

# 基于随机森林算法对二手车定价和成交周期的预测

## 摘要

在互联网时代下，O2O 门店模式应运而生。二手车零售作为 O2O 门店模式的典型应用场景，在二手车市场中占有重要地位。随之产生的，就是二手车价格制定的问题。不合理的价格会增加企业销售成本，降低盈利。因此，如何基于已有信息对二手车的零售交易价格进行合理预测便愈发重要。本文根据题目提供的估价数据和门店交易数据，针对二手车零售交易价格和车辆成交周期，利用线性回归、逻辑回归、回归树、随机森林、神经网络、LightGBM 和集成学习等方法，建立预测模型并深入研究。

针对问题一，在数据预处理阶段，剔除异常数值后，使用随机森林对缺失值进行回归和分类预测，并针对非数值型变量尝试多种编码方式，最终选择 **CountEncoder 编码**。在对数值进行 Min-Max 标准化的基础上，使用线性回归、回归树、随机森林、神经网络、LightGBM、利用 stacking 搭建学习器分别进行回归预测，score 值约为 0.17。为了进一步提高模型性能，选择由 800 棵树组成的 **随机森林**，得到 **Mape 为 0.1050**，**Accuracy<sub>s</sub> 为 0**，分数为 **0.1790**。

针对问题二，首先利用 **withdrawDate** 是否存在对车辆是否成交进行 **分类预测**，随后利用车辆成交周期进行 **回归预测**。通过将附件 1 和附件 4 连接，得到更为完整的数据集。随后采用与问题一相同的数据预处理方式，并针对 json 格式的变量构造新的变量。最终利用随机森林进行车辆是否成交预测，**F1 值达到 0.8827**；进行车辆成交周期的回归预测，**mse 约为 9.6299**。得到影响车辆成交的因素为 **价格变动幅度、年款、过户次数和二手车交易价格**；影响车辆成交周期的因素为 **价格调整次数、价格弹性和车辆所在城市**，并进行详细分析。

针对问题三，我们希望进一步探究和价格相关的变量。因此对附件 4 中数据进行聚类分析，可以将全部车辆分为两类：第一类样本的价格较低、价格调整频率低、价格调整幅度大；第二类样本的价格较高、价格调整频率大、价格调整幅度小。

**关键词** 二手车交易 CountEncoder 随机森林 集成学习 回归预测

目录

- 一、问题重述.....1
  - 1.1 问题背景.....1
  - 1.2 问题提出.....1
- 二、问题分析.....2
  - 2.1 问题一分析.....2
  - 2.2 问题二分析.....2
  - 2.3 问题三分析.....2
- 三、基本假设.....2
- 四、符号说明.....3
- 五、问题一的模型建立与求解.....3
  - 5.1 建模思路.....3
  - 5.2 数据预处理.....4
    - 5.2.1 数据清洗.....4
    - 5.2.2 异常值处理.....4
    - 5.2.3 划分训练集和测试集.....5
    - 5.2.4 缺失值处理.....5
  - 5.3 数据分析.....6
    - 5.3.1 离散特征编码处理.....6
    - 5.3.2 连续特征标准化处理.....7
    - 5.3.3 降维.....7
  - 5.4 构建模型.....10
    - 5.4.1 线性回归.....10
    - 5.4.3 随机森林.....13
    - 5.4.4 神经网络.....14
    - 5.4.5 LightGBM.....16
    - 5.4.6 集成学习.....17
  - 5.5 模型选择与问题求解.....18
- 六、问题二的模型建立与求解.....19
  - 6.1 建模思路.....19
  - 6.2 数据预处理.....19
    - 6.2.1 数据拼接与清洗.....19
    - 6.2.2 划分测试集和训练集.....20
    - 6.2.3 缺失值填充.....20
  - 6.3 数据分析.....20
    - 6.3.1 离散特征编码处理.....20
    - 6.3.2 连续特征标准化处理.....21
    - 6.3.3 降维.....21
  - 6.4 构建模型.....21
    - 6.4.1 线性回归.....21
    - 6.4.2 回归树.....22
    - 6.4.3 随机森林.....22
  - 6.5 模型选择与问题求解.....23

6.6 解决方法与预期效果.....	26
七、问题三的模型建立与求解.....	27
7.1 问题提出.....	27
7.2 建立模型.....	27
7.3 模型结果.....	28
7.4 结论归纳.....	30

# 一、问题重述

## 1.1 问题背景

随着国家的迅速发展、消费水平的不断提升，居民的消费结构不断升级、消费理念不断更新，人们更加注重实用性，二手车越来越频多地成为消费者购置汽车的选择。随着时间的推移，近年来二手车成交量不断上升，反映出二手车市场的巨大潜力<sup>1</sup>。二手车在电商平台的流通环节包括收车、拍卖、零售、置换等环节，而每一辆二手车都有其特殊化之处，使得二手车的交易比新车或其他电商产品更加复杂。车辆的定价不但会受到汽车本身的基础配置，如品牌、车系、动力、变速箱、出场时间、颜色等因素的影响，还会受到人为因素，如其老化磨损程度、维修情况、行驶里程数等个性化因素，因此很难通过一套统一的标准执行，定价的过程往往存在较强的主观性。一些二手车交易平台和第三方评估平台也根据自身的理论和经验制定出了一系列估价方法。

在互联网时代下，O2O 门店模式应运而生。O2O 营销模式又称离线商务模式，是指线上营销线上购买带动线下经营和线下消费。O2O 通过打折、提供信息、服务预订等方式，将线下商店的消息推送给互联网用户，从而将其转换为自己的线下客户。二手车零售行业中，O2O 模式的门店在收购二手车后，经由门店定价师为其定价后，进行展销和售卖。然而，定价的高低对二手车能否成功出售起到了关键性作用。若定价过低，不利于门店创造高盈利；若定价过高，则会导致二手车滞销，此时门店往往会以打折促销的方式推动产品的出售，甚至以更低的价格打包批发，直至商品最终卖出。

## 1.2 问题提出

在本题中，我们需要根据给定的二手车交易样本数据，通过构建数学模型，解决下列问题：

1. 基于附件 1 的数据，利用车辆基础情况、交易时间、价格等信息，建立合适的估价模型，自行划分训练集和测试集并对模型进行训练和测试后，运用模型对附件 2 中数据所对应的价格进行预测。
2. 基于附件 4 的数据，对车辆的成交周期进行分析，通过探究影响车辆成交周期的关键因素，从而为门店加快在库车辆的销售速度提出有效的解决手段，并说明这些手段所对应的适用条件和预期效果。
3. 基于给定的样本数据集，挖掘其他值得探究的问题，并针对所提出的问题，给出解决思路。

## 二、问题分析

### 2.1 问题一分析

问题一是较为典型的回归预测问题，可以采用传统的机器学习方法求解。由于附件 1 中的数据带有标签，因此可以采用机器学习中监督学习的方法。在对数据进行预处理后，通过建立线性回归、回归树、随机森林、神经网络、LightGBM 模型，预测二手车的零售交易价格。在此基础上，通过 stacking 建立集成学习模型，提高计算精度，得到预测结果。

### 2.2 问题二分析

问题二要求我们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。针对该问题，我们首先以是否成交作为因变量，建立分类预测模型，探究影响车辆成交的因素；随后以车辆成交周期为因变量，建立回归预测模型，探究影响成交周期的因素。最终综合考虑两个预测模型的影响因素，分析加快销售速度的手段和效果。

### 2.3 问题三分析

问题三要求我们自行提出问题并给出解决思路。我们所提出的问题是针对不同价格档次的二手车，销售商是否会采取不同的销售策略。首先通过聚类的方式将样本集划分为了两类，并分析了两类的特征，确认一类是中低端车，一类是高端车。其次，比较了两类样本各特征之间在均值和分布的差异。最后分析得出针对两类样本所采取的不同销售策略。

## 三、基本假设

1. 假设不存在给定特征外影响因素；
2. 假设给定数据真实可靠；

## 四、符号说明

表 X 符号说明

符号	符号说明
$x_{ij}$	第 i 列第 j 个数据
$\max(x_i)$	第 i 列数据的最大值
$\min(x_i)$	第 i 列数据的最小值
$\tilde{x}_j$	第 j 列数据均值
$s_j^2$	第 j 列数据方差
GI	基尼指数
VIM	变量重要性
$\Sigma(\sigma_{ij})$	协方差矩阵
f	公共因子向量
e	特殊因子向量
k	价格调整次数
T0	上架时间
Tk+1	下架时间
T <sub>avg</sub>	平均价格调整时间
elacity	价格弹性

## 五、问题一的模型建立与求解

### 5.1 建模思路

在给定的数据集附件 1 中数据是带有标签的，属于监督学习。通过观察数据集的特征，我们发现数据中部分特征的可读性较差、且存在异常值和缺失值。我们首先对数据进行预处理，之后对于分类特征进行编码处理。为了得到最优的预测结果，我们采用了包括线性回归、回归树、随机森林、神经网络、LightGBM、集成学习（stacking）等多种方法，通过对比不同模型的预测效果选出最佳的处理方式和回归模型，对附件 2 中的数据预测其价格。

## 5.2 数据预处理

数据预处理是建立模型前至关重要的一步，对最终结果起着决定性作用。通过数据预处理，识别出不完整、不正确等“脏数据”并对其进行清洗或清除，从而提高数据的质量、提升模型的效果。

### 5.2.1 数据清洗

首先，导入数据并查看其情况。由于“附件 1：估价训练数据”的数据集中包含 15 个匿名特征，为了方便后续处理，我们将各列命名为 feature1——feature15。

其次，修改数据类型。对于 feature11 而言，其存在形式较多，如“1”、“1+2”、“4+2”等，为了使模型更好地使用此列数据，我们将该列的数据视为离散的 object 类数据并为其进行了编码；对于 feature12 而言，其数据是由三个数字相乘，根据数据背景我们推断该属性所描述的是二手车的尺寸，因此将其分别拆分为“feature12-length”、“feature12-width”、“feature12-height”三个变量并将其乘积作为“feature12-volume”特征。

第三，对于时间的处理。为了计算时间跨度，我们以月为单位进行计算，分别计算了 tradeTime - registerDate、tradeTime - licenseDate、tradeTime - modelyear 和 tradeTime - feature13。

### 5.2.2 异常值处理

通过绘制各变量的通过绘制 price 变量的箱线图，可以发现在 price 列中存在一个数值在 100000 以上，这一数值与该列其他数值的量纲间存在巨大差距，根据数据的含义和经验判断该值为异常值，对该条数据进行了剔除。

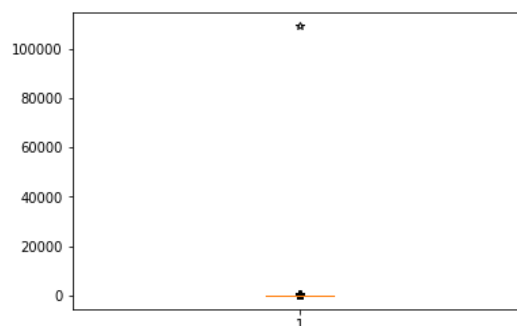


图 1 price 列的箱线图

### 5.2.3 划分训练集和测试集

通过调用 `sklearn.model_selection` 中 `train_test_split` 函数，以附件 1 中的 `price` 作为 `y`，以其他特征信息作为 `X`，设置 `test_size=0.3`，对数据集进行划分。划分后的结果为：

```
X_train.shape: (20999, 33)
X_test.shape: (9000, 33)
y_train.shape: (20999,)
y_test.shape: (9000,)
```

图 2 训练集和测试集规模

### 5.2.4 缺失值处理

首先，查看缺失值的分布情况。在附件一所包含的 30000 条数据中，缺失情况如下：

特征	缺失数据量
carCode	9
country	3757
maketype	3641
modelyear	312
gearbox	1
feature1	1582
feature4	12109
feature7	18044
feature8	3775
feature9	3744
feature10	6241
feature11	461
feature13	1691
feature15	27850

其余特征均无缺失。由于 `feature4`、`feature7` 和 `feature15` 的缺失值占比过大，即使将缺失值填充完整也很可能与真实数据产生较大差异，反而可能导致模型预测的偏差。因此，将 `feature4`、`feature7` 和 `feature15` 这三列缺失值过大的数据清除。

其次，分别采用随机森林回归树 `RandomForestRegressor` 和随机森林分类树 `RandomForestClassifier` 的方式对连续型和离散型的缺失数据进行了填充。随机森林填补通过构造多棵决策树对缺失值进行填补，使填补的数据具有随机性和



不确定性，更能反映出这些未知数据的真实分布。并且，由于在构造决策树过程中，每个分支节点选用随机的部分特征而不是全部特征，所以能很好的应用到高维数据的填补。随机森林填补缺失值的过程为：

对于一个有  $n$  个特征的数据来说，其中特征  $T$  有缺失，则将特征  $T$  当作标签，其他  $n-1$  个特征和原来的标签组成新的特征矩阵。对于特征  $T$  而言，其不存在缺失的部分视为  $y\_train$ ，对应的标签就是  $X\_train$ ；特征  $T$  的缺失部分视作  $y\_predict$ ，对应的标签就是  $X\_test$ 。通过 RandomForestRegressor 和 RandomForestClassifier 对  $X\_train$ 、 $y\_train$  进行拟合和转换后，对  $X\_test$  进行转换从而得到我们的填充值，即  $y\_predict$ 。

为了保证训练集和测试集之间的相互独立，对于数值型变量和分类型变量分别用 RandomForestRegressor 和 RandomForestClassifier 在  $X\_train$  进行拟合后，再对  $X\_train$  和  $X\_test$  进行了转换。

### 5.3 数据分析

#### 5.2.1 离散特征编码处理

在给定的数据集中，有众多离散特征信息。常见的处理离散特征的方式如下图所示：



图 3 常见的处理离散特征的方式

针对本文的数据背景和数据情况，我们小组分别采用了 OneHotEncoder 编码和 CountEncoder 编码对数据集中分类变量进行了编码处理，并通过对比两种编码方式下模型的预测效果从而选出最佳的编码方式。两种编码方式的原理如下：

OneHotEncoder 编码通过调用 OneHotEncoder() 函数，将其进行 one-hot 编码。One-hot 编码又称一位有效编码，若离散变量的种类有  $M$  个，One-hot 编码

就采用 M 位状态寄存器对这 M 种可能取值进行编码,每个可能的取值由独立的寄存器表示,即 M 位中只有一位有效,其本质就是采用二进制的方式表示变量中的每个值,最终得到的是一个 M 维的稀疏矩阵。

CountEncoder 编码统计了每个特征的出现频率并将分类特征替换为其出现次数。当频率信息与目标变量间存在一定关联时,使用 CountEncoder 编码往往能为模型效果带来提升。

在下面的模型中,我们会分别使用这两种编码方式并比较其效果,选择出更优的编码方式。

### 5.3.2 连续特征标准化处理

对于数值型数据,通过调用 MinMaxScaler() 函数进行归一化处理,将有量纲的数据经过变换,化为无量纲的数据。MinMaxScaler() 的原理如下:

$$\frac{x_{ij} - \min(x_i)}{\max(x_i) - \min(x_i)}$$

其中,  $x_{ij}$  为第 i 列第 j 个数据,  $\max(x_i)$  为第 i 列数据的最大值;  $\min(x_i)$  为第 i 列数据的最小值。通过 MinMaxScaler() 缩放后将所有数值型变量均缩放到了 [0, 1] 区间内。

### 5.3.3 降维

由于在给定的数据集中包含了 36 个变量特征,维度较高。通过降维,在保留了原有数据的主要信息的情况下,利用降维的数据进行机器学习模型的训练和预测可以大大提高时间效率。

降维的三种主要方式,包括特征选择、主成分分析和因子分析。我们小组分别采用了三种降维方式对数据进行处理,并分别运用到模型中,选择出效果更优的方式。

#### (1) 特征选择

随机森林模型在拟合数据后,会对数据属性列进行变量重要性的度量,在变量重要性度量数组中,数值越大的属性列对于预测的准确性更加重要。

用随机森林进行特征重要性评估的思想是计算每个特征在随机森林中的每棵树上的贡献并取个平均值,比较特征之间的贡献大小。贡献大小通常使用基尼指数(Gini index)或者袋外数据(out-of-bag, OOB)错误率作为评估指标来衡量。

将变量重要性评分(variable importance measures)用 VIM 表示,将 Gini 指数用 GI 表示,假设 m 个特征  $X_1, X_2, X_3, \dots, X_m$ , 计算出每个特征  $X_j$  的 Gini 指数评分  $VIM_j$  (Gini),即第 j 个特征在 RF 所有决策树中节点分裂不纯度的平均改

变量。Gini 指数的计算公式为：

$$GI_m = \sum_{k=1}^{|K|} \sum_{k' \neq k} p_{mk} p_{mk'} = 1 - \sum_{k=1}^{|K|} p_{mk}^2$$

其中，K 表示有 K 个类别。P<sub>mk</sub> 表示节点 m 中类别 k 所占的比例。特征 X<sub>j</sub> 在节点 m 的重要性，即节点 m 分支前后的 Gini 指数变化量为： $VIM_{ij}^{(Gini)} = \sum_{m \in M} VIM_{jm}^{(Gini)}$ 。

假设 RF 中共有 n 颗树，那么  $VIM_j^{(Gini)} = \sum_{i=1}^n VIM_{ij}^{(Gini)}$ 。最后，把所有求得的重要性评分做一个归一化处理： $VIM_j = \frac{VIM_j}{\sum_{i=1}^m VIM_i}$ 。

通过计算各特征的重要性值并进行排序后，其结果如下：

1)	newprice	0.670682
2)	tradeTime-licenseDate	0.119677
3)	tradeTime-registerDate	0.060832
4)	displacement	0.041107
5)	mileage	0.020193
6)	tradeTime-feature13	0.016606
7)	feature12-height	0.009485
8)	tradeTime-modelyear	0.008972
9)	brand	0.007857
10)	feature12-length	0.005302
11)	feature12-width	0.005091
12)	serial	0.004903
13)	feature12-volume	0.004403
14)	feature5	0.003482
15)	gearbox	0.003446
16)	model	0.003330
17)	country	0.002250
18)	feature2	0.002150
19)	feature9	0.001393
20)	cityId	0.001205
21)	feature8	0.001055
22)	feature6	0.000990
23)	oiltype	0.000929
24)	transferCount	0.000807
25)	color	0.000751
26)	feature11	0.000722
27)	seatings	0.000703
28)	maketype	0.000502
29)	carCode	0.000468
30)	feature10	0.000401
31)	feature14	0.000284
32)	feature3	0.000020
33)	feature1	0.000003

图 4 附件 1 中各变量特征的重要性排序

因此，我们可以以重要性为 0.01 或 0.005 为界限，选取重要性在此界限之上的变量特征放入模型中，在不大量损失数据集信息的前提下大大提高模型的效率。

## (2) 主成分分析

主成分分析（principal component analysis , PCA）是一种旋转数据集的方法，旋转后的特征在统计上不相关，将多个特征转化为少数几个综合特征。PCA 进行的是一个线性变换，将数据变换到新的坐标系中，使得任何数据投影的第一大

方差在第一主成分上，第二大方差在第二主成分上，以此类推。

首先，给定数据集中的  $p$  个特征可以视为  $p$  维向量， $x = [x_i]^T, i \in [1, p]$ ;  $n$  个样本  $X = [X_{ij}]^T, i \in [1, n], j \in [1, p]$ 。其次，对数据进行标准化：

$$Z = [z_{ij}], \text{其中 } i \in [1, n], j \in [1, p], z_{ij} = \frac{x_{ij} - \tilde{x}_j}{s_j}, \tilde{x}_j = \frac{\sum_{i=1}^n x_{ij}}{n},$$

$$s_j^2 = \frac{\sum_{i=1}^n (x_{ij} - \tilde{x}_j)^2}{n - 1}$$

计算相关系数矩阵（协方差矩阵）：

$$R = [r_{ij}]_{p \times p} = \frac{Z^T Z}{n - 1}, \text{其中 } r_{ij} = \frac{\sum_{i=1}^n z_{kj} * z_{ki}}{n - 1}, i, j \in [1, p]$$

获得特征根和特征向量： $|R - \lambda I_p| = 0$ ，及对应的特征向量  $u_1, u_2, u_3, \dots, u_m$ 。

计算特征值  $\lambda_j$  ( $j = 1, 2, \dots, m$ ) 的信息贡献率和累计贡献率： $b_j = \frac{\lambda_j}{\sum_{k=1}^m \lambda_k}$  为主成分  $y_i$  的信息贡献率； $a_p = \frac{\sum_{k=1}^p \lambda_k}{\sum_{k=1}^m \lambda_k}$  为主成分  $y_1, y_2, \dots, y_p$  的累计贡献率。

通过调用 `sklearn.decomposition` 中的 PCA 函数，对数据集进行了拟合和转换后，带入到后续模型中。

### (3) 因子分析

因子分析的目的是用少数的几个因子去描述多个变量之间的关系，以达到降维的效果。其核心思想是将联系紧密的变量归为同一类别，实现不同类别的变量之间有较低相关性。在同一类别内的变量，被认为是受到了某个共同影响而高度相关，这一共同影响称为共同因子。每一类变量代表了一类共同因子。在以相关性为基础之上，从协方差或相关矩阵入手，将大量的变异归结为少数几个共同因子，将剩下的变异称为特殊因子。其数学模型如下：

$$\begin{cases} x_1 = u_1 + a_{11}f_1 + a_{12}f_2 + a_{13}f_3 + \dots + a_{1m}f_m + e_1 \\ x_2 = u_2 + a_{21}f_1 + a_{22}f_2 + a_{23}f_3 + \dots + a_{2m}f_m + e_2 \\ x_3 = u_3 + a_{31}f_1 + a_{32}f_2 + a_{33}f_3 + \dots + a_{3m}f_m + e_3 \\ \dots \dots \\ x_p = u_p + a_{p1}f_1 + a_{p2}f_2 + a_{p3}f_3 + \dots + a_{pm}f_m + e_p \end{cases}$$

并且假设：

$E(f) = 0; E(e) = 0; V(f) = I; V(e) = D = \text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_p^2); \text{Cov}(f, e) = E(fe^T) = 0$  其中  $x = (x_1, x_2, x_3 \dots x_m)^T$  为  $P$  维可观测随机变量， $u = (u_1, u_2, u_3 \dots u_m)^T$  为可观测变量的均值， $\Sigma(\sigma_{ij})$  为协方差矩阵； $f = (f_1, f_2, f_3 \dots f_m)^T$  为公共因子向量， $e = (e_1, e_2, e_3 \dots e_m)^T$  为特殊因子向量； $A = (a_{ij})_{p \times m}$  为因子载荷矩阵。

在因子分析之前，首先确认了待分析的变量是否适合做因子分析，即对原有变量的做相关性分析，只有当原有变量直接有较强相关性时才满足因子分析的条件。本题数据集的巴特利特值和 `kmo` 值结果如下：

巴特利特值: 0.0 小于0.1, 则可以选择这种方法  
kmo值: 0.7140907291081602 大于0.6, 则可以选择这种方法

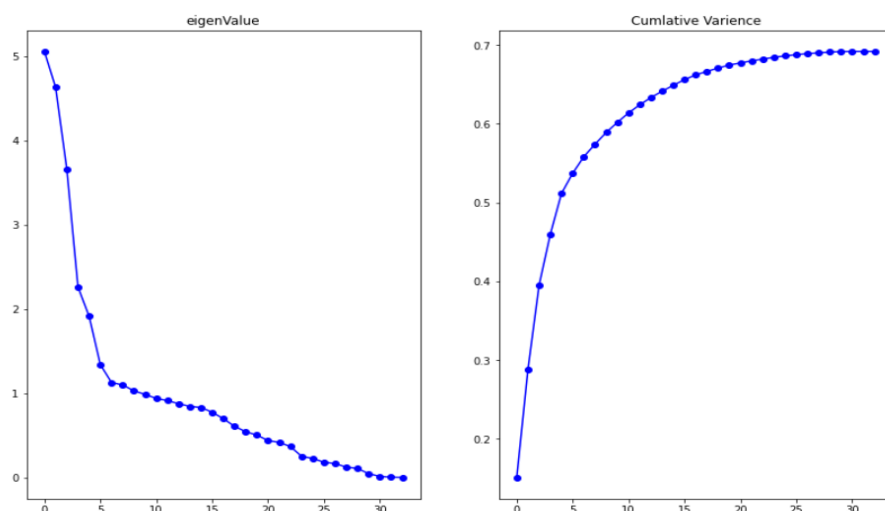


图 5 因子分析可行性检验

经过因子分析的可行性检验后, 数据集的巴特利特值和 kmo 值均符合条件, 因此可以使用因子分析进行降维。

## 5.4 构建模型

### 5.4.1 线性回归

线性回归 (Linear regression) 是利用称为线性回归方程的最小二乘函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。样本的回归函数为:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots + \beta_k x_{ik}$$

参数  $\beta_i$  的估计值  $\hat{\beta}_i$  由最小二乘原理得到, 即参数估计值是下列方程组的解

$$\begin{cases} \frac{\partial Q}{\partial b_0} = 0 \\ \frac{\partial Q}{\partial b_1} = 0 \\ \cdots \cdots \\ \frac{\partial Q}{\partial b_k} = 0 \end{cases}$$

其中,  $Q(b_0, b_1, \dots, b_k) = \sum_{i=1}^n \hat{u}_i^2 = \sum_{i=1}^n (y_i - b_0 - b_1 x_{i1} - \cdots - \beta_k x_{ik})^2$

由线性回归模型得到的因变量预测值为:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \hat{\beta}_3 x_{i3} + \cdots + \hat{\beta}_n x_{in}$$

通过调用 sklearn 中的 linear\_model.LinearRegression() 函数, 在不同数据预处理情况下得到的模型效果如下:

表 1 各数据处理方式下线性回归的模型效果

数据预处理方式	模型效果
---------	------

编码: CountEncoder 降维: 未降维	训练集 mse: 16.321902882953673 测试集 mse: 17.198400449765572 测试集 mape: 0.4153283109907366 测试集 accur: 0.0 测试集 score: 0.11693433780185268
编码: 均为数值型变量, 不存在编码问题 降维: 特征选择 (以 0.01 为重要性标准)	训练集 mse: 71.70336239413118 测试集 mse: 1319901.9618289284 测试集 mape: 0.5203050137187364 测试集 accur: 0.0 测试集 score: 0.09593899725625273
编码: OneHotEncoder 降维: 特征选择 (以 0.005 为重要性标准)	训练集 mse: 5.182449481758937e+24 测试集 mse: 3.0218206550470716e+24 测试集 mape: 85755150517.39914 测试集 accur: 0.0 测试集 score: -17151030103.279829
编码: CountEncoder 降维: 因子分析	训练集 mse: 97.21180067958802 测试集 mse: 62.228508208737864 测试集 mape: 0.5652591243628985 测试集 accur: 0.0 测试集 score: 0.08694817512742031

#### 5.4.2 回归树

决策树由结点(node)和有向边(directed edge)组成。结点有两种类型: 内部结点(internal node)和叶结点(leaf node)。内部结点表示一个特征或属性, 叶结点表示一个类别或者某个值。

一个回归树对应着输入空间 (即特征空间) 的一个划分以及在划分的单元上的输出值。假设已将输入空间划分为  $M$  个单元  $R_1, R_2, \dots, R_M$ , 并且在每个单元  $R_m$  上有一个固定的输出值  $c_m$ , 于是回归树模型可以表示为:

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m)$$

CART 回归树采用启发式的方法对输入空间进行划分, 选择第  $j$  个变量  $x^{(j)}$  和它取的值  $s$ , 作为切分变量 (splitting variable) 和切分点 (splitting point), 并定义两个区域:  $R_1(j, s) = \{x | x^{(j)} \leq s\}$  和  $R_2(j, s) = \{x | x^{(j)} > s\}$ 。通过求解:

来寻找最优切分点  $s$ 。

通过调用 sklearn 中 `tree.DecisionTreeRegressor()` 函数，在不同数据预处理情况下得到的模型效果如下：

表2 各数据处理方式下回归树的模型效果

数据预处理方式	模型效果
编码：CountEncoder 降维：未降维 调参：max_depth=19	训练集 mse: 4.621376071752553 测试集 mse: 4.718237236662554 测试集 mape: 0.17539879466788205 测试集 accur: 0.0 测试集 score: 0.16492024106642358
编码：均为数值型变量，不存在编码问题 降维：特征选择（以 0.01 为重要性标准） 参数：默认参数	训练集 mse: 42.61417227954587 测试集 mse: 1319926.223304201 测试集 mape: 0.18359200709926074 测试集 accur: 0.00033333333333333333 测试集 score: 0.16354826524681454
编码：OneHotEncoder 降维：特征选择（以 0.005 为重要性标准） 参数：默认参数	训练集 mse: 1131030.9464665093 测试集 mse: 32.38601453534583 测试集 mape: 0.13565348184466738 测试集 accur: 0.00011111111111111112 测试集 score: 0.17295819251995542
编码：OneHotEncoder 降维：PCA 调参：	训练集 mse: 1131364.8367311195 测试集 mse: 83.98005354340783 测试集 mape: 0.14671324035573838 测试集 accur: 0.0 测试集 score: 0.17065735192885234
编码：CountEncoder 降维：因子分析	训练集 mse: 69.39768115705638 测试集 mse: 44.08167012460929

	测试集 mape: 0.16309538531612636
	测试集 accur: 0.00011111111111111112
	测试集 score: 0.16746981182566364

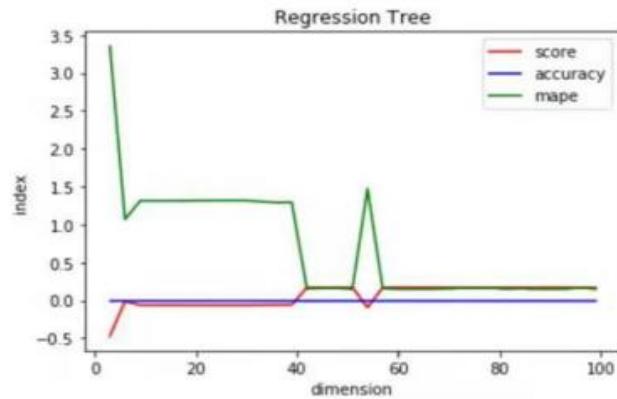


图 6 PCA 降维后回归树测试结果

### 5. 4. 3 随机森林

随机森林是在决策树的基础上做了改进。对于普通的决策树，我们会在节点上所有的  $n$  个样本特征中选择一个最优的特征来做决策树的左右子树划分，而随机森林通过随机选择节点上的一部分样本特征，这个数字小于  $n$ ，假设为  $n_{sub}$ 。在这些随机选择的  $n_{sub}$  个样本特征中，选择一个最优的特征来做决策树的左右子树划分。这样进一步增强了模型的泛化能力。

当输入的样本集  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ 、弱分类器迭代次数为  $T$ ，输出为最终的强分类器  $f(x)$ 。对于  $t=1, 2, \dots, T$ : 对训练集进行第  $t$  次随机采样，共采集  $m$  次，得到包含  $m$  个样本的采样集  $D_t$ 。用采样集  $D_t$  训练第  $t$  个决策树模型  $G_t(x)$ ，在训练决策树模型的节点时，在节点上所有的样本特征中选择一部分样本特征。在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分。 $T$  个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。

通过调用 `sklearn.ensemble` 中 `RandomForestRegressor()` 函数，在不同数据预处理情况下得到的模型效果如下：

表 3 各数据处理方式下随机森林的模型效果

数据预处理方式	模型效果
编码: CountEncoder	训练集 mse: 2.248651568903074
降维: 未降维	测试集 mse: 2.4794672075121524
参数: <code>n_estimators=200</code>	测试集 mape: 0.14142998030071374



	测试集 accur: 0.0001149029070435482 测试集 score: 0.1718059262654921
编码: 均为数值型变量, 不存在编码问题 降维: 特征选择 (以 0.01 为重要性标准) 参数: n_estimators=70	训练集 mse: 20.689009930723586 测试集 mse: 1319901.3686920642 测试集 mape: 0.1395811809459003 测试集 accur: 0.0 测试集 score: 0.17208376381081994
编码: OneHotEncoder 降维: 特征选择 (以 0.005 为重要性标准) 参数: 默认参数	训练集 mse: 674845.2133372057 测试集 mse: 248277.7030952193 测试集 mape: 0.5675642075541376 测试集 accur: 0.00033333333333333333 测试集 score: 0.08675382515583915
编码: OneHotEncoder 降维: PCA 调参: 默认参数	测试集 accur: 0.0001 测试集 score: 0.1334
编码: CountEncoder 降维: 因子分析 调参: 默认参数	训练集 mse: 55.188662111949405 测试集 mse: 15.154080034401051 测试集 mape: 0.12710326108545056 测试集 accur: 0.0 测试集 score: 0.1745793477829099
编码: CountEncoder 降维: 调参: n_estimators =800	训练集 mse: 47.9379965945527 测试集 mse: 13.493412508101459 测试集 mape: 0.1048994804649688 测试集 accur: 0.0 测试集 score: 0.17902010390700626

我们选取最后一个模型作为最终模型。

#### 5.4.4 神经网络

多层感知机 (multilayer perceptron, MLP) 也被称为 (普通) 前馈神经网络, 简称神经网络。MLP 可以被视为广义的线性模型, 通过执行多层处理后得到结论。在线性模型中回归的预测公式为:  $\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$ , 即  $\hat{y}$  是输入特征  $x[0]$  到  $x[p]$  的加权求和, 权重为学习得到的系数  $w[0]$  到  $w[p]$ 。而在 MLP 中, 则需要多次重复这个加权求和的过程。首先计算代表中间过程的隐单元 (hidden unit), 再计算这些隐单元的加权求和并得到最终结果。

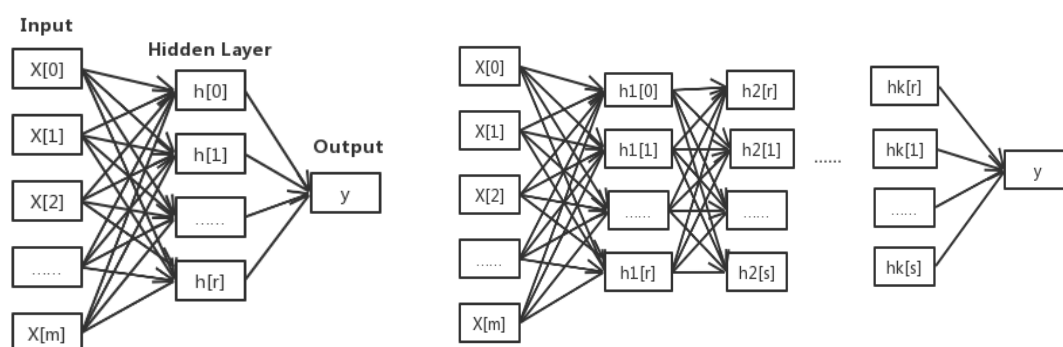


图 7 神经网络模型原理（左：单隐层，右：多隐层）

通过调用 `sklearn.neural_network` 中的 `MLPRegressor()` 函数，在不同数据预处理情况下得到的模型效果如下：

表 4 各数据处理方式下神经网络的模型效果

数据预处理方式	模型效果
编码：CountEncoder 降维：未降维 参数：默认参数	训练集 mse: 3.7509973844670528 测试集 mse: 3.636011262773957 测试集 mape: 0.017278330869320516 测试集 accur: 0.0001149029070435482 测试集 score: 0.16553526058699383
编码：均为数值型变量，不存在编码问题 降维：特征选择（以 0.01 为重要性标准） 参数：activation='tanh'	训练集 mse: 48.57959063187484 测试集 mse: 1319928.512109628 测试集 mape: 0.1966814546767688 测试集 accur: 0.00011111111111111112 测试集 score: 0.16075259795353514
编码：OneHotEncoder 降维：特征选择（以 0.005 为重要性标准） 参数：activation='tanh'	训练集 mse: 48.57959063187484 测试集 mse: 1319928.512109628 测试集 mape: 0.1966814546767688 测试集 accur: 0.00011111111111111112 测试集 score: 0.16075259795353514
编码：OneHotEncoder 降维：PCA 调参：	测试集 accur: 0.0001 测试集 score: 0.1334
编码：CountEncoder 降维：因子分析	训练集 mse: 35.699796563614186 测试集 mse: 12.366794312125139 测试集 mape: 0.1950985501410069 测试集 accur: 0.0

	测试集 score: 0.1609802899717986
--	-------------------------------

#### 5.4.5 LightGBM

GBDT 是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。GBDT 在工业界应用广泛，通常被用于多分类、点击率预测、搜索排序等任务。

LightGBM (Light Gradient Boosting Machine) 则是一个实现 GBDT 算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点。LightGBM 在传统 GBDT 算法上进行了如下优化<sup>1</sup>：

(1) 采用基于 Histogram 的决策树算法。

(2) 单边梯度采样 Gradient-based One-Side Sampling(GOSS)：使用 GOSS 可以减少大量只具有小梯度的数据实例，这样在计算信息增益的时候只利用剩下的具有高梯度的数据就可以了，相比 XGBoost 遍历所有特征值节省了不少时间和空间上的开销。

(3) 互斥特征捆绑 Exclusive Feature Bundling(EFB)：使用 EFB 可以将许多互斥的特征绑定为一个特征，这样达到了降维的目的。

(4) 带深度限制的 Leaf-wise 的叶子生长策略：大多数 GBDT 工具使用低效的按层生长 (level-wise) 的决策树生长策略，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。LightGBM 使用了带有深度限制的按叶子生长 (leaf-wise) 算法。

(5) 直接支持类别特征 (Categorical Feature)。

(6) 支持高效并行等。

通过调用 `lgbm.LGBMRegressor(objective='regression')` 函数，同时采用 `CountEncoder` 对数据集进行编码，我们得到最终结果如下：

表 5 各数据处理方式下 LightGBM 的模型效果

数据预处理方式	模型效果
编码：CountEncoder 降维：未降维	训练集 mse: 2.1856542573035114 测试集 mse: 2.39494960792052 测试集 mape: 0.14679727231552547 测试集 accur: 0.0001149029070435482 测试集 score: 0.17073246786252974
编码：CountEncoder	训练集 mse: 52.79601855380041

<sup>1</sup> 深入理解 LightGBM: <https://zhuanlan.zhihu.com/p/99069186>

降维：因子分析	测试集 mse: 17.53209141472146 测试集 mape: 0.19326885459158474 测试集 accur: 0.0 测试集 score: 0.16134622908168306
---------	---

#### 5.4.6 集成学习

集成学习（Ensemble learning）是一种将很多的机器学习算法结合在一起的算法。组成集成学习的算法为“个体学习器”，若个体学习器彼此相同，则将其称为“基学习器”。

将个体学习器结合在一起时使用的方法叫做结合策略。在集成学习中，有一种有效的组合策略叫做 Stacking。在 Stacking 方法中，把个体学习器叫做“初级学习器”，用于结合的学习器叫做“次级学习器”，次级学习器用于训练的数据叫做次级训练集。次级训练集是在训练集上用初级学习器得到的。Stacking 算法的具体流程如下图所示。

```

输入: 训练集  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ;
      初级学习算法  $\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_T$ ;
      次级学习算法  $\mathcal{L}$ .
过程:
1: for  $t = 1, 2, \dots, T$  do
2:    $h_t = \mathcal{L}_t(D)$ ;
3: end for
4:  $D' = \emptyset$ ;
5: for  $i = 1, 2, \dots, m$  do
6:   for  $t = 1, 2, \dots, T$  do
7:      $z_{it} = h_t(x_i)$ ;
8:   end for
9:    $D' = D' \cup ((z_{i1}, z_{i2}, \dots, z_{iT}), y_i)$ ;
10: end for
11:  $h' = \mathcal{L}(D')$ ;
输出:  $H(x) = h'(h_1(x), h_2(x), \dots, h_T(x))$ 

```

图 8 Stacking 算法

对于每个模型，使用交叉验证的方法来训练初级学习器，可以得到预测结果，再将其作为次级学习器的训练集训练模型，得到最终模型。具体模式如下图所示。

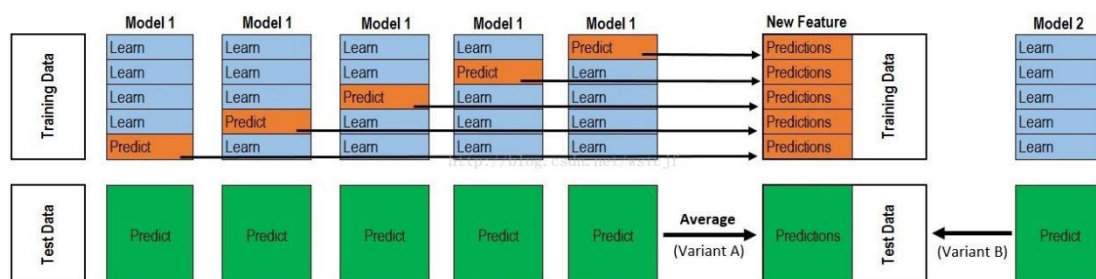


图 9 stacking 模式

针对问题一，参考不同个体学习器的表现，我们的 Stacking 模型设置如下：

初级学习器：线性回归、随机森林、LightGBM、回归树

次级学习器：线性回归

通过集成学习，可以在个体学习器的基础上有效提高模型的性能。最终在对数据进行 CountEncoder 基础上得到测试结果如下：

表 6 各数据处理方式下 stacking 的模型效果

数据预处理方式	模型效果
编码：CountEncoder 降维：未降维	测试集 mse: 2.27707817 测试集 mape: 0.13941684 测试集 accur: 0.0001149029070435482 测试集 score: 0.17220855
编码：CountEncoder 降维：因子分析	测试集 mse: [11.11337951] 测试集 mape: [0.14778379] 测试集 accur: 0.0 测试集 score: [0.17044324]

## 5.5 模型选择与问题求解

在编码方式的选择上，由于 OneHotEncoder 编码下数据维度过大，模型的性能得不到很好的改善，因此选择了使得模型效果更好的 CountEncoder 编码处理给定数据集中的分类变量。

在降维方法的选择上，由于特征选择下信息损失较大；主成分分析下因子的综合评价函数意义不明确、命名清晰性低；因子分析的方差贡献率比较小。通过模型结果发现在不降维的情况下模型效果比通过以上方法降维后的效果更好。

在模型的选择上，由于线性回归对变量间的线性关系要求较高，难以很好地表达高度复杂的数据；回归树有欠拟合的倾向；神经网络的效果对数据量和超参数的设置有较大依赖。综合对比上述不同数据处理方式下各模型的平均相对误差、误差准确率、均方误差等，最终我们选择随机森林模型。

综上所述，CountEncoder 编码处理后、在未降维的情况下，采用随机森林模型在附件 1 所划分的训练集上训练模型并在测试集上进行测试后，将附件 2 中的数据代入到模型中得到了预测结果，详见附件 3。模型参数如下：

模型：随机森林	训练集 mse: 47.9379965945527
编码：CountEncoder	测试集 mse: 13.493412508101459
降维：无	测试集 mape: 0.1048994804649688
调参：n_estimators =800	测试集 accur: 0.0
	测试集 score: 0.17902010390700626

## 六、问题二的模型建立与求解

### 6.1 建模思路

我们将问题二分解为两个部分处理：首先根据附件 4 中的车辆 id 信息，将其与附件 1 中的信息进行连接，得到包含更多特征的数据集，并将车辆是否成交，即成交时间是否存在作为因变量，同时根据 {价格调整时间：调整后价格} 构造新的变量，同其余 37 个变量一起作为自变量，利用逻辑回归、决策树、随机森林和支持向量机进行分类预测，探究车辆成交结果的影响因素。

随后，将车辆成交周期作为因变量，其余变量作为自变量，利用线性回归、回归树、随机森林等具有可解释性的模型进行回归预测，得到影响车辆成交周期的关键因素，进行进一步分析。

### 6.2 数据预处理

本问中我们同时使用了附件 1 和附件 4 中的数据。对于附件一中的数据，采取和问题一的相同处理方法。对于附件 4 的数据，我们在预处理的基础上最大程度利用数据信息，构造新的变量，提升模型预测效果。

#### 6.2.1 数据拼接与清洗

首先，通过观察发现，附件 2 的 carid 是附件 1 中 carid 的子集，使用 merge 函数连接附件 2 和附件 1 两张表，设置参数 on="carid", how="left"，完成两张表以 carid 作为关键词的拼接。

随后，我们对数据类型进行了修改。updatePriceTimeJson 列的信息可读性差，我们通过 split() 等函数，将 updatePriceTimeJson 列的信息转化为：调整时间跨度（pull\_pushTime 列）、价格调整次数（update\_num 列）、价格弹性（update\_T\_el 列）、价格相对变动幅度（delta\_price 列）。其中，价格弹性的定义为：

假设价格调整次数为  $k$ ，上架时间为  $T_0$ ，下架时间为  $T_{k+1}$ ，

$$\text{则平均价格调整时间 } T_{avg} = \frac{\sum_{i=1}^k T_i - T_{i-1}}{k} = \frac{T_k - T_0}{k}$$

$$\text{则价格弹性为 } elasticity = \frac{1}{T_{avg}}$$

特别的，若  $k=0$ ，则将 elasticity 记为 0。

最后，针对 withdrawDate 列，我们将其作为判断车辆是否成交的变量。若为空值，则车辆未成交，将其记为 0；反之则说明车辆成交，记为 1。

### 6.2.2 划分测试集和训练集

通过调用 `sklearn.model_selection` 中 `train_test_split` 函数，分别以 `withdraw` 和 `pull_pushTime` 作为 `y`，以其他特征信息作为 `X`，设置 `test_size=0.3`，对数据集进行划分。划分后的结果为：

```
X_train.shape: (7000, 39)
X_test.shape: (3000, 39)
y_train.shape: (7000,)
y_test.shape: (3000,)
```

图 10 训练集和测试集规模

### 6.2.3 缺失值填充

首先，查看缺失值的分布情况。在附件二所包含的 10000 条数据中，确实情况如下：

特征	缺失数据量
carid	0
pushDate	0
pushPrice	0
updatePriceTimeJson	6762
pullDate	0
withdrawDate	2000

针对附件 1 中的缺失值，采用和问题一相同的方法，使用随机森林预测填充。

对于附加 4，若 `updatePriceTimeJson` 列的数据为空，则将 `update_num`、`update_T_el`、`delta_price` 均记为 0。通过观察数据，我们发现 `feature12` 列出现一条新的缺失值，采用众数的方法填充。

## 6.3 数据分析

### 6.3.1 离散特征编码处理

采用和问题一中相同的方法，用 `CountEncoder` 编码对数据集中的分类变量进行编码。

### 6.3.2 连续特征标准化处理

与问题一的连续特征标准化相似，通过调用 `MinMaxScaler()` 函数进行归一化处理，将所有数值型变量均缩放到了 `[0,1]` 区间内，避免了因不同数据间的量纲不同而对模型产生影响，使模型预测更加准确。

### 6.3.3 降维

在问题二中我们尝试了以下降维方法：

#### （1）低方差滤波

在对数值型变量做 `MinMaxScaler()` 的归一化处理后，查看各变量的方差。若某个特征的方差过小，则其所包含的信息过于有限，为了降低数据维度，将方差过小的特征进行丢弃。

通过计算各变量的方差，我们可以选择性地舍弃方差较小的变量，如：“`withdrawDate`”、“`carCode`”、“`country`”、“`maketype`”、“`gearbox`”、“`feature1`”、“`feature8`”到“`feature11`”。

#### （2）特征选择

通过调用 `RandomForestClassifier` 的 `feature_importances_()` 函数提取特征重要性，选择重要性高的变量放入模型中。数据集中的“`update_num`”、“`update_T_el`”、“`cityId`”三个特征重要性最高，可以选择性地将这三个特征放入模型中。

## 6.4 构建模型

由于各模型的作用原理在问题一构建模型时已经阐述，此处不再赘述。以下是各模型的作用效果。

### 6.4.1 线性回归

线性回归模型是输入特征的线性函数进行预测。通过调用 `sklearn` 中的 `linear_model.LinearRegression()` 函数，在不同数据预处理情况下得到的模型效果如下：

表 7 各数据处理方式下线性回归的模型效果

数据预处理方式	模型效果
编码：CountEncoder 降维：未降维	训练集 mse: 13.159272705800444 测试集 mse: 12.4171359197618



	测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf
编码: CountEncoder 降维: 低方差滤波	训练集 mse: 13.154614178766135 测试集 mse: 12.417135919761805 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf
编码: CountEncoder 降维: 低方差滤波+特征选择 (3 个)	训练集 mse: 13.400382072487025 测试集 mse: 12.785794818304653 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf

#### 6.4.2 回归树

决策树本质上是从一层层 if/else 问题中进行学习。树中每个结点代表一个问题或一个包含答案的叶结点。通过调用 sklearn 中 tree.DecisionTreeRegressor() 函数，在不同数据预处理情况下得到的模型效果如下：

表 8 各数据处理方式下回归树的模型效果

数据预处理方式	模型效果
编码: CountEncoder 降维: 未降维	训练集 mse: 11.754110527354403 测试集 mse: 11.319678265680626 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf

#### 6.4.3 随机森林

随机森林本质上是许多回归树的集合，即使每棵树中存在过拟合，通过构造很多棵树且每棵以不同的方式存在过拟合，随机森林对这些树的结果求平均值可以即减少过拟合又能保持树的预测能力。通过调用 sklearn.ensemble 中 RandomForestRegressor() 函数，在不同数据预处理情况下得到的模型效果如下：

表 9 各数据处理方式下随机森林的模型效果

数据预处理方式	模型效果
编码: CountEncoder	训练集 mse: 10.428544778320312

降维：未降维	测试集 mse: 9.723242597656245 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf
编码：CountEncoder 降维：低方差滤波	训练集 mse: 10.421124948242186 测试集 mse: 9.74402552734374 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf
编码：CountEncoder 降维：低方差滤波+特征选择（3 个）	训练集 mse: 10.418439561940959 测试集 mse: 9.629900447920914 测试集 mape: inf 测试集 accur: 0.0 测试集 score: -inf

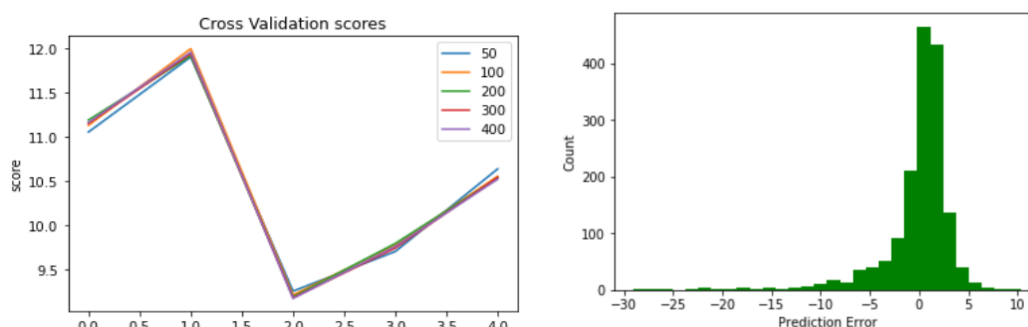
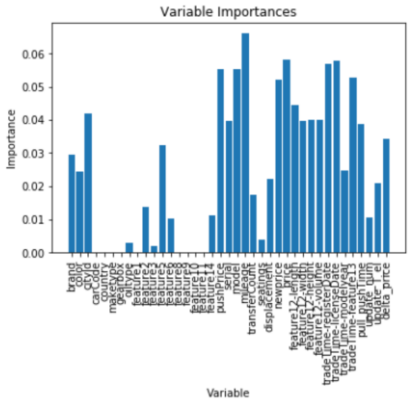
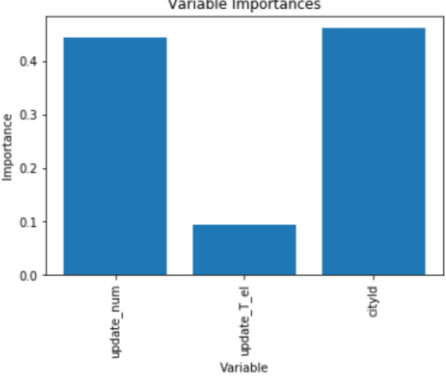


图 11 低方差滤波和特征选择后在测试集和训练集上表现

## 6.5 模型选择与问题求解

在综合比较多种机器学习方法后，我们选取了 F1 值最高的随机森林模型作为分类预测模型；选择 MSE 最小的随机森林作为回归预测模型。两种模型在测试集上性能如下：

模型	分类模型	回归模型
因变量	是否成交	车辆成交周期
编码方式	CountEncoder	CountEncoder
性能	precision: 0.801432958034800 recall: 0.985312631137222 F1: 0.8827067669172933	mse: 9.629900447920914
重要性分数		
主要影响因素	价格变动幅度 年款 二手车交易价格 过户次数	价格调整次数 价格弹性 车辆所在城市

为了进一步挖掘影响车辆成交周期的关键因素，我们对随机森林分类和回归周期做了可视化处理，结果如下：

(1) 分类模型（由于树的深度较长，只截取部分作为图例，完整图片详见附件）

如下图所示，该可视化结果反映了不同变量的重要性程度以及该变量对于分类结果的影响：重要性程度从根节点到叶子节点依次递减，评价标准为信息熵，信息熵越大，则该变量包含信息越多；每个父节点含有不同的子节点，表示父节点表示的变量取值对最终分类结果的影响。

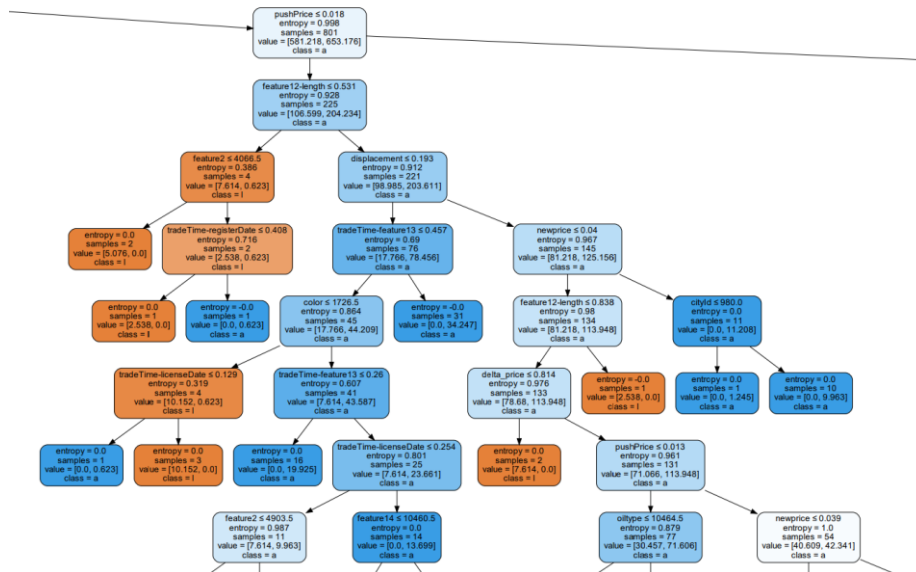


图 12 随机森林分类过程可视化

由于二手车能否成交的影响因素较多，我们选取重要性分数大于 0.05 的变量进行重点研究。结合上文可知，这些变量距离根节点位置更近，所含信息更多，应当作为首要考虑因素（以下数值均为最大-最小值标准化后数据，非真实数据）：

- 价格变动幅度：**临界值为 0.833，若小于该值，则交易成功可能性较小，此时需要考虑年款；反之，交易成功可能性较大，下一步考虑因素为交易价格。这也在一定程度上反映了消费者的消费心理：对于价格变动较小的车，消费者往往不愿接受高价购买，此时需要考虑车的详细款式与价格是否匹配；反之，若价格变动幅度较大，则希望详细了解价格信息。
- 年款：**临界值为 0.508，若小于该值，购买可能性较高，下一步继续考虑过户次数；若大于 0.508，则考虑车辆所在城市信息。
- 二手车交易价格：**临界值为 0.006，可以看出影响消费者决策的价格远低于二手车平均价格。若小于 0.0006，消费者会将其与 0.004 比较，做出下一步决策，购买可能性较高；反之则继续考虑车型是否符合期望。
- 过户次数：**临界值为 0.056，若小于该值，则考虑注册距离展销时间；反之则考虑新车价格。这可以解释为：如果二手车过户次数较少，人们会选择查看注册日期；若过户次数较多，人们会直接考虑新车价格，购买可能性较低。

（2）回归模型（由于树的深度较长，只截取部分作为图例，完整图片详见附件）

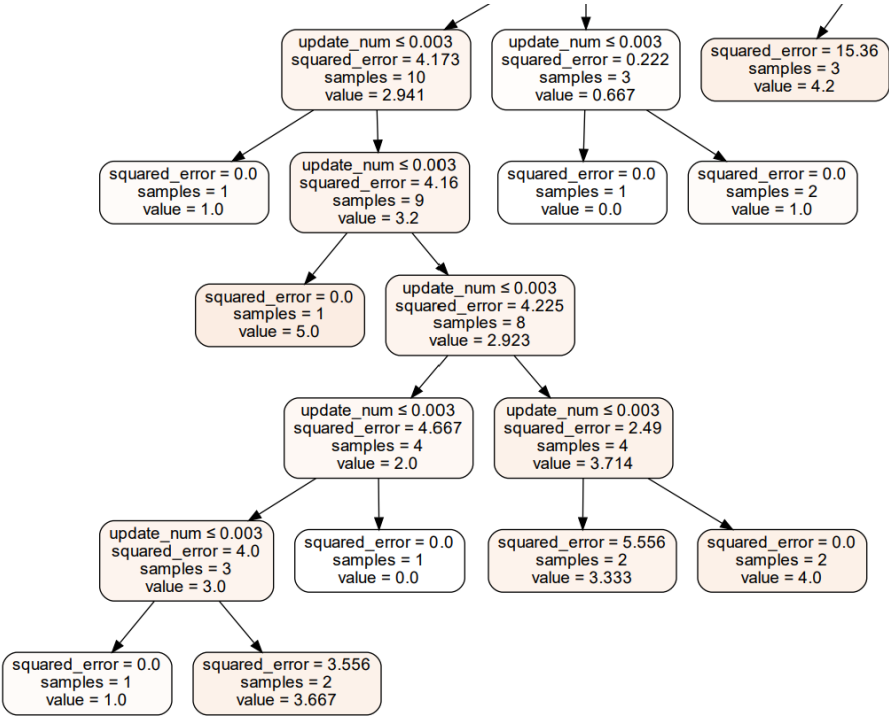


图 13 随机森林预测过程可视化

在（1）的基础上，进一步考虑影响二手车成交周期的影响因素（以下数值均为最大-最小值标准化后数据，非真实数据）。

- **车辆所在城市：**作为首要考虑因素，临界值为 0.003，若大于该值，则考虑价格调整次数；反之则对城市信息进一步分类分析。
- **价格调整次数：**临界值为 0.002，若小于该值则继续对调整次数分类分析；若大于 0.002，可以直接预测车辆价格。
- **价格弹性：**在考虑车辆所在城市的基础上，根据价格弹性进一步分析。一般来讲，若价格弹性较小，价格越高；反之则越低。这与经济现象相符：**若市场价格对于需求敏感，则价格变动频繁，价格降低可能性较大；反之则较小。**

## 6.6 解决方法与预期效果

作为中间商，实现盈利的途径在于提高收益、降低成本。对二手车零售商来讲，实现收益的途径在于合理定价，如问题一；而降低成本的方法，就是尽量缩短二手车成交周期，如问题二。在问题二模型的基础上，我们给出如下建议：

(1) **价格变动幅度：**当二手车滞销或超售时，首要的应对方法就是调整价格。临界值为 0.83（标准化后百分比），即若某辆车的价格调整幅度大于最大调整幅度的 83%时，该车成功出售的可能性更大。因此，若选择调整价格，可以将调价幅度设置为历史最大调整幅度的 0.83。

(2) **年款：**一般来讲，款式越新，人们购买可能性越高。但也存在部分人群偏爱旧款汽车。因此，可以将平均年款作为临界值，对高于和低于该值的产品针对不同消费者定向销售。

(3) **二手车交易价格：**当价格较低时，成功出售的可能性显著增大。但当价格无法降低时，可以考虑向消费者提供车辆的具体信息，如车型。在增加消费者对产品的了解后，其购买意愿会大幅增加。

(4) **过户次数：**一般过户次数越小，意味着车的崭新程度越高，人们需求越大。临界值约为 0.056（标准化后百分比）。当过户次数较高时，人们会倾向于考虑购买新车。此时，若新车价格较高，人们购买二手车的意愿会增加；反之，则需要重新定价。

(5) **车辆所在城市：**这是缩短车辆成交周期的首要考虑因素不同城市的车辆成交周期存在较大差异，因此应当对不同城市的销售情况建模，独立分析。对于周期较短的城市，可以增加车辆数目；反之则减少。

(6) **价格调整次数：**一般来讲，价格调整频繁的车辆被购买的可能性较高。这与人们的消费心理相符。在无法大幅降价的情况下，可以适量增大初始价格，随后频繁调整价格，有利于车辆的快速出售。

## 七、问题三的模型建立与求解

### 7.1 问题提出

基于给定的样本数据集，结合二手车市场的情况，在此问中我们将研究：针对不同价格档次的车，销售商是否会采取不同的销售策略。

### 7.2 建立模型

我们采用 KMeans 聚类的方式，对数据集的样本进行分类。KMeans 聚类模型的原理如下：

在给定  $K$  值和  $K$  个初始类簇中心点的情况下，把每个点（亦即数据记录）分到离其最近的类簇中心点所代表的类簇中，所有点分配完毕之后，根据一个类簇内的所有点重新计算该类簇的中心点（取平均值），然后再迭代地进行分配点和更新类簇中心点的步骤，直至类簇中心点的变化很小，或者达到指定的迭代次数。假定给定数据样本  $X$ ，包含了  $n$  个对象  $X=(X_1, X_2, \dots, X_n)$ ，其中每个对象都具有  $m$  个维度的属性。KMeans 算法的目标是将  $n$  个对象依据对象间的相似性聚集到指定的  $k$  个类簇中，每个对象属于且仅属于一个其到类簇中心距离最小的类簇中。对于 KMeans，首先需要初始化  $k$  个聚类中心  $(C_1, C_2, \dots, C_k)$ ，然后通过计算每一个对象到每一个聚类中心的欧式距离，如下式所示：

$$\text{dis}(X_i, C_j) = \sqrt{\sum_{t=1}^m (X_{it} - C_{jt})^2}$$

上式中， $X_i$  表示第  $i$  个对象  $1 \leq i \leq n$ ， $C_j$  表示第  $j$  个聚类中心  $1 \leq j \leq k$ ， $X_{it}$  表示第  $i$  个对象的第  $t$  个属性， $1 \leq t \leq m$ ， $C_{jt}$  表示第  $j$  个聚类中心的第  $t$  个属性。

依次比较每一个对象到每一个聚类中心的距离，将对象分配到距离最近的聚类中心的类簇中，得到  $k$  个类簇  $(S_1, S_2, \dots, S_k)$ 。

KMeans 算法用中心定义了类簇的原型，类簇中心就是类簇内所有对象在各个维度的均值，其计算公式如下：

$$C_l = \frac{\sum_{X_i \in S_l} X_i}{|S_l|}$$

上式中， $C_l$  表示第  $l$  个聚类的中心， $1 \leq l \leq k$ ， $|S_l|$  表示第  $l$  个类簇中对象的个数， $X_i$  表示第  $l$  个类簇中第  $i$  个对象， $1 \leq i \leq |S_l|$ 。

## 7.3 模型结果

通过调用 `sklearn.cluster` 中 `KMeans` 函数并设置参数 `n_clusters=2`，我们将样本数据集划分为了两类。通过对比两类数据的各特征的均值后，以下变量在两类间存在明显差异：

	第一类簇均值(1)	第二类簇均值 (2)	$((2)-(1))/(1)$
price	11.32510338	45.25123549	2.99565760922059
pushPrice	11.52335963	45.60379823	2.95750889512615
newprice	23.30100197	101.0835493	3.33816320216155
transferCount	0.410685549	0.768777614	0.871937341009506
delta_price	0.016940773	0.004994095	-0.705202671850291

通过对比可以发现，第一类的价格总体较低，属于普通车。第二类的价格总体较高，属于高档车。其中第一类的数据集中包含了 **9321** 个样本，第二类的数据集中包含了 **679** 个样本，这与市场是绝大多数是中低档汽车、少数是高档的高端汽车情况相一致。为了更进一步分析，我们绘制了这几类变量的小提琴图。

首先是反映二手车价格的三个变量：`price`、`pushPrice`、`newprice`，结果如下图所示。

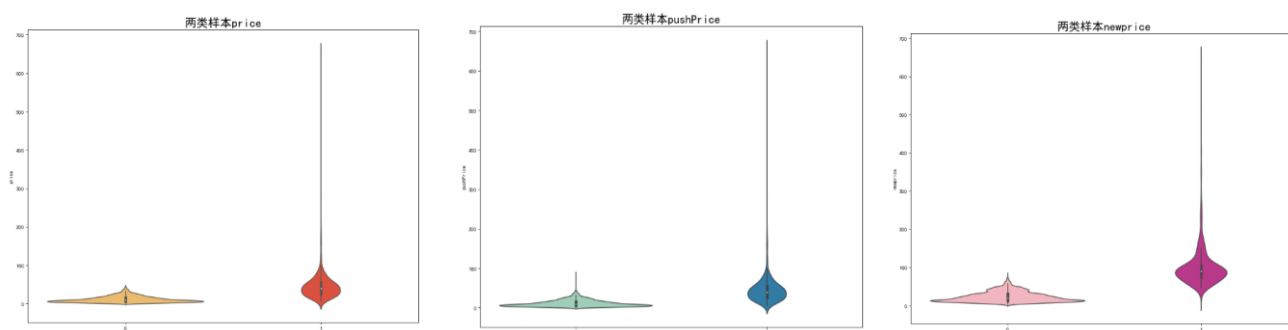


图 14 两类二手车价格分布的小提琴图（左 `price`，中 `pushprice`，右 `newprice`）

从上图可以直观看出，`price`、`pushPrice`、`newprice` 三个变量的总体分布非常相似；第一类车的均价明显低于第二类车。第一类样本的分布更加集中，第二类数据集内部间的样本差异比第一类更大。

在明确了两类车辆的价格区别后，在关于价格调整次数和价格调整幅度上，两类样本也存在显著差别。

首先是价格调整次数的差异：

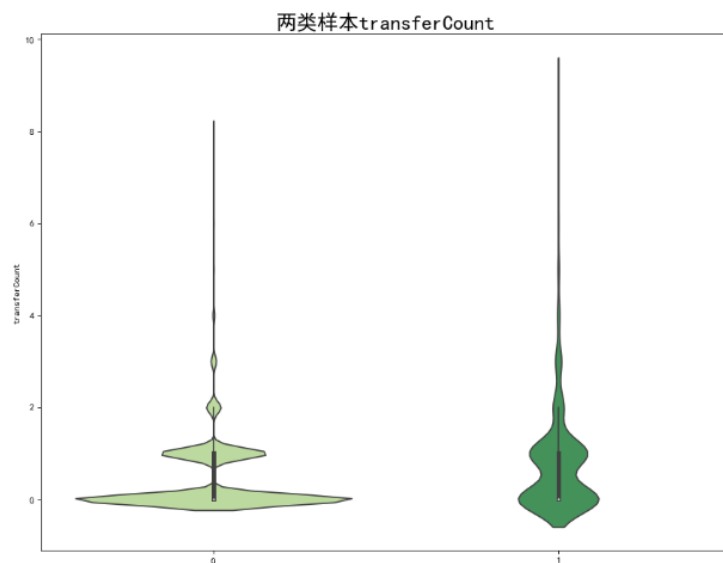


图 15 两类二手车价格调整次数的小提琴图

从上图可以看出，第一类车的价格调整集中在 0 值附近，说明多数第一类车是不进行价格调整或仅进行少数次数的价格变动。而第二类车的价格变动次数均值高于第一辆车，其变动的分布也更加离散，总体来说高端车的调价次数更频繁。

其次是价格调整幅度的差异：

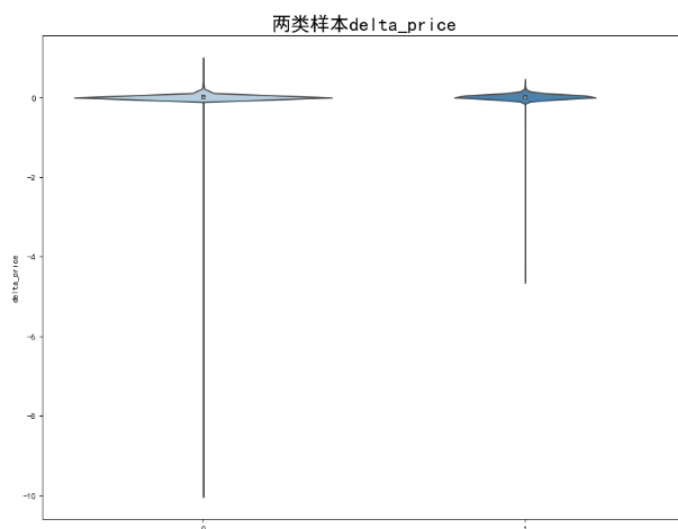


图 16 两类二手车价格调整幅度的小提琴图

对于价格变动幅度，采用的公式是（上架价格 - 最后一次调整的价格）/上架价格，即所计算出的价格变动幅度是相对程度。结合数据和图进行分析后，第一类的价格相对变动程度的均值为 0.016940773，即中低端车的最后一次调整价格平均比上架时低 1.69%；第二类的价格相对变动程度的均值为 0.004994095，即高端车的最后一次价格调整仅比上架时低 0.499%。但是从图中可以发现，价格调整不仅存在下调的情况（即图中处于 0 值以上的部分），也存在上调的可能性（即



图中处于 0 值以下的部分），且第一类车的上调幅度更大。

此外，值得注意的是，在 15 个匿名变量中，feature2 也在两类样本集之间存在明显差异：

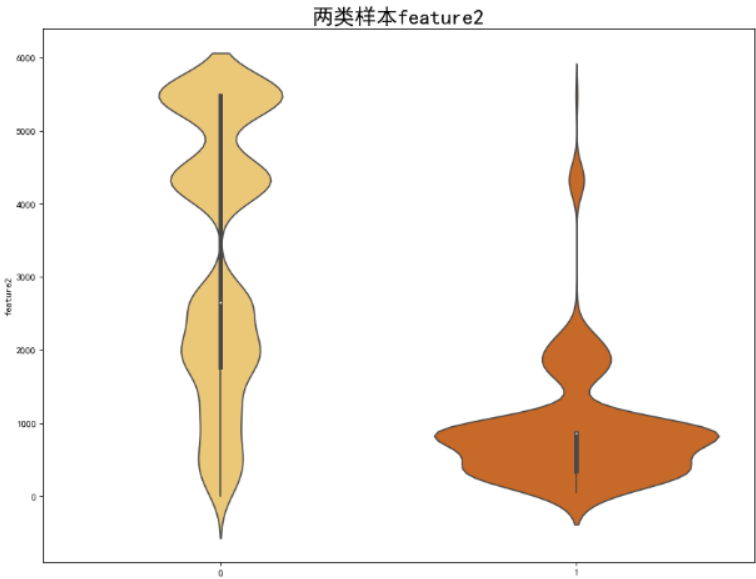


图 17 两类二手车 feature2 的小提琴图

在第一类样本中，feature2 的均值是 3288.26209634159；在第二类样本中，feature2 的均值是 911.101620029455。在图中，第一类样本的 feature2 分布较为分散、第二类样本的 feature2 则大多集中在均值附近。由于 feature2 是匿名变量，我们无法准确推断 feature2 的含义，若 feature2 的实际含义与价格或价格变动等销售策略相关，则也可以作为不同档次的二手车的销售策略差异。

#### 7.4 结论归纳

在此问题中，首先通过聚类的方式，将样本数据集划分为了两类。经过检验，所划分的两类样本可以被视为中低端类二手车和高端二手车。其次，对比了两类样本的各特征的均值和分布，从而推测出以下销售策略：

（1）中低端车在上架后，价格调整频率较低；一旦实行价格调整，调整的相对幅度较大。

（2）高端车在上架后，价格调整较为频繁；每次调价的相对幅度较低。

## 参考文献

- [1]贾鹏翔. 基于 LightGBM 的二手车价格预测[D]. 山东师范大学, 2021.

## 附 录

```
1. import pandas as pd
2. import numpy as np
3. from sklearn.preprocessing import StandardScaler
4. from sklearn import preprocessing
5. from collections import Counter
6. import warnings; warnings.simplefilter('ignore')
7. import matplotlib.pyplot as plt
8. #显示所有列, 把行显示设置成最大
9. pd.set_option('display.max_columns', None)
10. df = pd.read_csv('附件/附件 1: 估价训练数据.csv', engine='python', header=None, dtype=object)
11. # 列命名
12. col = ['carid', 'tradeTime', 'brand', 'serial', 'model',
13.        'mileage', 'color', 'cityId', 'carCode', 'transferCount',
14.        'seatings', 'registerDate', 'licenseDate', 'country',
15.        'maketype', 'modelyear', 'displacement', 'gearbox',
16.        'oiltype', 'newprice', 'feature1', 'feature2', 'feature3',
17.        'feature4', 'feature5', 'feature6', 'feature7', 'feature8',
18.        'feature9', 'feature10', 'feature11', 'feature12',
19.        'feature13', 'feature14', 'feature15', 'price']
20. df.columns = col
21. #类型修改
22. # df['mileage'] = df['mileage'].astype('float64')
23. df['transferCount'] = df['transferCount'].astype('float64')
24. df['seatings'] = df['seatings'].astype('float64')
25. df['displacement'] = df['displacement'].astype('float64')
26. df['newprice'] = df['newprice'].astype('float64')
27. df['price'] = df['price'].astype('float64')
28. df.head(3)
29. #统计有多少缺失值
30. missing_count = df.isnull().sum()
31. #第 23、26 和 34 列缺失值超过 50%, 无法填补缺失值, 删去这两列
32. df=df.drop(df[["feature4", "feature7", "feature15"]], axis=1)
33. missing_count = df.isnull().sum()
34. #绘制箱线图
35. y=df.loc[:, "price"]
36. plt.boxplot(y, sym='s')
37. #剔除异常值, 再次绘制箱线图
38. df = df.drop(index=22115, axis=0)
39. y=df.loc[:, "price"]
40. plt.boxplot(y, sym='s')
41.
42. df.loc[df['feature11']=='1+2', 'feature11'] = '1'
```

```

43. df.loc[df['feature11']=='1', 'feature11'] = '2'
44. df.loc[df['feature11']=='3+2', 'feature11'] = '3'
45. df.loc[df['feature11']=='1+2,4+2', 'feature11'] = '4'
46. df.loc[df['feature11']=='5', 'feature11'] = '5'
47. df.loc[df['feature11']=='1,3+2', 'feature11'] = '6'
48.
49. df['feature11'] = df['feature11'].astype(object)
50.
51.
52. # 拆2: feature12
53.
54. str_slice = pd.Series(df["feature12"])
55. info = [s for s in str_slice.str.split()]
56. length = []
57. width = []
58. height = []
59. volume = []
60.
61. for i in info:
62.     sub_info = i[0].split('*')
63.     length.append(int(sub_info[0]))
64.     width.append(int(sub_info[1]))
65.     height.append(int(sub_info[2]))
66.     volume.append(int(sub_info[0])*int(sub_info[1])*int(sub_info[2]))
67.
68. df.insert(df.shape[1], 'feature12-length', length)
69. df.insert(df.shape[1], 'feature12-width', width)
70. df.insert(df.shape[1], 'feature12-height', height)
71. df.insert(df.shape[1], 'feature12-volume', volume)
72.
73. df['feature12-length'] = df['feature12-length'].astype('float64')
74. df['feature12-width'] = df['feature12-width'].astype('float64')
75. df['feature12-height'] = df['feature12-height'].astype('float64')
76. df['feature12-volume'] = df['feature12-volume'].astype('float64')
77.
78. # 删除原列
79. df=df.drop(df[["feature12"]], axis=1)
80.
81. #将字符串转为日期类型,分别记为变量 tradeTime、
82. tradeTime = pd.to_datetime(df['tradeTime'])
83. registerDate = pd.to_datetime(df['registerDate'])
84. licenseDate = pd.to_datetime(df['licenseDate'])
85.

```

```

86. #计算 registerDate-tradeTime, licenseDate-tradeTime
87. delta_reg = tradeTime-registerDate
88. delta_lic = tradeTime-licenseDate
89.
90. #以月为单位表示时间差
91. delta_weeks_reg = []
92. for delta in delta_reg:
93.     if delta.days % 7 <= 3:
94.         weeks = delta.days//7
95.     else:
96.         weeks = delta.days//7 + 1
97.     delta_weeks_reg.append(weeks)
98. delta_weeks_lic = []
99. for delta in delta_lic:
100.    if delta.days % 7 <= 3:
101.        weeks = delta.days//7
102.    else:
103.        weeks = delta.days//7 + 1
104.    delta_weeks_lic.append(weeks)
105.
106.
107. #转为 datetime 类型
108. feature13 = pd.to_datetime(df['feature13'],format='%Y%m')
109. modelyear = pd.to_datetime(df['modelyear'])
110.
111. #计算 tradeTime-modelyear, 以年为单位
112. trade_model = []
113. for i in modelyear.index:
114.    trade_model.append(tradeTime[i].year-modelyear[i].year)
115.
116. #计算 tradeTime-feature13,以月为单位
117. trade_f13 = []
118. for i in feature13.index:
119.    trade_f13.append(tradeTime[i].year*12+tradeTime[i].month-featu
        re13[i].year*12-feature13[i].month)
120.
121. # #加入
122. df.insert(df.shape[1],'tradeTime-registerDate',delta_weeks_reg)
123. df.insert(df.shape[1],'tradeTime-licenseDate',delta_weeks_lic)
124. df.insert(df.shape[1],'tradeTime-modelyear',trade_model)
125. df.insert(df.shape[1],'tradeTime-feature13',trade_f13)
126. #删除原列
127. df = df.drop(df[["tradeTime","registerDate","licenseDate","feature
        13","modelyear"]],axis=1)

```

```

128. df.head()
129. # 修改数据类型
130. df["tradeTime-registerDate"] = df["tradeTime-registerDate"].astype(
    ('float64')
131. df["tradeTime-licenseDate"] = df["tradeTime-licenseDate"].astype('
    float64')
132. # 删除 car_id 列
133. df = df.drop(df[["carid"]],axis=1)
134.
135. #删除 gearbox 缺失数据: 只有一条
136.
137. df = df.dropna(subset=["gearbox"],axis=0)
138.
139. #price 插到最后
140. price = df['price']
141. df.drop(labels=['price'], axis=1,inplace=True)
142. df.insert(33, 'price', price)
143.
144. #划分训练集和测试集
145. from sklearn.model_selection import train_test_split
146.
147. lst = [i for i in range(0,33)]
148.
149. X=df.iloc[:,lst]
150. y=df.loc[:, "price"]
151.
152. X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
153.
154. print("X_train.shape:",X_train.shape)
155. print("X_test.shape:",X_test.shape)
156. print("y_train.shape:",y_train.shape)
157. print("y_test.shape:",y_test.shape)
158. # 缺失变量:
159. null_lst1 = ['carCode', 'country', 'maketype', 'feature1', 'feature
    8', 'feature9', 'feature10', 'feature11']
160. null_lst2 = [ 'tradeTime-modelyear', 'tradeTime-feature13']
161.
162.
163. # 用随机森林分类树填充缺失值
164.
165. from sklearn.ensemble import RandomForestClassifier
166. def set_missing_classier_train(df, feature):
167.     df = df[['brand', 'serial', 'model', 'gearbox',

```

```

168.         'mileage', 'color', 'cityId', 'transferCount',
169.         'seatings', 'displacement', 'oiltype', 'newprice',
170.         'feature2', 'feature3', 'feature5', 'feature6',
171.         'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
172.         'feature12-length', 'feature12-width', 'feature12-height', 'feature12-volume', feature]]
173.
174.     known = df[df[feature].notnull()].values
175.     unknown = df[df[feature].isnull()].values
176.
177.     y = known[:, -1]
178.     X = known[:, :-1]
179.
180.     rfc = RandomForestClassifier(random_state=0, n_estimators=100, n
        _jobs=-1)
181.
182.     rfc.fit(X, y)
183.
184.     predict = rfc.predict(unknown[:, :-1])
185.
186.     df.loc[(df[feature].isnull()), feature] = predict
187.
188.     #return rfr 是为后续预测训练集 feature 缺失数据
189.     return df, rfc
190.
191. from sklearn.ensemble import RandomForestClassifier
192. def set_missing_classifier_test(rfc, df, feature):
193.     df = df[['brand', 'serial', 'model', 'gearbox',
194.         'mileage', 'color', 'cityId', 'transferCount',
195.         'seatings', 'displacement', 'oiltype', 'newprice',
196.         'feature2', 'feature3', 'feature5', 'feature6',
197.         'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
198.         'feature12-length', 'feature12-width', 'feature12-height', 'feature12-volume', feature]]
199.
200.     known = df[df[feature].notnull()].values
201.     unknown = df[df[feature].isnull()].values
202.
203.     y = known[:, -1]
204.     X = known[:, :-1]
205.
206.     # rfc.fit(X, y)

```

```

207.
208.     predict = rfc.predict(unknown[:, :-1])
209.
210.     df.loc[(df[feature].isnull()), feature] = predict
211.
212.     #return rfr 是为后续预测训练集 feature 缺失数据
213.     return df
214. rfc_lst = []
215. for i in null_lst1:
216.     train_result, rfc = set_missing_classier_train(X_train, i)
217.     test_result = set_missing_classier_test(rfc, X_test, i)
218.     X_train[i] = train_result[i]
219.     X_test[i] = test_result[i]
220.     rfc_lst.append(rfc)
221.     # print(i+"已完成! ")
222.     # 用随机森林回归树填充缺失值
223.
224. from sklearn.ensemble import RandomForestRegressor
225. def set_missing_regress_train(df, feature):
226.     df = df[['brand', 'serial', 'model', 'gearbox',
227.             'mileage', 'color', 'cityId', 'transferCount',
228.             'seatings', 'displacement', 'oiltype', 'newprice',
229.             'feature2', 'feature3', 'feature5', 'feature6',
230.             'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
231.             'feature12-length', 'feature12-width', 'feature12-height', 'feature12-volume', feature]]
232.
233.     known = df[df[feature].notnull()].values
234.     unknown = df[df[feature].isnull()].values
235.
236.     y = known[:, -1]
237.     X = known[:, :-1]
238.
239.     rfr = RandomForestRegressor(random_state=0, n_estimators=100, n_
        jobs=-1)
240.
241.     rfr.fit(X, y)
242.
243.     predict = rfr.predict(unknown[:, :-1])
244.
245.     df.loc[(df[feature].isnull()), feature] = predict
246.
247.     #return rfr 是为后续预测训练集 feature 缺失数据

```



```

248.         return df,rfr
249.
250.
251.     # 用随机森林回归树填充缺失值
252.
253.     from sklearn.ensemble import RandomForestRegressor
254.     def set_missing_regress_test(rfr, df, feature):
255.         df = df[['brand','serial','model','gearbox',
256.                 'mileage','color','cityId','transferCount',
257.                 'seatings','displacement','oiltype','newprice',
258.                 'feature2','feature3', 'feature5','feature6',
259.                 'feature14','tradeTime-registerDate','tradeTime-licenseDate',
260.                 'feature12-length','feature12-width','feature12-height','feature12-volume', feature]]
261.
262.         known = df[df[feature].notnull()].values
263.         unknown = df[df[feature].isnull()].values
264.
265.         y = known[:,-1]
266.         X = known[:, :-1]
267.
268.         #     rfr.fit(X,y)
269.
270.         predict = rfr.predict(unknown[:, :-1])
271.         df.loc[(df[feature].isnull()),feature] = predict
272.         #return rfr 是为后续预测训练集 feature 缺失数据
273.         return df
274.     rfr_lst = []
275.     for i in null_lst2:
276.         train_result, rfr = set_missing_regress_train(X_train, i)
277.         test_result = set_missing_regress_test(rfr, X_test, i)
278.         X_train[i] = train_result[i]
279.         X_test[i] = test_result[i]
280.         rfr_lst.append(rfr)
281.     #     print(i+"已完成! ")
282.     # # 经观察，不同类别数据的分布并不均匀，因此尝试提取 frequency 作为非数值
        型变量的表示
283.     # # 对于 brand、serial、model: brand 用每个品牌占总数的频率，serial 用每个
        serial 占该 brand 所有汽车的占比，model 用每个 model 占该 serial 的占比
284.     # 下面的 brands、serials、models 这三个变量可以作为固定的数据使用
285.     serials = X_train['serial'].value_counts()
286.     brands = X_train['brand'].value_counts()
287.     models = X_train['model'].value_counts()

```

```

288. #对训练集:
289. X_train_1 = X_train.copy()
290. #转变数据类型
291. X_train_1['serial'] = X_train_1['serial']
292. X_train_1['model'] = X_train_1['model']
293. X_train_1['brand'] = X_train_1['brand']
294. for i in X_train_1.index:
295.     modeli = X_train_1.loc[i, 'model']
296.     seriali = X_train_1.loc[i, 'serial']
297.     brandi = X_train_1.loc[i, 'brand']
298.     X_train_1.loc[i, 'model'] = models[modeli]/serials[seriali]
299.     X_train_1.loc[i, 'serial'] = serials[seriali]/brands[brandi]
300.
301. #对测试集:
302. X_test_1 = X_test.copy()
303. #转变数据类型
304. X_test_1['serial'] = X_test_1['serial']
305. X_test_1['model'] = X_test_1['model']
306. X_test_1['brand'] = X_test_1['brand']
307. for i in X_test_1.index:
308.     modeli = X_test_1.loc[i, 'model']
309.     seriali = X_test_1.loc[i, 'serial']
310.     brandi = X_test_1.loc[i, 'brand']
311.     #处理未知值
312.     if modeli not in models:
313.         models = models.append(X_test_1.loc[X_test_1['model']==modeli, 'model'].value_counts())
314.     if seriali not in serials:
315.         serials = serials.append(X_test_1.loc[X_test_1['serial']==seriali, 'serial'].value_counts())
316.     if brandi not in brands:
317.         brands = brands.append(X_test_1.loc[X_test_1['brand']==brandi, 'brand'].value_counts())
318.     X_test_1.loc[i, 'model'] = models[modeli]/serials[seriali]
319.     X_test_1.loc[i, 'serial'] = serials[seriali]/brands[brandi]
320.
321. #frequency 编码: 首先安装编码库 pip install category-encoders
322. from category_encoders import CountEncoder
323. #选出其他非数值列
324. notnum_columns = ['brand', 'color', 'cityId', 'carCode', 'country',
325.                    'maketype', 'gearbox', 'oiltype', 'feature
326.                    1', 'feature2', 'feature3', 'feature5',
327.                    'feature6', 'feature8', 'feature9', 'feature
328.                    e10', 'feature11', 'feature14']

```

```

327. #定义编码器：传入要编码的列
328. count_encoder = CountEncoder(cols=notnum_columns).fit(X_train_1)
329. #分别对测试集和训练集编码：无需再合并到数据集或删除列了
330. X_train_code = count_encoder.transform(X_train_1)
331. X_test_code = count_encoder.transform(X_test_1)
332.
333. X_train_code.head(3)
334. #都不算小
335. X_train_code.var()
336. from sklearn.model_selection import train_test_split
337. from sklearn.ensemble import RandomForestClassifier
338. from sklearn.model_selection import cross_val_score
339.
340. forest = RandomForestRegressor(n_estimators=100, random_state=0, n
    _jobs=-1)
341. forest.fit(X_train, y_train)
342. scores = []
343. n_estimators = [20,50,100,200,300,400,500]
344.
345. for n in n_estimators:
346.     forest = RandomForestRegressor(n_estimators=n, random_state=0,
        n_jobs=-1)
347.     score = cross_val_score(forest,X_train,y_train,cv=5)
348.     scores.append(score)
349.
350. plt.plot(scores)
351. plt.title('Cross Validation scores')
352. plt.xlabel('n_estimators')
353. plt.ylabel('score')
354. plt.legend(n_estimators)
355.
356. plt.show()
357. from sklearn.model_selection import train_test_split
358. from sklearn.ensemble import RandomForestClassifier
359.
360. importance_list = []
361. forest = RandomForestRegressor(n_estimators=100, random_state=0, n
    _jobs=-1)
362. forest.fit(X_train, y_train)
363.
364. importances = forest.feature_importances_
365. indices = np.argsort(importances)[::-1]
366. feat_labels = X_train.columns[:]
367. for f in range(X_train.shape[1]):

```

```

368.         print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], im
            portances[indices[f]]))
369.
370.     importance_list.append(importances)
371.     # 归一化处理: 选择性运行这一个代码块, 如果前面已经归一化过了, 就不需要在运
        行一遍了
372.     from sklearn import preprocessing
373.
374.     min_max_scaler = preprocessing.MinMaxScaler().fit(X_train_code)
375.     X_train_sc = min_max_scaler.transform(X_train_code)
376.     X_test_sc = min_max_scaler.transform(X_test_code)
377.
378.     #充分性检验: 确认是否可以做因子分析
379.     from factor_analyzer.factor_analyzer import calculate_kmo
380.     from factor_analyzer.factor_analyzer import calculate_bartlett_sph
        ericity
381.     chi_square_value, p_value = calculate_bartlett_sphericity(X_train_
        sc)
382.     print('巴特利特值: ', p_value, '小于 0.1, 则可以选择这种方法')
383.     kmo_all, kmo_model = calculate_kmo(X_train_sc)
384.     print("kmo 值: ", kmo_model, '大于 0.6, 则可以选择这种方法')
385.     #选择合适的维数
386.     from factor_analyzer import FactorAnalyzer
387.     fa = FactorAnalyzer(X_train_code.shape[1]+1, rotation=None)
388.     fa.fit(X_train_code)
389.     ev, v = fa.get_eigenvalues() # 计算特征值和特征向量
390.     var = fa.get_factor_variance()#给出方差贡献率
391.     #绘图展示
392.     fig, axes = plt.subplots(1, 2, figsize=(14, 9))
393.     axes[0].plot(ev, 'ob-')
394.     axes[0].set_title('eigenValue')
395.     axes[1].plot(var[2], 'ob-')
396.     axes[1].set_title('Cumlative Variance')
397.
398.     plt.show()
399.     #根据上述结果, 选择降到 9 维
400.     from factor_analyzer import FactorAnalyzer
401.     fa = FactorAnalyzer(n_factors=9, rotation='varimax')
402.     fa.fit(X_train_sc)
403.     X_train_fa = fa.transform(X_train_sc)
404.     X_test_fa = fa.transform(X_test_sc)
405.
406.     #可视化展示因子变换的矩阵, 如果不需要, 可以注释掉
407.     #图中 X 轴 1-9 是 9 个因子, 图中数字是做变换的矩阵的数字

```

```

408. import seaborn as sns
409. df_cm = pd.DataFrame(np.abs(fa.loadings_),index=X_train_code.columns
    ns)
410.
411. fig,ax = plt.subplots(figsize=(12,10))
412. sns.heatmap(df_cm,annot=True,cmap='BuPu',ax=ax)
413. # 设置y轴字体的大小
414. ax.tick_params(axis='x',labelsize=15)
415. ax.set_title("Factor Analysis",fontsize=12)
416. ax.set_ylabel("Variable")
417. # 归一化处理：选择性运行这一个代码块，如果前面已经归一化过了，就不需要在运行一遍了
418.
419. from sklearn import preprocessing
420.
421. min_max_scaler = preprocessing.MinMaxScaler().fit(X_train_code)
422. X_train_sc = min_max_scaler.transform(X_train_code)
423. X_test_sc = min_max_scaler.transform(X_test_code)
424. from sklearn.decomposition import PCA
425.
426. pca = PCA(n_components='mle').fit(X_train_sc)
427. X_train_pca = pca.transform(X_train_sc)
428. X_test_pca = pca.transform(X_test_sc)
429.
430. pd.DataFrame(X_test_pca).describe()
431. # 归一化处理：选择性运行这一个代码块，如果前面已经归一化过了，就不需要在运行一遍了
432.
433. from sklearn import preprocessing
434.
435. min_max_scaler = preprocessing.MinMaxScaler().fit(X_train_code)
436. X_train_sc = min_max_scaler.transform(X_train_code)
437. X_test_sc = min_max_scaler.transform(X_test_code)
438. # 损失函数：MSE
439. def mse_loss(y_pred, y_test):
440.     y_pred = list(y_pred)
441.     y_test = list(y_test)
442.
443.     mse = 0
444.     for i in range(len(y_pred)):
445.         mse += (y_pred[i]-y_test[i])**2
446.     mse /= len(y_pred)
447.
448.     return mse

```

```

449.
450.
451. # 损失函数: Accuracy = count(Ape 0.05) / count(total)
452. def accuracy(y_pred, y_test):
453.     y_pred = list(y_pred)
454.     y_test = list(y_test)
455.
456.     ape = 0
457.     count = 0
458.     for i in range(len(y_pred)):
459.         ape += abs(y_pred[i]-y_test[i])/y_test[i]
460.         if ape<=0.05:
461.             count += 1
462.
463.     mape = ape/len(y_pred)
464.     accuracy = count/len(y_pred)
465.     score = 0.2*(1-mape)+0.8*accuracy
466.     return mape, accuracy, score
467. from sklearn import linear_model
468. from sklearn.model_selection import cross_val_score
469.
470. lreg = linear_model.LinearRegression()
471. lreg.fit(X_train_sc, y_train)
472.
473. # 验证集
474. loss1 = -cross_val_score(lreg, X_train_sc, y_train, cv = 10, scoring='neg_mean_squared_error').mean()
475.
476. # 训练集
477. y_pred = lreg.predict(X_test_sc)
478. mse1 = mse_loss(y_pred, y_test)
479. mape1, accuracy1, score1 = accuracy(y_pred, y_test)
480.
481. print("线性回归:countEncoder")
482. print("训练集 mse:", loss1)
483. print("测试集 mse:", mse1)
484. print("测试集 mape:", mape1)
485. print("测试集 accur:", accuracy1)
486. print("测试集 score:",score1)
487. from sklearn import linear_model
488. from sklearn.model_selection import cross_val_score
489.
490. lreg = linear_model.LinearRegression()
491. lreg.fit(X_train_fa, y_train)

```

```

492.
493. # 验证集
494. loss1 = -cross_val_score(lireg, X_train_fa, y_train, cv = 10, scor
    ing='neg_mean_squared_error').mean()
495.
496. # 训练集
497. y_pred = lireg.predict(X_test_fa)
498. mse1 = mse_loss(y_pred, y_test)
499. mape1, accuracy1, score1 = accuracy(y_pred, y_test)
500.
501. print("线性回归:因子分析")
502. print("训练集 mse:", loss1)
503. print("测试集 mse:", mse1)
504. print("测试集 mape:", mape1)
505. print("测试集 accur:", accuracy1)
506. print("测试集 score:", score1)#交叉验证
507. from sklearn import tree
508. from sklearn.model_selection import cross_val_score
509.
510. scores = []
511. max_depths = [18,19,20,21,22]
512.
513. for n in max_depths:
514.     clf = tree.DecisionTreeRegressor(max_depth=n,random_state=0)
515.     score = -cross_val_score(clf,X_train_sc,y_train,cv=5,scoring='
        neg_mean_squared_error')
516.     scores.append(score)
517.
518. for i in scores:
519.     plt.plot(i)
520. plt.title('Cross Validation scores')
521. plt.xlabel('n_estimators')
522. plt.ylabel('score')
523. plt.legend(max_depths)
524.
525. plt.show()
526.
527. from sklearn import tree
528.
529. clf = tree.DecisionTreeRegressor(max_depth=20,random_state=0)
530. clf = clf.fit(X_train_sc, y_train)
531.
532. # 验证集

```

```

533. loss3 = -cross_val_score(clf, X_train_sc, y_train, cv = 10, scoring='neg_mean_squared_error').mean()
534.
535. # 训练集
536. clf = clf.fit(X_train_sc, y_train)
537. y_pred = clf.predict(X_test_sc)
538. mse3 = mse_loss(y_pred, y_test)
539. mape3, accuracy3, score3 = accuracy(y_pred, y_test)
540.
541. print("回归树:Countencoder")
542. print("训练集 mse:", loss3)
543. print("测试集 mse:", mse3)
544. print("测试集 mape:", mape3)
545. print("测试集 accur:", accuracy3)
546. print("测试集 score:", score3)
547. # 交叉验证
548. from sklearn import tree
549. from sklearn.model_selection import cross_val_score
550.
551. scores = []
552. max_depths = [20,25,28,30]
553.
554. for n in max_depths:
555.     clf = tree.DecisionTreeRegressor(max_depth=n,random_state=0)
556.     score = -cross_val_score(clf,X_train_fa,y_train,cv=5,scoring='neg_mean_squared_error')
557.     scores.append(score)
558.
559. for i in scores:
560.     plt.plot(i)
561. plt.title('Cross Validation scores')
562. plt.xlabel('n_estimators')
563. plt.ylabel('score')
564. plt.legend(max_depths)
565.
566. plt.show()
567. from sklearn import tree
568.
569. clf = tree.DecisionTreeRegressor(max_depth=25,random_state=0)
570. clf = clf.fit(X_train_fa, y_train)
571.
572. # 验证集
573. loss3 = -cross_val_score(clf, X_train_fa, y_train, cv = 10, scoring='neg_mean_squared_error').mean()

```



```

574.
575. # 训练集
576. clf = clf.fit(X_train_fa, y_train)
577. y_pred = clf.predict(X_test_fa)
578. mse3 = mse_loss(y_pred, y_test)
579. mape3, accuracy3, score3 = accuracy(y_pred, y_test)
580.
581. print("回归树:Countencoder")
582. print("训练集 mse:", loss3)
583. print("测试集 mse:", mse3)
584. print("测试集 mape:", mape3)
585. print("测试集 accur:", accuracy3)
586. print("测试集 score:", score3)
587. #交叉验证
588. from sklearn.ensemble import RandomForestRegressor
589.
590. scores = []
591. n_estimators = [50,100,200,300,400]
592.
593. for n in n_estimators:
594.     forest = RandomForestRegressor(n_estimators=n,random_state=0)
595.     score = -cross_val_score(forest,X_train_sc,y_train,cv=5,scoring='neg_mean_squared_error')
596.     scores.append(score)
597.
598. print(scores)
599. for i in scores:
600.     plt.plot(i)
601. plt.title('Cross Validation scores')
602. plt.xlabel('n_estimators')
603. plt.ylabel('score')
604. plt.legend(n_estimators)
605.
606. plt.show()
607. from sklearn.ensemble import RandomForestRegressor
608.
609. rfr = RandomForestRegressor(n_estimators=800, random_state=0)
610. rfr.fit(X_train_sc, y_train)
611.
612. # 验证集
613. loss4 = -cross_val_score(rfr, X_train_sc, y_train, cv = 10, scoring='neg_mean_squared_error').mean()
614.

```

```

615. # 训练集
616. y_pred = rfr.predict(X_test_sc)
617. mse4 = mse_loss(y_pred, y_test)
618. mape4, accuracy4, score4 = accuracy(y_pred, y_test)
619.
620. print("随机森林:CountEncoder")
621. print("训练集 mse:", loss4)
622. print("测试集 mse:", mse4)
623. print("测试集 mape:", mape4)
624. print("测试集 accur:", accuracy4)
625. print("测试集 score:", score4)
626. from sklearn.neural_network import MLPRegressor
627.
628. nn = MLPRegressor(hidden_layer_sizes=(25,25), activation='relu', r
        andom_state=1, max_iter=200)
629. loss6 = -cross_val_score(nn, X_train_sc, y_train, cv = 10, scoring
        ='neg_mean_squared_error').mean()
630. nn.fit(X_train_sc, y_train)
631.
632. # 训练集
633. y_pred = nn.predict(X_test_sc)
634. mse6 = mse_loss(y_pred, y_test)
635. mape6, accuracy6, score6 = accuracy(y_pred, y_test)
636.
637. print("神经网络:Countencoder")
638. print("训练集 mse:", loss6)
639. print("测试集 mse:", mse6)
640. print("测试集 mape:", mape6)
641. print("测试集 accur:", accuracy6)
642. print("测试集 score:", score6)
643. from sklearn.neural_network import MLPRegressor
644.
645. nn = MLPRegressor(hidden_layer_sizes=(25,25), activation='relu', r
        andom_state=1, max_iter=200)
646. loss6 = -cross_val_score(nn, X_train_fa, y_train, cv = 10, scoring
        ='neg_mean_squared_error').mean()
647. nn.fit(X_train_fa, y_train)
648.
649. # 训练集
650. y_pred = nn.predict(X_test_fa)
651. mse6 = mse_loss(y_pred, y_test)
652. mape6, accuracy6, score6 = accuracy(y_pred, y_test)
653.
654. print("神经网络:因子分析")

```

```

655. print("训练集 mse:", loss6)
656. print("测试集 mse:", mse6)
657. print("测试集 mape:", mape6)
658. print("测试集 accur:", accuracy6)
659. print("测试集 score:", score6)
660. import lightgbm as lgbm
661. from sklearn import metrics
662. from sklearn import model_selection
663. from sklearn.model_selection import cross_val_score
664.
665. def lightgbm(X_train, y_train, X_test):
666.     lgbm_reg = lgbm.LGBMRegressor(objective='regression')
667.     lgbm_reg.fit(X_train, y_train)
668.
669.     loss7 = -cross_val_score(lgbm_reg, X_train, y_train, cv = 5, scoring='neg_mean_squared_error').mean()
670.
671.     y_pred = lgbm_reg.predict(X_test)
672.     mse7 = mse_loss(y_pred, y_test)
673.     mape7, accuracy7, score7 = accuracy(y_pred, y_test)
674.
675.     return loss7, mse7, mape7, accuracy7, score7
676. loss7, mse7, mape7, accuracy7, score7 = lightgbm(X_train_sc, y_train_sc, X_test_sc)
677. print("LightGBM: Countencoder")
678. print("训练集 mse:", loss7)
679. print("测试集 mse:", mse7)
680. print("测试集 mape:", mape7)
681. print("测试集 accur:", accuracy7)
682. print("测试集 score:", score7)
683. from sklearn.svm import SVR
684. from sklearn.model_selection import cross_val_score
685.
686. def svm_regression(X_train, y_train, X_test):
687.     svr = SVR(kernel='rbf')
688.     svr.fit(X_train, y_train)
689.
690.     # 验证集
691.     loss5 = -cross_val_score(svr, X_train, y_train, cv = 5, scoring='neg_mean_squared_error').mean()
692.
693.     # 训练集
694.     y_pred = svr.predict(X_test)
695.     mse5 = mse_loss(y_pred, y_test)

```

```

696.         mape5, accuracy5, score5 = accuracy(y_pred, y_test)
697.
698.         return loss5, mse5, mape5, accuracy5, score5
699.     from sklearn.model_selection import KFold
700.     from sklearn.base import BaseEstimator, RegressorMixin, Transform
        erMixin, clone
701.     #定义 stacking 类
702.     class StackingAveragedModels(BaseEstimator, RegressorMixin, Transf
        ormerMixin):
703.         #base_model 第一层模型：传入列表；meta_model 第二层模型：单个模型
704.         #n_folds=几折
705.         def __init__(self, base_models, meta_model, n_folds=5):
706.             self.base_models = base_models
707.             self.meta_model = meta_model
708.             self.n_folds = n_folds
709.
710.         # 将原来的模型 clone 出来，并且进行实现 fit 功能
711.         def fit(self, X, y):
712.             self.base_models_ = [list() for x in self.base_models]
713.             self.meta_model_ = clone(self.meta_model)
714.             kfold = KFold(n_splits=self.n_folds, shuffle=True, random_
                state=156)
715.
716.             #对于每个模型，使用交叉验证的方法来训练初级学习器，并且得到次级训
                练集
717.             out_of_fold_predictions = np.zeros((X.shape[0], len(self.b
                ase_models)))
718.             for i, model in enumerate(self.base_models):
719.                 for train_index, holdout_index in kfold.split(X, y):
720.                     # self.base_models_[i].append(instance)
721.                     instance = clone(model)
722.                     instance.fit(X.loc[train_index,:], y.loc[train_ind
                        ex,:])
723.                     y_pred = instance.predict(X.loc[holdout_index,:])
724.
725.                     out_of_fold_predictions[holdout_index, i] = y_pred
726.                     .reshape(1,-1)
727.                     self.base_models_[i].append(instance)
728.
729.             # 使用次级训练集来训练次级学习器
730.             self.meta_model_.fit(out_of_fold_predictions, y)
731.             return self

```

```

731.         #在上面的 fit 方法当中，已经将我们训练出来的初级学习器和次级学习器保存
           下来了
732.         #predict 的时候只需要用这些学习器构造我们的次级预测数据集并且进行预测
           就可以了
733.         def predict(self, X):
734.             meta_features = np.column_stack([
735.                 np.column_stack([model.predict(X) for model in base_mo
                    dels]).mean(axis=1)
736.                 for base_models in self.base_models_ ])
737.             return self.meta_model_.predict(meta_features)
738.         #定义模型
739.         #建议第一层选择之前已经训练的效果较好的(因为咱们前面模型也没几个，其实都可
           以加上)，第二层学习器选择简单的线性回归
740.         from sklearn import tree
741.         from sklearn.ensemble import RandomForestRegressor
742.         from sklearn import linear_model
743.         import lightgbm as lgbm
744.
745.         clf2 = tree.DecisionTreeRegressor(max_depth=20,random_state=0)
746.         forest2 = RandomForestRegressor(n_estimators=200,random_state=0)
747.         lightgbm2 = lgbm.LGBMRegressor(objective='regression')
748.         lreg2 = linear_model.LinearRegression()
749.         base_models = [lreg2,clf2,forest2,lightgbm2]
750.         meta_model = linear_model.LinearRegression()
751.
752.         stacking = StackingAveragedModels(base_models=base_models,meta_mod
                    el=meta_model)
753.
754.         #训练模型
755.         stacking.fit(pd.DataFrame(X_train_sc),pd.DataFrame(y_train.values)
                    )
756.
757.         #测试
758.         y_pred = stacking.predict(X_test_sc)
759.         mse8 = mse_loss(y_pred, y_test)
760.         mape8, accuracy8, score8 = accuracy(y_pred, y_test)
761.
762.         print("stacking:Countencoder")
763.         # print("训练集 mse:", loss8)
764.         print("测试集 mse:", mse8)
765.         print("测试集 mape:", mape8)
766.         print("测试集 accur:", accuracy8)
767.         print("测试集 score:", score8)

```

```

768. df_val = pd.read_csv('附件/附件 2: 估价验证数
    据.csv',engine='python',header=None,dtype=object)
769. # 列命名
770. col = ['carid','tradeTime','brand','serial','model',
771.         'mileage','color','cityId','carCode','transferCount',
772.         'seatings','registerDate','licenseDate','country',
773.         'maketype','modelyear','displacement','gearbox',
774.         'oiltype','newprice','feature1','feature2','feature3',
775.         'feature4','feature5','feature6','feature7','feature8',
776.         'feature9','feature10','feature11','feature12',
777.         'feature13','feature14','feature15']
778. df_val.columns = col
779. #类型修改
780. # df['mileage'] = df['mileage'].astype('float64')
781. df_val['transferCount'] = df_val['transferCount'].astype('float64'
    )
782. df_val['seatings'] = df_val['seatings'].astype('float64')
783. df_val['displacement'] = df_val['displacement'].astype('float64')

784. df_val['newprice'] = df_val['newprice'].astype('float64')
785. # df_val['price'] = df_val['price'].astype('float64')
786. df_val=df_val.drop(df_val[["feature4","feature7","feature15"]], ax
    is=1)
787. df_val.loc[df_val['feature11']=='1+2', 'feature11'] = '1'
788. df_val.loc[df_val['feature11']=='1', 'feature11'] = '2'
789. df_val.loc[df_val['feature11']=='3+2', 'feature11'] = '3'
790. df_val.loc[df_val['feature11']=='1+2,4+2', 'feature11'] = '4'
791. df_val.loc[df_val['feature11']=='5', 'feature11'] = '5'
792. df_val.loc[df_val['feature11']=='1,3+2', 'feature11'] = '6'
793.
794. df_val['feature11'] = df_val['feature11'].astype(object)
795. # 拆 2: feature12
796.
797. str_slice = pd.Series(df_val["feature12"])
798. info = [s for s in str_slice.str.split()]
799. length = []
800. width = []
801. height = []
802. volume = []
803.
804. for i in info:
805.     sub_info = i[0].split('*')
806.     length.append(int(sub_info[0]))
807.     width.append(int(sub_info[1]))

```

```

808.         height.append(int(sub_info[2]))
809.         volume.append(int(sub_info[0])*int(sub_info[1])*int(sub_info[2]
    )))
810.
811.     df_val.insert(df_val.shape[1], 'feature12-length', length)
812.     df_val.insert(df_val.shape[1], 'feature12-width', width)
813.     df_val.insert(df_val.shape[1], 'feature12-height', height)
814.     df_val.insert(df_val.shape[1], 'feature12-volume', volume)
815.
816.     df_val['feature12-length'] = df_val['feature12-length'].astype('float64')
817.     df_val['feature12-width'] = df_val['feature12-width'].astype('float64')
818.     df_val['feature12-height'] = df_val['feature12-height'].astype('float64')
819.     df_val['feature12-volume'] = df_val['feature12-volume'].astype('float64')
820.
821.     # 删除原列
822.     df_val=df_val.drop(df_val[["feature12"]], axis=1)
823.     #将字符串转为日期类型,分别记为变量 tradeTime、
824.     tradeTime = pd.to_datetime(df_val['tradeTime'])
825.     registerDate = pd.to_datetime(df_val['registerDate'])
826.     licenseDate = pd.to_datetime(df_val['licenseDate'])
827.
828.     #计算 registerDate-tradeTime, licenseDate-tradeTime
829.     delta_reg = tradeTime-registerDate
830.     delta_lic = tradeTime-licenseDate
831.
832.     #以月为单位表示时间差
833.     delta_weeks_reg = []
834.     for delta in delta_reg:
835.         if delta.days % 7 <= 3:
836.             weeks = delta.days//7
837.         else:
838.             weeks = delta.days//7 + 1
839.         delta_weeks_reg.append(weeks)
840.     delta_weeks_lic = []
841.     for delta in delta_lic:
842.         if delta.days % 7 <= 3:
843.             weeks = delta.days//7
844.         else:
845.             weeks = delta.days//7 + 1
846.         delta_weeks_lic.append(weeks)

```

```

847.
848.
849. #转为 datetime 类型
850. feature13 = pd.to_datetime(df_val['feature13'],format='%Y%m')
851. modelyear = pd.to_datetime(df_val['modelyear'])
852.
853. #计算 tradeTime-modelyear, 以年为单位
854. trade_model = []
855. for i in modelyear.index:
856.     trade_model.append(tradeTime[i].year-modelyear[i].year)
857.
858. #计算 tradeTime-feature13,以月为单位
859. trade_f13 = []
860. for i in feature13.index:
861.     trade_f13.append(tradeTime[i].year*12+tradeTime[i].month-feature13[i].year*12-feature13[i].month)
862.
863. # #加入
864. df_val.insert(df_val.shape[1],'tradeTime-registerDate',delta_weeks_reg)
865. df_val.insert(df_val.shape[1],'tradeTime-licenseDate',delta_weeks_lic)
866. df_val.insert(df_val.shape[1],'tradeTime-modelyear',trade_model)
867. df_val.insert(df_val.shape[1],'tradeTime-feature13',trade_f13)
868. #删除原列
869. df_val = df_val.drop(df_val[["tradeTime","registerDate","licenseDate","feature13","modelyear"]],axis=1)
870. df_val.head()
871. # 修改数据类型
872. df_val["tradeTime-registerDate"] = df_val["tradeTime-registerDate"].astype('float64')
873. df_val["tradeTime-licenseDate"] = df_val["tradeTime-licenseDate"].astype('float64')
874.
875. df_val.head(3)
876. # 缺失变量: feature11 在 null_lst1 中会报错
877. null_lst1 = ['country', 'maketype', 'feature1', 'feature8', 'feature9', 'feature10', 'feature11']
878. null_lst2 = ['tradeTime-modelyear', 'tradeTime-feature13']
879. from sklearn.ensemble import RandomForestClassifier
880. def set_missing_classier_valid(rfc, df, feature):
881.     df = df[['brand', 'serial', 'model', 'gearbox',
882.             'mileage', 'color', 'cityId', 'transferCount',
883.             'seatings', 'displacement', 'oiltype', 'newprice',

```



```

884.         'feature2', 'feature3', 'feature5', 'feature6',
885.         'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDat
           e',
886.         'feature12-length', 'feature12-width', 'feature12-height', 'f
           eature12-volume', feature]]
887.
888.     known = df[df[feature].notnull()].values
889.     unknown = df[df[feature].isnull()].values
890.
891.     y = known[:, -1]
892.     X = known[:, :-1]
893.
894.     #     rfc.fit(X,y)
895.
896.     predict = rfc.predict(unknown[:, :-1])
897.
898.     df.loc[(df[feature].isnull()), feature] = predict
899.
900.     #return rfr 是为后续预测训练集 feature 缺失数据
901.     return df
902.     for n in range(len(null_lst1)):
903.         i = null_lst1[n]
904.         rfc = rfc_lst[n]
905.         valid_result = set_missing_classier_valid(rfc, df_val, i)
906.         df_val[i] = valid_result[i]
907.         print(i+'已完成')
908.     # 用随机森林回归树填充缺失值
909.
910.     from sklearn.ensemble import RandomForestRegressor
911.     def set_missing_regress_valid(rfr, df, feature):
912.         df = df[['brand', 'serial', 'model', 'gearbox',
913.                  'mileage', 'color', 'cityId', 'transferCount',
914.                  'seatings', 'displacement', 'oiltype', 'newprice',
915.                  'feature2', 'feature3', 'feature5', 'feature6',
916.                  'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDat
           e',
917.                  'feature12-length', 'feature12-width', 'feature12-height', 'f
           eature12-volume', feature]]
918.
919.         known = df[df[feature].notnull()].values
920.         unknown = df[df[feature].isnull()].values
921.
922.         y = known[:, -1]
923.         X = known[:, :-1]

```

```

924.
925. #     rfr.fit(X,y)
926.
927.     predict = rfr.predict(unknown[:, :-1])
928.
929.     df.loc[(df[feature].isnull()), feature] = predict
930.
931.     #return rfr 是为后续预测训练集 feature 缺失数据
932.     return df
933. for n in range(len(null_lst2)):
934.     i = null_lst2[n]
935.     rfr = rfr_lst[n]
936.     valid_result = set_missing_regress_valid(rfr, df_val, i)
937.     df_val[i] = valid_result[i]
938.     print(i+'已完成')
939. #对验证集:
940. X_val_1 = df_val.copy()
941. for i in X_val_1.index:
942.     modeli = X_val_1.loc[i, 'model']
943.     seriali = X_val_1.loc[i, 'serial']
944.     brandi = X_val_1.loc[i, 'brand']
945.     #处理未知值
946.     if modeli not in models:
947.         models = models.append(X_val_1.loc[X_val_1['model']==modeli, 'model'].value_counts())
948.     if seriali not in serials:
949.         serials = serials.append(X_val_1.loc[X_val_1['serial']==seriali, 'serial'].value_counts())
950.     if brandi not in brands:
951.         brands = brands.append(X_val_1.loc[X_val_1['brand']==brandi, 'brand'].value_counts())
952.     X_val_1.loc[i, 'model'] = models[modeli]/serials[seriali]
953.     X_val_1.loc[i, 'serial'] = serials[seriali]/brands[brandi]
954.
955. #frequency 编码: 首先安装编码库 pip install category-encoders
956. from category_encoders import CountEncoder
957. #选出其他非数值列
958. notnum_columns = ['brand', 'color', 'cityId', 'carCode', 'country',
959.                    'maketype', 'gearbox', 'oiltype', 'feature
960.                    1', 'feature2', 'feature3', 'feature5',
961.                    'feature6', 'feature8', 'feature9', 'feature
962.                    e10', 'feature11', 'feature14']

```

```

963. X_val_code = count_encoder.transform(X_val_1)
964.
965. X_val_code.head(3)
966. X_val_code = min_max_scaler.transform(X_val_code)
967. y_val_pred = stacking.predict(X_val_code)
968.
969. carid = list(carid)
970. y_val_pred = list(y_val_pred)
971.
972. f = open("附件/附件 3: 估价模型结果.txt", "w")
973.
974. for i in range(len(carid)):
975.     f.write(carid[i]+' '+str(y_val_pred[i][0]) + '\n')
976.
977. f.close()
978. #随机森林模型-800
979. y_val_pred = rfr.predict(X_val_code)
980.
981. carid = list(carid)
982. y_val_pred = list(y_val_pred)
983.
984. f = open("附件/附件 3: 估价模型结果(800).txt", "w")
985.
986. for i in range(len(carid)):
987.     f.write(carid[i]+' '+str(y_val_pred[i]) + '\n')
988.
989. f.close()
990. import pandas as pd
991. import numpy as np
992. from sklearn.preprocessing import StandardScaler
993. from sklearn import preprocessing
994. from collections import Counter
995. import warnings; warnings.simplefilter('ignore')
996. import matplotlib.pyplot as plt
997. #之前对附件 1 的处理直接 copy 过来
998. df1 = pd.read_csv('附件 1: 估价训练数据.csv', engine='python', header=None, dtype=object)
999.
1000. col = ['carid', 'tradeTime', 'brand', 'serial', 'model',
1001.        'mileage', 'color', 'cityId', 'carCode', 'transferCount',
1002.        'seatings', 'registerDate', 'licenseDate', 'country',
1003.        'maketype', 'modelyear', 'displacement', 'gearbox',
1004.        'oiltype', 'newprice', 'feature1', 'feature2', 'feature3',
1005.        'feature4', 'feature5', 'feature6', 'feature7', 'feature8',

```

```

1006.         'feature9', 'feature10', 'feature11', 'feature12',
1007.         'feature13', 'feature14', 'feature15', 'price']
1008.     df1.columns = col
1009.
1010.
1011.     #类型修改
1012.     df1['mileage'] = df1['mileage'].astype('float64')
1013.     df1['transferCount'] = df1['transferCount'].astype('float64')
1014.     df1['seatings'] = df1['seatings'].astype('float64')
1015.     df1['displacement'] = df1['displacement'].astype('float64')
1016.     df1['newprice'] = df1['newprice'].astype('float64')
1017.     df1['price'] = df1['price'].astype('float64')
1018.
1019.     # 拆 1: feature11
1020.     df1.loc[df1['feature11']=='1+2', 'feature11'] = '1'
1021.     df1.loc[df1['feature11']=='1', 'feature11'] = '2'
1022.     df1.loc[df1['feature11']=='3+2', 'feature11'] = '3'
1023.     df1.loc[df1['feature11']=='1+2,4+2', 'feature11'] = '4'
1024.     df1.loc[df1['feature11']=='5', 'feature11'] = '5'
1025.     df1.loc[df1['feature11']=='1,3+2', 'feature11'] = '6'
1026.
1027.     df1['feature11'] = df1['feature11'].astype(object)
1028.
1029.
1030.     # 拆 2: feature12
1031.     str_slice = pd.Series(df1["feature12"])
1032.     info = [s for s in str_slice.str.split()]
1033.     length = []
1034.     width = []
1035.     height = []
1036.     volume = []
1037.
1038.     for i in info:
1039.         sub_info = i[0].split('*')
1040.         length.append(int(sub_info[0]))
1041.         width.append(int(sub_info[1]))
1042.         height.append(int(sub_info[2]))
1043.         volume.append(int(sub_info[0])*int(sub_info[1])*int(sub_info[2]))
1044.
1045.     df1.insert(df1.shape[1], 'feature12-length', length)
1046.     df1.insert(df1.shape[1], 'feature12-width', width)
1047.     df1.insert(df1.shape[1], 'feature12-height', height)
1048.     df1.insert(df1.shape[1], 'feature12-volume', volume)
1049.

```

```

1050. df1['feature12-length'] = df1['feature12-length'].astype('float64')
1051. df1['feature12-width'] = df1['feature12-width'].astype('float64')
1052. df1['feature12-height'] = df1['feature12-height'].astype('float64')
1053. df1['feature12-volume'] = df1['feature12-volume'].astype('float64')
1054.
1055. # 删除原列
1056. df1=df1.drop(df1[["feature12"]], axis=1)
1057.
1058.
1059. #将字符串转为日期类型,分别记为变量 tradeTime、
1060. tradeTime = pd.to_datetime(df1['tradeTime'])
1061. registerDate = pd.to_datetime(df1['registerDate'])
1062. licenseDate = pd.to_datetime(df1['licenseDate'])
1063.
1064. #计算 registerDate-tradeTime, licenseDate-tradeTime
1065. delta_reg = tradeTime-registerDate
1066. delta_lic = tradeTime-licenseDate
1067.
1068. #以月为单位表示时间差
1069. delta_weeks_reg = []
1070. for delta in delta_reg:
1071.     if delta.days % 7 <= 3:
1072.         weeks = delta.days//7
1073.     else:
1074.         weeks = delta.days//7 + 1
1075.     delta_weeks_reg.append(weeks)
1076. delta_weeks_lic = []
1077. for delta in delta_lic:
1078.     if delta.days % 7 <= 3:
1079.         weeks = delta.days//7
1080.     else:
1081.         weeks = delta.days//7 + 1
1082.     delta_weeks_lic.append(weeks)
1083.
1084.
1085. #转为 datetime 类型
1086. feature13 = pd.to_datetime(df1['feature13'],format='%Y%m')
1087. modelyear = pd.to_datetime(df1['modelyear'])
1088.
1089. #计算 tradeTime-modelyear, 以年为单位
1090. trade_model = []
1091. for i in range(modelyear.shape[0]):
1092.     trade_model.append(tradeTime[i].year-modelyear[i].year)
1093.

```

```

1094.     #计算 tradeTime-feature13,以月为单位
1095.     trade_f13 = []
1096.     for i in range(feature13.shape[0]):
1097.         trade_f13.append(tradeTime[i].year*12+tradeTime[i].month-feature13[
            i].year*12-feature13[i].month)
1098.
1099.
1100.     # #加入
1101.     df1.insert(df1.shape[1], 'tradeTime-registerDate', delta_weeks_reg)
1102.     df1.insert(df1.shape[1], 'tradeTime-licenseDate', delta_weeks_lic)
1103.     df1.insert(df1.shape[1], 'tradeTime-modelyear', trade_model)
1104.     df1.insert(df1.shape[1], 'tradeTime-feature13', trade_f13)
1105.     #删除原列
1106.     df1 = df1.drop(df1[["tradeTime", "registerDate", "licenseDate", "feature13
        ", "modelyear"]], axis=1)
1107.
1108.     # 修改数据类型
1109.     df1["tradeTime-registerDate"] = df1["tradeTime-registerDate"].astype('f
        loat64')
1110.     df1["tradeTime-licenseDate"] = df1["tradeTime-licenseDate"].astype('flo
        at64')
1111.
1112.     df1.head(3)
1113.     #第 23、26 和 34 列缺失值超过 50%，无法填补缺失值，删去这两列
1114.     df1=df1.drop(df1[["feature4", "feature7", "feature15"]], axis=1)
1115.
1116.     #对 df1 划分训练集和测试集
1117.
1118.     # 删除 gearbox 缺失数据：只有一条
1119.     df1 = df1.dropna(subset=["gearbox"], axis=0)
1120.
1121.     #price 插到最后
1122.     price = df1['price']
1123.     df1.drop(labels=['price'], axis=1, inplace=True)
1124.     df1.insert(34, 'price', price)
1125.
1126.     #划分训练集和测试集
1127.     from sklearn.model_selection import train_test_split
1128.
1129.     lst = [i for i in range(0,34)]
1130.
1131.     X=df1.iloc[:,lst]
1132.     y=df1.loc[:, "price"]
1133.

```

```

1134. X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
1135.
1136. print("X_train.shape:",X_train.shape)
1137. print("X_test.shape:",X_test.shape)
1138. print("y_train.shape:",y_train.shape)
1139. print("y_test.shape:",y_test.shape)
1140. # 缺失变量: feature11 在 null_lst1 中会报错
1141. null_lst1 = ['carCode','country', 'maketype', 'feature1', 'feature8', '
    feature9', 'feature10','feature11']
1142. null_lst2 = [ 'tradeTime-modelyear','tradeTime-feature13']
1143.
1144.
1145. # 用随机森林分类树填充缺失值
1146.
1147. from sklearn.ensemble import RandomForestClassifier
1148. def set_missing_classier_train(df, feature):
1149.     df = df[['brand','serial','model','gearbox',
1150.             'mileage','color','cityId','transferCount',
1151.             'seatings','displacement','oiltype','newprice',
1152.             'feature2','feature3', 'feature5','feature6',
1153.             'feature14','tradeTime-registerDate','tradeTime-licenseDate',
1154.             'feature12-length','feature12-width','feature12-height','featur
    e12-volume', feature]]
1155.
1156.     known = df[df[feature].notnull()].values
1157.     unknown = df[df[feature].isnull()].values
1158.
1159.     y = known[:, -1]
1160.     X = known[:, :-1]
1161.
1162.     rfc = RandomForestClassifier(random_state=0,n_estimators=100,n_jobs
    =-1)
1163.
1164.     rfc.fit(X,y)
1165.
1166.     predict = rfc.predict(unknown[:, :-1])
1167.
1168.     df.loc[(df[feature].isnull()),feature] = predict
1169.
1170.     #return rfr 是为后续预测训练集 feature 缺失数据
1171.     return df,rfc
1172.
1173. from sklearn.ensemble import RandomForestClassifier
1174. def set_missing_classier_test(rfc, df, feature):

```

```

1175.         df = df[['brand', 'serial', 'model', 'gearbox',
1176.                 'mileage', 'color', 'cityId', 'transferCount',
1177.                 'seatings', 'displacement', 'oiltype', 'newprice',
1178.                 'feature2', 'feature3', 'feature5', 'feature6',
1179.                 'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
1180.                 'feature12-length', 'feature12-width', 'feature12-height', 'feature12-volume', feature]]
1181.
1182.         known = df[df[feature].notnull()].values
1183.         unknown = df[df[feature].isnull()].values
1184.
1185.         y = known[:, -1]
1186.         X = known[:, :-1]
1187.
1188.         rfc.fit(X, y)
1189.
1190.         predict = rfc.predict(unknown[:, :-1])
1191.
1192.         df.loc[(df[feature].isnull()), feature] = predict
1193.
1194.         #return rfr 是为后续预测训练集 feature 缺失数据
1195.         return df
1196.
1197.     for i in null_lst1:
1198.         train_result, rfc = set_missing_classier_train(X_train, i)
1199.         test_result = set_missing_classier_test(rfc, X_test, i)
1200.         X_train[i] = train_result[i]
1201.         X_test[i] = test_result[i]
1202.         print(i+"已完成! ")
1203.
1204.     # 用随机森林回归树填充缺失值
1205.
1206.     from sklearn.ensemble import RandomForestRegressor
1207.     def set_missing_regress_train(df, feature):
1208.         df = df[['brand', 'serial', 'model', 'gearbox',
1209.                 'mileage', 'color', 'cityId', 'transferCount',
1210.                 'seatings', 'displacement', 'oiltype', 'newprice',
1211.                 'feature2', 'feature3', 'feature5', 'feature6',
1212.                 'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
1213.                 'feature12-length', 'feature12-width', 'feature12-height', 'feature12-volume', feature]]
1214.
1215.         known = df[df[feature].notnull()].values
1216.         unknown = df[df[feature].isnull()].values

```



```

1217.
1218.     y = known[:, -1]
1219.     X = known[:, :-1]
1220.
1221.     rfr = RandomForestRegressor(random_state=0, n_estimators=100, n_jobs=
-1)
1222.
1223.     rfr.fit(X, y)
1224.
1225.     predict = rfr.predict(unknown[:, :-1])
1226.
1227.     df.loc[(df[feature].isnull()), feature] = predict
1228.
1229.     #return rfr 是为后续预测训练集 feature 缺失数据
1230.     return df, rfr
1231.
1232.     # 用随机森林回归树填充缺失值
1233.
1234.     from sklearn.ensemble import RandomForestRegressor
1235.     def set_missing_regress_test(rfr, df, feature):
1236.         df = df[['brand', 'serial', 'model', 'gearbox',
1237.                 'mileage', 'color', 'cityId', 'transferCount',
1238.                 'seatings', 'displacement', 'oiltype', 'newprice',
1239.                 'feature2', 'feature3', 'feature5', 'feature6',
1240.                 'feature14', 'tradeTime-registerDate', 'tradeTime-licenseDate',
1241.                 'feature12-length', 'feature12-width', 'feature12-height', 'featur
e12-volume', feature]]
1242.
1243.         known = df[df[feature].notnull()].values
1244.         unknown = df[df[feature].isnull()].values
1245.
1246.         y = known[:, -1]
1247.         X = known[:, :-1]
1248.
1249.         rfr.fit(X, y)
1250.
1251.         predict = rfr.predict(unknown[:, :-1])
1252.
1253.         df.loc[(df[feature].isnull()), feature] = predict
1254.
1255.         #return rfr 是为后续预测训练集 feature 缺失数据
1256.         return df
1257.
1258.     for i in null_lst2:

```

```

1259.     train_result, rfr = set_missing_regress_train(X_train, i)
1260.     test_result = set_missing_regress_test(rfr, X_test, i)
1261.     X_train[i] = train_result[i]
1262.     X_test[i] = test_result[i]
1263.     print(i+"已完成! ")
1264.
1265.     df1 = pd.concat([X_train,X_test],axis=0)
1266.     pd.set_option('display.max_columns', None)
1267.
1268.     df4 = pd.read_csv('附件 4: 门店交易训练数据.csv',engine='python',header=None)
1269.     df4.columns = ["carid","pushDate","pushPrice","updatePriceTimeJson","pullDate","withdrawDate"]
1270.
1271.     df4['carid'] = df4['carid'].astype('str')
1272.
1273.     #转化成时间
1274.     df4['pushDate'] = pd.to_datetime(df4['pushDate'])
1275.     df4['pullDate'] = pd.to_datetime(df4['pullDate'])
1276.     df4['withdrawDate'] = pd.to_datetime(df4['withdrawDate'])
1277.     df4.head(3)
1278.     #成交了的车，下架时间=成交时间
1279.     #没成交的车，下架时间有，成交时间无
1280.     #成交周期=上架时间-成交时间，所以没成交的车的成交时间填 0 也不合适
1281.     df4_success=df4[~pd.isna(df4["withdrawDate"])]
1282.     df4_unsuccess=df4[pd.isna(df4["withdrawDate"])]
1283.
1284.     df=pd.merge(df4,df1,on="carid" ,how="left")
1285.     df = df.dropna(subset=["brand"],axis=0)
1286.
1287.     outputpath = 'E:/Fanny_Python_Files/MergeResult.csv'
1288.     (df).to_csv(outputpath,sep=',',index=False,header=True)
1289.
1290.     import pandas as pd
1291.     import numpy as np
1292.
1293.     df = pd.read_csv('第二问编码后的数据.csv',engine='python')
1294.
1295.     # 删除已处理变量
1296.
1297.     df = df.drop(df[["carid"]], axis=1)
1298.     withdraw = df['withdrawDate']
1299.     df = df.drop(labels=['withdrawDate'], axis=1)
1300.     df.insert(39, 'withdraw', withdraw)

```

```

1301.     #划分训练集和测试集
1302.     from sklearn.model_selection import train_test_split
1303.
1304.     lst = [i for i in range(0,39)]
1305.
1306.     X=df.iloc[:,lst]
1307.     y=df.loc[:,"withdraw"]
1308.
1309.     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
1310.
1311.     print("X_train.shape:",X_train.shape)
1312.     print("X_test.shape:",X_test.shape)
1313.     print("y_train.shape:",y_train.shape)
1314.     print("y_test.shape:",y_test.shape)
1315.     # 数值型变量归一化处理
1316.
1317.     num_columns = ['pushPrice','serial','model','mileage','transferCount','
1318.                    'displacement','newprice','price','feature12-length','fe
1319.                    'feature12-height','feature12-volume','tradeTime-register
1320.                    'tradeTime-licenseDate','tradeTime-modelyear','tradeTime
1321.                    'pull_pushTime','update_num','update_T_el','delta_price']
1322.
1323.     X_train_num = X_train.loc[:,num_columns]
1324.     X_test_num = X_test.loc[:,num_columns]
1325.
1326.     X_train_num.columns = num_columns
1327.
1328.     from sklearn import preprocessing
1329.
1330.     min_max_scaler = preprocessing.MinMaxScaler().fit(X_train_num)
1331.     X_train_num = min_max_scaler.transform(X_train_num)
1332.     X_test_num = min_max_scaler.transform(X_test_num)
1333.
1334.     X_train.drop(labels=num_columns,axis=1,inplace=True)
1335.     X_test.drop(labels=num_columns,axis=1,inplace=True)
1336.
1337.     X_train_num = pd.DataFrame(X_train_num,index=X_train.index)
1338.     X_train = pd.concat([X_train,X_train_num],axis=1)
1339.
1340.     X_test_num = pd.DataFrame(X_test_num,index=X_test.index)

```

```

1340.     X_test = pd.concat([X_test,X_test_num],axis=1)
1341.     def scores(y_pred, y_test):
1342.         y_pred = list(y_pred)
1343.         y_test = list(y_test)
1344.
1345.         TP, TN, FP, FN = 0, 0, 0, 0
1346.         for i in range(len(y_pred)):
1347.             if y_pred[i] == 1 and y_test[i]==1:
1348.                 TP += 1
1349.             elif y_pred[i] == 1 and y_test[i]==0:
1350.                 FP += 1
1351.             elif y_pred[i] == 0 and y_test[i]==0:
1352.                 TN += 1
1353.             else:
1354.                 FN += 1
1355.         precision = TP/(TP+FP)
1356.         recall = TP/(TP+FN)
1357.         f1 = 2*precision*recall/(recall+precision)
1358.         return precision, recall, f1
1359.
1360.     from sklearn.linear_model import LogisticRegression
1361.     from sklearn.model_selection import cross_val_score
1362.
1363.     def logistic_regression(X_train, X_test):
1364.         logit = LogisticRegression(class_weight='balanced')
1365.         logit.fit(X_train, y_train)
1366.
1367.         y_pred = logit.predict(X_test)
1368.         coef = logit.coef_
1369.         intercept = logit.intercept_
1370.
1371.         return y_pred, coef, intercept
1372.
1373.     y_pred1, coef, intercept = logistic_regression(X_train, X_test)
1374.
1375.     precision1, recall1, f11 = scores(y_pred1, y_test)
1376.
1377.     print("逻辑回归: ")
1378.     print(precision1)
1379.     print(recall1)
1380.     print(f11)
1381.     import pydotplus
1382.     import numpy as np
1383.     import pandas as pd

```

```

1384. import scipy.stats as stats
1385. import matplotlib.pyplot as plt
1386. from sklearn import metrics
1387. from openpyxl import load_workbook
1388. from sklearn.tree import export_graphviz
1389. from sklearn.ensemble import RandomForestRegressor
1390.
1391. from sklearn.ensemble import RandomForestClassifier
1392.
1393. def random_forest_classifier(X_train, X_test):
1394.     forest = RandomForestClassifier(criterion='entropy', class_weight='
balanced',n_estimators=100)
1395.     forest.fit(X_train, y_train)
1396.
1397.     y_pred = forest.predict(X_test)
1398.
1399.     estimator = forest.estimators_
1400.
1401.     return y_pred, estimator, forest
1402.
1403. y_pred2, estimator, forest = random_forest_classifier(X_train, X_test)
1404.
1405. precision2, recall2, f12 = scores(y_pred2, y_test)
1406.
1407. print("随机森林: ")
1408. print(precision2)
1409. print(recall2)
1410. print(f12)
1411.
1412. # 绘制误差图像
1413.
1414. plt.figure(1)
1415. plt.clf()
1416. ax=plt.axes(aspect='equal')
1417. plt.scatter(y_test, y_pred2)
1418. plt.xlabel('True Values')
1419. plt.ylabel('Predictions')
1420. Lims=[0,1]
1421. plt.xlim(Lims)
1422. plt.ylim(Lims)
1423. plt.plot(Lims,Lims)
1424. plt.grid(False)
1425.

```

```

1426.     error = y_pred2 - y_test
1427.     plt.figure(2)
1428.     plt.clf()
1429.     plt.hist(error,bins=30)
1430.     plt.xlabel('Prediction Error')
1431.     plt.ylabel('Count')
1432.     plt.grid(False)
1433.
1434.     import os
1435.
1436.     os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38
        /bin/'
1437.     tree_graph_dot_path='D:/tree.dot'
1438.
1439.     from sklearn import tree
1440.     from IPython.display import Image
1441.
1442.     Estimators = forest.estimators_
1443.     for index, model in enumerate(Estimators):
1444.         filename = 'RFC_min_sample500' + str(index) + '.pdf'
1445.         dot_data = tree.export_graphviz(model , out_file=None, feature_name
            s=names,
1446.                                         class_names='label',
1447.                                         filled=True, rounded=True,
1448.                                         special_characters=True)
1449.         graph = pydotplus.graph_from_dot_data(dot_data)
1450.
1451.         # 使用 ipython 的终端 jupyter notebook 显示
1452.         Image(graph.create_png())
1453.         graph.write_pdf(filename)
1454.         break
1455.     names = ['brand', 'color', 'cityId', 'carCode', 'country', 'maketype', 'gea
        rbox', 'oiltype', 'feature1', 'feature2',
1456.             'feature3', 'feature5', 'feature6', 'feature8', 'feature9',
            'feature10', 'feature11', 'feature14',
1457.             'pushPrice', 'serial', 'model', 'mileage', 'transferCount', 'seating
            s',
1458.             'displacement', 'newprice', 'price', 'feature12-length', 'fe
            ature12-width',
1459.             'feature12-height', 'feature12-volume', 'tradeTime-register
            Date',
1460.             'tradeTime-licenseDate', 'tradeTime-modelyear', 'tradeTime
            -feature13',

```

```

1461.         'pull_pushTime', 'update_num', 'update_T_el', 'delta_price']

1462.
1463.     # Calculate the importance of variables
1464.
1465.     random_forest_importance=list(forest.feature_importances_)
1466.     random_forest_feature_importance=[(feature,round(importance,8))
1467.                                         for feature, importance in zip(names,
        random_forest_importance)]
1468.     random_forest_feature_importance=sorted(random_forest_feature_importanc
        e,key=lambda x:x[1],reverse=True)
1469.     plt.figure(3)
1470.     plt.clf()
1471.     importance_plot_x_values=list(range(len(random_forest_importance)))
1472.     plt.bar(importance_plot_x_values,random_forest_importance,orientation='
        vertical')
1473.     plt.xticks(importance_plot_x_values,names,rotation='vertical')
1474.     plt.xlabel('Variable')
1475.     plt.ylabel('Importance')
1476.     plt.title('Variable Importances')
1477.
1478.     # 损失函数: MSE
1479.     def mse_loss(y_pred, y_test):
1480.         y_pred = list(y_pred)
1481.         y_test = list(y_test)
1482.
1483.         mse = 0
1484.         for i in range(len(y_pred)):
1485.             mse += (y_pred[i]-y_test[i])**2
1486.         mse /= len(y_pred)
1487.
1488.         return mse
1489.
1490.
1491.     # 损失函数: Accuracy = count(Ape 0.05) / count(total)
1492.     def accuracy(y_pred, y_test):
1493.         y_pred = list(y_pred)
1494.         y_test = list(y_test)
1495.
1496.         ape = 0
1497.         count = 0
1498.         for i in range(len(y_pred)):
1499.             ape += abs(y_pred[i]-y_test[i])/y_test[i]
1500.         if ape<=0.05:

```

```

1501.             count += 1
1502.
1503.             mape = ape/len(y_pred)
1504.             accuracy = count/len(y_pred)
1505.             score = 0.2*(1-mape)+0.8*accuracy
1506.             return mape, accuracy, score
1507.     #未降维
1508.     from sklearn import linear_model
1509.     from sklearn.model_selection import cross_val_score
1510.
1511.     lereg = linear_model.LinearRegression()
1512.     lereg.fit(X_train_sc, y_train)
1513.
1514.     # 验证集
1515.     loss1 = -cross_val_score(lereg, X_train_sc, y_train, cv = 10, scoring='
        neg_mean_squared_error').mean()
1516.
1517.     # 训练集
1518.     y_pred = lereg.predict(X_test_sc)
1519.     mse1 = mse_loss(y_pred, y_test)
1520.     mape1, accuracy1, score1 = accuracy(y_pred, y_test)
1521.
1522.     print("线性回归:未降维")
1523.     print("训练集 mse:", loss1)
1524.     print("测试集 mse:", mse1)
1525.     print("测试集 mape:", mape1)
1526.     print("测试集 accur:", accuracy1)
1527.     print("测试集 score:", score1)
1528.
1529.     #低方差滤波
1530.     from sklearn import linear_model
1531.     from sklearn.model_selection import cross_val_score
1532.
1533.     lereg = linear_model.LinearRegression()
1534.     lereg.fit(X_train_dp, y_train)
1535.
1536.     # 验证集
1537.     loss1 = -cross_val_score(lereg, X_train_dp, y_train, cv = 10, scoring='
        neg_mean_squared_error').mean()
1538.
1539.     # 训练集
1540.     y_pred = lereg.predict(X_test_dp)
1541.     mse1 = mse_loss(y_pred, y_test)
1542.     mape1, accuracy1, score1 = accuracy(y_pred, y_test)

```



```

1543.
1544.     print("线性回归:低方差滤波")
1545.     print("训练集 mse:", loss1)
1546.     print("测试集 mse:", mse1)
1547.     print("测试集 mape:", mape1)
1548.     print("测试集 accur:", accuracy1)
1549.     print("测试集 score:", score1)
1550.     #特征选择+低方差滤波
1551.     from sklearn import linear_model
1552.     from sklearn.model_selection import cross_val_score
1553.
1554.     lereg = linear_model.LinearRegression()
1555.     lereg.fit(X_train_kb, y_train)
1556.
1557.     # 验证集
1558.     loss1 = -cross_val_score(lereg, X_train_kb, y_train, cv = 10, scoring='
neg_mean_squared_error').mean()
1559.
1560.     # 训练集
1561.     y_pred = lereg.predict(X_test_kb)
1562.     mse1 = mse_loss(y_pred, y_test)
1563.     mape1, accuracy1, score1 = accuracy(y_pred, y_test)
1564.
1565.     print("线性回归:低方差滤波")
1566.     print("训练集 mse:", loss1)
1567.     print("测试集 mse:", mse1)
1568.     print("测试集 mape:", mape1)
1569.     print("测试集 accur:", accuracy1)
1570.     print("测试集 score:", score1)
1571.
1572.     print('未降维')
1573.     #交叉验证
1574.     from sklearn import tree
1575.     from sklearn.model_selection import cross_val_score
1576.
1577.     scores = []
1578.     max_depths = [2,3,5,7,10,12,15]
1579.
1580.     for n in max_depths:
1581.         clf = tree.DecisionTreeRegressor(max_depth=n,random_state=0)
1582.         score = -cross_val_score(clf,X_train_sc,y_train,cv=5,scoring='neg_m
ean_squared_error')
1583.         scores.append(score)
1584.

```

```

1585.     for i in scores:
1586.         plt.plot(i)
1587.     plt.title('Cross Validation scores')
1588.     plt.xlabel('n_estimators')
1589.     plt.ylabel('score')
1590.     plt.legend(max_depths)
1591.
1592.     plt.show()
1593.     print('未降维')
1594.     #交叉验证
1595.     from sklearn.ensemble import RandomForestRegressor
1596.
1597.     scores = []
1598.     n_estimators = [100,200,300,400]
1599.
1600.     for n in n_estimators:
1601.         forest = RandomForestRegressor(n_estimators=n,random_state=0)
1602.         score = -cross_val_score(forest,X_train_sc,y_train,cv=5,scoring='neg_mean_squared_error')
1603.         scores.append(score)
1604.
1605.     print(scores)
1606.     for i in scores:
1607.         plt.plot(i)
1608.     plt.title('Cross Validation scores')
1609.     plt.xlabel('n_estimators')
1610.     plt.ylabel('score')
1611.     plt.legend(n_estimators)
1612.
1613.     plt.show()
1614.     import pandas as pd
1615.     import numpy as np
1616.     from sklearn.preprocessing import StandardScaler
1617.     from sklearn import preprocessing
1618.     from collections import Counter
1619.     import warnings; warnings.simplefilter('ignore')
1620.     import matplotlib.pyplot as plt
1621.     from sklearn.cluster import KMeans
1622.     import matplotlib as mpl
1623.     import seaborn as sns
1624.     plt.rcParams['font.sans-serif']=['SimHei']
1625.     plt.rcParams['axes.unicode_minus'] = False
1626.
1627.     df = pd.read_csv('附件 1 处理后数据.csv',engine='python')

```

```

1628.     df=df[["newprice", "tradeTime-licenseDate", "tradeTime-registerDate", "dis
           placement", "mileage", "tradeTime-feature13", "price"]]
1629.
1630.     kmeans=KMeans(n_clusters=3)
1631.     kmeans.fit(df)
1632.     y_pred=kmeans.predict(df)
1633.
1634.     print(kmeans.cluster_centers_[ :,6])
1635.     plt.scatter(y_pred,df["price"],c=y_pred,s=30)
1636.     plt.scatter(0,kmeans.cluster_centers_[0,6],marker="^",s=100)
1637.     plt.scatter(1,kmeans.cluster_centers_[1,6],marker="^",s=100)
1638.     plt.scatter(2,kmeans.cluster_centers_[2,6],marker="^",s=100)
1639.
1640.     print(kmeans.cluster_centers_[ :,0])
1641.     plt.scatter(y_pred,df["newprice"],c=y_pred,s=30)
1642.     plt.scatter(0,kmeans.cluster_centers_[0,0],marker="^",s=100)
1643.     plt.scatter(1,kmeans.cluster_centers_[1,0],marker="^",s=100)
1644.     plt.scatter(2,kmeans.cluster_centers_[2,0],marker="^",s=100)
1645.
1646.     kmeans2=KMeans(n_clusters=2)
1647.     kmeans2.fit(df)
1648.     y_pred2=kmeans2.predict(df)
1649.
1650.     print(kmeans2.cluster_centers_[ :,6])
1651.     plt.scatter(y_pred2,df["price"],c=y_pred2,s=30)
1652.     plt.scatter(0,kmeans2.cluster_centers_[0,6],marker="^",s=100)
1653.     plt.scatter(1,kmeans2.cluster_centers_[1,6],marker="^",s=100)
1654.
1655.     df = pd.read_csv('第二问编码后的数据 2.csv',engine='python')
1656.     df=df[["pushPrice", "withdrawDate", "newprice", "pull_pushTime", "update_nu
           m", "update_T_el", "delta_price", "price"]]
1657.     df.head(3)
1658.     kmeans=KMeans(n_clusters=2)
1659.     kmeans.fit(df)
1660.     y_pred=kmeans.predict(df)
1661.
1662.     for i in range(df.shape[1]):
1663.         print("-----以下是",df.columns[i], "-----")
1664.         print(kmeans.cluster_centers_[ :,i])
1665.         plt.scatter(y_pred,df.iloc[:,i],c=y_pred,s=30)
1666.         plt.scatter(0,kmeans.cluster_centers_[0,i],marker="^",s=100)
1667.         plt.scatter(1,kmeans.cluster_centers_[1,i],marker="^",s=100)
1668.         plt.ylim=(0,1)
1669.

```

```

1670.         plt.show()
1671.     location1=[]
1672.     location2=[]
1673.     for i in range(len(y_pred)):
1674.         if y_pred[i]==0:
1675.             location1.append(i)
1676.         else:
1677.             location2.append(i)
1678.     print("location1",len(location1))
1679.     print("location2",len(location2))
1680.
1681.     df = pd.read_csv('第二问编码后的数据 2.csv',engine='python')
1682.     #分别提取两类
1683.     df1=pd.DataFrame()
1684.     df2=pd.DataFrame()
1685.     for i in range(df.shape[0]):
1686.         if i in location1:
1687.             df1=pd.concat([df1,(df.iloc[i,:].T)],axis=1)
1688.         else:
1689.             df2=pd.concat([df2,(df.iloc[i,:].T)],axis=1)
1690.
1691.     df1=df1.T
1692.     df2=df2.T
1693.     df1=df1.reset_index()
1694.     df2=df2.reset_index()
1695.
1696.     discount1=[]
1697.     for i in range(df1.shape[0]):
1698.         discount1.append((df1.loc[i,"pushPrice"])/(df1.loc[i,"newprice"]))
1699.
1700.     discount2=[]
1701.     for i in range(df2.shape[0]):
1702.         discount2.append((df2.loc[i,"pushPrice"])/(df2.loc[i,"newprice"]))
1703.
1704.     df1.insert(43, 'discount', discount1)
1705.     df2.insert(43, 'discount', discount2)
1706.
1707.     #计算两类样本的各变量均值
1708.     class_compare=pd.DataFrame(columns=["第一类均值","第二类均值"])
1709.     for i in range(df1.shape[1]):
1710.         class_compare.loc[df1.columns[i]]=df1.mean()[i],df2.mean()[i]]
1711.

```

```

1712.     #设置 dataframe 不以科学计数法打印
1713.     pd.set_option('display.float_format', lambda x: '%.5f' % x)
1714.
1715.     class_compare
1716.
1717.     df_all = pd.read_csv('第二问编码后的数据 2.csv',engine='python')
1718.     df=df_all[["pushPrice","withdrawDate","newprice","pull_pushTime","update_num","update_T_el","delta_price","price"]]
1719.     df.head(3)
1720.     kmeans=KMeans(n_clusters=2)
1721.     kmeans.fit(df)
1722.     y_pred=kmeans.predict(df)
1723.     #transferCount 有明显差异
1724.
1725.     plt.figure(figsize=(13,10), dpi= 80)
1726.     sns.violinplot(x=y_pred,y=df_all["transferCount"],palette=sns.color_palette("YlGn",2))
1727.
1728.     # Decoration
1729.     plt.title('两类样本 transferCount', fontsize=22)
1730.
1731.     plt.show()

```