.commons.utils import JsCode from pyecharts.globals import ThemeType

list2 = [

{"value": 5304, "percent": 5304 / (5304 + 1459)},

{"value": 1763, "percent": 1763 / (1763 + 363)},

{"value": 586, "percent": 586 / (586 + 104)},

{"value": 220, "percent": 220 / (220 + 49)},

{"value": 80, "percent": 80 / (80 + 17)},

{"value": 25, "percent": 25 / (25 + 6)},

{"value": 13, "percent": 13 / (13 + 1)},

{"value": 4, "percent": 4 / (4 + 1)}, {"value": 3, "percent": 3 / (3 + 0)},

{"value": 2, "percent": 2 / (2 + 0)},

]

list3 = [

{"value": 1459, "percent": 1459 / (1459 + 5304)},

{"value": 363, "percent": 363 / (363 + 1763)},

{"value": 104, "percent": 104 / (104+ 586)},

{"value": 49, "percent": 49/ (49 + 220)},

{"value": 17, "percent": 17 / (17 + 80)},

{"value": 6, "percent": 6 / (6 + 25)}, {"value": 1, "percent": 1 / (1 + 13)},

{"value": 1, "percent": 1 / (1 + 4)}, {"value": 0, "percent": 0 / (0 + 3)},

{"value": 0, "percent": 0 / (0 + 2)},

]

c = (

Bar(init\_opts=opts.InitOpts(theme=ThemeType.LIGHT))

.add\_xaxis([0,1, 2, 3, 4, 5,6,7,8,12])

.add\_yaxis("成交车辆", list2, stack="stack1", category\_gap="10%")

.add\_yaxis("未成交车辆", list3, stack="stack2", category\_gap="50%")

.set\_series\_opts( label\_opts=opts.LabelOpts( position="right", formatter=JsCode(

"function(x){return Number(x.data.percent \* 100).toFixed() +

'%';}"

),

) )

.render("成交车辆与未成交车辆更新频次占比图.html")

)

from pyecharts import options as opts from pyecharts.charts import Bar from pyecharts.faker import Faker

c = (

Bar()

.add\_xaxis([0,1, 2, 3, 4, 5,6,7,8,12])

.add\_yaxis("成交", [5304, 1763, 586, 220, 80, 25, 13, 4, 3, 2])

.add\_yaxis("未成交", [1459, 363, 104, 49, 17, 6, 1, 1, 0, 0])

.set\_global\_opts( title\_opts=opts.TitleOpts(title="Bar-显示 ToolBox"), toolbox\_opts=opts.ToolboxOpts(),

legend\_opts=opts.LegendOpts(is\_show=False),

)

.render("成交车辆与未成交车辆更新频次对比图.html")

)

import pandas as pd from collections import Counter data2 = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4\_hhh.csv", encoding='utf-8')

print(" 未 成 交 车 辆 更 新 次 数 平 均 值 ： ",data2[data2.withdrawDate ==

"0"].updatenum.mean())

print(" 成 交 车 辆 更 新 次 数 平 均 值 ： ",data2[data2.withdrawDate !=

"0"].updatenum.mean()) import pyecharts.options as opts x\_data = data2.updatenum.tolist() y\_data = data2.withdrawPeriod.tolist() from pyecharts.charts import Scatter

(

Scatter(init\_opts=opts.InitOpts(width="1600px", height="1000px"))

.add\_xaxis(xaxis\_data=x\_data)

.add\_yaxis( series\_name="", y\_axis=y\_data, symbol\_size=20,

label\_opts=opts.LabelOpts(is\_show=False),

)

.set\_series\_opts() .set\_global\_opts(

xaxis\_opts=opts.AxisOpts( type\_="value", splitline\_opts=opts.SplitLineOpts(is\_show=True)

),

yaxis\_opts=opts.AxisOpts( type\_="value", axistick\_opts=opts.AxisTickOpts(is\_show=True), splitline\_opts=opts.SplitLineOpts(is\_show=True),

),

tooltip\_opts=opts.TooltipOpts(is\_show=False),

)

.render("更新次数—成交周期散点图.html")

)

##### 2.2 综合附件一与附件四

###### 2.2.1 数据分析

# 数据处理

import numpy as np import pandas as pd import datetime

# 读取附件四

df = pd.read\_table('./data/附件 4：门店交易训练数据.txt', sep='\t', header=None, encoding='gbk')

# 修改中文标签

columns = ['车辆 id', '上架时间', '上架价格', '{价格调整时间：调整后价格}', '下架

时间(成交车辆下架时间和成交时间相同)', '成交时间']

df.columns = columns

# 保存文件附件 4-1（附件四含中文标签）

df.to\_csv('./data/附件 4-1.csv', index = False)

# 读取附件 1

df1 = pd.read\_table('./data/附件 1：估价训练数据.txt', sep='\t', header=None, encoding='gbk')

# 修改中文标签

columns1 = ['车辆 id', '展销时间', '品牌 id ', '车系 id ', '车型 id ', '里程', '车辆颜色', '车辆所在城市 id', '国标码', '过户次数', '载客人数', '注册日期', '上牌日期', '国别', '厂商类型', '年款', '排量', '变速箱', '燃油类型', '新车价', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', ' 匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '二手车交易

价格（预测目标）']

df1.columns = columns1

# 保存

df1.to\_csv('./data/附件 1-1.csv', index = False)

# 合并附件一与附件四

df2 = pd.merge(df1, df, how = 'inner', on = ['车辆 id']) # 按照车辆 id 合并

df2.to\_csv('./data/附件 4-2.csv', index = False)

# 整理销售周期（天数）

df2['上架时间'] = pd.to\_datetime(df['上架时间'])

df2['下架时间(成交车辆下架时间和成交时间相同)'] = pd.to\_datetime(df2['下架时间(成交车辆下架时间和成交时间相同)'])

df2['成交周期(天)'] = df2['下架时间(成交车辆下架时间和成交时间相同)'] - df2[' 上架时间']

df2['成交周期(天)'] = df2['成交周期(天)'].apply(lambda x: x.days) df2.to\_csv('./data/附件 4-3（含销售周期天数）.csv', index = False)

# 划分销售周期区间

cycle\_bins = [0, 1, 2, 3, 4, 5, 8, 18, 51, 92, 278]

cycle\_labels = ['0', '1', '2', '3', '4', '5-7', '8-17', '18-50', '51-91', '92-277']

a, b = pd.cut(x = df2['成交周期(天)'], bins = cycle\_bins, right = False, labels = cycle\_labels, retbins = True)

df2['周期区间'], b = pd.cut(x = df2['成交周期(天)'], bins = cycle\_bins, right = False, retbins = True)

df2['周期标签'], b = pd.cut(x = df2['成交周期(天)'], bins = cycle\_bins, right = False, labels = cycle\_labels, retbins = True) df2.to\_csv('./data/附件 4-4（含周期区间）.csv', index = False)

2.2.2 销售周期天数分析

import pandas as pd import numpy as np from collections import Counter

from pyecharts import options as opts from pyecharts.charts import Scatter from pyecharts.charts import Line

def counter(list):

from collections import Counter

# 对列表内的元素进行排序，计数，以列表的形式返回

list\_counter = Counter(list) list\_sorted = sorted(list\_counter.items(), key = lambda x:x[0], reverse = False)

# lambda x:x[1]按值排序，reverse = True 倒序 即大值在前。 list\_dict = dict(list\_sorted) return list\_dict

# 对销售周期天数的分析

df = pd.read\_csv('./data/附件 4-3（含销售周期天数）.csv') transaction = list(df['成交周期(天)'])

# print(max(transaction)) # print(min(transaction)) cycle = list(range(0, 278))

transaction\_counter = Counter(transaction)

transaction\_sorted = sorted(transaction\_counter.items(), key = lambda x: x[0], reverse = False)

transaction\_dict = dict(transaction\_sorted)

transaction\_sorted1 = sorted(transaction\_counter.items(), key = lambda x: x[1], reverse = False)

transaction\_dict1 = dict(transaction\_sorted1)

# 绘制销售周期的散点图

c = (

Scatter()

.add\_xaxis(list(transaction\_dict))

.add\_yaxis(

"次",

list(transaction\_dict.values()),

)

.set\_global\_opts(

title\_opts = opts.TitleOpts(title = '销售周期散点图')

)

.render('./charts/销售周期散点图.html')

)

# 比较品牌 id 对销售周期的影响

brandid\_dict = counter(list(df['品牌 id ']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data=list(brandid\_dict))

.add\_yaxis("数量", y\_axis=list(brandid\_dict.values()))

.render('./charts/品牌 id 对销售周期的影响折线图.html')

)

brandid\_dict1 = sorted(brandid\_dict.items(), key = lambda x:x[1], reverse = True)

# 整体销量前十的品牌

brandid\_top10 = dict(brandid\_dict1[:10]) brand\_top10 = list(brandid\_top10)

brandeal = df.groupby('品牌 id ')['成交周期(天)'].apply(list).to\_dict()

# 每个周期出现多少品牌

dealbrand = df.groupby('成交周期(天)')['品牌 id '].apply(list).to\_dict() brand\_0 = brandeal[brand\_top10[0]] brand\_1 = brandeal[brand\_top10[1]] brand\_2 = brandeal[brand\_top10[2]] brand\_3 = brandeal[brand\_top10[3]] brand\_4 = brandeal[brand\_top10[4]] brand\_5 = brandeal[brand\_top10[5]] brand\_6 = brandeal[brand\_top10[6]] brand\_7 = brandeal[brand\_top10[7]] brand\_8 = brandeal[brand\_top10[8]] brand\_9 = brandeal[brand\_top10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", brand\_0) .add\_yaxis("top\_2", brand\_1) .add\_yaxis("top\_3", brand\_2) .add\_yaxis("top\_4", brand\_3) .add\_yaxis("top\_5", brand\_4) .add\_yaxis("top\_6", brand\_5) .add\_yaxis("top\_7", brand\_6) .add\_yaxis("top\_8", brand\_7) .add\_yaxis("top\_9", brand\_8)

.add\_yaxis("top\_10", brand\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "品牌前 10 的销售周期")

# )

.render('./charts/品牌前 10 的销售周期散点图.html')

)

# 销量最少的十个

brandid\_dreciprocal10 = dict(brandid\_dict1[-10:]) print(brandid\_dreciprocal10) # len(brandid\_reciprocal10)

brand\_reciprocal10 = list(brandid\_dreciprocal10) print(brand\_reciprocal10) brand\_d0 = brandeal[brand\_reciprocal10[0]] brand\_d1 = brandeal[brand\_reciprocal10[1]] brand\_d2 = brandeal[brand\_reciprocal10[2]] brand\_d3 = brandeal[brand\_reciprocal10[3]] brand\_d4 = brandeal[brand\_reciprocal10[4]] brand\_d5 = brandeal[brand\_reciprocal10[5]] brand\_d6 = brandeal[brand\_reciprocal10[6]] brand\_d7 = brandeal[brand\_reciprocal10[7]] brand\_d8 = brandeal[brand\_reciprocal10[8]] brand\_d9 = brandeal[brand\_reciprocal10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", brand\_d0) .add\_yaxis("reci\_2", brand\_d1) .add\_yaxis("reci\_3", brand\_d2) .add\_yaxis("reci\_4", brand\_d3) .add\_yaxis("reci\_5", brand\_d4) .add\_yaxis("reci\_6", brand\_d5) .add\_yaxis("reci\_7", brand\_d6) .add\_yaxis("reci\_8", brand\_d7) .add\_yaxis("reci\_9", brand\_d8)

.add\_yaxis("reci\_10", brand\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "品牌后 10 的销售周期")

# )

.render('./charts/品牌后 10 的销售周期散点图.html')

)

c.render\_notebook()

# 车系 id

chexiid\_dict = counter(list(df['车系 id ']))

print("len: ", len(chexiid\_dict)) print("max: ", max(chexiid\_dict)) print("min: ", min(chexiid\_dict))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data=list(chexiid\_dict))

.add\_yaxis("数量", y\_axis=list(chexiid\_dict.values()))

.render('./charts/车系 id 销量折线图.html')

)

# 按销量排序

chexiid\_dict1 = sorted(chexiid\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理车系 id 与成交周期

chexideal = df.groupby('车系 id ')['成交周期(天)'].apply(list).to\_dict()

# 车系销量前十

chexiid\_top10 = dict(chexiid\_dict1[:10]) print(chexiid\_top10) chexi\_top10 = list(chexiid\_top10) print(chexi\_top10)

# 前 10

chexi\_0 = chexideal[chexi\_top10[0]] chexi\_1 = chexideal[chexi\_top10[1]] chexi\_2 = chexideal[chexi\_top10[2]] chexi\_3 = chexideal[chexi\_top10[3]] chexi\_4 = chexideal[chexi\_top10[4]] chexi\_5 = chexideal[chexi\_top10[5]] chexi\_6 = chexideal[chexi\_top10[6]] chexi\_7 = chexideal[chexi\_top10[7]] chexi\_8 = chexideal[chexi\_top10[8]] chexi\_9 = chexideal[chexi\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("1", chexi\_0) .add\_yaxis("2", chexi\_1) .add\_yaxis("3", chexi\_2) .add\_yaxis("4", chexi\_3) .add\_yaxis("5", chexi\_4) .add\_yaxis("6", chexi\_5) .add\_yaxis("7", chexi\_6) .add\_yaxis("8", chexi\_7) .add\_yaxis("9", chexi\_8)

.add\_yaxis("10", chexi\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/车系 id 前 10 的销售周期散点图.html')

)

# 车系销量后十

chexiid\_dreciprocal10 = dict(chexiid\_dict1[-10:]) print(chexiid\_dreciprocal10) chexi\_reciprocal10 = list(chexiid\_dreciprocal10) print(chexi\_reciprocal10)

# 后 10

chexiid\_dreciprocal10 = dict(chexiid\_dict1[-10:]) print(chexiid\_dreciprocal10) # len(chexiid\_reciprocal10)

chexi\_reciprocal10 = list(chexiid\_dreciprocal10) print(chexi\_reciprocal10) chexi\_d0 = chexideal[chexi\_reciprocal10[0]] chexi\_d1 = chexideal[chexi\_reciprocal10[1]] chexi\_d2 = chexideal[chexi\_reciprocal10[2]] chexi\_d3 = chexideal[chexi\_reciprocal10[3]] chexi\_d4 = chexideal[chexi\_reciprocal10[4]] chexi\_d5 = chexideal[chexi\_reciprocal10[5]] chexi\_d6 = chexideal[chexi\_reciprocal10[6]] chexi\_d7 = chexideal[chexi\_reciprocal10[7]] chexi\_d8 = chexideal[chexi\_reciprocal10[8]] chexi\_d9 = chexideal[chexi\_reciprocal10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", chexi\_d0) .add\_yaxis("reci\_2", chexi\_d1) .add\_yaxis("reci\_3", chexi\_d2) .add\_yaxis("reci\_4", chexi\_d3) .add\_yaxis("reci\_5", chexi\_d4) .add\_yaxis("reci\_6", chexi\_d5) .add\_yaxis("reci\_7", chexi\_d6) .add\_yaxis("reci\_8", chexi\_d7) .add\_yaxis("reci\_9", chexi\_d8)

.add\_yaxis("reci\_10", chexi\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期") # )

.render('./charts/车系 id 后 10 的销售周期散点图.html')

)

# 车型 id

chexingid\_dict = counter(list(df['车型 id ']))

print("len: ", len(chexingid\_dict)) print("max: ", max(chexingid\_dict)) print("min: ", min(chexingid\_dict))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data=list(chexingid\_dict))

.add\_yaxis("数量", y\_axis=list(chexingid\_dict.values()))

.render('./charts/车型 id 销量折线图.html')

)

# 按销量排序

chexingid\_dict1 = sorted(chexingid\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理车系 id 与成交周期

chexingdeal = df.groupby('车型 id ')['成交周期(天)'].apply(list).to\_dict()

# 前 10

chexingid\_top10 = dict(chexingid\_dict1[:10]) print(chexingid\_top10) chexing\_top10 = list(chexingid\_top10) print(chexing\_top10) chexing\_0 = chexingdeal[chexing\_top10[0]] chexing\_1 = chexingdeal[chexing\_top10[1]] chexing\_2 = chexingdeal[chexing\_top10[2]] chexing\_3 = chexingdeal[chexing\_top10[3]] chexing\_4 = chexingdeal[chexing\_top10[4]] chexing\_5 = chexingdeal[chexing\_top10[5]] chexing\_6 = chexingdeal[chexing\_top10[6]] chexing\_7 = chexingdeal[chexing\_top10[7]] chexing\_8 = chexingdeal[chexing\_top10[8]] chexing\_9 = chexingdeal[chexing\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", chexing\_0) .add\_yaxis("top\_2", chexing\_1) .add\_yaxis("top\_3", chexing\_2) .add\_yaxis("top\_4", chexing\_3) .add\_yaxis("top\_5", chexing\_4) .add\_yaxis("top\_6", chexing\_5) .add\_yaxis("top\_7", chexing\_6) .add\_yaxis("top\_8", chexing\_7) .add\_yaxis("top\_9", chexing\_8)

.add\_yaxis("top\_10", chexing\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/车系 id 销量前 10 销售周期散点图.html')

)

# 后 10

chexingid\_dreciprocal10 = dict(chexingid\_dict1[-10:]) print(chexingid\_dreciprocal10) # len(chexingid\_reciprocal10)

chexing\_reciprocal10 = list(chexingid\_dreciprocal10) print(chexing\_reciprocal10) chexing\_d0 = chexingdeal[chexing\_reciprocal10[0]] chexing\_d1 = chexingdeal[chexing\_reciprocal10[1]] chexing\_d2 = chexingdeal[chexing\_reciprocal10[2]] chexing\_d3 = chexingdeal[chexing\_reciprocal10[3]] chexing\_d4 = chexingdeal[chexing\_reciprocal10[4]] chexing\_d5 = chexingdeal[chexing\_reciprocal10[5]] chexing\_d6 = chexingdeal[chexing\_reciprocal10[6]] chexing\_d7 = chexingdeal[chexing\_reciprocal10[7]] chexing\_d8 = chexingdeal[chexing\_reciprocal10[8]] chexing\_d9 = chexingdeal[chexing\_reciprocal10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", chexing\_d0) .add\_yaxis("reci\_2", chexing\_d1) .add\_yaxis("reci\_3", chexing\_d2) .add\_yaxis("reci\_4", chexing\_d3) .add\_yaxis("reci\_5", chexing\_d4) .add\_yaxis("reci\_6", chexing\_d5) .add\_yaxis("reci\_7", chexing\_d6) .add\_yaxis("reci\_8", chexing\_d7) .add\_yaxis("reci\_9", chexing\_d8)

.add\_yaxis("reci\_10", chexing\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期")

# )

.render('./charts/车系 id 销量后 10 销售周期散点图.html')

)

# 里程

# 排序，按里程

licheng\_dict = counter(list(df['里程'])) print("len: ", len(licheng\_dict)) print( "max: " , max(list(df['里程']))) print( "min: " , min(list(df['里程'])))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data = list(licheng\_dict))

.add\_yaxis("数量", y\_axis = list(licheng\_dict.values()))

.render('./charts/里程对销售周期的影响折线图.html')

)

# 按销量排序

licheng\_dict1 = sorted(licheng\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理里程与成交周期

lichengdeal = df.groupby('里程')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dlicheng\_top10 = dict(licheng\_dict1[:10]) print(dlicheng\_top10) licheng\_top10 = list(dlicheng\_top10) print(licheng\_top10) licheng\_0 = lichengdeal[licheng\_top10[0]] licheng\_1 = lichengdeal[licheng\_top10[1]] licheng\_2 = lichengdeal[licheng\_top10[2]] licheng\_3 = lichengdeal[licheng\_top10[3]] licheng\_4 = lichengdeal[licheng\_top10[4]] licheng\_5 = lichengdeal[licheng\_top10[5]] licheng\_6 = lichengdeal[licheng\_top10[6]] licheng\_7 = lichengdeal[licheng\_top10[7]] licheng\_8 = lichengdeal[licheng\_top10[8]] licheng\_9 = lichengdeal[licheng\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", licheng\_0) .add\_yaxis("top\_2", licheng\_1) .add\_yaxis("top\_3", licheng\_2) .add\_yaxis("top\_4", licheng\_3) .add\_yaxis("top\_5", licheng\_4) .add\_yaxis("top\_6", licheng\_5) .add\_yaxis("top\_7", licheng\_6) .add\_yaxis("top\_8", licheng\_7) .add\_yaxis("top\_9", licheng\_8)

.add\_yaxis("top\_10", licheng\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/里程销量前 10 销售周期散点图.html')

)

# 后 10

dlicheng\_dreciprocal10 = dict(licheng\_dict1[-10:]) print(dlicheng\_dreciprocal10) # len(dlicheng\_reciprocal10)

licheng\_reciprocal10 = list(dlicheng\_dreciprocal10) print(licheng\_reciprocal10) licheng\_d0 = lichengdeal[licheng\_reciprocal10[0]] licheng\_d1 = lichengdeal[licheng\_reciprocal10[1]] licheng\_d2 = lichengdeal[licheng\_reciprocal10[2]] licheng\_d3 = lichengdeal[licheng\_reciprocal10[3]] licheng\_d4 = lichengdeal[licheng\_reciprocal10[4]] licheng\_d5 = lichengdeal[licheng\_reciprocal10[5]] licheng\_d6 = lichengdeal[licheng\_reciprocal10[6]] licheng\_d7 = lichengdeal[licheng\_reciprocal10[7]] licheng\_d8 = lichengdeal[licheng\_reciprocal10[8]] licheng\_d9 = lichengdeal[licheng\_reciprocal10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", licheng\_d0) .add\_yaxis("reci\_2", licheng\_d1) .add\_yaxis("reci\_3", licheng\_d2) .add\_yaxis("reci\_4", licheng\_d3) .add\_yaxis("reci\_5", licheng\_d4) .add\_yaxis("reci\_6", licheng\_d5) .add\_yaxis("reci\_7", licheng\_d6) .add\_yaxis("reci\_8", licheng\_d7) .add\_yaxis("reci\_9", licheng\_d8)

.add\_yaxis("reci\_10", licheng\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期")

# )

.render('./charts/里程销量后 10 销售周期散点图.html')

)

# 车辆颜色

# 排序，按车辆颜色

color\_dict = counter(list(df['车辆颜色']))

# 按销量排序

color\_dict1 = sorted(color\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理车辆颜色与成交周期

colordeal = df.groupby('车辆颜色')['成交周期(天)'].apply(list).to\_dict()

dcolor\_top10 = dict(color\_dict1[:10]) print(dcolor\_top10) color\_top10 = list(dcolor\_top10) print(color\_top10) color\_0 = colordeal[color\_top10[0]] color\_1 = colordeal[color\_top10[1]] color\_2 = colordeal[color\_top10[2]] color\_3 = colordeal[color\_top10[3]] color\_4 = colordeal[color\_top10[4]] color\_5 = colordeal[color\_top10[5]] color\_6 = colordeal[color\_top10[6]] color\_7 = colordeal[color\_top10[7]] color\_8 = colordeal[color\_top10[8]] color\_9 = colordeal[color\_top10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", color\_0) .add\_yaxis("top\_2", color\_1) .add\_yaxis("top\_3", color\_2) .add\_yaxis("top\_4", color\_3) .add\_yaxis("top\_5", color\_4) .add\_yaxis("top\_6", color\_5) .add\_yaxis("top\_7", color\_6) .add\_yaxis("top\_8", color\_7) .add\_yaxis("top\_9", color\_8)

.add\_yaxis("top\_10", color\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/颜色对交易周期影响分布图.html')

)

# 车辆所在城市 id

# 排序，按车辆所在城市 id

city\_dict = counter(list(df['车辆所在城市 id']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data=list(city\_dict))

.add\_yaxis("数量", y\_axis=list(city\_dict.values()))

.render('./charts/车辆所在城市对销售周期的影响折线图.html')

)

# 按销量排序

city\_dict1 = sorted(city\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理车辆所在城市 id 与成交周期

citydeal = df.groupby('车辆所在城市 id')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dcity\_top10 = dict(city\_dict1[:10]) print(dcity\_top10) city\_top10 = list(dcity\_top10) print(city\_top10) city\_0 = citydeal[city\_top10[0]] city\_1 = citydeal[city\_top10[1]] city\_2 = citydeal[city\_top10[2]] city\_3 = citydeal[city\_top10[3]] city\_4 = citydeal[city\_top10[4]] city\_5 = citydeal[city\_top10[5]] city\_6 = citydeal[city\_top10[6]] city\_7 = citydeal[city\_top10[7]] city\_8 = citydeal[city\_top10[8]] city\_9 = citydeal[city\_top10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", city\_0) .add\_yaxis("top\_2", city\_1) .add\_yaxis("top\_3", city\_2) .add\_yaxis("top\_4", city\_3) .add\_yaxis("top\_5", city\_4) .add\_yaxis("top\_6", city\_5) .add\_yaxis("top\_7", city\_6) .add\_yaxis("top\_8", city\_7) .add\_yaxis("top\_9", city\_8)

.add\_yaxis("top\_10", city\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/车辆所在城市销量前 10 销售周期散点图.html')

)

# 后 10

dcity\_dreciprocal10 = dict(city\_dict1[-10:]) print(dcity\_dreciprocal10) # len(dcity\_reciprocal10)

city\_reciprocal10 = list(dcity\_dreciprocal10) print(city\_reciprocal10) city\_d0 = citydeal[city\_reciprocal10[0]] city\_d1 = citydeal[city\_reciprocal10[1]] city\_d2 = citydeal[city\_reciprocal10[2]] city\_d3 = citydeal[city\_reciprocal10[3]] city\_d4 = citydeal[city\_reciprocal10[4]] city\_d5 = citydeal[city\_reciprocal10[5]] city\_d6 = citydeal[city\_reciprocal10[6]] city\_d7 = citydeal[city\_reciprocal10[7]] city\_d8 = citydeal[city\_reciprocal10[8]] city\_d9 = citydeal[city\_reciprocal10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", city\_d0) .add\_yaxis("reci\_2", city\_d1) .add\_yaxis("reci\_3", city\_d2) .add\_yaxis("reci\_4", city\_d3) .add\_yaxis("reci\_5", city\_d4) .add\_yaxis("reci\_6", city\_d5) .add\_yaxis("reci\_7", city\_d6) .add\_yaxis("reci\_8", city\_d7) .add\_yaxis("reci\_9", city\_d8)

.add\_yaxis("reci\_10", city\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期")

# )

.render('./charts/车辆所在城市销量后 10 销售周期散点图.html')

)

# 过户次数

# 排序，按过户次数

guohucount\_dict = counter(list(df['过户次数']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data = list(guohucount\_dict))

.add\_yaxis("数量", y\_axis = list(guohucount\_dict.values()))

.render('./charts/过户次数对销售周期的影响折线图.html')

)

# 按销量排序

guohucount\_dict1 = sorted(guohucount\_dict.items(), key = lambda x:x[1], reverse

= True)

# 整理过户次数与成交周期

guohucountdeal = df.groupby('过户次数')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dguohucount\_top10 = dict(guohucount\_dict1[:10]) print(dguohucount\_top10)

guohucount\_top10 = list(dguohucount\_top10) print(guohucount\_top10) guohucount\_0 = guohucountdeal[guohucount\_top10[0]] guohucount\_1 = guohucountdeal[guohucount\_top10[1]] guohucount\_2 = guohucountdeal[guohucount\_top10[2]] guohucount\_3 = guohucountdeal[guohucount\_top10[3]] guohucount\_4 = guohucountdeal[guohucount\_top10[4]] guohucount\_5 = guohucountdeal[guohucount\_top10[5]] guohucount\_6 = guohucountdeal[guohucount\_top10[6]] guohucount\_7 = guohucountdeal[guohucount\_top10[7]] guohucount\_8 = guohucountdeal[guohucount\_top10[8]] guohucount\_9 = guohucountdeal[guohucount\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", guohucount\_0) .add\_yaxis("top\_2", guohucount\_1) .add\_yaxis("top\_3", guohucount\_2) .add\_yaxis("top\_4", guohucount\_3) .add\_yaxis("top\_5", guohucount\_4) .add\_yaxis("top\_6", guohucount\_5) .add\_yaxis("top\_7", guohucount\_6) .add\_yaxis("top\_8", guohucount\_7) .add\_yaxis("top\_9", guohucount\_8)

.add\_yaxis("top\_10", guohucount\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/ 过户次数对销售周期散点图.html')

)

# 载客人数

# 排序，按载客人数

zaikerenshu\_dict = counter(list(df['载客人数']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(list(zaikerenshu\_dict))

.add\_yaxis("数量", list(zaikerenshu\_dict.values()))

.render('./charts/载客人数对销售周期的影响折线图.html')

)

# 按销量排序

zaikerenshu\_dict1 = sorted(zaikerenshu\_dict.items(), key = lambda x:x[1], reverse

= True)

# 整理载客人数与成交周期

zaikerenshudeal = df.groupby('载客人数')['成交周期(天)'].apply(list).to\_dict()

# 前 10

zaikerenshu\_top10 = list(zaikerenshudeal) print(zaikerenshu\_top10) zaikerenshu\_0 = zaikerenshudeal[zaikerenshu\_top10[0]] zaikerenshu\_1 = zaikerenshudeal[zaikerenshu\_top10[1]] zaikerenshu\_2 = zaikerenshudeal[zaikerenshu\_top10[2]] zaikerenshu\_3 = zaikerenshudeal[zaikerenshu\_top10[3]] zaikerenshu\_4 = zaikerenshudeal[zaikerenshu\_top10[4]] zaikerenshu\_5 = zaikerenshudeal[zaikerenshu\_top10[5]] zaikerenshu\_6 = zaikerenshudeal[zaikerenshu\_top10[6]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", zaikerenshu\_0) .add\_yaxis("top\_2", zaikerenshu\_1) .add\_yaxis("top\_3", zaikerenshu\_2) .add\_yaxis("top\_4", zaikerenshu\_3) .add\_yaxis("top\_5", zaikerenshu\_4) .add\_yaxis("top\_6", zaikerenshu\_5)

.add\_yaxis("top\_7", zaikerenshu\_6)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/载客人数对销售周期影响散点图.html')

)

# 排量

# 排序，按排量

pailiang\_dict = counter(list(df['排量']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data = list(pailiang\_dict))

.add\_yaxis("数量", y\_axis = list(pailiang\_dict.values()))

.render('./charts/排量对销售周期的影响折线图.html')

)

# 按销量排序

pailiang\_dict1 = sorted(pailiang\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理排量与成交周期

pailiangdeal = df.groupby('排量')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dpailiang\_top10 = dict(pailiang\_dict1[:10]) print(dpailiang\_top10) pailiang\_top10 = list(dpailiang\_top10) print(pailiang\_top10) pailiang\_0 = pailiangdeal[pailiang\_top10[0]] pailiang\_1 = pailiangdeal[pailiang\_top10[1]] pailiang\_2 = pailiangdeal[pailiang\_top10[2]] pailiang\_3 = pailiangdeal[pailiang\_top10[3]] pailiang\_4 = pailiangdeal[pailiang\_top10[4]] pailiang\_5 = pailiangdeal[pailiang\_top10[5]] pailiang\_6 = pailiangdeal[pailiang\_top10[6]] pailiang\_7 = pailiangdeal[pailiang\_top10[7]] pailiang\_8 = pailiangdeal[pailiang\_top10[8]] pailiang\_9 = pailiangdeal[pailiang\_top10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", pailiang\_0) .add\_yaxis("top\_2", pailiang\_1) .add\_yaxis("top\_3", pailiang\_2) .add\_yaxis("top\_4", pailiang\_3) .add\_yaxis("top\_5", pailiang\_4) .add\_yaxis("top\_6", pailiang\_5) .add\_yaxis("top\_7", pailiang\_6) .add\_yaxis("top\_8", pailiang\_7) .add\_yaxis("top\_9", pailiang\_8)

.add\_yaxis("top\_10", pailiang\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/排量销量前 10 销售周期散点图.html')

)

# 后 10

dpailiang\_dreciprocal10 = dict(pailiang\_dict1[-10:]) print(dpailiang\_dreciprocal10) # len(dpailiang\_reciprocal10)

pailiang\_reciprocal10 = list(dpailiang\_dreciprocal10) print(pailiang\_reciprocal10) pailiang\_d0 = pailiangdeal[pailiang\_reciprocal10[0]] pailiang\_d1 = pailiangdeal[pailiang\_reciprocal10[1]] pailiang\_d2 = pailiangdeal[pailiang\_reciprocal10[2]] pailiang\_d3 = pailiangdeal[pailiang\_reciprocal10[3]] pailiang\_d4 = pailiangdeal[pailiang\_reciprocal10[4]] pailiang\_d5 = pailiangdeal[pailiang\_reciprocal10[5]] pailiang\_d6 = pailiangdeal[pailiang\_reciprocal10[6]] pailiang\_d7 = pailiangdeal[pailiang\_reciprocal10[7]] pailiang\_d8 = pailiangdeal[pailiang\_reciprocal10[8]] pailiang\_d9 = pailiangdeal[pailiang\_reciprocal10[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", pailiang\_d0) .add\_yaxis("reci\_2", pailiang\_d1) .add\_yaxis("reci\_3", pailiang\_d2) .add\_yaxis("reci\_4", pailiang\_d3) .add\_yaxis("reci\_5", pailiang\_d4) .add\_yaxis("reci\_6", pailiang\_d5) .add\_yaxis("reci\_7", pailiang\_d6) .add\_yaxis("reci\_8", pailiang\_d7) .add\_yaxis("reci\_9", pailiang\_d8)

.add\_yaxis("reci\_10", pailiang\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期")

# )

.render('./charts/排量销量后 10 销售周期散点图.html')

)

# 变速箱

# 排序，按变速箱

biansuxiang\_dict = counter(list(df['变速箱']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data = list(biansuxiang\_dict))

.add\_yaxis("数量", y\_axis = list(biansuxiang\_dict.values()))

.render('./charts/变速箱对销售周期的影响折线图.html')

)

# 按销量排序

biansuxiang\_dict1 = sorted(biansuxiang\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理变速箱与成交周期

biansuxiangdeal = df.groupby('变速箱')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dbiansuxiang\_top10 = dict(biansuxiang\_dict1[:10]) print(dbiansuxiang\_top10)

biansuxiang\_top10 = list(dbiansuxiang\_top10) print(biansuxiang\_top10) biansuxiang\_0 = biansuxiangdeal[biansuxiang\_top10[0]] biansuxiang\_1 = biansuxiangdeal[biansuxiang\_top10[1]] biansuxiang\_2 = biansuxiangdeal[biansuxiang\_top10[2]] biansuxiang\_3 = biansuxiangdeal[biansuxiang\_top10[3]] biansuxiang\_4 = biansuxiangdeal[biansuxiang\_top10[4]] biansuxiang\_5 = biansuxiangdeal[biansuxiang\_top10[5]] biansuxiang\_6 = biansuxiangdeal[biansuxiang\_top10[6]] biansuxiang\_7 = biansuxiangdeal[biansuxiang\_top10[7]] biansuxiang\_8 = biansuxiangdeal[biansuxiang\_top10[8]] biansuxiang\_9 = biansuxiangdeal[biansuxiang\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", biansuxiang\_0) .add\_yaxis("top\_2", biansuxiang\_1) .add\_yaxis("top\_3", biansuxiang\_2) .add\_yaxis("top\_4", biansuxiang\_3) .add\_yaxis("top\_5", biansuxiang\_4) .add\_yaxis("top\_6", biansuxiang\_5) .add\_yaxis("top\_7", biansuxiang\_6) .add\_yaxis("top\_8", biansuxiang\_7) .add\_yaxis("top\_9", biansuxiang\_8)

.add\_yaxis("top\_10", biansuxiang\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/变速箱销量前 10 销售周期散点图.html')

)

# 后 10

dbiansuxiang\_dreciprocal10 = dict(biansuxiang\_dict1[-10:]) print(dbiansuxiang\_dreciprocal10) # len(dbiansuxiang\_reciprocal10)

biansuxiang\_reciprocal10 = list(dbiansuxiang\_dreciprocal10) print(biansuxiang\_reciprocal10) biansuxiang\_d0 = biansuxiangdeal[biansuxiang\_reciprocal10[0]] biansuxiang\_d1 = biansuxiangdeal[biansuxiang\_reciprocal10[1]] biansuxiang\_d2 = biansuxiangdeal[biansuxiang\_reciprocal10[2]] biansuxiang\_d3 = biansuxiangdeal[biansuxiang\_reciprocal10[3]] biansuxiang\_d4 = biansuxiangdeal[biansuxiang\_reciprocal10[4]] biansuxiang\_d5 = biansuxiangdeal[biansuxiang\_reciprocal10[5]] biansuxiang\_d6 = biansuxiangdeal[biansuxiang\_reciprocal10[6]] biansuxiang\_d7 = biansuxiangdeal[biansuxiang\_reciprocal10[7]] biansuxiang\_d8 = biansuxiangdeal[biansuxiang\_reciprocal10[8]] # biansuxiang\_d9 = biansuxiangdeal[biansuxiang\_reciprocal10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("reci\_1", biansuxiang\_d0) .add\_yaxis("reci\_2", biansuxiang\_d1) .add\_yaxis("reci\_3", biansuxiang\_d2) .add\_yaxis("reci\_4", biansuxiang\_d3) .add\_yaxis("reci\_5", biansuxiang\_d4) .add\_yaxis("reci\_6", biansuxiang\_d5) .add\_yaxis("reci\_7", biansuxiang\_d6) .add\_yaxis("reci\_8", biansuxiang\_d7) .add\_yaxis("reci\_9", biansuxiang\_d8)

# .add\_yaxis("reci\_10", biansuxiang\_d9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "后 10 的销售周期")

# )

.render('./charts/变速箱销量后 10 销售周期散点图.html')

)

# 燃油类型

# 排序，按燃油类型

ranyouleixing\_dict = counter(list(df['燃油类型']))

# 绘制折线图显示

c = (

Line()

.add\_xaxis(xaxis\_data = list(ranyouleixing\_dict))

.add\_yaxis("数量", y\_axis = list(ranyouleixing\_dict.values()))

.render('./charts/燃油类型对销售周期的影响折线图.html')

)

# 按销量排序

ranyouleixing\_dict1 = sorted(ranyouleixing\_dict.items(), key = lambda x:x[1], reverse = True)

# 整理燃油类型与成交周期

ranyouleixingdeal = df.groupby('燃油类型')['成交周期(天)'].apply(list).to\_dict()

# 前 10

dranyouleixing\_top10 = dict(ranyouleixing\_dict1) print(dranyouleixing\_top10)

ranyouleixing\_top10 = list(dranyouleixing\_top10) print(ranyouleixing\_top10) ranyouleixing\_0 = ranyouleixingdeal[ranyouleixing\_top10[0]] ranyouleixing\_1 = ranyouleixingdeal[ranyouleixing\_top10[1]] ranyouleixing\_2 = ranyouleixingdeal[ranyouleixing\_top10[2]] ranyouleixing\_3 = ranyouleixingdeal[ranyouleixing\_top10[3]] ranyouleixing\_4 = ranyouleixingdeal[ranyouleixing\_top10[4]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", ranyouleixing\_0) .add\_yaxis("top\_2", ranyouleixing\_1) .add\_yaxis("top\_3", ranyouleixing\_2)

.add\_yaxis("top\_4", ranyouleixing\_3)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/燃油类型对销售周期影响散点图.html')

)

# 新车价—— 变化较大

# df['价格折损'] = df['新车价'] - df['上架价格']

# df\_time = pd.concat([df['展销时间'], df['注册日期'], df['上牌日期'], df['上架时间'], df['成交时间']], axis = 1)

# 基于皮尔逊相关系数

pearson1 = df.corr() print(pearson1)

# 相关系数计算

df['价格差'] = df['新车价'] - df['上架价格'] df['上架时间'] = pd.to\_datetime(df['上架时间']) df['注册日期'] = pd.to\_datetime(df['注册日期']) df['日期差'] = df['上架时间'] - df['注册日期'] df['日期差'] = df['日期差'].apply(lambda x : x.days)

# 基于皮尔逊相关系数

pearson2 = df.corr() print(pearson2)

xishu = dict(zip(list(pearson2.index), list(np.abs(np.array(list(pearson2[' 成交周期

(天)']))))))

xishu = sorted(xishu.items(), key = lambda x:x[1], reverse = True) xishu = dict(xishu)

print(xishu)

2.2.3 销售周期区间分析

import pandas as pd import numpy as np from collections import Counter

from pyecharts import options as opts from pyecharts.charts import Scatter from pyecharts.charts import Line from pyecharts.charts import Pie

def counter(list):

from collections import Counter

# 对列表内的元素进行排序，计数，以列表的形式返回

list\_counter = Counter(list) list\_sorted = sorted(list\_counter.items(), key = lambda x:x[0], reverse = False)

# lambda x:x[1]按值排序，reverse = True 倒序 即大值在前。 list\_dict = dict(list\_sorted) return list\_dict

df = pd.read\_csv('./data/附件 4-4（含周期区间）.csv')

# 成交周期

transaction = list(df['成交周期(天)']) # transaction

# 成交周期区间

cycle\_interval = list(df['周期标签']) # print(type(cycle\_interval[0]))

cycle\_interval\_dict0 = counter(cycle\_interval)

cycle\_interval\_list = sorted(cycle\_interval\_dict0.items(), key = lambda x : x[1], reverse = False) cycleon = list(cycle\_interval\_dict0)

# 绘制销售周期分布图

c = (

Pie()

.add("", cycle\_interval\_list)

.set\_global\_opts(title\_opts = opts.TitleOpts(title = "销售周期区间分布饼图"))

.set\_series\_opts(label\_opts = opts.LabelOpts(formatter = "{b}:{c}"))

.render('./charts/销售周期区间分布饼图.html')

)

# 分析品牌对销售周期区间的影响

brandid\_dict = counter(list(df['品牌 id ']))

brandid\_dict1 = sorted(brandid\_dict.items(), key = lambda x:x[1], reverse = True) brandid\_top10 = dict(brandid\_dict1[:10]) brand\_top10 = list(brandid\_top10) brandcycle = df.groupby('品牌 id ')['周期标签'].apply(list).to\_dict()

brand\_0 = brandcycle[brand\_top10[0]] brand\_1 = brandcycle[brand\_top10[1]] brand\_2 = brandcycle[brand\_top10[2]] brand\_3 = brandcycle[brand\_top10[3]] brand\_4 = brandcycle[brand\_top10[4]] brand\_5 = brandcycle[brand\_top10[5]] brand\_6 = brandcycle[brand\_top10[6]] brand\_7 = brandcycle[brand\_top10[7]] brand\_8 = brandcycle[brand\_top10[8]] brand\_9 = brandcycle[brand\_top10[9]]

brand0 = list(counter(brand\_0).values()) brand1 = list(counter(brand\_1).values()) brand2 = list(counter(brand\_2).values()) brand3 = list(counter(brand\_3).values()) brand4 = list(counter(brand\_4).values()) brand5 = list(counter(brand\_5).values()) brand6 = list(counter(brand\_6).values()) brand7 = list(counter(brand\_7).values()) brand8 = list(counter(brand\_8).values()) brand9 = list(counter(brand\_9).values())

c = (

Scatter()

.add\_xaxis(cycleon)

.add\_yaxis("top\_1", brand0) .add\_yaxis("top\_2", brand1) .add\_yaxis("top\_3", brand2) .add\_yaxis("top\_4", brand3) .add\_yaxis("top\_5", brand4) .add\_yaxis("top\_6", brand5) .add\_yaxis("top\_7", brand6) .add\_yaxis("top\_8", brand7) .add\_yaxis("top\_9", brand8)

.add\_yaxis("top\_10", brand9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "品牌前 10 的销售周期区间")

# )

.render('./charts/品牌前 10 的销售周期区间散点图.html')

)

# 车辆颜色

color\_dict = counter(list(df['车辆颜色'])) colorlist = list(color\_dict)

# 整理车辆颜色与成交周期区间

colorcycle = df.groupby('车辆颜色')['周期标签'].apply(list).to\_dict()

color\_0 = colorcycle[colorlist[0]] color\_1 = colorcycle[colorlist[1]] color\_2 = colorcycle[colorlist[2]] color\_3 = colorcycle[colorlist[3]] color\_4 = colorcycle[colorlist[4]] color\_5 = colorcycle[colorlist[5]] color\_6 = colorcycle[colorlist[6]] color\_7 = colorcycle[colorlist[7]] color\_8 = colorcycle[colorlist[8]] color\_9 = colorcycle[colorlist[9]]

color0 = list(counter(color\_0).values()) color1 = list(counter(color\_1).values()) color2 = list(counter(color\_2).values()) color3 = list(counter(color\_3).values()) color4 = list(counter(color\_4).values()) color5 = list(counter(color\_5).values()) color6 = list(counter(color\_6).values()) color7 = list(counter(color\_7).values()) color8 = list(counter(color\_8).values()) color9 = list(counter(color\_9).values()) c = (

Scatter()

.add\_xaxis(cycleon)

.add\_yaxis("top\_1", color0) .add\_yaxis("top\_2", color1) .add\_yaxis("top\_3", color2) .add\_yaxis("top\_4", color3) .add\_yaxis("top\_5", color4) .add\_yaxis("top\_6", color5) .add\_yaxis("top\_7", color6) .add\_yaxis("top\_8", color7) .add\_yaxis("top\_9", color8)

.add\_yaxis("top\_10", color9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期区间")

# )

.render('./charts/车辆颜色销售区间分布情况散点图.html')

)

# 过户次数

# 排序，按过户次数

guohucount\_dict = counter(list(df['过户次数'])) guohucountlist = list(guohucount\_dict)

# 整理过户次数与成交周期区间

guohucountcycle = df.groupby('过户次数')['周期标签'].apply(list).to\_dict()

guohucount\_0 = guohucountcycle[guohucountlist[0]] guohucount\_1 = guohucountcycle[guohucountlist[1]] guohucount\_2 = guohucountcycle[guohucountlist[2]] guohucount\_3 = guohucountcycle[guohucountlist[3]] guohucount\_4 = guohucountcycle[guohucountlist[4]] guohucount\_5 = guohucountcycle[guohucountlist[5]] guohucount\_6 = guohucountcycle[guohucountlist[6]] guohucount\_7 = guohucountcycle[guohucountlist[7]] guohucount\_8 = guohucountcycle[guohucountlist[8]] guohucount\_9 = guohucountcycle[guohucountlist[9]]

guohucount0 = list(counter(guohucount\_0).values()) guohucount1 = list(counter(guohucount\_1).values()) guohucount2 = list(counter(guohucount\_2).values()) guohucount3 = list(counter(guohucount\_3).values()) guohucount4 = list(counter(guohucount\_4).values()) guohucount5 = list(counter(guohucount\_5).values()) guohucount6 = list(counter(guohucount\_6).values()) guohucount7 = list(counter(guohucount\_7).values()) guohucount8 = list(counter(guohucount\_8).values()) guohucount9 = list(counter(guohucount\_9).values())

c = (

Scatter()

.add\_xaxis(cycleon)

.add\_yaxis("top\_1", guohucount0) .add\_yaxis("top\_2", guohucount1) .add\_yaxis("top\_3", guohucount2) .add\_yaxis("top\_4", guohucount3) .add\_yaxis("top\_5", guohucount4) .add\_yaxis("top\_6", guohucount5) .add\_yaxis("top\_7", guohucount6) .add\_yaxis("top\_8", guohucount7) .add\_yaxis("top\_9", guohucount8)

.add\_yaxis("top\_10", guohucount9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期区间")

# )

.render('./charts/过户次数销售区间分布情况散点图.html')

)

# 载客人数

# 排序，按载客人数

zaikerenshu\_dict = counter(list(df['载客人数'])) zaikerenshulist = list(zaikerenshu\_dict)

# 整理载客人数与成交周期区间

zaikerenshucycle = df.groupby('载客人数')['周期标签'].apply(list).to\_dict()

zaikerenshu\_0 = zaikerenshucycle[zaikerenshulist[0]] zaikerenshu\_1 = zaikerenshucycle[zaikerenshulist[1]] zaikerenshu\_2 = zaikerenshucycle[zaikerenshulist[2]] zaikerenshu\_3 = zaikerenshucycle[zaikerenshulist[3]] zaikerenshu\_4 = zaikerenshucycle[zaikerenshulist[4]] zaikerenshu\_5 = zaikerenshucycle[zaikerenshulist[5]] zaikerenshu\_6 = zaikerenshucycle[zaikerenshulist[6]]

zaikerenshu0 = list(counter(zaikerenshu\_0).values()) zaikerenshu1 = list(counter(zaikerenshu\_1).values()) zaikerenshu2 = list(counter(zaikerenshu\_2).values()) zaikerenshu3 = list(counter(zaikerenshu\_3).values()) zaikerenshu4 = list(counter(zaikerenshu\_4).values()) zaikerenshu5 = list(counter(zaikerenshu\_5).values()) zaikerenshu6 = list(counter(zaikerenshu\_6).values())

c = (

Scatter()

.add\_xaxis(cycleon)

.add\_yaxis("top\_1", zaikerenshu0) .add\_yaxis("top\_2", zaikerenshu1) .add\_yaxis("top\_3", zaikerenshu2) .add\_yaxis("top\_4", zaikerenshu3) .add\_yaxis("top\_5", zaikerenshu4) .add\_yaxis("top\_6", zaikerenshu5)

.add\_yaxis("top\_7", zaikerenshu6)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期区间")

# )

.render('./charts/载客人数销售区间分布情况散点图.html')

)

# 燃油类型

# 排序，按燃油类型

ranyouleixing\_dict = counter(list(df['燃油类型'])) ranyouleixinglist = list(ranyouleixing\_dict)

# 整理燃油类型与成交周期区间

ranyouleixingcycle = df.groupby('燃油类型')['周期标签'].apply(list).to\_dict()

ranyouleixing\_0 = ranyouleixingcycle[ranyouleixinglist[0]] ranyouleixing\_1 = ranyouleixingcycle[ranyouleixinglist[1]] ranyouleixing\_2 = ranyouleixingcycle[ranyouleixinglist[2]] ranyouleixing\_3 = ranyouleixingcycle[ranyouleixinglist[3]] ranyouleixing\_4 = ranyouleixingcycle[ranyouleixinglist[4]] ranyouleixing0 = list(counter(ranyouleixing\_0).values()) ranyouleixing1 = list(counter(ranyouleixing\_1).values()) ranyouleixing2 = list(counter(ranyouleixing\_2).values()) ranyouleixing3 = list(counter(ranyouleixing\_3).values()) ranyouleixing4 = list(counter(ranyouleixing\_4).values())

c = (

Scatter()

.add\_xaxis(cycleon)

.add\_yaxis("top\_1", ranyouleixing0) .add\_yaxis("top\_2", ranyouleixing1) .add\_yaxis("top\_3", ranyouleixing2)

.add\_yaxis("top\_4", ranyouleixing3)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期区间")

# )

.render('./charts/燃油类型销售区间分布情况散点图.html')

)

2.2.4 销售周期区间分析 2

import pandas as pd import numpy as np from collections import Counter

from pyecharts import options as opts from pyecharts.charts import Scatter from pyecharts.charts import Line from pyecharts.charts import Pie

def counter(list):

from collections import Counter

# 对列表内的元素进行排序，计数，以列表的形式返回

list\_counter = Counter(list) list\_sorted = sorted(list\_counter.items(), key = lambda x:x[0], reverse = False)

# lambda x:x[1]按值排序，reverse = True 倒序 即大值在前。 list\_dict = dict(list\_sorted) return list\_dict

df = pd.read\_csv('./data/附件 4-4（含周期区间）.csv')

# 成交周期

transaction = list(df['成交周期(天)']) cycle = list(range(0, 278))

# 成交周期区间

cycle\_interval = list(df['周期标签'])

cycle\_interval\_dict0 = counter(cycle\_interval)

cycle\_interval\_list = sorted(cycle\_interval\_dict0.items(), key = lambda x : x[1], reverse = False)

print(cycle\_interval\_list)

cycle\_interval\_dict1 = dict(cycle\_interval\_list) cycleon = list(cycle\_interval\_dict0)

# 绘制销售周期分布图

# 绘制销售周期的散点图

c = (

Scatter()

.add\_xaxis(list(cycle\_interval\_dict1))

.add\_yaxis(

"次",

list(cycle\_interval\_dict1.values()),

)

.set\_global\_opts( title\_opts = opts.TitleOpts(title = '销售周期散点图')

)

.render('./charts/销售周期区间分布散点图 1.html')

)

c.render\_notebook()

# 绘制销售周期的散点图

c = (

Scatter()

.add\_xaxis(list(cycle))

.add\_yaxis( " ",

list(cycle\_interval),

)

.set\_global\_opts( title\_opts = opts.TitleOpts(title = '销售周期散点图')

)

)

# 分析品牌对销售周期区间的影响

brandid\_dict = counter(list(df['品牌 id '])) brandid\_dict1 = sorted(brandid\_dict.items(), key = lambda x:x[1], reverse = True) brandid\_top10 = dict(brandid\_dict1[:10]) brand\_top10 = list(brandid\_top10) brandcycle = df.groupby('品牌 id ')['周期标签'].apply(list).to\_dict()

brand\_0 = brandcycle[brand\_top10[0]] brand\_1 = brandcycle[brand\_top10[1]] brand\_2 = brandcycle[brand\_top10[2]] brand\_3 = brandcycle[brand\_top10[3]] brand\_4 = brandcycle[brand\_top10[4]] brand\_5 = brandcycle[brand\_top10[5]] brand\_6 = brandcycle[brand\_top10[6]] brand\_7 = brandcycle[brand\_top10[7]] brand\_8 = brandcycle[brand\_top10[8]] brand\_9 = brandcycle[brand\_top10[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", brand\_0) .add\_yaxis("top\_2", brand\_1) .add\_yaxis("top\_3", brand\_2) .add\_yaxis("top\_4", brand\_3) .add\_yaxis("top\_5", brand\_4) .add\_yaxis("top\_6", brand\_5) .add\_yaxis("top\_7", brand\_6) .add\_yaxis("top\_8", brand\_7) .add\_yaxis("top\_9", brand\_8)

.add\_yaxis("top\_10", brand\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "品牌前 10 的销售周期区间")

# )

.render('./charts/品牌前 10 的销售周期区间散点图 1.html')

)

# 车辆颜色

color\_dict = counter(list(df['车辆颜色'])) colorlist = list(color\_dict)

# 整理车辆颜色与成交周期区间

colorcycle = df.groupby('车辆颜色')['周期标签'].apply(list).to\_dict()

color\_0 = colorcycle[colorlist[0]] color\_1 = colorcycle[colorlist[1]] color\_2 = colorcycle[colorlist[2]] color\_3 = colorcycle[colorlist[3]] color\_4 = colorcycle[colorlist[4]] color\_5 = colorcycle[colorlist[5]] color\_6 = colorcycle[colorlist[6]] color\_7 = colorcycle[colorlist[7]] color\_8 = colorcycle[colorlist[8]] color\_9 = colorcycle[colorlist[9]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", color\_0) .add\_yaxis("top\_2", color\_1) .add\_yaxis("top\_3", color\_2) .add\_yaxis("top\_4", color\_3) .add\_yaxis("top\_5", color\_4) .add\_yaxis("top\_6", color\_5) .add\_yaxis("top\_7", color\_6) .add\_yaxis("top\_8", color\_7) .add\_yaxis("top\_9", color\_8)

.add\_yaxis("top\_10", color\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期区间")

# )

.render('./charts/车辆颜色销售区间分布情况散点图 1.html')

)

# 过户次数

# 排序，按过户次数

guohucount\_dict = counter(list(df['过户次数']))

guohucountlist = list(guohucount\_dict)

# 整理过户次数与成交周期区间

guohucountcycle = df.groupby('过户次数')['周期标签'].apply(list).to\_dict()

guohucount\_0 = guohucountcycle[guohucountlist[0]] guohucount\_1 = guohucountcycle[guohucountlist[1]] guohucount\_2 = guohucountcycle[guohucountlist[2]] guohucount\_3 = guohucountcycle[guohucountlist[3]] guohucount\_4 = guohucountcycle[guohucountlist[4]] guohucount\_5 = guohucountcycle[guohucountlist[5]] guohucount\_6 = guohucountcycle[guohucountlist[6]] guohucount\_7 = guohucountcycle[guohucountlist[7]] guohucount\_8 = guohucountcycle[guohucountlist[8]] guohucount\_9 = guohucountcycle[guohucountlist[9]] c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", guohucount\_0) .add\_yaxis("top\_2", guohucount\_1) .add\_yaxis("top\_3", guohucount\_2) .add\_yaxis("top\_4", guohucount\_3) .add\_yaxis("top\_5", guohucount\_4) .add\_yaxis("top\_6", guohucount\_5) .add\_yaxis("top\_7", guohucount\_6) .add\_yaxis("top\_8", guohucount\_7) .add\_yaxis("top\_9", guohucount\_8)

.add\_yaxis("top\_10", guohucount\_9)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/过户次数销售区间分布情况散点图 1.html')

)

# 载客人数

# 排序，按载客人数

zaikerenshu\_dict = counter(list(df['载客人数'])) zaikerenshulist = list(zaikerenshu\_dict)

# 整理载客人数与成交周期区间

zaikerenshucycle = df.groupby('载客人数')['周期标签'].apply(list).to\_dict() # zaikerenshucycle

zaikerenshu\_0 = zaikerenshucycle[zaikerenshulist[0]] zaikerenshu\_1 = zaikerenshucycle[zaikerenshulist[1]] zaikerenshu\_2 = zaikerenshucycle[zaikerenshulist[2]] zaikerenshu\_3 = zaikerenshucycle[zaikerenshulist[3]] zaikerenshu\_4 = zaikerenshucycle[zaikerenshulist[4]] zaikerenshu\_5 = zaikerenshucycle[zaikerenshulist[5]] zaikerenshu\_6 = zaikerenshucycle[zaikerenshulist[6]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", zaikerenshu\_0) .add\_yaxis("top\_2", zaikerenshu\_1) .add\_yaxis("top\_3", zaikerenshu\_2) .add\_yaxis("top\_4", zaikerenshu\_3) .add\_yaxis("top\_5", zaikerenshu\_4) .add\_yaxis("top\_6", zaikerenshu\_5)

.add\_yaxis("top\_7", zaikerenshu\_6)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/载客人数销售区间分布情况散点图 1.html')

)

# 燃油类型

# 排序，按燃油类型

ranyouleixing\_dict = counter(list(df['燃油类型'])) ranyouleixinglist = list(ranyouleixing\_dict)

# 整理燃油类型与成交周期区间

ranyouleixingcycle = df.groupby('燃油类型')['周期标签'].apply(list).to\_dict()

ranyouleixing\_0 = ranyouleixingcycle[ranyouleixinglist[0]] ranyouleixing\_1 = ranyouleixingcycle[ranyouleixinglist[1]] ranyouleixing\_2 = ranyouleixingcycle[ranyouleixinglist[2]] ranyouleixing\_3 = ranyouleixingcycle[ranyouleixinglist[3]] ranyouleixing\_4 = ranyouleixingcycle[ranyouleixinglist[4]]

c = (

Scatter()

.add\_xaxis(cycle)

.add\_yaxis("top\_1", ranyouleixing\_0) .add\_yaxis("top\_2", ranyouleixing\_1) .add\_yaxis("top\_3", ranyouleixing\_2) .add\_yaxis("top\_4", ranyouleixing\_3)

.add\_yaxis("top\_5", ranyouleixing\_4)

# .set\_global\_opts(

# title\_opts = opts.TitleOpts(title = "前 10 的销售周期")

# )

.render('./charts/燃油类型销售区间分布情况散点图 1.html')

)

2.3 进一步挖掘

import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from collections import Counter

data = pd.read\_csv(r"C:\Users\Lenovo\Desktop\总.csv") df = pd.DataFrame(data)

print(df.corr()) print(df.corr('spearman'))

print(df.corr('kendall')) hm = plt.subplots(figsize = (10,10))

hm = sns.heatmap(data.corr(), square=True, annot=True) plt.show()

data = pd.read\_csv("C:\\Users\Lenovo\Desktop\\4.csv", encoding='utf-8')

#print(data) for i in data.updatePriceTimeJson: i = i.split('"') if len(i) > 1: print(i) print(len(i)) print(i[1])

data.updateTime = data.updatePriceTimeJson.apply(lambda x:i[1]) data.updatePrice = i[3] print(data)

data = pd.read\_csv(r"C:\Users\Lenovo\Desktop\4\_all.csv") data.withdrawDate = data.withdrawDate.astype(str) data.withdraw\_month = data.withdrawDate.apply(lambda x:x[5:6]) print(data)

data.to\_csv("C:\\Users\Lenovo\Desktop\\4\_all2.csv", encoding='utf-8')

data = pd.read\_csv(r"C:\\Users\Lenovo\Desktop\\4\_all2.csv", encoding='utf-8') print(Counter(data.withdraw\_month))

2.4 问题二模型 2.4.1 仅附件四模型 import pandas as pd import numpy as np import re from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestRegressor

df\_1=pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件

4：门店交易训练数据.txt',sep='\t',

parse\_dates=[1,4,5],header=None,names=['carid','pushDate','pushPrice','updateP riceTimeJson'

,'pullDate','withdrawDate'])

data=pd.DataFrame(data=df\_1) data['pushDate\_year']=data['pushDate'].dt.year data['pushDate\_month']=data['pushDate'].dt.month data['pushDate\_day']=data['pushDate'].dt.day

data['withdrawDate\_year']=data['withdrawDate'].dt.year data['withdrawDate\_month']=data['withdrawDate'].dt.month data['withdrawDate\_day']=data['withdrawDate'].dt.day

data\_1=data

data\_1['period']=data\_1['withdrawDate']-data\_1['pushDate']

# pd.to\_datetime(data['period']) dtype timedelta64[ns] cannot be converted to datetime64[ns]

data\_1['period']=data\_1['period'].apply(lambda x: str(x).split(' ')[0]) data\_1['period']=data\_1['period'].apply(lambda x: int(x) if x!='NaT' else np.nan)

data\_1['updatePriceTimeJson'].astype('string') def deal(data=None):

li=[] for i in data.index: count=len(re.findall('\d+-\d+-\d+',str(data.loc[i,'updatePriceTimeJson']))) li.append(count) return li data\_1['decreasing\_count']=deal(data\_1) data\_1.info()

data\_2=data\_1.drop(['pushDate','pullDate','withdrawDate','updatePriceTimeJson'], axis=1)

data\_2=data\_2.dropna()

y\_=data\_2['period']

data\_2=data\_2.drop(['period'],axis=1) data\_2['period']=y\_ data\_2.info()

pearson=data\_2.corr()

index = pearson['period'][:-1].abs() > 0.01

X=data\_2.iloc[:,:-1]

X\_submit=X.loc[:,index]

X\_submit.columns

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X\_submit,data\_2['period'],test\_size=

0.2)

random\_model=RandomForestRegressor(n\_estimators=300,random\_state=33,n

\_jobs=-1) random\_model.fit(X\_train,y\_train) y\_pred=random\_model.predict(X\_test) random\_model.score(X\_test,y\_test)

2.4.2 结合附件四和附件一模型

import pandas as pd import numpy as np import re from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestRegressor

# df = pd.read\_table('../data/附件 1：估价训练数据.txt',sep='\t',encoding='gbk',

# parse\_dates=[1,11,12],header=None)

# df\_1 = pd.read\_table('../data/ 附 件 4 ： 门 店 交 易 训 练 数

据.txt',sep='\t',encoding='gbk',

# parse\_dates=[1,4,5],header=None)

df = pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件 1：

估价训练数据.txt',sep='\t',encoding='gbk',

parse\_dates=[1,11,12],header=None)

df\_1=pd.read\_table(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\附件\附件

4：门店交易训练数据.txt',sep='\t',

parse\_dates=[1,4,5],header=None,names=['carid','pushDate','pushPrice','updateP riceTimeJson'

,'pullDate','withdrawDate'])

def add\_columns(data=None):

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color', 'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country',

'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1, 16): str\_ = 'anonymousFeature' + str(i) columns.append(str\_)

columns.append('price') data.columns = columns

return data

df=add\_columns(df)

#

df\_1.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJso n',4:'pullDate',5:'withdrawDate'},inplace=True) data = pd.merge(df,df\_1,how='right')

data.info()

data=data.drop(['country','anonymousFeature4','anonymousFeature7','anonymous

Feature10','anonymousFeature15'

,'maketype','anonymousFeature1','anonymousFeature8','anony

mousFeature9'],axis=1)

data.info() nn = ['carCode', 'modelyear', 'gearbox'] for i in nn:

x = int(data[i].mode()) data[i].fillna(x, inplace=True)

#data[i]=data[i].dropna()

"""

特征工程

"""

data['tradeTime\_year']=data['tradeTime'].dt.year data['tradeTime\_month']=data['tradeTime'].dt.month data['tradeTime\_day']=data['tradeTime'].dt.day data['registerDate\_year']=data['registerDate'].dt.year data['registerDate\_month']=data['registerDate'].dt.month data['registerDate\_day']=data['registerDate'].dt.day

# 这个特征不适用与预测周期

data['old\_year'] = data['tradeTime'] - data['registerDate'] data['old\_year'] = data['old\_year'].apply(lambda x: str(x).split(' ')[0]) data['old\_year'] = data['old\_year'].astype(int)

# 处理匿名特征 11 data\_1=data.dropna() #

data\_1.info()

len(list(data\_1['anonymousFeature11']))

# data\_1['anonymousFeature11'].fillna('0',inplace=True) def deal\_11(x):

return sum([float(x) for x in re.findall("\d",x)])

data\_1['anonymousFeature11']=data\_1['anonymousFeature11'].map(deal\_11) data\_1.info()

# 处理匿名特征 12 def deal\_12(x):

li=[float(x) for x in re.findall("\d+",x)] return li[0]\*li[1]\*li[2]

data\_1['anonymousFeature12\_length']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[0]))

data\_1['anonymousFeature12\_width']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[1]))

data\_1['anonymousFeature12\_height']=data\_1['anonymousFeature12'].apply(lam bda x:int(x.split('\*')[2]))

data\_1['anonymousFeature12']=data\_1['anonymousFeature12'].map(deal\_12)

# 处理匿名特征 13 def deal\_13(x):

return x[:4], x[4:6]

data\_1['anonymousFeature13']=data\_1['anonymousFeature13'].astype('string') data\_1['anonymousFeature13\_year']=data\_1['anonymousFeature13'].map(deal\_1

3) # (2017, 09)

data\_1['anonymousFeature13\_month']=data\_1['anonymousFeature13\_year'].appl y(lambda x: int(x[1]))

data\_1['anonymousFeature13\_year']=data\_1['anonymousFeature13\_year'].apply(l ambda x: int(x[0])) data\_1['anonymousFeature13']=data\_1['anonymousFeature13'].astype('float')

data\_1['old\_year\_1']=data\_1['tradeTime']-data\_1['licenseDate'] data\_1['old\_year\_1']=data\_1['old\_year\_1'].apply(lambda x:str(x).split(' ')[0]) data\_1['old\_year\_1']=data\_1['old\_year\_1'].astype(int)

data\_1['period']=data\_1['withdrawDate']-data\_1['pushDate']

# pd.to\_datetime(data['period']) dtype timedelta64[ns] cannot be converted to datetime64[ns]

data\_1['period']=data\_1['period'].apply(lambda x: str(x).split(' ')[0]) data\_1['period']=data\_1['period'].apply(lambda x: int(x) if x!='NaT' else np.nan)

"""

特征处理

"""

data\_new = data\_1.dropna(subset=['period'])

data\_new.info() data\_new.isnull().sum()

data\_new\_2=data\_new.dropna() data\_new\_2.info()

# 处理 updatePriceTimeJson

data\_new\_2['updatePriceTimeJson'].astype('string') def deal(data=None):

li=[] for i in data.index: count=len(re.findall('\d+-\d+-\d+',str(data.loc[i,'updatePriceTimeJson']))) li.append(count) return li

data\_new\_2['decreasing\_count']=deal(data\_new\_2) data\_new\_2.info()

data\_new\_3=data\_new\_2.select\_dtypes(include=['float64', 'int32','int64'])

y\_=data\_new\_3['period']

data\_new\_3=data\_new\_3.drop(['period'],axis=1) data\_new\_3['period']=y\_ data\_new\_3.info()

# data\_new\_3.to\_csv(r'D:\比赛\2021 年 MathorCup 大数据竞赛赛道 A\任务二

result.csv')

pearson=data\_new\_3.corr() index=pearson['price'][:-1].abs() > 0.1

X=data\_new\_3.iloc[:,:-1]

X\_subset = X.loc[:, index]

X\_train,X\_test,y\_train,y\_test=train\_test\_split(data\_new\_3.iloc[:,:-1],data\_new\_3['p eriod'])

random\_model=RandomForestRegressor(n\_estimators=300,random\_state=33,n

\_jobs=-1) random\_model.fit(X\_train,y\_train) y\_pred=random\_model.predict(X\_test)

random\_model.score(X\_test,y\_test) random\_model.score(X\_train,y\_train)

# float("inf") 无穷大

## 附录3：成交周期预估分析代码

### 3.1 只进行回归

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import re

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.feature\_selection import SelectKBest

import xgboost as xgb

import lightgbm as lgb

import warnings

warnings.filterwarnings('ignore')

"""

评估数据集的构造

"""

# df\_5 = pd.read\_csv('./data/附件5：门店交易验证数据.txt', sep='\t', parse\_dates=[1], header=None, encoding='utf-8') # 附件五

# df\_2 = pd.read\_csv('../data/附件2：估价验证数据.txt', sep='\t', parse\_dates=[1, 11, 12], header=None, encoding='utf-8') # 附件二的信息

df\_5 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\复赛\data\附件5：门店交易验证数据.txt', sep='\t', parse\_dates=[1], header=None, encoding='utf-8') # 附件五

df\_2 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件2：估价验证数据.txt', sep='\t', parse\_dates=[1, 11, 12], header=None, encoding='utf-8') # 附件二的信息

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color', 'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country', 'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1,16):

columns.append('anonymousFeature'+str(i))

# 重命名数据列名

df\_2.columns = columns

# df\_5.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件五.csv', encoding='utf-8')

df\_5.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson'} ,inplace=True)

# df\_5.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件五1.csv', encoding='utf-8')

print(df\_2)

df\_test = pd.merge(df\_5, df\_2, on='carid', how='left')

# df\_test.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一test.csv')

"""

训练集的构造

"""

# df\_1 = pd.read\_csv('../data/附件1：估价训练数据.txt', sep='\t', parse\_dates=[1, 11, 12], encoding='utf-8', header=None) # 附件一

# df\_4 = pd.read\_csv('../data/附件4：门店交易训练数据.txt', sep='\t', parse\_dates=[1, 4, 5], encoding='utf-8', header=None) # 附件四

df\_1 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件1：估价训练数据.txt', sep='\t', parse\_dates=[1, 11, 12], encoding='utf-8', header=None) # 附件一

df\_4 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件4：门店交易训练数据.txt', sep='\t', parse\_dates=[1, 4, 5], encoding='utf-8', header=None) # 附件四

columns.append('price')

df\_1.columns=columns

df\_4.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson',4:'pullDate',5:'withdrawDate'},inplace=True)

df\_train = pd.merge(df\_4, df\_1, how='left', on='carid')

# print(df\_train.columns)

# print(df\_test.columns)

# 调整trian的数据特征，保持与test一致

df\_train['cycle'] = df\_train['withdrawDate']-df\_train['pushDate']

df\_train=df\_train.drop(['price', 'pullDate', 'withdrawDate'], axis=1)

df\_train['cycle']=df\_train['cycle'].astype('timedelta64[D]')

# df\_train.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一train.csv')

"""

数据处理

"""

df\_train=df\_train.dropna(subset=['cycle'])

df\_train.loc[df\_train['country']==0,'country']=779410

# df\_train['country'].unique()

# feature\_num=['mileage','transferCount','seatings','displacement','gearbox','newprice','cycle']

# feature\_class=['brand','serial','model','color','cityId','carCode','country','maketype','modelyear','oiltype']

# 数值型的相关系数

# data\_1=df\_train[feature\_num]

# pearson=data\_1.corr()

# pearson['cycle'].abs().sort\_values(ascending=False)

# pearson.index.to\_list()

# df\_train.dropna(how='any',subset=['carCode','gearbox'],inplace=True) # 训练集中有缺失值，缺失值较少 ：6个，1个

def fill\_(data):

n=['country','maketype','modelyear']

for i in n:

x=int(data[i].mode())

data[i].fillna(x,inplace=True)

return data

df\_train=fill\_(df\_train)

df\_test=fill\_(df\_test)

df\_train.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature15'],axis=1,inplace=True)

df\_test.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature15'],axis=1,inplace=True)

"""

异常值检验

"""

df\_train.isnull().sum()

def replace(x):

QU = x.quantile(0.75)

QL = x.quantile(0.25)

IQR = QU -QL

x[(x>(QU+1.5\*IQR)) | (x<(QL - 1.5\*IQR))] = np.nan

return x

nn=['mileage','transferCount','seatings','displacement','gearbox','newprice','pushPrice']

for i in nn:

df\_train[i] = replace(df\_train[i])

"""

特征工程

"""

def deal\_time(df\_train):

df\_train['old\_year']=df\_train['tradeTime']-df\_train['registerDate']

df\_train['old\_year']=df\_train['old\_year'].apply(lambda x:str(x).split(' ')[0])

df\_train['old\_year']=df\_train['old\_year'].astype(int)

df\_train['old\_year\_1']=df\_train['tradeTime']-df\_train['licenseDate']

df\_train['old\_year\_1']=df\_train['old\_year\_1'].apply(lambda x:str(x).split(' ')[0])

df\_train['old\_year\_1']=df\_train['old\_year\_1'].astype(int)

return df\_train

df\_train=deal\_time(df\_train)

df\_test=deal\_time(df\_test)

def f(x):

if re.findall(': ".+?"',str(x)):

res=re.findall(': ".+?"',str(x))[-1]

a=res.replace('"','').replace(':','').replace(' ','')

a=float(a)

else:

a=0.0

return a

# res=re.findall(': ".+?"','{"2020-12-29": "17.68", "2020-12-30": "17.8799", "2021-01-15": "16.8799"}')

# a=res[-1].replace('"','').replace(':','').replace(' ','')

# a=float(a)

# res=re.findall(': ".+?"','{}')[-1]

"""

求出最终的价格

"""

df\_train['last\_updatePrice']=df\_train['updatePriceTimeJson'].map(f)

df\_train.loc[df\_train['last\_updatePrice']==0.0000,'last\_updatePrice']=df\_train.loc[df\_train['last\_updatePrice']==0.0000,'pushPrice']

df\_test['last\_updatePrice']=df\_test['updatePriceTimeJson'].map(f)

df\_test.loc[df\_test['last\_updatePrice']==0.0000,'last\_updatePrice']=df\_test.loc[df\_test['last\_updatePrice']==0.0000,'pushPrice']

"""

构造特征

"""

def deal\_ano\_feature(df\_train):

df\_train['anonymousFeature11'].fillna('1+2',inplace=True)

def deal\_11(x):

return sum([float(x) for x in re.findall("\d",x)])

df\_train['anonymousFeature11']=df\_train['anonymousFeature11'].map(deal\_11)

# 处理匿名特征12

def deal\_12(x):

li=[float(x) for x in re.findall("\d+",x)]

return li[0]\*li[1]\*li[2]

df\_train['anonymousFeature12\_length']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[0]))

df\_train['anonymousFeature12\_width']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[1]))

df\_train['anonymousFeature12\_height']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[2]))

df\_train['anonymousFeature12']=df\_train['anonymousFeature12'].map(deal\_12)

# 处理匿名特征13

df\_train['anonymousFeature13'].fillna(int(df\_train['anonymousFeature13'].mode()),inplace=True)

def deal\_13(x):

return x[:4], x[4:6]

df\_train['anonymousFeature13']=df\_train['anonymousFeature13'].astype('string')

df\_train['anonymousFeature13\_year']=df\_train['anonymousFeature13'].map(deal\_13) # (2017, 09)

df\_train['anonymousFeature13\_month']=df\_train['anonymousFeature13\_year'].apply(lambda x: int(x[1]))

df\_train['anonymousFeature13\_year']=df\_train['anonymousFeature13\_year'].apply(lambda x: int(x[0]))

df\_train['anonymousFeature13']=df\_train['anonymousFeature13'].astype('float')

df\_train.isnull().sum()

# df\_train.dtypes

def deal(data=None):

li=[]

for i in data.index:

count=len(re.findall('\d+-\d+-\d+',str(data.loc[i,'updatePriceTimeJson'])))

li.append(count)

return li

df\_train['decreasing\_count']=deal(df\_train)

"""

对剩余的缺失值进行填充

"""

# 匿名变量1 , 8 ,9 ,10先用众数填充； 或者删除有缺失值的行

def fill\_v(data):

n=['anonymousFeature1','anonymousFeature8','anonymousFeature9','anonymousFeature10']

for i in n:

x=int(data[i].mode())

data[i].fillna(x,inplace=True)

return data

# df\_train=fill\_v(df\_train)

df\_train['pushDate\_year'] = df\_train['pushDate'].dt.year

df\_train['pushDate\_month'] = df\_train['pushDate'].dt.month

df\_train['pushDate\_day'] = df\_train['pushDate'].dt.day

return df\_train

df\_train=deal\_ano\_feature(df\_train)

df\_test=deal\_ano\_feature(df\_test)

# 将cycle调整到最后一列

# 数据分箱

bin = [i\*5 for i in range(9)]

df\_train['pushPrice\_bin'] = pd.cut(df\_train['pushPrice'], bin, labels=False)

df\_test['pushPrice\_bin'] = pd.cut(df\_test['pushPrice'], bin, labels=False)

df\_train=df\_train.select\_dtypes(include=['int64','float64','int32'])

df\_test=df\_test.select\_dtypes(include=['int64','float64','int32'])

# 对类别特征进行 OneEncoder

# df\_train=pd.get\_dummies(df\_train,columns=['brand','serial','model','pushPrice\_bin',

# 'color','cityId','carCode','country','maketype','modelyear','oiltype'])

# df\_test=pd.get\_dummies(df\_test,columns=['brand','serial','model','pushPrice\_bin',

# 'color','cityId','carCode','country','maketype','modelyear','oiltype'])

cycle=df\_train['cycle']

df\_train=df\_train.drop('cycle',axis=1)

df\_train['cycle']=cycle

df\_train['seatings'].unique()

df\_train=df\_train.drop('seatings',axis=1)

pearson=df\_train.corr()

index = pearson['cycle'][:-1].abs() > 0.03

X\_submit=df\_train.iloc[:,:-1]

X\_submit=X\_submit.loc[:,index]

features=X\_submit.columns

y=np.log1p(df\_train['cycle'])

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X\_submit.iloc[:,:],y,test\_size=0.3)

clf=lgb.LGBMRegressor(

learning\_rate=0.01,

max\_depth=-1,

n\_estimators=5000,

boosting\_type='gbdt',

random\_state=2022,

objective='regression',

)

clf.fit(X\_train,y\_train)

y\_pred=clf.predict(X\_test)

y\_pred=np.expm1(y\_pred)

res=mean\_absolute\_error(y\_test,y\_pred)

print("lightGbm: ",res)

# plt.scatter(X\_test['pushPrice'],y\_test, color='blue')

# plt.scatter(X\_test['pushPrice'],y\_pred, color='red')

# plt.xlabel('pushPrice')

# plt.ylabel('cycle')

# plt.legend(['ture','pred'])

# 13.206856546563632

X\_=df\_test[features]

X\_.columns

y\_res=clf.predict(X\_)

y\_res=np.expm1(y\_res)

df\_test['cycle']=y\_res

data=df\_test[['carid','cycle']]

# data.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\附件6：门店交易模型结果.txt',sep='\t',index=None,header=None)

### 3.2 分类和回归双模型

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import re

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.metrics import classification\_report

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

import xgboost as xgb

import lightgbm as lgb

import joblib

import warnings

warnings.filterwarnings('ignore')

"""

评估数据集的构造

"""

# df\_5 = pd.read\_csv('./data/附件5：门店交易验证数据.txt', sep='\t', parse\_dates=[1], header=None, encoding='utf-8') # 附件五

# df\_2 = pd.read\_csv('../data/附件2：估价验证数据.txt', sep='\t', parse\_dates=[1, 11, 12], header=None, encoding='utf-8') # 附件二的信息

df\_5 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\复赛\data\附件5：门店交易验证数据.txt', sep='\t', parse\_dates=[1], header=None, encoding='utf-8') # 附件五

df\_2 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件2：估价验证数据.txt', sep='\t', parse\_dates=[1, 11, 12], header=None, encoding='utf-8') # 附件二的信息

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color', 'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country', 'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1,16):

columns.append('anonymousFeature'+str(i))

# 重命名数据列名

df\_2.columns = columns

# df\_5.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件五.csv', encoding='utf-8')

df\_5.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson'} ,inplace=True)

# df\_5.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件五1.csv', encoding='utf-8')

print(df\_2)

df\_test = pd.merge(df\_5, df\_2, on='carid', how='left')

# df\_test.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一test.csv')

"""

训练集的构造

"""

# df\_1 = pd.read\_csv('../data/附件1：估价训练数据.txt', sep='\t', parse\_dates=[1, 11, 12], encoding='utf-8', header=None) # 附件一

# df\_4 = pd.read\_csv('../data/附件4：门店交易训练数据.txt', sep='\t', parse\_dates=[1, 4, 5], encoding='utf-8', header=None) # 附件四

df\_1 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件1：估价训练数据.txt', sep='\t', parse\_dates=[1, 11, 12], encoding='utf-8', header=None) # 附件一

df\_4 = pd.read\_csv(r'D:\应用\pycharm\PyCharm 2021.1.3\source\mathorcup\data\附件4：门店交易训练数据.txt', sep='\t', parse\_dates=[1, 4, 5], encoding='utf-8', header=None) # 附件四

columns.append('price')

df\_1.columns=columns

df\_4.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson',4:'pullDate',5:'withdrawDate'},inplace=True)

df\_train = pd.merge(df\_4, df\_1, how='left', on='carid')

# print(df\_train.columns)

# print(df\_test.columns)

# 调整trian的数据特征，保持与test一致

df\_train['cycle'] = df\_train['withdrawDate']-df\_train['pushDate']

df\_train=df\_train.drop(['price', 'pullDate', 'withdrawDate'], axis=1)

df\_train['cycle']=df\_train['cycle'].astype('timedelta64[D]')

# df\_train.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一train.csv')

"""

数据处理

"""

df\_train.dropna(how='any',subset=['carCode','gearbox'],inplace=True) # 训练集中有缺失值，缺失值较少 ：6个，1个

def fill\_(data):

n=['country','maketype','modelyear']

for i in n:

x=int(data[i].mode())

data[i].fillna(x,inplace=True)

return data

df\_train=fill\_(df\_train)

df\_test=fill\_(df\_test)

df\_train.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature15'],axis=1,inplace=True)

df\_test.drop(['anonymousFeature4','anonymousFeature7','anonymousFeature15'],axis=1,inplace=True)

"""

异常值检验，

"""

"""

特征工程

"""

def deal\_ano\_feature(df\_train):

df\_train['anonymousFeature11'].fillna('1+2',inplace=True)

def deal\_11(x):

return sum([float(x) for x in re.findall("\d",x)])

df\_train['anonymousFeature11']=df\_train['anonymousFeature11'].map(deal\_11)

# 处理匿名特征12

def deal\_12(x):

li=[float(x) for x in re.findall("\d+",x)]

return li[0]\*li[1]\*li[2]

df\_train['anonymousFeature12\_length']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[0]))

df\_train['anonymousFeature12\_width']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[1]))

df\_train['anonymousFeature12\_height']=df\_train['anonymousFeature12'].apply(lambda x:int(x.split('\*')[2]))

df\_train['anonymousFeature12']=df\_train['anonymousFeature12'].map(deal\_12)

# 处理匿名特征13

df\_train['anonymousFeature13'].fillna(int(df\_train['anonymousFeature13'].mode()),inplace=True)

def deal\_13(x):

return x[:4], x[4:6]

df\_train['anonymousFeature13']=df\_train['anonymousFeature13'].astype('string')

df\_train['anonymousFeature13\_year']=df\_train['anonymousFeature13'].map(deal\_13) # (2017, 09)

df\_train['anonymousFeature13\_month']=df\_train['anonymousFeature13\_year'].apply(lambda x: int(x[1]))

df\_train['anonymousFeature13\_year']=df\_train['anonymousFeature13\_year'].apply(lambda x: int(x[0]))

df\_train['anonymousFeature13']=df\_train['anonymousFeature13'].astype('float')

df\_train.isnull().sum()

# df\_train.dtypes

def deal(data=None):

li=[]

for i in data.index:

count=len(re.findall('\d+-\d+-\d+',str(data.loc[i,'updatePriceTimeJson'])))

li.append(count)

return li

df\_train['decreasing\_count']=deal(df\_train)

# df\_train['decreasing\_count'].unique()

# df\_train['decreasing\_count'].value\_counts()

df\_train.isnull().sum()

"""

对剩余的缺失值进行填充

"""

# 匿名变量1 , 8 ,9 ,10先用众数填充； 或者删除有缺失值的行

def fill\_v(data):

n=['anonymousFeature1','anonymousFeature8','anonymousFeature9','anonymousFeature10']

for i in n:

x=int(data[i].mode())

data[i].fillna(x,inplace=True)

return data

df\_train=fill\_v(df\_train)

df\_train['pushDate\_year'] = df\_train['pushDate'].dt.year

df\_train['pushDate\_month'] = df\_train['pushDate'].dt.month

df\_train['pushDate\_day'] = df\_train['pushDate'].dt.day

return df\_train

df\_train=deal\_ano\_feature(df\_train)

df\_test=deal\_ano\_feature(df\_test)

# 将cycle调整到最后一列

cycle=df\_train['cycle']

df\_train=df\_train.drop('cycle',axis=1)

df\_train['cycle']=cycle

"""

以上思路是清晰的

"""

# 分类

df\_train\_class=df\_train.copy() # 让文件指向不同的值

df\_test\_class=df\_test.copy()

"""

开始分开处理

"""

# df\_train\_class is df\_train

df\_train\_class['label']=np.where(df\_train\_class['cycle'].isnull(),-1,1)

# df\_train\_class.dtypes

df\_train\_class=df\_train\_class.select\_dtypes(include=['float64', 'int64', 'int32'])

df\_train\_class=df\_train\_class.drop(['cycle','carid'],axis=1)

df\_test\_class=df\_test\_class.select\_dtypes(include=['float64', 'int64', 'int32'])

df\_test\_class=df\_test\_class.drop('carid',axis=1)

# df\_train\_class.dtypes

# df\_test\_class.dtypes

# df\_train\_class.isnull().sum()

# df\_test\_class.isnull().sum()

"""

X,y的数据已经满足机器学习的要求了

"""

pearson=df\_train\_class.corr()

index = pearson['label'][:-1].abs() > 0.03

X=df\_train\_class.iloc[:,:-1]

X\_submit=X.loc[:,index]

feature=X\_submit.columns

df\_test\_class=df\_test\_class[feature]

# df\_test\_class.columns

"""

分类模型,判断是否能卖出

"""

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X\_submit,df\_train\_class['label'],test\_size=0.2)

model=RandomForestClassifier(n\_estimators=300,random\_state=33,n\_jobs=-1)

model.fit(X\_train,y\_train)

model\_path='./models'

joblib.dump(model,f"{model\_path}/model.joblib")

y\_pred=model.predict(X\_test)

# sum(y\_pred==y\_test)/len(y\_pred)

res=classification\_report(y\_test,y\_pred)

print(res)

"""

以上思路清楚

"""

y\_=model.predict(df\_test\_class)

df\_test['label']=y\_

# data=pd.DataFrame(y\_)

# data[0].unique()

# data[0].value\_counts()

df\_test['label'].value\_counts()

"""

1 1918

-1 82

Name: label, dtype: int64

"""

"""

对分类模型判断可以卖出的数据，进行回归预测交易周期

"""

df\_train\_reg\_ori=df\_train

df\_test\_reg\_ori=df\_test

# 预测卖不出去的结果保存

res\_test\_1=df\_test[df\_test['label']==-1].copy()

res\_test\_1.rename(columns={'label':'cycle'},inplace=True)

"""

可以卖出的回归预测即可

"""

# 删除cycle为空的，

df\_train\_reg\_ori=df\_train\_reg\_ori.dropna()

# df\_train\_reg\_ori.isnull().sum()

# df\_train\_reg\_ori.dtypes

df\_train\_reg=df\_train\_reg\_ori.select\_dtypes(include=['float64','int64'])

pearson\_reg=df\_train\_reg.corr()

index\_reg=pearson\_reg['cycle'][:-1].abs() > 0.04

X\_=df\_train\_reg.iloc[:,:-1]

# X\_.columns

X\_submit\_reg=X\_.loc[:,index\_reg]

X\_submit\_reg.columns

X\_train\_reg,X\_test\_reg,y\_train\_reg,y\_test\_reg=train\_test\_split(X\_submit\_reg,df\_train\_reg['cycle'])

model\_reg=RandomForestRegressor(n\_estimators=500,random\_state=33,n\_jobs=-1)

model\_reg.fit(X\_train\_reg,y\_train\_reg)

joblib.dump(model\_reg,f"{model\_path}/model\_reg.joblib")

y\_reg=model\_reg.predict(X\_test\_reg)

res=mean\_absolute\_error(y\_test\_reg,y\_reg)

print(res)

data\_0=df\_test[df\_test['label']==1].copy() # 取出经分类模型判断cycle不为空的test数据

data=data\_0.select\_dtypes(include=['float64','int64'])

# data.columns

data=data.loc[:,index\_reg]

y\_reg\_=model\_reg.predict(data)

data\_0['cycle']=y\_reg\_

# data\_0['label'].unique()

data\_0=data\_0.drop('label',axis=1)

finall\_res=pd.concat([data\_0,res\_test\_1]) # 把分类的结果和回归的结果文件拼接到一起

finall\_res=finall\_res[['carid','cycle']]

finall\_res.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件6：门店交易模型结果.txt',sep='\t',index=None,header=None)

# # 取出经过分类模型预测的测试集中可以卖出的数据

# df\_test\_reg=df\_test\_reg\_ori[df\_test['label']==1]

# len(df\_test\_reg)

# df\_test\_reg=df\_test\_reg\_ori.drop('label',axis=1)

#

# df\_test\_reg\_1=df\_test\_reg.drop('carid', axis=1) # 去除id

# df\_test\_reg=df\_test\_reg\_1.copy()

# # df\_train\_reg.columns

# # df\_test\_reg.columns

# df\_train\_reg=df\_train\_reg.select\_dtypes(include=['float64', 'int64', 'int32'])

# df\_test\_reg=df\_test\_reg.select\_dtypes(include=['float64', 'int64', 'int32'])

#

# len(df\_train\_reg)

#

# data\_feature=SelectKBest(chi2,k=10).fit\_transform(df\_train\_reg.iloc[:,:-1],df\_train\_reg.iloc[:,-1])

# data\_feature\_test=SelectKBest(chi2,k=10).fit\_transform(df\_test\_reg.iloc[:,:-1],df\_test\_reg.iloc[:,-1])

#

# X\_train\_reg,X\_test\_reg,y\_train\_reg,y\_test\_reg=train\_test\_split(data\_feature,df\_train\_reg.iloc[:,-1])

#

# model\_reg=RandomForestRegressor(n\_estimators=500,random\_state=33,n\_jobs=-1)

# model\_reg.fit(X\_train\_reg,y\_train\_reg)

# y\_reg=model\_reg.predict(data\_feature\_test)

#

# len(y\_reg)

# len(y\_test\_reg)

# mean\_absolute\_error(y\_test\_reg,y\_reg)

#

# df\_test\_reg\_ori['cycle']=y\_reg

#

# res\_data=pd.concat([res\_test\_1,df\_test\_reg\_ori])

# print(res\_data)

# res\_data=res\_data[['carid','cycle']]

#

# res\_data.to\_csv('./data/附件6：门店交易模型结果.txt',sep='\t',index=None,header=None)

### 3.3 可视化代码(jupyter notebook 编写)

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import re

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.feature\_selection import SelectKBest

import xgboost as xgb

import lightgbm as lgb

import warnings

warnings.filterwarnings('ignore')

"""

评估数据集的构造

"""

df\_5 = pd.read\_csv('./data/附件5：门店交易验证数据.txt', sep='\t', parse\_dates=[1], header=None, encoding='utf-8') # 附件五

df\_2 = pd.read\_csv('../data/附件2：估价验证数据.txt', sep='\t', parse\_dates=[1, 11, 12], header=None, encoding='utf-8') # 附件二的信息

columns = ['carid', 'tradeTime', 'brand', 'serial', 'model', 'mileage', 'color', 'cityId', 'carCode',

'transferCount', 'seatings', 'registerDate', 'licenseDate', 'country', 'maketype', 'modelyear',

'displacement', 'gearbox', 'oiltype', 'newprice']

for i in range(1,16):

columns.append('anonymousFeature'+str(i))

# 重命名数据列名

df\_2.columns = columns

df\_5.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson'} ,inplace=True)

# df\_5.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\附件五.csv', encoding='utf-8')

df\_test = pd.merge(df\_5, df\_2, on='carid', how='left')

# df\_test.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一test.csv')

"""

训练集的构造

"""

df\_1 = pd.read\_csv('../data/附件1：估价训练数据.txt', sep='\t', parse\_dates=[1, 11, 12], encoding='utf-8', header=None) # 附件一

df\_4 = pd.read\_csv('../data/附件4：门店交易训练数据.txt', sep='\t', parse\_dates=[1, 4, 5], encoding='utf-8', header=None) # 附件四

columns.append('price')

df\_1.columns=columns

df\_4.rename(columns={0:'carid',1:'pushDate',2:'pushPrice',3:'updatePriceTimeJson',4:'pullDate',5:'withdrawDate'},inplace=True)

df\_train = pd.merge(df\_4, df\_1, how='left', on='carid')

# print(df\_train.columns)

# print(df\_test.columns)

# 调整trian的数据特征，保持与test一致

df\_train['cycle'] = df\_train['withdrawDate']-df\_train['pushDate']

df\_train=df\_train.drop(['price', 'pullDate', 'withdrawDate'], axis=1)

df\_train['cycle']=df\_train['cycle'].astype('timedelta64[D]')

# df\_train.to\_csv(r'D:\比赛\2021年MathorCup大数据竞赛赛道A\复赛\任务一train.csv')

## 附录4：结合库存管理的门店经营模型构建分析代码

### 4.1 K近邻模型预测数据分析代码

1. 数据处理

# 数据分析处理

import numpy as np

import pandas as pd

df1 = pd.read\_table('./data/附件4：门店交易训练数据.txt', sep='\t', header=None, encoding='gbk')

columns1 = ['车辆id', '上架时间', '上架价格', '{价格调整时间：调整后价格}', '下架时间(成交车辆下架时间和成交时间相同)', '成交时间']

df1.columns = columns1

df1['上架时间'] = pd.to\_datetime(df1['上架时间'])

df1['下架时间(成交车辆下架时间和成交时间相同)'] = pd.to\_datetime(df1['下架时间(成交车辆下架时间和成交时间相同)'])

df1['成交周期(天)'] = df1['下架时间(成交车辆下架时间和成交时间相同)'] - df1['上架时间']

df1['成交周期(天)'] = df1['成交周期(天)'].apply(lambda x: x.days)

df1.to\_csv('./data/附件4：门店交易训练数据(含标签和销售周期).csv', index = False)

df = pd.read\_csv('./data/附件4：门店交易训练数据(含标签和销售周期).csv')

# 打标签

bina = [0, 16, 100, 278]

label = ['热销', '正常', '滞销']

df['label'], b = pd.cut(x = df['成交周期(天)'], bins = bina, right = False, labels = label, retbins = True)

# 成交时间为空，说明为销售出去，滞销

for i in range(len(df)):

if pd.isnull(df['成交时间'][i]) == True:

df['label'][i] = '滞销'

df['价格调整次数'] = df['{价格调整时间：调整后价格}'].apply(lambda x:len(eval(x)))

tiaozhengmax = max(list(df['价格调整次数']))

df['final-time'] = 0

df['final-jiage'] = 0

df['是否调价'] = 0

for i in range(len(df['价格调整次数'])):

cou = df['价格调整次数'][i]

d1 = list(eval(df['{价格调整时间：调整后价格}'][i]))

d2 = list(eval(df['{价格调整时间：调整后价格}'][i]).values())

if cou == 0: # 将未进行价格调整的综合到调整的里面进行数据分析统计

df['final-time'][i] = df['上架时间'][i]

df['final-jiage'][i] = df['上架价格'][i]

df['是否调价'][i] = 0

else:

df['final-time'][i] = d1[-1]

df['final-jiage'][i] = d2[-1]

df['是否调价'][i] = 1

df['time-difference'] = 0

df['price-difference'] = 0

df['price-range'] = 0 # 百分比

df['上架时间'] = pd.to\_datetime(df['上架时间'])

df['下架时间(成交车辆下架时间和成交时间相同)'] = pd.to\_datetime(df['下架时间(成交车辆下架时间和成交时间相同)'])

df['final-time'] = pd.to\_datetime(df['final-time'])

df['final-jiage'] = df['final-jiage'].astype('float')

df['上架价格'] = df['上架价格'].astype('float')

print(type(df['final-jiage'][0]))

print(type(df['上架价格'][0]))

df['time-difference'] = df['下架时间(成交车辆下架时间和成交时间相同)'] - df['final-time']

df['time-difference'] = df['time-difference'].apply(lambda x:x.days)

df['price-difference'] = df['final-jiage'] - df['上架价格'] # 正值是涨价，负值为降价

df['price-range'] = (df['price-difference']/df['上架价格']) \* 100 # 百分比

df['timecha'] = df['final-time'] - df['上架时间']

df['timecha'] = df['timecha'].apply(lambda x:x.days)

df.to\_csv('./data/最终价格调整时间价格幅度(含标签）.csv', index = False)

df\_4\_2 = pd.read\_csv('./data/最终价格调整时间价格幅度(含标签）.csv')

df\_1 = pd.read\_table('./data/附件1：估价训练数据.txt', sep='\t', header=None, encoding='gbk')

column = ['车辆id', '展销时间', '品牌id ', '车系id ', '车型id ', '里程', '车辆颜色', '车辆所在城市id', '国标码', '过户次数', '载客人数', '注册日期', '上牌日期', '国别', '厂商类型', '年款', '排量', '变速箱', '燃油类型', '新车价', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '匿名特征', '二手车交易价格（预测目标）']

df\_1.columns = column

df\_1.to\_csv('./data/附件1（标签化）.csv')

df\_14\_2 = pd.concat([df\_1, df\_4\_2], axis = 1, join = 'inner')

df\_14\_2.to\_csv('./data/附件1-4（只保留相同索引-大量标签）.csv', index = False)

2. 模型架构

import numpy as np

import pandas as pd

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

df = pd.read\_csv('./data/附件1-4（只保留相同索引-大量标签）.csv')

bina = [0, 16, 100, 278]

label = [0, 1, 2]

df['z\_label'], b = pd.cut(x = df['成交周期(天)'], bins = bina, right = False, labels = label, retbins = True)

# 成交时间为空，说明为销售出去，滞销

for i in range(len(df)):

if pd.isnull(df['成交时间'][i]) == True:

df['z\_label'][i] = 2

df['z\_label'] = df['z\_label'].astype('int')

pr\_df = pd.DataFrame()

pr\_df['车辆id'] = df['车辆id']

pr\_df['上架时间'] = df['上架时间']

pr\_df['z\_label'] = df['z\_label']

pr\_df['上架时间'] = pd.to\_datetime(pr\_df['上架时间'])

pr\_df['pushDate\_year'] = pr\_df['上架时间'].dt.year

pr\_df['pushDate\_month'] = pr\_df['上架时间'].dt.month

pr\_df['pushDate\_day'] = pr\_df['上架时间'].dt.day

pr\_df['上架价格'] = df['上架价格']

pr\_df[ '品牌id '] = df[ '品牌id ']

pr\_df['车系id '] = df[ '车系id ']

pr\_df[ '车型id '] = df['车型id ']

pr\_df = pr\_df.fillna(value = -1)

df = df.fillna(value = -1) # 用-1填补空值

df['price-range'] = df['price-range'].astype('int')

df['上架时间'] = pd.to\_datetime(df['上架时间'])

df['下架时间(成交车辆下架时间和成交时间相同)'] = pd.to\_datetime(df['下架时间(成交车辆下架时间和成交时间相同)'])

df['pushDate\_year'] = df['上架时间'].dt.year

df['pushDate\_month'] = df['上架时间'].dt.month

df['pushDate\_day'] = df['上架时间'].dt.day

x\_data = df.loc[:, [ '品牌id ', 'z\_label', 'pushDate\_year', 'pushDate\_month', 'pushDate\_day', '上架价格', '车辆颜色', '车辆所在城市id', '载客人数', '国标码', '过户次数', '年款']]

y\_data = df.loc[:, '是否调价']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.2, random\_state = 22)

transfer = StandardScaler()

x\_train = transfer.fit\_transform(x\_train)

x\_test = transfer.transform(x\_test)

estimator = KNeighborsClassifier()

estimator.fit(x\_train, y\_train)

y\_pre = estimator.predict(x\_test)

pr\_df['pr是否调价'] = estimator.predict(x\_data)

x\_data = df.loc[:, [ 'z\_label', 'pushDate\_year', 'pushDate\_month', 'pushDate\_day', '上架价格', '车辆颜色', '车辆所在城市id', '载客人数', '国标码', '过户次数', '年款']]

y\_data = df.loc[:, 'timecha']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.2, random\_state = 22)

transfer = StandardScaler()

x\_train = transfer.fit\_transform(x\_train)

x\_test = transfer.transform(x\_test)

estimator = KNeighborsClassifier()

estimator.fit(x\_train, y\_train)

y\_pre = estimator.predict(x\_test)

pr\_df['timecha'] = estimator.predict(x\_data)

x\_data = df.loc[:, [ '品牌id ', '车系id ', '车型id ', 'z\_label', 'pushDate\_year', 'pushDate\_month', 'pushDate\_day', '上架价格', '车辆颜色', '车辆所在城市id', '载客人数', '国标码', '过户次数', '年款']]

y\_data = df.loc[:, 'price-range']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.2, random\_state = 22)

transfer = StandardScaler()

x\_train = transfer.fit\_transform(x\_train)

x\_test = transfer.transform(x\_test)

estimator = KNeighborsClassifier()

estimator.fit(x\_train, y\_train)

y\_pre = estimator.predict(x\_test)

pr\_df['price-range'] = estimator.predict(x\_data)

# 根据预测得到的数据再次进行预测销售周期

pr\_df['周期'] = df['成交周期(天)']

pr\_df['上架时间'] = pd.to\_datetime(df['上架时间'])

pr\_df['调价日期'] = 0

pr\_df['timecha'] = pr\_df['timecha'].astype('float')

import datetime

for i in range(len(pr\_df)):

pr\_df['调价日期'][i] = pr\_df['上架时间'][i] + datetime.timedelta(days= pr\_df['timecha'][i])

pr\_df['调价日期'] = pd.to\_datetime(pr\_df['调价日期'])

pr\_df['update\_year'] = pr\_df['调价日期'].dt.year

pr\_df['update\_month'] = pr\_df['调价日期'].dt.month

pr\_df['updatee\_day'] = pr\_df['调价日期'].dt.day

x\_data = df.loc[:, [ '品牌id ', 'z\_label', 'pushDate\_year', 'pushDate\_month', 'pushDate\_day', '上架价格', '车辆颜色', '车辆所在城市id', '载客人数', '国标码', '过户次数', '年款']]

y\_data = pr\_df.loc[:, '周期']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.2, random\_state = 22)

transfer = StandardScaler()

x\_train = transfer.fit\_transform(x\_train)

x\_test = transfer.transform(x\_test)

estimator = KNeighborsClassifier()

estimator.fit(x\_train, y\_train)

y\_pre = estimator.predict(x\_test)

pr\_df['pr周期'] = estimator.predict(x\_data)

year = set(list(df['pushDate\_year']))

df\_20 = df[df['pushDate\_year'] == 2020]

df\_21 = df[df['pushDate\_year'] == 2021]

pr\_df['pr周期'] = pr\_df['pr周期'].astype('float64')

pr\_df['出售日期'] = 0

import datetime

for i in range(len(pr\_df)):

pr\_df['出售日期'][i] = pr\_df['上架时间'][i] + datetime.timedelta(days= pr\_df['pr周期'][i])

pr\_df['出售日期'] = pd.to\_datetime(pr\_df['出售日期'])

pr\_df['chushoudate\_year'] = pr\_df['出售日期'].dt.year

pr\_df['chushoudate\_month'] = pr\_df['出售日期'].dt.month

pr\_df['chushoudate\_day'] = pr\_df['出售日期'].dt.day

pr\_df['上架价格'] = df['上架价格']

pr\_df['新车价'] = df['新车价']

pr\_df['售价'] = (100 + df['price-range'] ) \* df['上架价格'] /100

pr\_df['利润'] = pr\_df['售价'] - pr\_df['新车价']

df\_20 = pr\_df[pr\_df['chushoudate\_year'] == 2020]

df\_21 = pr\_df[pr\_df['chushoudate\_year'] == 2021]

lirun\_20 = []

lirun\_21 = []

for i in range(1, 13):

dd = df\_20[df\_20['chushoudate\_month'] == i]

lli = dd.loc[:, '利润'].sum(axis = 0)

lirun\_20.append(lli)

print(lirun\_20)

for i in range(1, 13):

dd = df\_21[df\_21['chushoudate\_month'] == i]

lli = dd.loc[:, '利润'].sum(axis = 0)

lirun\_21.append(lli)

print(lirun\_21)

### 4.2 遗传算法建模分析代码

import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import cm  
from mpl\_toolkits.mplot3d import Axes3D  
import math  
DNA\_SIZE = 24#DNA数量  
POP\_SIZE = 80  
CROSSOVER\_RATE = 0.6#交叉概率  
MUTATION\_RATE = 0.01#变异概率  
N\_GENERATIONS = 100  
X\_BOUND = [0, 30]  
Y\_BOUND = [-1000, 1000]  
  
  
def F(x, y):  
   #return （模型检验函数）  
  
  
def plot\_3d(ax):  
    X = np.linspace(\*X\_BOUND, 30)  
    Y = np.linspace(\*Y\_BOUND, 100)  
    X, Y = np.meshgrid(X, Y)  
    Z = F(X, Y)  
    ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm)  
    ax.set\_xlabel(&apos;x&apos;)  
    ax.set\_ylabel(&apos;y&apos;)  
    ax.set\_zlabel(&apos;z&apos;)  
    plt.pause(3)  
    plt.show()  
  
  
def get\_fitness(pop):  
    x, y = translateDNA(pop)  
    pred = F(x, y)  
    return pred  
  
    # return pred - np.min(pred)+1e-3  # 求最大值时的适应度  
    # return np.max(pred) - pred + 1e-3  # 求最小值时的适应度，通过这一步fitness的范围为[0, np.max(pred)-np.min(pred)]  
  
  
def translateDNA(pop):  # pop表示种群矩阵，一行表示一个二进制编码表示的DNA，矩阵的行数为种群数目  
    x\_pop = pop[:, 0:DNA\_SIZE]  # 前DNA\_SIZE位表示X  
    y\_pop = pop[:, DNA\_SIZE:]  # 后DNA\_SIZE位表示Y  
  
    x = x\_pop.dot(2 \*\* np.arange(DNA\_SIZE)[::-1]) / float(2 \*\* DNA\_SIZE - 1) \* (X\_BOUND[1] - X\_BOUND[0]) + X\_BOUND[0]  
    y = y\_pop.dot(2 \*\* np.arange(DNA\_SIZE)[::-1]) / float(2 \*\* DNA\_SIZE - 1) \* (Y\_BOUND[1] - Y\_BOUND[0]) + Y\_BOUND[0]  
    return x, y  
  
  
def crossover\_and\_mutation(pop, CROSSOVER\_RATE=0.8):  
    new\_pop = []  
    for father in pop:  # 遍历种群中的每一个个体，将该个体作为父亲  
        child = father  # 孩子先得到父亲的全部基因（这里我把一串二进制串的那些0，1称为基因）  
        if np.random.rand() < CROSSOVER\_RATE:  # 产生子代时不是必然发生交叉，而是以一定的概率发生交叉  
            mother = pop[np.random.randint(POP\_SIZE)]  # 再种群中选择另一个个体，并将该个体作为母亲  
            cross\_points = np.random.randint(low=0, high=DNA\_SIZE \* 2)  # 随机产生交叉的点  
            child[cross\_points:] = mother[cross\_points:]  # 孩子得到位于交叉点后的母亲的基因  
        mutation(child)  # 每个后代有一定的机率发生变异  
        new\_pop.append(child)  
  
    return new\_pop  
  
  
def mutation(child, MUTATION\_RATE=0.003):  
    if np.random.rand() < MUTATION\_RATE:  # 以MUTATION\_RATE的概率进行变异  
        mutate\_point = np.random.randint(0, DNA\_SIZE)  # 随机产生一个实数，代表要变异基因的位置  
        child[mutate\_point] = child[mutate\_point] ^ 1  # 将变异点的二进制为反转  
  
  
def select(pop, fitness):  # nature selection wrt pop&apos;s fitness  
    idx = np.random.choice(np.arange(POP\_SIZE), size=POP\_SIZE, replace=True,  
                           p=(fitness) / (fitness.sum()))  
    return pop[idx]  
  
  
def print\_info(pop):  
    fitness = get\_fitness(pop)  
    max\_fitness\_index = np.argmax(fitness)  
    print("max\_fitness:", fitness[max\_fitness\_index])  
    x, y = translateDNA(pop)  
    print("最优的基因型：", pop[max\_fitness\_index])  
    print("(x, y):", (x[max\_fitness\_index], y[max\_fitness\_index]))  
    print(F(x[max\_fitness\_index], y[max\_fitness\_index]))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    fig = plt.figure()  
    ax = Axes3D(fig)  
    plt.ion()  # 将画图模式改为交互模式，程序遇到plt.show不会暂停，而是继续执行  
    plot\_3d(ax)  
  
    pop = np.random.randint(2, size=(POP\_SIZE, DNA\_SIZE \* 2))  # matrix (POP\_SIZE, DNA\_SIZE)  
    for \_ in range(N\_GENERATIONS):  # 迭代N代  
        x, y = translateDNA(pop)  
        if &apos;sca&apos; in locals():  
            sca.remove()  
        sca = ax.scatter(x, y, F(x, y), c=&apos;black&apos;, marker=&apos;o&apos;)  
        plt.show()  
        plt.pause(0.1)  
        pop = np.array(crossover\_and\_mutation(pop, CROSSOVER\_RATE))  
        fitness = get\_fitness(pop)  
        pop = select(pop, fitness)  # 选择生成新的种群  
  
    print\_info(pop)  
    plt.ioff()  
    plot\_3d(ax)

96