队伍编号	MCB2100593
赛道	A

基于 Stacking 模型的二手车价格预测

摘 要

随着机动车数量的增长，二手车交易的市场蓬勃发展。为了帮助二手车交易平台解决二手车定价问题以及加快销售速度的问题。本文建立基于 Stacking 技术和贝叶斯优化的模型，将估价模型进行融合，提出了更准确的二手车的估价策略；建立基于 XGBoost 模型，对影响交易周期的关键因素进行研究。

针对问题一，主要需要确定影响成交价格的因素并建立合适的预测模型。本文建立了基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型来进行估价，并运用贝叶斯优化进行模型超参数调优。首先，我们对数据进行异常值剔除、正态转换和缺省值填补的预处理，构造出便于统计分析的特征，选取与成交价格有密切关系的 28 个特征并做检验。基于所选特征利用 Stacking 融合技术的模型对成交价格进行预测，在此过程中用 HERO 算法对基模型 XGBoost、LightGBM 和 CatBoost 进行贝叶斯优化，最终得到训练模型。依据此训练模型在验证集上所得预估价格的评测值为 0.5324，且优于单一模型评测效果。

针对问题二，主要需要研究影响交易周期的关键因素以及改善策略。本文建立了 XGBoost 模型来提取关键因素并进行分析。在第一问数据预处理的基础上，选取与成交周期有关的 29 个特征。基于所选特征利用 XGBoost 算法确定影响交易周期的关键因素有：降价比率、评估方法 1、评估方法 2、里程、定价、新车价、车辆所在城市交易价格总值。根据上述因素在收车、定价、维修保养方面提出了促进销售速度加快的方法。

针对问题三，由于二手车交易“一车一况”的特性，我们期望在原有数据集的基础上获得更多能够衡量车辆老旧损伤程度的特征。因此建议车辆在定价之前做一次全面的检测，从而得到车辆老旧损耗程度的特征，便于我们预估价格。另外，不同于车辆的市场特征以及性能特征，二手车的车系车型对成交价格的影响不具有明显的统计规律，我们建议采用图嵌入方式研究其对价格的影响。

关键词：数据预处理、HEBO 算法、Stacking 技术、XGBoost 模型

目录

一、问题重述

1.1 问题的背景

随着社会的进步和人民生活水平的不断提高，我国汽车市场不断增大，机动车数量不断增长，人均保有量也随之增加，机动车以“二手车”形式在流通环节，包括二手车收车、二手车拍卖、二手车零售、二手车置换等环节的流通需求越来越大。然而相对于新车交易，二手车作为一种特殊的“电商商品”，因为其“一车一况”的特性比一般电商商品的交易要复杂得多，究其原因是二手车价格难于准确估计和设定，不但受到车本身基础配置，如品牌、车系、动力等的影响，还受到车况如行驶里程、车身受损和维修情况等的影响，甚至新车价格的变化也会对二手车价格带来作用。随着电商平台的发展，出现了一些专业汽车交易平台，但是目前国家尚未出台一个评判二手车资产价值的标准。一些二手车交易平台和二手车第三方估价平台都从自身的角度建立了一系列估价方法用于评估二手车资产的价值。

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

1.2 问题的提出

问题一：基于给定的二手车交易样本数据（附件 1：估价训练数据），选用合适的估价方法，构建模型，预测二手车的零售交易价格，数据中会对 id 类，主要特征类等信息进行脱敏。采用附件 1 中的“估价训练数据”（带标签）训练模型和测试模型，自行设置测试集，使用训练完成后的模型对附件 2 中的“估价验证数据”（不带标签）进行预测。

问题二：结合附件 4“门店交易训练数据”对车辆的成交周期（从车辆上架到成交的时间长度，单位：天）进行分析，挖掘影响车辆成交周期的关键因素。假如需要加快门店在库车辆的销售速度，研究可以结合这些关键因素采取哪些行之有效的手段，并进一步说明这些手段的适用条件和预期效果。

问题三：依据给出的样本数据集，探究还有哪些问题值得研究，并给出思路。

二、问题的分析

2.1 问题一的分析

问题一主要需要我们提取附件 1 中估价训练数据的车辆特征，对特征进行构造和筛选，因此我们对各个特征进行 EDA，并基于分析结果对题目进行求解。首先，由于有些特征严重长尾或缺省值过多，不利于预测价格，因此我们将这些特征舍弃。其次，我们利用 label encoding 方法将离散特征转化为连续特征进行处理。通过相关性分析和 XGBoost 算法获取特征重要性，按照从低到高的水平逐次筛除检验模型在验证集上的效果从而得到预测价格所需的重要特征。然后利用 Stacking 技术将经过贝叶斯优化的 XGBoost、LightGBM 和 CatBoost 三种模型进行融合，并基于所选特征进行模型训练，并对价格进行预估。

2.2 问题二的分析

问题二主要需要我们根据附件 4 的数据对车辆的成交周期进行分析，挖掘影响车辆成交周期的关键因素，因此根据第一问所建立的 XGBoost 模型以及所用的方法，对附件 4 的数据和特征进行处理和筛选，并选出重要性最好的特征来确定影响车辆成交周期的关键因素，结合关键因素继而找出行之有效的手段提高门店在库车辆的销售速度，并找到对应手段的适用条件和估计对应手段的预期效果。

2.3 问题三的分析

由于二手车交易“一车一况”的特性，车辆的老旧损伤程度对成交价格的影响十分关键，而样本数据在这方面的信息并不丰富。因此我们期望在原有数据集的基础上获得更多能够衡量车辆老旧损伤程度的特征。针对此问题建议车辆在定价之前做一次全面的检测，从而得到车辆老旧损耗程度的特征，便于我们预估价格；在目前特征的基础上也可基于根据现有特征建模挖掘出车辆的车况可以使价格预测更加准确。

另外，不同于车辆的市场特征以及性能特征，二手车的车系车型对成交价格的影响不具有明显的统计规律，我们建议采用图嵌入方式研究其对价格的影响。将二手车作为

图中的节点，相同车系或者相同车型的车对应的节点连接起来构造图，使用 item2vec 或者 node2vec 算法计算图嵌入矩阵得到表征车系车型的向量，然后直接把它作为特征的一部分进行模型的训练。

三、模型的假设

1、假设所调查的二手车交易样本数据都比较准确；

2、假设二手车交易价格不受第三方平台自身因素的影响；

3、假设买家和卖家是在公平、公正的条件下进行买卖。

四、基于 Stacking 模型的二手车价格预测

在本问题中，利用数据 EDA 方法对二手车的 36 种变量数据进行处理，并对二手车信息变量间的相互关系以及变量与二手车估价之间的关系进行探索；在此基础上构造特征并选择 Stacking 模型将经过贝叶斯优化的 XGBoost、LightGBM 和 CatBoost 三种模型融合来进行价格预测。

利用 EDA 对已有的数据在尽量少的先验假定下进行探索，通过作图、制表、方程拟合、计算特征量等手段探索数据的结构和规律能够帮助了解数据集，对数据集进行处理并获取重要特征，从而使得数据集的结构和特征集让所研究的预测问题更加可靠。

4.1 数据处理

基于 python 中 pandas 库对附件一中的数据进行读取，对数据进行以下预处理：

1、将数据信息按照类别特征和数值特征进行分类。

2、剔除异常数据，并将所给数据进行类型的转化，方便之后的操作。其中剔除的数据包括一条 target > 10000 的数据以及 target 过低的数据以及 target 比新车价高的数据。剔除原因为：target 过低的数据是一些车况很差的数据，不多见；target 比新车价高的数据有可能是因为车为绝版车，不多见。

3、对数据进行正态转换。我们采用三种函数对数据进行拟合，比较拟合效果，效果如下图：

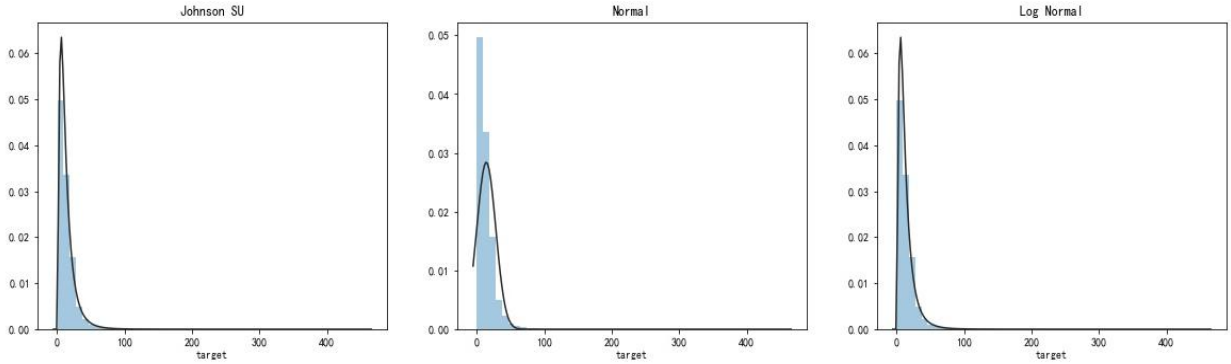


图 1 左图（Johnson SU 函数拟合） 中图（正态函数拟合） 右图（log 函数拟合）

将价格的总体分布画出来后，我们发现价格数据分布存在右偏，说明存在过大的极端值，因此需要对数据中的过大的价格值进行处理。我们通过上图发现利用 Johnson SU 函数对数据进行转换效果较好，因此得到偏度为-0.001057，峰度为 0.075610 的正态分布。转换后的数据如下图：

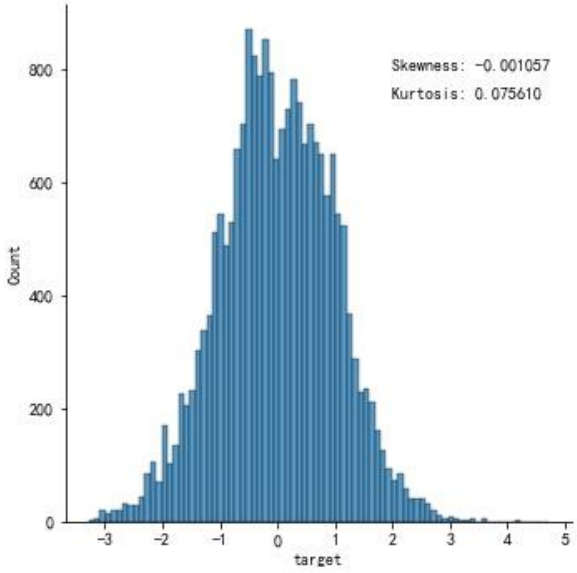


图 2 价格数据正态转换图像

4、根据车型 id 和车系 id 相同的判断准则，对数据中的缺省值，用对应的样本的众数进行填充。

5、对附件 1：估价训练数据.txt 中的数据按照展销时间进行排序，并按照 7：3 的比例进行划分，构造训练数据集和验证数据集，训练集用于模型训练，验证集用于模型效果的评估。

4.2 特征构造

取出二手车的数字特征（能够刻画随机变量某些方面的性质特征的量）和类别特征（有限选项内取值的特征），对严重长尾和缺省值过多的特征进行舍弃。具体特征的舍弃如下：

严重长尾：	车辆颜色, 燃油类型, 载客人数, Feature_1, Feature_3
缺省值过多：	Feature_4, Feature_7, Feature_10, Feature_15

为了处理二手车的离散型特征，可以采用 one-hot encoding 和 label encoding 两种转化方法。由于已知离散特征种类较多，转化后参数维度太大，不利于模型训练与预测。因此本文采用 label encoding 方法，通过提取不同变量下的交易价格数值特征构造所在城市交易价格总值（所在城市_sum）、车辆品牌交易价格均值（品牌_mean）、车辆品牌交易价格最大值（品牌_max）等特征。

另外，基于汽车年限的评估方法引入新特征（评估方法 1）：可将新车使用 10 年报废（按照老的规定算）视为 100 分，把 15%作为不折旧的固定部分为残值，其余 85%为浮动折旧值。可分三个阶段：3 年—4 年—3 年来折旧，折旧率分别为 11%、10%和 9%。然后加残值，构成了折旧值，计算为：

评估价=市场现行新车售价×[15%（不动残值）+85%（浮动值）×（1-（分阶段折旧率））]+评估值。

构造后的特征类别如下：

表 1 关于二手车交易价格的特征

序号	特征	序号	特征
1	年款	15	所在城市_sum
2	排量	16	品牌_mean
3	变速箱	17	品牌_count
4	载客人数	18	品牌_max
5	新车价	19	品牌_min
6	里程	20	品牌_median
7	注册年份	21	品牌_sum
8	Feature_2	22	厂商类型_mean
9	Feature_5	23	国标码-mean

10	Feature_13	24	年款_mean
11	Feature_12_num_sqrt	25	变速箱_mean
12	Feature_12_0	26	Feature_2_mean
13	Feature_12_1	27	Feature_8_mean
14	Feature_12_2	28	评估方法 1

由于不同特征之间的量纲差异过大时，样本之间的相似度评估结果将会受到较大的影响，导致对样本相似度的计算存在偏差，所以采用最小最大化处理，将数据统一映射到[0,1]区间上，归一化公式如下：

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

其中x_{min}指样本中的最小值，x_{max}指样本的最大值。

4.3 特征筛选

首先对构造后得到的连续特征进行相关性分析：计算变量间的 Pearson 系数，正值表示正相关，负值表示负相关，绝对值越大表示线性相关程度越高。

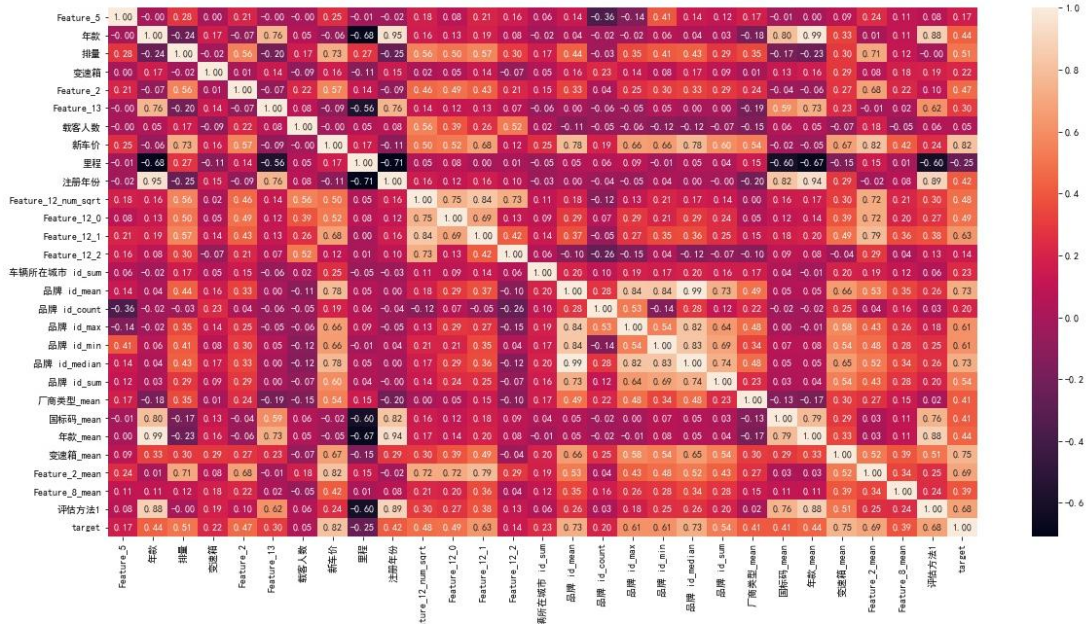


图 3 成交价格连续特征协方差矩阵热力图

根据上图我们可以发现汽车的成交价格与新车价、feature_12_1、品牌 id_mean、品牌 id_median、变速箱_mean、feature_2_mean、评估方法 1 的相关性较强。

然后利用 XGBoost 算法（在 4.4.1 进行介绍）获取特征重要性，依据模型在验证集上的效果对特征按照重要性程度从低到高进行排序，结果如下图：

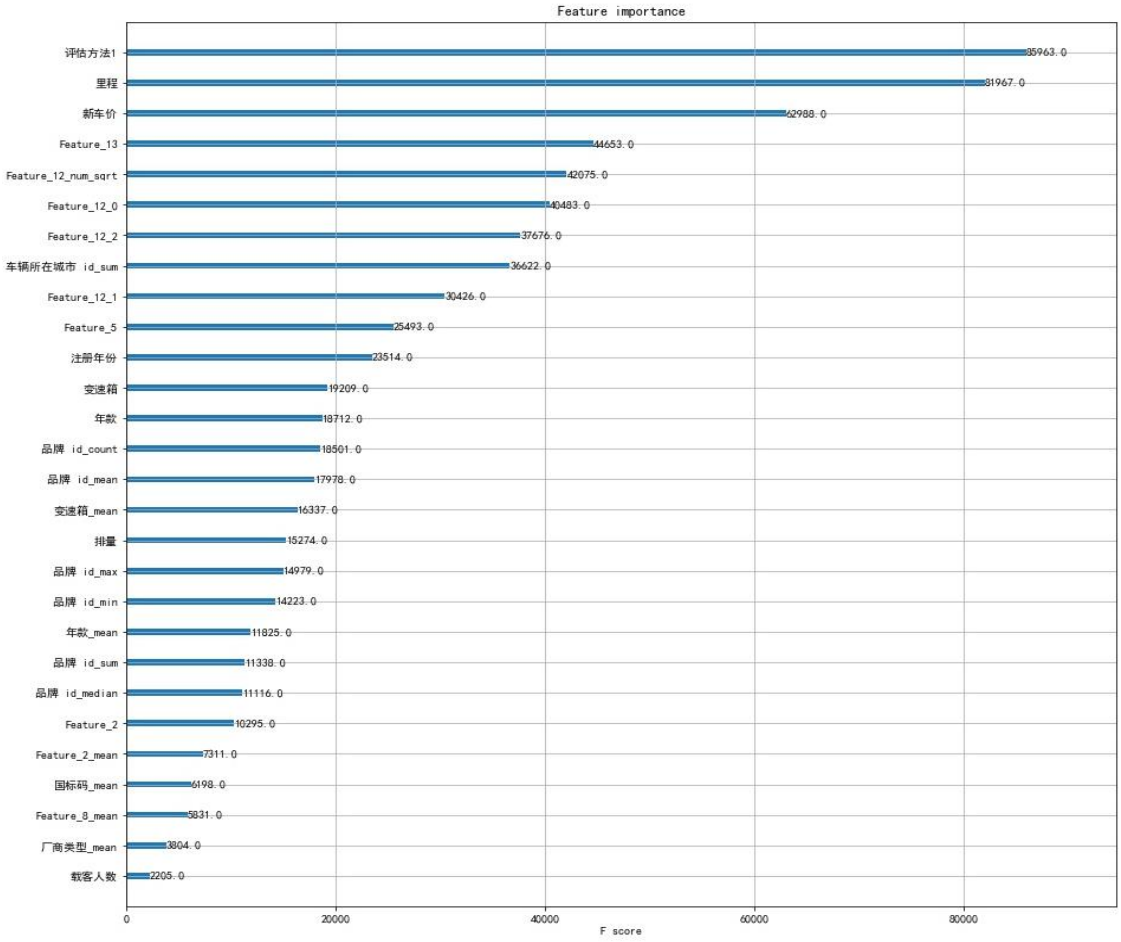


图 4 成交价格特征重要性排序图

由上图我们可以发现评估价格 1 特征的重要性最大，其次为里程、新车价、feature_13……利用模型训练，我们根据重要性从低到高的顺序删除对应的因素，继而检验我们所选的因素与正确的预测值的差距，数据如下图：

```
Feature Name=载客人数 Thresh=0.003, n=28, Accuracy: 52.78%
Feature Name=厂商类型_mean Thresh=0.005, n=27, Accuracy: 52.29%
Feature Name=Feature_8_mean Thresh=0.008, n=26, Accuracy: 52.22%
```

图 5 删除相应因素之后的预测准确率

根据上述的数据，我们可以发现即使删除重要性水平较低的因素，预测价格的准确值也会明显变小，因此我们选择图 4 的所有特征作为预测价格的主要特征。

4.4 模型选择

4.4.1 XGBoost 模型

XGBoost 算法是一种以 CART 决策树模型为基础的集成学习方法，通过构建多棵 CART 决策树来提供预测模型的准确性，最后将每轮训练得到决策树的预测结果求和得到最终的预测值[1]。其第 t 棵 CART 决策树的目标函数定义如下[2]：

$$L^{(t)} = \sum_{i=1}^{n} l\left(y_i, y_i^{'(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{2}$$

其中：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\|w\|^2 \tag{3}$$

n 为样本总数，$y_i^{'(t-1)}$为第 t-1 个学习器对 i 样本的预测值，$f_t(x_i)$为新加入的第 t 个学习器，$\Omega(f_t)$为正则项。

运用泰勒公式对目标函数进行展开

$$\mathrm{f(x+\Delta x) = f(x) + f^{'}(x)\Delta x + \frac{1}{2}f^{''}(x)\Delta x^2} \tag{4}$$

损失函数的二阶泰勒展开结果为

$$\sum_i \mathrm{L}\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}} + \mathrm{f_t(x_i)}\right) = \sum_i \left[\mathrm{L}\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}}\right) + \mathrm{L^{'}}\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}}\right)\mathrm{t(x_i)} + \frac{1}{2}\mathrm{L^{''}}\left(\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}}\right)\mathrm{f_t^2(x_i)}\right)\right] \tag{5}$$

用$\mathrm{g_i}$记为第 i 个样本损失函数的一阶导数，$\mathrm{h_i}$记为第 i 个样本损失函数的二阶导数

$$\mathrm{g_i} = \mathrm{L^{'}}\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}}\right) \tag{6}$$

$$\mathrm{h_i} = \mathrm{L^{''}}\left(\left(\mathrm{y_i}, \hat{\mathrm{y}}_\mathrm{i}^{\mathrm{t-1}}\right)\right)$$

在进行特征节点选择时，遍历训练集所有特征变量的取值，用节点分裂前的目标函数值减去分裂后 2 个叶子节点的目标函数值之和，计算增益值，得到树模型最优的切分点，其中增益值计算式为：

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{7}$$

4.4.2 LightGBM 模型

LightGBM 是一个高效实现 GBDT 算法的框架。它的原理与 XGBoost 相似，但其在处理数据上的速度是 XGB 模型的 10 倍，内存占用率也仅仅为 XGB 模型的 1/6，且准确

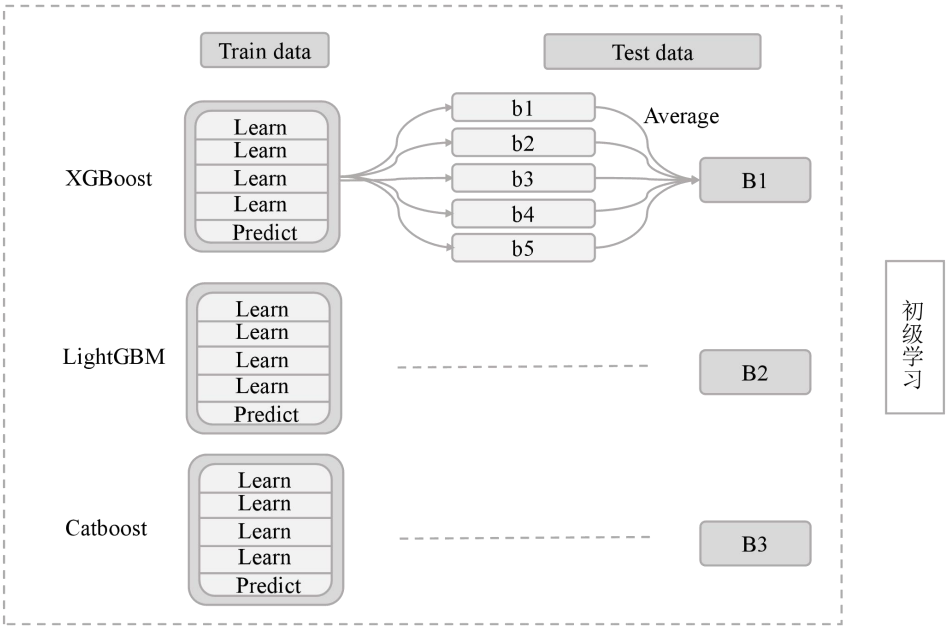
率也有提升。LGB 模型采用直方图算法遍历寻找最优的分割点，并且用直方图做差加速从而将遍历速度提升一倍，而且还用带深度限制的 Leaf-wise 的叶子生长算法在保证高效率的同时防止过拟合[3]。

4.4.3 CatBoost 模型

CatBoost 是一种以对称决策树为基学习器，参数较少、支持类别型变量和高准确性的 GBDT 框架。通过采用排序提升（Or-dered Boosting）的方式替代传统算法中的梯度估计方法，进而减小梯度估计的偏差，因此能够高效合理地处理类别型特征，解决以往 GBDT 框架的机器学习算法中常出现的梯度偏差和预测偏移问题，从而减少了过拟合的发生，提高了算法的泛化能力[4]。

4.4.4 Stacking 技术

Stacking 是一种集成学习技术，通过元分类器或元回归器聚合多个分类或回归模型。基础层模型基于完整的训练集进行训练，元模型基于基础层模型的输出进行训练[5]。本文采用 Stacking 融合模型预测过程，以 XGBoost、LightGBM、Catboost 三种单一模型为基模型进行训练预测；以 linear regression 作为元模型进行次级训练预测。如下图所示。



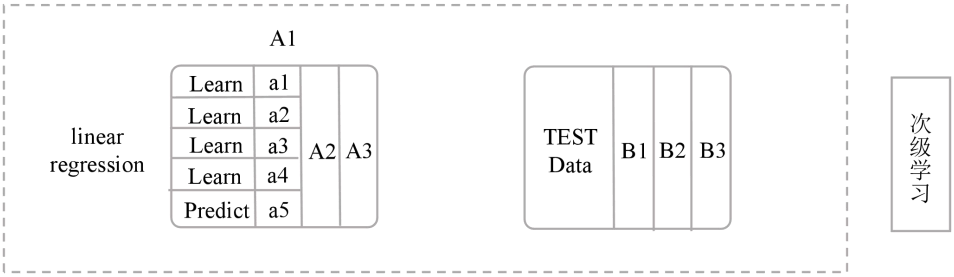


图 6 交叉验证的 Stacking 融合模型图

其中初级学习过程：假设已有数据集$D=\{x_i,y_i\}(=1,\cdots n)$，其中$x_i$为输入的样本特征，$y_i$为预测标签。将数据集$D$划分为训练集和测试集，基础层模型$M_k$基于五折划分的训练集进行模型训练得到基学习器$L_k$,$k=1,2,3$。基模型$L_k$分别基于划分测试集和原始测试集进行预测，输出结果$A_k$和$B_k$，$A_k$和$B_k$构成了新的数据样本，其中$k=1,2,3$。次级学习过程：元模型将初级学习过程的预测输出作为输入数据进行学习得到元学习器L_{new}，利用元学习器对数据学习预测得到预测结果。即：

$$L_{predicted}=L_k(L_1(x_i),L_2(x_i),L_3(x_i)) \tag{8}$$

本文根据所给数据进行了 XGB、LGB、CAT 模型以及 Stacking 技术的处理，结果如下表所示：

表 2 模型预测效果比较

模型	评测标准 $0.2*(1-Mape)+0.8*Accuracy_5$
XGBoost	0.5286
LightGBM	0.5256
CatBoost	0.5261
Stacking Model	**0.5324**

对比分析可以得出，Stacking 融合模型的预测效果均高于单一模型的预测效果，相比于单一模型，Stacking 融合模型拥有更好的预测精度和模型效果。

4.5 基于贝叶斯框架的超参数调优

贝叶斯超参数优化算法的基本思想是根据过去目标的评估结果建立替代函数，以找到目标函数的最小值。在此过程中建立的替代函数比原始目标函数更容易优化，并且通过对采集函数应用特定标准来选择要评估的输入值，相较于遗传算法，优化模型参数的过程更加简单，可以节省大量的时间。贝叶斯参数优化算法流程如下图所示。

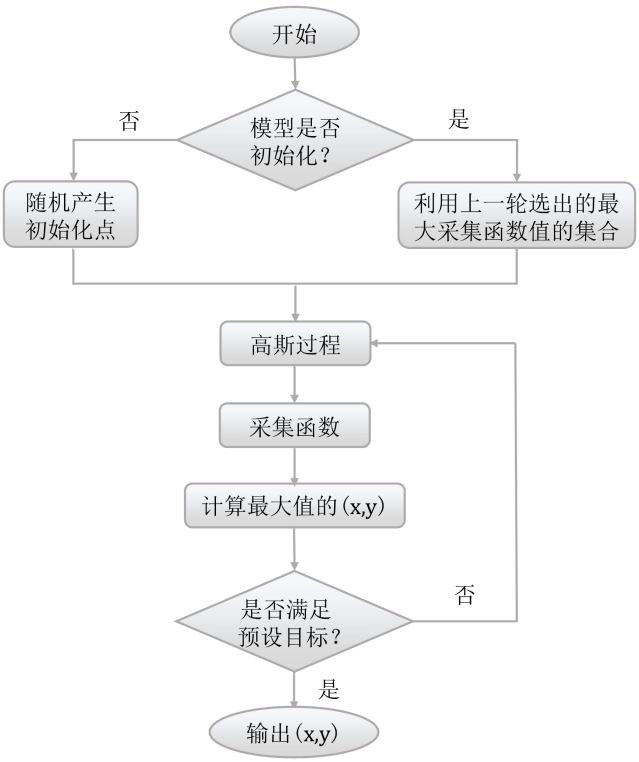


图 7 贝叶斯参数优化流程图

本文采用 HEBO 算法进行贝叶斯优化。

4.5.1 HEBO 算法

相比于普通的贝叶斯优化，HEBO 算法在以下两方面进行优化：带鲁棒性的采集函数、多目标采集函数[6]。

针对采集函数而言，从贝叶斯优化算法流程来看，对于非凸、非凹的黑盒函数，没有全局的解；代理函数和采集函数这两步是分开建模的，没有统一的计算框架让梯度在这两步间相互传播。为了解决这两个问题，HEBO 算法引入了带鲁棒性的采集函数：

$$\max_{\mathrm{x}} \alpha_{\mathrm{rob.}}(\cdot) \equiv \max_{\mathrm{x}} \mathrm{E}_{\epsilon\sim \mathrm{N}(0,\sigma_\epsilon^2)\mathrm{I}}[\alpha^{\theta+\epsilon}(\mathrm{x}|\mathrm{D})] \tag{9}$$

带鲁棒性的采集函数将尝试在一系列代理函数分布中寻找表现良好的候选点。带鲁棒性的采集函数的实现，只需要获取代理函数的均值和方差，而且如果满足以下条件：

$$\bar{\alpha}^{\theta}(x|D) = \alpha^{\theta}(x|D) + \eta\sigma_n \qquad \eta\sim N(0,1) \tag{10}$$

则对于任意的$\rho \in (0,1)$，我们有：

$$\left|\bar{\alpha}^{\theta}(x|D) - \mathrm{E}_{\epsilon\sim \mathrm{N}(0,\sigma_\epsilon^2)\mathrm{I}}[\alpha^{\theta+\epsilon}(\mathrm{x}|\mathrm{D})]\right| \le \rho \tag{11}$$

所以针对采集函数而言，引入高斯噪音$\epsilon\sim \mathrm{N}(0,\sigma_\epsilon^2)\mathrm{I}$，能够使采集函数更加鲁棒。

针对多目标采集函数，HEBO 算法引入多目标采集函数来获取帕累托最优解，防止经常输出相反的候选点，多目标采集函数为：

$$\max(\alpha_{\mathrm{EI}}^{\theta}(\mathrm{x}|\mathrm{D}), \alpha_{\mathrm{PI}}^{\theta}(\mathrm{x}|\mathrm{D}), \alpha_{\mathrm{UBC}}^{\theta}(\mathrm{x}|\mathrm{D})) \tag{12}$$

$\alpha_{\mathrm{Type}}^{\theta}(\mathrm{x}|\mathrm{D})$，$\mathrm{Type} \in \{\mathrm{EI}, \mathrm{PI}, \mathrm{UCB}\}$，为前面介绍的带鲁棒性的采集函数，HEBO 算法使用 NSGA—II 的进化算法进行求解。

HEBO 算法对数据的输入输出进行变换校准，将数据变换校准和 GPs 核函数联合在一起优化，并引入多目标采集函数，进行更鲁棒的探索，从而找到最精确的最优解。具体参数空间设置见代码。

4.5.2 优化流程

利用贝叶斯优化 Stacking 进行二手车估价预测的执行步骤为：

1、设定待优化参数空间（具体详细见代码），优化目标为均方根误差。

2、利用 HEBO 对 XGBoost、LightGBM、CatBoost 模型进行优化。

3、利用 Stacking 模型将上述三种模型融合。

4、将 Stacking 模型作为训练模型，对二手车估价进行预测。

4.5.3 HEBO 优化结果

1、LGBParams

表 3 LGBParams 的超参数优化结果

超参数	优化结果
learning_rate	0.010035982594961306
subsample	0.35028462644730735
colsample_bytree	0.3796965007412236
reg_lambda	0.0006865914755758266
reg_alpha	0.0026163019185854106
num_leaves	189
max_depth	37
n_estimators	1965
objective	mse

min_child_samples	25

2、XGBParams

表 4 XGBParams 的超参数优化结果

超参数	优化结果
learning_rate	0.010211967985703159
subsample	0.30001115463199896
colsample_bytree	0.48460986059028555
lambda	0.032780563044634864
alpha	0.06981554843421037
max_depth	26
min_child_weight	10
n_estimators	1845

3、CATParams

表 5 CATParams 的超参数优化结果

超参数	优化结果
learning_rate	0.08491056514750207
l2_leaf_reg	1
max_depth	8
n_estimators	2000
min_child_samples	25
loss_function	RMSE

4.6 实验结果

利用上述提及的 Stacking 技术对经过 HEBO 优化的 XGBoost、LightGBM、CatBoost 三种模型的融合所建立的新的模型，对二手车进行价格预估。根据题目要求的模型评测准则的算法，我们求得对应值为：0.5324。

详细预估价格见附件 3。

五、基于 XGBoost 的关键因素挖掘

5.1 数据处理

按照第一问的数据处理原则对样本数据进行异常值剔除、正态转换和缺省值填补等处理。得到成交周期经过正态变换后的数据集。

5.2 特征构造

取出二手车的数字特征和类别特征，对严重长尾和缺省值过多的特征进行舍弃。通过 label encoding 方法提取各变量下成交周期的数值统计特征构造出新特征。并引入两个评估方法作为新特征。

评估方法 1：使用年限计算法。将新车使用 10 年报废（按照老的规定算）视为 100 分，把 15%作为不折旧的固定部分为残值，其余 85%为浮动折旧值。可分三个阶段：3 年—4 年—3 年来折旧，折旧率分别为 11%、10%和 9%。前三年每年折 11%，总折 28%（85%×33%）。然后加残值，构成了折旧值，计算为：

评估价=市场现行新车售价×[15%（不动残值）+85%（浮动值）×（分阶段折旧率）]+评估值。

评估方法 2：里程分段“54321 法”。具体为：假设一部车有效寿命 30 万公里，将其分为 5 段，每段 6 万公里，每段价值依序为新车价的 5/15、4/15、3/15、2/15、1/15。用此方法评估二手车价格，按照公里数分段，得出每年的折损价格，最后做减到相应用新车价格减掉公里数对应的价格部分即可，详细实现方式见代码。

构造后的特征类别如下：

表 6 关于二手车成交周期的特征

序号	特征	序号	特征
1	评估方法 1	16	Feature_12_num_sqrt
2	评估方法 2	17	所在城市_sum
3	里程	18	品牌_mean
4	注册年份	19	品牌_count

5	变速箱	20	品牌_max
6	年款	21	品牌_min
7	排量	22	品牌_median
8	载客人数	23	品牌_sum
9	新车价	24	厂商类型_mean
10	Feature_2	25	国标码-mean
11	Feature_5	26	年款_mean
12	Feature_13	27	变速箱_mean
13	Feature_12_0	28	Feature_2_mean
14	Feature_12_1	29	Feature_8_mean
15	Feature_12_2		

采用与第一问同样的方法对特征数据采用最小最大化处理进行归一化。

5.3 关键因素挖掘

首先对构造后得到的连续特征进行相关性分析：

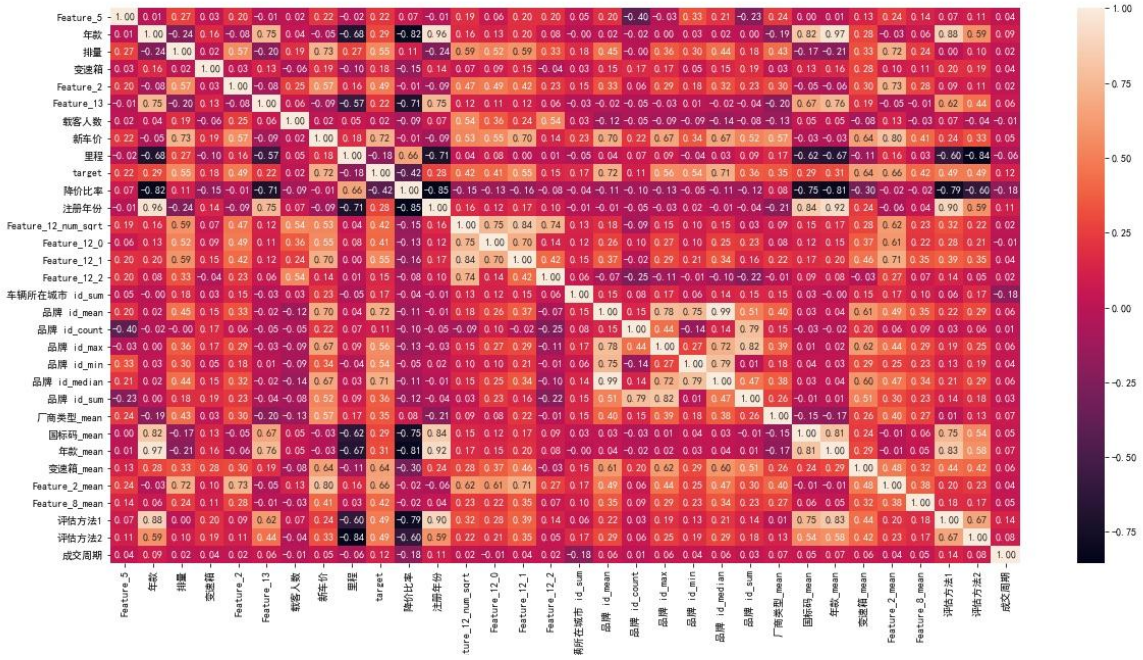


图 8 交易周期连续特征协方差矩阵热力图

根据上图发现汽车的交易周期与各特征的协方差数值均较小，表明各变量与交易周期的线性关系较弱。

然后利用 XGBoost 算法获取特征重要性，依据模型在验证集上的效果对特征按照重要性程度从低到高进行逐次排序，结果如下图：

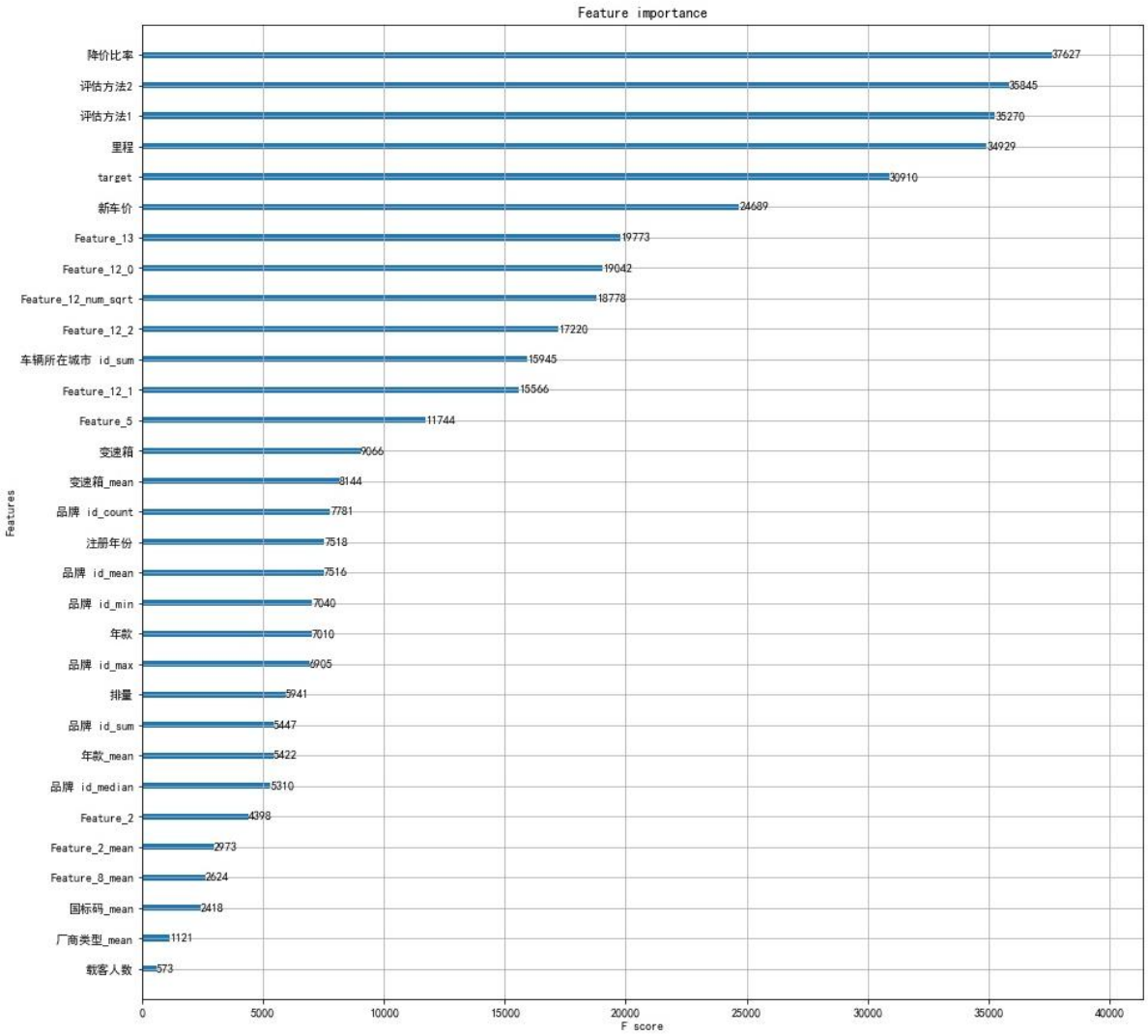


图 9 交易周期特征重要性排序图

由上图可知，对交易周期影响较大的关键因素有降价比率、评估方法 1、评估方法 2、里程、定价、新车价、Feature_13、Feature_12_0、Feature_12_num_sqrt、Feature_12_2、车辆所在城市 id_sum、Feature_12_1。

综上，我们所得影响车辆成交周期的关键因素主要有：降价比率、评估方法 1、评估方法 2、里程、定价、新车价、Feature_13、Feature_12_0、Feature_12_num_sqrt、Feature_12_2、车辆所在城市 id_sum、Feature_12_1。

5.4 销售建议

1、评估方法 1

根据“*评估价=市场现行新车售价×[15%（不动残值）+85%（浮动值）×（1-（分阶段折旧率））]+评估值*”的计算公式，可以分析得出评估价与新车售价成正相关，与使用年限成负相关。具体计算可得，前三年车辆折旧速度最快，汽车贬值速度最快。为了加快销售速度，在收车时要收取使用时间较短的车辆，而且尽可能多收使用时间在三年以内的车。

2、评估方法 2

根据“54321 法”的计算原则可以发现，车辆累积行驶距离越短，其价值越高；且在车辆越新，行驶距离对价值影响越大。具体分析得，当车辆行驶其寿命的 2/5 时，该车依据此方法的评估价只剩原车价的 40%；而车辆行驶其寿命的 1/5 时，该车评估价为原车价的 67%。为了加快销售速度，在收车时要收取寿命较高的车辆，而且尽可能多收寿命仍在 4/5 以上的车。

3.里程

里程数与交易周期呈现一定的负相关关系。里程可以客观反映二手车的老旧程度，这一性质表明车辆较旧不利于进行交易。因此在收车时尽量收取里程数较小的车辆，另外可以对车辆维修和保养以达到车辆品质改善的效果。

4、定价和新车价

定价对交易周期起着重要的影响作用。定价较高时，交易周期较长。因此门店定价时要根据车辆的实际情况进行定价，若车辆急于出售可把其价格定的比正常价格低，加大车辆的吸引力，从而让车辆更容易出售，降低交易周期。

新车价对于交易周期同样为比较重要的特征。新车价较高时，交易周期较长。因此在收取二手车时可以重点收取近期市场价较高的车辆。

5、降价比率

降价比率对于交易周期是最为重要的特征。降价比率越高，交易周期较短。因此在定价二手车时可以根据车辆的新车价来决定车辆的价格，控制降价比率在一定范围内，从而保障车辆的成交周期不会太长甚至无法出售。在车辆无法出售的情况下，可以降低车辆的定价，从而使降价比率提高，从而吸引买家购买车辆。

6、车辆所在城市交易价格总值

这一变量可以总体反应车辆所在城市的交易活跃度以及价格水平。选择交易价格总值较高城市的车辆，更易获得品质较高的车辆，也更易进行交易。XGBoost 模型得到的结果也同样表明该变量值越高，交易周期越短。因此门店在收车时可重点选取交易值较

高的城市的车辆。

综上分析，门店可采取以下几方面的销售手段；

收车；尽可能收取使用时间较短的、寿命较高的、品质较好的、近期市场价较高的、交易值较高的城市的车辆。

定价：定价时在符合车辆品质估价的基础上尽可能价格亲民，保障自己利益不会减少太多的情况下，适当提高降价比率。

维修保养：销售之前对二手车进行维修和保养，以在寿命和折旧方面对车辆进行最大程度的改善。

六、问题拓展

6.1 二手车车况信息研究

问题提出：不同于一般商品交易，二手车交易具有“一车一况”的特性。在所给样本数据中，我们可以获得车辆、展销时间、品牌、车系、车型、里程、车辆颜色、车辆所在城市、国标码、过户次数、载客人数、注册日期、上牌日期、国别、厂商类型、年款、排量、变速箱、燃油类型、新车价等特征。但是由于收购后的车辆变速箱、车内部件等数据未获取，车况老旧损伤程度未知，二手车性能没有很好的指标来衡量。在实验过程中，发现存在不少的样本车系车型相同，车龄较低，里程数较少，但是成交价却较低的情况，应该是因为车辆车况差的原因。由于二手车估价与车况紧密相关，且通过题目给出数据难以获取车况信息，因此若能基于根据现有特征挖掘出车辆的车况可以使价格预测更加准确。

解决思路：针对此问题，我们队伍给出一些思路：1. 我们可以基于变速箱、排量等汽车硬件数据，对于汽车每公里的损伤程度进行建模，再结合里程数，获得汽车的车况信息，使用该车况信息进行定价模型的训练和预测，会对定价效果有一定的帮助；2. 在汽车定价之前对二手车进行全面的检测，知道二手车的具体损耗，根据损耗的程度对数据进行再一次的分析，控制汽车的品牌、车系、车型，根据具体的损耗程度，再一次根据评价方法来对汽车的定价进行预估，从而使车辆的成交周期较小，卖家的利润也不会过于低。

6.2 图嵌入研究车系车型对车辆价格的单一影响

问题提出：经过研究发现，二手车在车系、车型不同时，尽管其他变量数值接近，但成交价格仍有较大差异，车系与车型对价格的影响同样关键。而不同于新车价、里程等能够体现车辆市场以及车况的特征，车系与车型相当于车辆的本身属性，它对成交价格的影响往往不具有统计规律，因此比较难研究车系车型对车辆的价格的影响。虽然我们在模型中使用了 label encoding 处理了车系特征，但是 label encoding 还是没能够很好地表征相应车型车系车辆的内在特性。

解决思路：在研究车系、车型对交易价格的影响时，我们采用图嵌入的方式。要使用图嵌入算法，首先要构造图。将二手车作为图中的节点，将相同车系或者相同车型的车对应的节点连接起来，我们可以控制车系边或车型边的权重，让图嵌入矩阵是更倾向于相同车系还是相同车型。构建好图后，我们可以使用 item2vec 或者 node2vec 算法计算图嵌入矩阵，得到的向量就是表征车系车型的向量。得到表征向量后，我们可以直接把它作为特征的一部分进行模型的训练。

七、模型的评价

7.1 模型的优点

1、本文使用基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型来进行估价，相比于单一模型而言，有利于提高模型的准确性。

2、利用 HEBO 算法来对模型超参数调优，大大提高模型的准确性。

3、使用相关性分析与 XGBoost 算法结合来进行特征筛选，并采用递归特征消除法对重要特征进行检验，以达到准确筛选出相关特征，除去无关及冗余特征的结果，提升算法精度并减少过拟合。

7.2 模型的缺点

1、基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型对价格的预测速度较慢。

2、通过实验发现，剔除边缘样本后，模型预测的效果变好，换而言之，我们的模型对于边缘样本的预测效果稍显不足。

参考文献

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System.[J]. CoRR, 2016, abs/1603.02754

[2]李原吉,李丹,唐淇.基于 XGBoost 算法的依据车辆信息预估二手车成交价格模型[J].电子测试,2021(21):47-49.DOI:10.16520/j.cnki.1000-8519.2021.21.016.

[3] Wang D , Yang Z , Yi Z . LightGBM: An Effective miRNA Classification Method in Breast Cancer Patients[C]// the 2017 International Conference. 2017.

[4] Prokhorenkova L, Gusev G, Vorobev A, et al. CatBoost: unbiased boosting with categorical features[J]. arXiv preprint arXiv:1706.09516, 2017.

[5]赵梦想,胡敏.基于 Stacking 融合的电商平台二手车定价研究[J].信息与电脑(理论版),2021,33(19):35-38.

[6] Cowen-Riv Er S A I , Lyu W , Wang Z , et al. HEBO: Heteroscedastic Evolutionary Bayesian Optimisation[J]. 2020.

附录

附录 1： 问题一 (jupyter lab 编写)

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import random
import numpy as np
import math
import warnings
import missingno as msno
from pprint import pprint
from chinese_calendar import is_workday, is_holiday
from pathlib import Path
from sklearn.preprocessing import PowerTransformer
import scipy.stats as st
from collections import defaultdict

# 固定随机种子，稳定模型效果
random.seed(2021)
np.random.seed(2021)
%matplotlib inline
warnings.filterwarnings("ignore")
plt.rc("font",family="SimHei",size="10")
plt.rcParams['axes.unicode_minus']=False
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
pd.set_option('max_colwidth',200)
```

```
pd.set_option('expand_frame_repr', False)

# 数据集路径
root = Path(os.getcwd()).resolve().parent / "data"
root2 = Path(os.getcwd()).resolve().parent / "pic"
train = root / "附件 1：估价训练数据.txt"
test = root / "附件 2：估价验证数据.txt"

# index2name 存放的是所有特征的名称
index2name = ["车辆 id", "展销时间", "品牌 id", "车系 id", "车型 id", "里程", "车辆颜色", "车辆所在城市 id", "国标码",
              "过户次数", "载客人数", "注册日期", "上牌日期", "国别", "厂商类型", "年款", "排量", "变速箱", "燃油类型",
              "新车价"]

# category 存放的是类别特征的名称
category = ["品牌 id", "车系 id", "车型 id", "车辆颜色", "车辆所在城市 id", "厂商类型", "燃油类型"] # 类别类型特征

# numerical 存放的是数值特征的名称
numerical = ["Feature_5", "年款", "排量", "变速箱", "Feature_2", "Feature_13", "载客人数", "新车价", "里程"] #  "厂商类型"实数类型特征, "Feature_12", "Feature_8"

# cross_num 存放的是要进行特征交叉的特征的名称
cross_num = ["新车价", "Feature_2", "Feature_12_1"]

# 匿名特征为 Feature_i，此处将 15 个匿名特征加入到 index2name 中
for _ in range(1, 16):
    index2name.append("Feature_"+str(_))
index2name.append("target")
```

```
    # i2n 函数：将 dataframe 的 header 改为 index2name
    def i2n(df):
        df = df.rename(columns={index: _ for index, _ in enumerate(index2name)})
        return df

    # metric 为比赛设定的模型评价指标
    def metric(y_, y):
        ape = np.abs(y_ - y) / y
        mape = np.mean(ape)
        accuracy = np.sum(ape <= 0.05) / len(ape)
        # print(f'mape: {float(1 - mape)} accuracy: {float(accuracy)}')
        return float(0.2 * (1 - mape) + 0.8 * accuracy)

    # 读取数据集
    data_df = pd.read_csv(train, sep="\t", header=None)
    data_df = i2n(data_df)
    # 将 target==10900 的那条数据删除
    data_df = data_df[data_df["target"] <= 10000]
    # data_df = data_df[((data_df["新车价"]-data_df["target"]) / data_df["新车价"]) <
0.969]
    # data_df = data_df[data_df["新车价"] > data_df["target"]]

    # 将 data_df 打乱
    data_df.sort_values(by="展销时间", inplace=True)

    # 将时间转为 datetime 类型，便于之后的特征工作
    data_df["展销时间"] = pd.to_datetime(data_df["展销时间"])
    data_df["注册日期"] = pd.to_datetime(data_df["注册日期"])
    data_df["上牌日期"] = pd.to_datetime(data_df["上牌日期"])
```

```
    # ds_size 为数据集的大小，此处将数据集按照 7：3 的比例划分为训练集和验证集，使用训练集训练模型，使用验证集测试模型，提交结果时会将训练集和验证集合并一起训练模型，然后做预测
    ds_size = len(data_df)
    tr_size = int(ds_size * 0.7)
    te_size = ds_size - tr_size

    # 这里做了两个筛除异常数据的操作：1. 将 target 过低的数据删除，主要考虑是一些车况很差的数据，并不多见。2.将 target 比新车价高的数据删除，这些数据有可能是一些绝版车的数据，不多见。
    tr = data_df.iloc[:tr_size]
    tr = tr.sample(tr_size)
    tr = tr[((tr["新车价"]-tr["target"]) / tr["新车价"]) < 0.969]
    tr = tr[tr["新车价"] > tr["target"]]
    te = data_df.iloc[tr_size:]

    # 使用 yeo-johnson 对 target 进行处理，使 target 的分布符合正态分布，为什么使用 yeo-johnson 我会在下一次开会讲
    pt_tr = PowerTransformer(method="yeo-johnson")

    tr["target"] = pt_tr.fit_transform(np.array(tr['target']).reshape(-1,1))
    plt.figure(figsize=[20, 10])
    sns.displot(tr['target'])
    plt.text(2, 800, "Skewness: %f" % tr['target'].skew())
    plt.text(2, 750, "Kurtosis: %f" % tr['target'].kurt())
    plt.savefig(root2 / "b.jpg")

    ## 使用众数填补缺省值
    sFea = {}
    data_df_all = pd.concat([tr, te], axis=0)
```

```
    cx = data_df_all.groupby("车系  id")
    for a, b in cx:
        sFea[a] = list(set(b["车型  id"]))

    for cx_idx in sFea:
        for cxing_idx in sFea[cx_idx]:
            df = data_df_all[(data_df_all["车系  id"]==cx_idx) & (data_df_all["车型
id"]==cxing_idx)]
            for _ in df:
                if _ == "target":
                    continue
                a = df[_].value_counts()
                if len(a) <= 0:
                    continue
                b = a.index[0]
                tr[_] = tr[_].fillna(b)
                te[_] = te[_].fillna(b)

    for _ in tr:
        tr[_] = tr[_].fillna(tr[_].value_counts().index[0])

    for _ in te:
        te[_] = te[_].fillna(te[_].value_counts().index[0])

    # ******************构造新特征********************
    class FeatureEngine:
        def __init__(self):
            self.ID = defaultdict(lambda: defaultdict(int))
            self.sFea = {}
            self.target_cxx = defaultdict(lambda: defaultdict(int))
```

```
    @staticmethod
    def add2list(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            numerical.append(name)

    @staticmethod
    def add2list_cat(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            category.append(name)

    # strongFea 没啥用，忽略
    def strongFea(self, data_df, stage):
        if stage == "train":
            cx = tr.groupby("车系  id")
            for a, b in cx:
                self.sFea[a] = list(set(b["车型  id"]))

            for cx_idx in self.sFea:
                for cxing_idx in self.sFea[cx_idx]:
                    if len(data_df[(data_df["车系  id"]==cx_idx) & (data_df["车型
id"]==cxing_idx)]) <= 5:
                        self.target_cxx[cx_idx][cxing_idx] = 0
                    else:
                        self.target_cxx[cx_idx][cxing_idx] =
float(data_df[(data_df["车系  id"]==cx_idx) & (data_df["车型
id"]==cxing_idx)]["target"].mean())
```

```
                data_df.insert(len(data_df.columns)-1, "车系车型_mean",
data_df.apply(lambda x: self.target_cxx[x["车系  id"]][x["车型  id"]], axis=1))
                data_df["车系车型_mean"] = data_df["车系车型_mean"].fillna(data_df["车系
车型_mean"].mean())
                self.add2list(len(index2name)-1, "车系车型_mean", stage)


        # 下面一系列的 get 函数是类似的，按照某个特征进行聚类，然后求 mean，
count，max.....
        def getMean(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_mean"][list(set(v[fea_name]))[0]] =
v["target"].mean()
                data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
                data_df[fea_name+"_mean"] =
data_df[fea_name+"_mean"].fillna(data_df[fea_name+"_mean"].mean())
            self.add2list(len(index2name)-1, fea_name+"_mean", stage)



        def getCount(self, data_df, fea_name, stage):
            if stage == "train":
                self.ID[fea_name+"_count"] = dict(data_df[fea_name].value_counts())
                data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
```

```
                else:
                    data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
                    data_df[fea_name+"_count"] =
data_df[fea_name+"_count"].fillna(data_df[fea_name+"_count"].value_counts().index[0])
                self.add2list(len(index2name)-1, fea_name+"_count", stage)



        def getMax(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_max"][list(set(v[fea_name]))[0]] =
v["target"].max()
                data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
                data_df[fea_name+"_max"] =
data_df[fea_name+"_max"].fillna(data_df[fea_name+"_max"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_max", stage)



        def getMin(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
```

```
                    self.ID[fea_name+"_min"][list(set(v[fea_name]))[0]] =
v["target"].min()
                    data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
                else:
                    data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
                    data_df[fea_name+"_min"] =
data_df[fea_name+"_min"].fillna(data_df[fea_name+"_min"].value_counts().index[0])
                self.add2list(len(index2name)-1, fea_name+"_min", stage)



        def getMedian(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_median"][list(set(v[fea_name]))[0]] =
v["target"].median()
                    data_df.insert(len(data_df.columns)-1, fea_name+"_median",
data_df[fea_name].map(self.ID[fea_name+"_median"]))
                else:
                    data_df.insert(len(data_df.columns)-1, fea_name+"_median",
data_df[fea_name].map(self.ID[fea_name+"_median"]))
                    data_df[fea_name+"_median"] =
data_df[fea_name+"_median"].fillna(data_df[fea_name+"_median"].value_counts().index[0])
                self.add2list(len(index2name)-1, fea_name+"_median", stage)



        def getSum(self, data_df, fea_name, stage):
```

```
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_sum"][list(set(v[fea_name]))[0]] =
v["target"].sum()
                data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
                data_df[fea_name+"_sum"] =
data_df[fea_name+"_sum"].fillna(data_df[fea_name+"_sum"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_sum", stage)



        def getStd(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_std"][list(set(v[fea_name]))[0]] =
v["target"].std()
                data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
                data_df[fea_name+"_std"] =
data_df[fea_name+"_std"].fillna(data_df[fea_name+"_std"].value_counts().index[0])
```

```
                self.add2list(len(index2name)-1, fea_name+"_std", stage)


        def newFeature(self, data_df, stage="train"):
                # 构造特征
                pd.set_option('mode.use_inf_as_na', True)

                data_df.insert(len(data_df.columns)-1,"展销是否为假期",data_df["展销时间
"].map(lambda x: is_holiday(x)))
                self.add2list_cat(len(index2name)-1, "展销是否为假期", stage)

                data_df.insert(len(data_df.columns)-1,"是否转户", data_df["过户次数
"].map(lambda x: int(x > 0)))
                # self.add2list(len(index2name)-1, "是否转户", stage)
                # self.add2list_cat(len(index2name)-1, "是否转户", stage)

                data_df.insert(len(data_df.columns)-1, "注册年份", data_df["注册日期
"].map(lambda x: x.year))
                self.add2list(len(index2name)-1, "注册年份", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_num_sqrt",
data_df["Feature_12"].map(lambda x: eval(x) ** (1./3.)))
                self.add2list(len(index2name)-1, "Feature_12_num_sqrt", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_0",
data_df["Feature_12"].map(lambda x: int(x.split('*')[0])))
                self.add2list(len(index2name)-1, "Feature_12_0", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_1",
data_df["Feature_12"].map(lambda x: int(x.split('*')[1])))
                self.add2list(len(index2name)-1, "Feature_12_1", stage)
```

```
                data_df.insert(len(data_df.columns)-1, "Feature_12_2",
data_df["Feature_12"].map(lambda x: int(x.split('*')[2])))
                self.add2list(len(index2name)-1, "Feature_12_2", stage)

                data_df.insert(len(data_df.columns)-1, "汽车年龄", (data_df["展销时间"] -
data_df["上牌日期"]).map(lambda x: x.days))
                # self.add2list(len(index2name)-1, "汽车年龄", stage)



                self.getSum(data_df, "车辆所在城市  id", stage)

                self.getMean(data_df, "品牌  id", stage)
                self.getCount(data_df, "品牌  id", stage)
                self.getMax(data_df, "品牌  id", stage)
                self.getMin(data_df, "品牌  id", stage)
                self.getMedian(data_df, "品牌  id", stage)
                self.getSum(data_df, "品牌  id", stage)

                # self.getMean(data_df, "是否转户", stage)
                self.getMean(data_df, "厂商类型", stage)
                self.getMean(data_df, "排量", stage)
                # self.getMean(data_df, "燃油类型", stage)
                self.getMean(data_df, "国标码", stage)
                self.getMean(data_df, "年款", stage)
                self.getMean(data_df, "变速箱", stage)
                self.getMean(data_df, "Feature_2", stage)

                # self.getMean(data_df, "Feature_6", stage)
                self.getMean(data_df, "Feature_8", stage)
```

```
                # self.getMean(data_df, "Feature_9", stage)
                # self.getMean(data_df, "展销是否为假期", stage)
                # self.getCount(data_df, "车型  id", stage)

                data_df["新车价"] = np.log(data_df["新车价"])
                data_df.insert(len(data_df.columns)-1, "评估方法 1",
np.log(data_df.apply(lambda x: x["新车价"] * (0.15 + 0.85*(1 - 0.11*min(3, max(0., x["汽车
年龄"] / 365)) - 0.1 * min(4., max(0, x["汽车年龄"] / 365-3)) - 0.09 * min(4., max(0, x["汽车
年龄"] / 365-7)))), axis=1)))
                self.add2list(len(index2name)-1, "评估方法 1", stage)
                # data_df.insert(len(data_df.columns)-1, "评估方法 2",
np.log(data_df.apply(lambda x: x["新车价"] * (15 - 5 * min(6, max(0, x["里程"])) / 6 - 4 *
min(6, max(0, x["里程"]-6)) / 6 - 3 * min(6, max(0, x["里程"]-12)) / 6 - 2 * min(6, max(0, x["
里程"]-18)) / 6 - min(6, max(0, x["里程"]-24)) / 6) / 15, axis=1)))
                # self.add2list(len(index2name)-1, "评估方法 2", stage)
                # data_df["评估方法 2"].fillna(data_df["评估方法 1"], inplace=True)



        def crossFeature(self, data_df, stage="train"):
                #  特征交叉
                for f1 in data_df:
                        for f2 in data_df:
                                if f1 != f2 and f1 in cross_num and f2 in cross_num and (f2+'_'+f1)
not in index2name:
                                        data_df.insert(len(data_df.columns)-1, f1+'_'+f2, (data_df[f1] -
data_df[f1].mean()) / data_df[f1].std() * data_df[f2].mean()/ data_df[f2].std())
                                        self.add2list(len(index2name)-1, f1+'_'+f2, stage)


    #  生成新特征
    FE = FeatureEngine()
```

```
FE.newFeature(tr, "train")
FE.newFeature(te, "test")
# FE.crossFeature(tr, "train")
# FE.crossFeature(te, "test")

# 取出 numerical 特征和 categories 特征
tr_x = tr.iloc[:, :-1]
tr_y = tr.iloc[:, -1:]
te_x = te

tr_x_num = tr[numerical].astype("float")
te_x_num = te[numerical].astype("float")
tr_x_cat = tr[category]
te_x_cat = te[category]

# 特征归一化：  (x-min) / (max-min)
for _ in tr_x_num:
    if not _ in ["厂商类型", "载客人数", "Feature_8", "是否转户"]:
        min_ = tr_x_num[_].min()
        max_ = tr_x_num[_].max()
        tr_x_num[_] = (tr_x_num[_] - min_) / (max_ - min_)
        te_x_num[_] = (te_x_num[_] - min_) / (max_ - min_)

tr_x_num_array = np.array(tr_x_num)
tr_y_array = np.array(tr_y)
te_x_num_array = np.array(te_x_num)

# 取出 numerical 特征和 categories 特征
tr_x = tr.iloc[:, :-1]
tr_y = tr.iloc[:, -1:]
```

```
te_x = te.iloc[:, :-1]
te_y = te.iloc[:, -1:]
tr_x_num = tr[numerical].astype("float")
te_x_num = te[numerical].astype("float")
tr_x_cat = tr[category]
te_x_cat = te[category]

# 特征归一化：  (x-min) / (max-min)
for _ in tr_x_num:
    if not _ in ["厂商类型", "载客人数", "Feature_8", "是否转户"]:
        min_ = tr_x_num[_].min()
        max_ = tr_x_num[_].max()
        tr_x_num[_] = (tr_x_num[_] - min_) / (max_ - min_)
        te_x_num[_] = (te_x_num[_] - min_) / (max_ - min_)

tr_x_num_array = np.array(tr_x_num)
tr_y_array = np.array(tr_y)
te_x_num_array = np.array(te_x_num)
te_y_array = np.array(te_y)

# 协方差矩阵热力图
plt.figure(figsize=(20,10))
tr_num = pd.concat([tr_x_num, tr_y], axis=1)
sns.heatmap(tr_num.corr(), annot=True, fmt='.2f')

from xgboost import XGBRegressor
from lightgbm.sklearn import LGBMRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor
```

```
from pprint import pprint

lgb_params = {"num_leaves": 256, "reg_alpha": 0., "reg_lambda": 0.01, "objective": 'mse', "max_depth": -1,
                "learning_rate": 0.03, "min_child_samples": 25,    "n_estimators": 2000, "subsample": 0.7, "colsample_bytree": 0.45}
xgb_params = {'learning_rate': 0.09995869693725284,
                'subsample': 0.869693514217955,
                'colsample_bytree': 0.9999236721373504,
                'lambda': 0.19406502844686765,
                'alpha': 0.07701113887171819,
                'n_estimators': 2000}
cat_params = {'n_estimators': 1000,
                # 'score_function': 'NewtonL2',
                "min_child_samples": 25,
                'learning_rate':0.03,
                'loss_function':'RMSE',
                'verbose': False,
                'l2_leaf_reg': 0.01,
                'task_type': 'CPU'}

from sklearn.metrics import r2_score, mean_squared_error
from sklearn_tuner import sklearn_tuner
from sklearn.metrics import mean_absolute_error

xgb_cfg = [
    {'name' : "learning_rate", 'type' : 'num', 'lb' : 0.01, 'ub' : 0.1},
    {'name' : "subsample", 'type' : 'num', 'lb' : 0.3, 'ub' : 1.0},
    {'name' : "colsample_bytree", 'type' : 'num', 'lb' : 0.1, 'ub' : 1.0},
    {'name' : "lambda", 'type' : 'num', 'lb' : 0, 'ub' : 0.2},
```

```
            {'name' : "alpha", 'type' : 'num', 'lb' : 0, 'ub' : 0.2},
            {'name' : "max_depth", 'type' : 'int', 'lb' : 10, 'ub' : 50},
            {'name' : "min_child_weight", 'type' : 'int', 'lb' : 1, 'ub' : 10},
            {'name' : "n_estimators", 'type' : 'int', 'lb' : 500, 'ub' : 2000},
            ]

        lgb_cfg = [
            {'name' : "learning_rate", 'type' : 'num', 'lb' : 0.01, 'ub' : 0.1},
            {'name' : "subsample", 'type' : 'num', 'lb' : 0.3, 'ub' : 1.0},
            {'name' : "colsample_bytree", 'type' : 'num', 'lb' : 0.1, 'ub' : 1.0},
            {'name' : "reg_lambda", 'type' : 'num', 'lb' : 0, 'ub' : 0.2},
            {'name' : "reg_alpha", 'type' : 'num', 'lb' : 0, 'ub' : 0.2},
            {'name' : "num_leaves", 'type' : 'int', 'lb' : 100, 'ub' : 500},
            {'name' : "max_depth", 'type' : 'int', 'lb' : 10, 'ub' : 50},
            {'name' : "n_estimators", 'type' : 'int', 'lb' : 500, 'ub' : 2000},
            ]

        cat_cfg = [
            {'name' : "learning_rate", 'type' : 'num', 'lb' : 0.01, 'ub' : 0.1},
            {'name' : "l2_leaf_reg", 'type' : 'int', 'lb' : 1, 'ub' : 10},
            {'name' : "max_depth", 'type' : 'int', 'lb' : 5, 'ub' : 10},
            {'name' : "n_estimators", 'type' : 'int', 'lb' : 500, 'ub' : 2000},
            {'name' : "min_child_samples", 'type' : 'int', 'lb' : 5, 'ub' : 30},
            ]

        lgbParams = sklearn_tuner(LGBMRegressor, lgb_cfg, tr_x_num_array,
tr_y_array.ravel(), metric = mean_squared_error, max_iter = 25, greater_is_better = False)
        catParams = sklearn_tuner(CatBoostRegressor, cat_cfg, tr_x_num_array,
tr_y_array.ravel(), metric = mean_squared_error, max_iter = 25, greater_is_better = False)
        xgbParams = sklearn_tuner(XGBRegressor, xgb_cfg, tr_x_num_array, tr_y_array.ravel(),
```

```
metric = mean_squared_error, max_iter = 25, greater_is_better = False)


    for _ in xgb_params:
        if _ not in xgbParams:
            xgbParams[_] = xgb_params[_]


    for _ in lgb_params:
        if _ not in lgbParams:
            lgbParams[_] = lgb_params[_]


    for _ in cat_params:
        if _ not in catParams:
            catParams[_] = cat_params[_]


    from xgboost import plot_importance
    from numpy import sort
    from sklearn.feature_selection import SelectFromModel


    # Fit model using each importance as a threshold
    model_XGB = XGBRegressor(**xgbParams, importance_type="weight").fit(tr_x_num,
tr_y.target)
    thresholds = sort(model_XGB.feature_importances_)
    f_i = dict(zip(numerical, model_XGB.feature_importances_))
    f_i = sorted(f_i.items(), key=lambda x: x[1])
    print(f_i)


    fig,ax = plt.subplots(figsize=(15,15))
    plot_importance(model_XGB, importance_type="weight", ax=ax)
    plt.show()
```

```
from sklearn.model_selection import KFold
from sklearn.base import BaseEstimator, RegressorMixin, TransformerMixin, clone
import numpy as np

# stacking
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # 将原来的模型 clone 出来，并且进行实现 fit 功能
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=42)

        #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
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                self.base_models_[i].append(instance)
                out_of_fold_predictions[holdout_index, i] = y_pred

        # 使用次级训练集来训练次级学习器
        self.meta_model_.fit(out_of_fold_predictions, y)
```

```
            return self


        #在上面的 fit 方法当中，我们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
        #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
        def predict(self, X):
            meta_features = np.column_stack([
                np.column_stack([model.predict(X) for model in base_models]).mean(axis=1)
                for base_models in self.base_models_ ])
            return self.meta_model_.predict(meta_features)


    base_models = [
                    CatBoostRegressor(**catParams),
                    LGBMRegressor(**lgbParams),
                    XGBRegressor(**xgbParams),
                   ]


    meta_model = LinearRegression()
    stacking_model = StackingAveragedModels(base_models=base_models, meta_model=meta_model)


    stacking_model.fit(tr_x_num_array, tr_y_array.ravel())
    y_predict = stacking_model.predict(te_x_num_array)
```

附录 2： 问题二 (jupyter lab 编写)

```
    import pandas as pd
    import os
    import matplotlib.pyplot as plt
```

```
import matplotlib
import seaborn as sns
import random
import numpy as np
import math
import warnings
import missingno as msno
from pprint import pprint
from chinese_calendar import is_workday, is_holiday
from pathlib import Path
from sklearn.preprocessing import PowerTransformer
import scipy.stats as st
from collections import defaultdict

# 固定随机种子，稳定模型效果
random.seed(2021)
np.random.seed(2021)
%matplotlib inline
warnings.filterwarnings("ignore")
plt.rc("font",family="SimHei",size="10")
plt.rcParams['axes.unicode_minus']=False
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
pd.set_option('max_colwidth',200)
pd.set_option('expand_frame_repr', False)

# 数据集路径
root = Path(os.getcwd()).resolve().parent / "data"
root2 = Path(os.getcwd()).resolve().parent / "pic"
train = root / "Q2.xlsx"
```

```
    data_df = pd.read_excel(train)

    # index2name 存放的是所有特征的名称
    index2name = ["carid", "pushDate", "pushPrice", "updatePriceTimeJson", "pullDate", "
展销时间",
                  "品牌 id", "车系 id", "车型 id", "里程", "车辆颜色", "车辆所在城
市 id", "国标码",
                  "过户次数", "载客人数", "注册日期", "上牌日期", "国别", "厂商类
型", "年款", "排量", "变速箱", "燃油类型",
                  "新车价"]

    # category 存放的是类别特征的名称
    category = ["品牌 id", "车系 id", "车型 id", "车辆颜色", "车辆所在城市 id", "厂商
类型", "燃油类型"] # 类别类型特征

    # numerical 存放的是数值特征的名称
    numerical = ["Feature_5", "年款", "排量", "变速箱", "Feature_2", "Feature_13", "载客
人数", "新车价", "里程", "target"] #  "厂商类型"实数类型特征, "Feature_12", "Feature_8"

    # cross_num 存放的是要进行特征交叉的特征的名称
    cross_num = ["新车价", "Feature_2", "Feature_12_1"]

    # 匿名特征为 Feature_i，此处将 15 个匿名特征加入到 index2name 中
    for _ in range(1, 16):
        index2name.append("Feature_"+str(_))
    index2name.append("target")
    index2name.append("成交周期")

    sFea = {}
    cx = data_df.groupby("车系 id")
```

```
    for a, b in cx:
        sFea[a] = list(set(b["车型  id"]))

    # 将 target==10900 的那条数据删除
    data_df = data_df[data_df["target"] <= 10000]
    # data_df = data_df[((data_df["新车价"]-data_df["target"]) / data_df["新车价"]) <
0.969]
    # data_df = data_df[data_df["新车价"] > data_df["target"]]

    # 将 data_df 打乱
    data_df.sort_values(by="展销时间", inplace=True)

    # 将时间转为 datetime 类型，便于之后的特征工作
    data_df["展销时间"] = pd.to_datetime(data_df["展销时间"])
    data_df["注册日期"] = pd.to_datetime(data_df["注册日期"])
    data_df["上牌日期"] = pd.to_datetime(data_df["上牌日期"])
    data_df["pushDate"] = pd.to_datetime(data_df["pushDate"])
    data_df["pullDate"] = pd.to_datetime(data_df["pullDate"])


    # 这里做了两个筛除异常数据的操作：1. 将 target 过低的数据删除，主要考虑是一
些车况很差的数据，并不多见。2.将 target 比新车价高的数据删除，这些数据有可能是
一些绝版车的数据，不多见。
    tr = data_df
    tr = tr[((tr["新车价"]-tr["target"]) / tr["新车价"]) < 0.969]
    tr = tr[tr["新车价"] > tr["target"]]

    # 使用众数填补缺省值
    for _ in tr:
        tr[_] = tr[_].fillna(tr[_].value_counts().index[0])
```

```
# ******************构造新特征********************
class FeatureEngine:
    def __init__(self):
        self.ID = defaultdict(lambda: defaultdict(int))
        self.sFea = {}
        self.target_cxx = defaultdict(lambda: defaultdict(int))

    @staticmethod
    def add2list(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            numerical.append(name)

    @staticmethod
    def add2list_cat(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            category.append(name)

    # strongFea 没啥用，忽略
    def strongFea(self, data_df, stage):
        if stage == "train":
            cx = tr.groupby("车系  id")
            for a, b in cx:
                self.sFea[a] = list(set(b["车型  id"]))

            for cx_idx in self.sFea:
                for cxing_idx in self.sFea[cx_idx]:
                    if len(data_df[(data_df["车系  id"]==cx_idx) & (data_df["车型
```

```
id"]==cxing_idx)]) <= 5:
                                    self.target_cxx[cx_idx][cxing_idx] = 0
                                else:
                                    self.target_cxx[cx_idx][cxing_idx] =
float(data_df[(data_df["车系 id"]==cx_idx) & (data_df["车型
id"]==cxing_idx)]["target"].mean())

            data_df.insert(len(data_df.columns)-1, "车系车型_mean",
data_df.apply(lambda x: self.target_cxx[x["车系 id"]][x["车型 id"]], axis=1))
            data_df["车系车型_mean"] = data_df["车系车型_mean"].fillna(data_df["车系
车型_mean"].mean())
            self.add2list(len(index2name)-1, "车系车型_mean", stage)

        # 下面一系列的 get 函数是类似的，按照某个特征进行聚类，然后求 mean，
count，max.....
        def getMean(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_mean"][list(set(v[fea_name]))[0]] =
v["target"].mean()
                data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
                data_df[fea_name+"_mean"] =
data_df[fea_name+"_mean"].fillna(data_df[fea_name+"_mean"].mean())
            self.add2list(len(index2name)-1, fea_name+"_mean", stage)
```

```
        def getCount(self, data_df, fea_name, stage):
            if stage == "train":
                self.ID[fea_name+"_count"] = dict(data_df[fea_name].value_counts())
                data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
                data_df[fea_name+"_count"] =
data_df[fea_name+"_count"].fillna(data_df[fea_name+"_count"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_count", stage)


        def getMax(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_max"][list(set(v[fea_name]))[0]] =
v["target"].max()
                data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
                data_df[fea_name+"_max"] =
data_df[fea_name+"_max"].fillna(data_df[fea_name+"_max"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_max", stage)
```

```
        def getMin(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_min"][list(set(v[fea_name]))[0]] =
v["target"].min()
                data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
                data_df[fea_name+"_min"] =
data_df[fea_name+"_min"].fillna(data_df[fea_name+"_min"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_min", stage)



        def getMedian(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_median"][list(set(v[fea_name]))[0]] =
v["target"].median()
                data_df.insert(len(data_df.columns)-1, fea_name+"_median",
data_df[fea_name].map(self.ID[fea_name+"_median"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_median",
```

```
data_df[fea_name].map(self.ID[fea_name+"_median"]))
                    data_df[fea_name+"_median"] =
data_df[fea_name+"_median"].fillna(data_df[fea_name+"_median"].value_counts().index[0])
                self.add2list(len(index2name)-1, fea_name+"_median", stage)



        def getSum(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_sum"][list(set(v[fea_name]))[0]] =
v["target"].sum()
                data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
                data_df[fea_name+"_sum"] =
data_df[fea_name+"_sum"].fillna(data_df[fea_name+"_sum"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_sum", stage)



        def getStd(self, data_df, fea_name, stage):
            if stage == "train":
                cs = data_df.loc[:, [fea_name, "target"]]
                cs = cs.groupby(fea_name)
                for k, v in cs:
                    self.ID[fea_name+"_std"][list(set(v[fea_name]))[0]] =
v["target"].std()
```

```
                    data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
                else:
                    data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
                    data_df[fea_name+"_std"] =
data_df[fea_name+"_std"].fillna(data_df[fea_name+"_std"].value_counts().index[0])
                self.add2list(len(index2name)-1, fea_name+"_std", stage)

        def newFeature(self, data_df, stage="train"):
            # 构造特征
            pd.set_option('mode.use_inf_as_na', True)

            data_df.insert(len(data_df.columns)-1, "降价比率", (data_df["新车价"] -
data_df["target"]) / data_df["新车价"])
            self.add2list(len(index2name)-1, "降价比率", stage)

            data_df.insert(len(data_df.columns)-1,"展销是否为假期",data_df["展销时间
"].map(lambda x: is_holiday(x)))
            self.add2list_cat(len(index2name)-1, "展销是否为假期", stage)

            data_df.insert(len(data_df.columns)-1,"是否转户", data_df["过户次数
"].map(lambda x: int(x > 0)))
            # self.add2list(len(index2name)-1, "是否转户", stage)
            # self.add2list_cat(len(index2name)-1, "是否转户", stage)

            data_df.insert(len(data_df.columns)-1, "注册年份", data_df["注册日期
"].map(lambda x: x.year))
            self.add2list(len(index2name)-1, "注册年份", stage)
```

```
                data_df.insert(len(data_df.columns)-1, "Feature_12_num_sqrt",
data_df["Feature_12"].map(lambda x: eval(x) ** (1./3.)))
                self.add2list(len(index2name)-1, "Feature_12_num_sqrt", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_0",
data_df["Feature_12"].map(lambda x: int(x.split('*')[0])))
                self.add2list(len(index2name)-1, "Feature_12_0", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_1",
data_df["Feature_12"].map(lambda x: int(x.split('*')[1])))
                self.add2list(len(index2name)-1, "Feature_12_1", stage)

                data_df.insert(len(data_df.columns)-1, "Feature_12_2",
data_df["Feature_12"].map(lambda x: int(x.split('*')[2])))
                self.add2list(len(index2name)-1, "Feature_12_2", stage)

                data_df.insert(len(data_df.columns)-1, "汽车年龄", (data_df["展销时间"] -
data_df["上牌日期"]).map(lambda x: x.days))
                # self.add2list(len(index2name)-1, "汽车年龄", stage)


                self.getSum(data_df, "车辆所在城市  id", stage)

                self.getMean(data_df, "品牌  id", stage)
                self.getCount(data_df, "品牌  id", stage)
                self.getMax(data_df, "品牌  id", stage)
                self.getMin(data_df, "品牌  id", stage)
                self.getMedian(data_df, "品牌  id", stage)
                self.getSum(data_df, "品牌  id", stage)
```

```
                # self.getMean(data_df, "是否转户", stage)
                self.getMean(data_df, "厂商类型", stage)
                # self.getMean(data_df, "燃油类型", stage)
                self.getMean(data_df, "国标码", stage)
                self.getMean(data_df, "年款", stage)
                self.getMean(data_df, "变速箱", stage)
                self.getMean(data_df, "Feature_2", stage)
                # self.getMean(data_df, "Feature_6", stage)
                self.getMean(data_df, "Feature_8", stage)
                # self.getMean(data_df, "Feature_9", stage)
                # self.getMean(data_df, "展销是否为假期", stage)
                # self.getCount(data_df, "车型  id", stage)

                data_df["新车价"] = np.log(data_df["新车价"])
                data_df.insert(len(data_df.columns)-1, "评估方法 1",
np.log(data_df.apply(lambda x: x["新车价"] * (0.15 + 0.85*(1 - 0.11*min(3, max(0., x["汽车
年龄"] / 365)) - 0.1 * min(4., max(0, x["汽车年龄"] / 365-3)) - 0.09 * min(4., max(0, x["汽车
年龄"] / 365-7)))), axis=1)))
                self.add2list(len(index2name)-1, "评估方法 1", stage)
                data_df.insert(len(data_df.columns)-1, "评估方法 2",
np.log(data_df.apply(lambda x: x["新车价"] * (15 - 5 * min(6, max(0, x["里程"])) / 6 - 4 *
min(6, max(0, x["里程"]-6)) / 6 - 3 * min(6, max(0, x["里程"]-12)) / 6 - 2 * min(6, max(0, x["
里程"]-18)) / 6 - min(6, max(0, x["里程"]-24)) / 6) / 15, axis=1)))
                self.add2list(len(index2name)-1, "评估方法 2", stage)
                data_df["评估方法 2"].fillna(data_df["评估方法 1"], inplace=True)

    # 生成新特征
    FE = FeatureEngine()
    FE.newFeature(tr, "train")
    # FE.newFeature(te, "test")
```

```
# FE.crossFeature(tr, "train")
# FE.crossFeature(te, "test")

# 取出 numerical 特征和 categories 特征
tr_x = tr.iloc[:, :-1]
tr_y = tr.iloc[:, -1:]

tr_x_num = tr[numerical].astype("float")
tr_x_cat = tr[category]

# 特征归一化： (x-min) / (max-min)
for _ in tr_x_num:
    if not _ in ["厂商类型", "载客人数", "Feature_8", "是否转户"]:
        min_ = tr_x_num[_].min()
        max_ = tr_x_num[_].max()
        tr_x_num[_] = (tr_x_num[_] - min_) / (max_ - min_)

tr_x_num_array = np.array(tr_x_num)
tr_y_array = np.array(tr_y)

# 协方差矩阵热力图
plt.figure(figsize=(20,10))
tr_num = pd.concat([tr_x_num, tr_y], axis=1)
sns.heatmap(tr_num.corr(), annot=True, fmt='.2f')
plt.savefig(root2 / "corr_q2.jpg")

from xgboost import XGBRegressor
from lightgbm.sklearn import LGBMRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

```
from catboost import CatBoostRegressor
from pprint import pprint

# 优化结果：模型参数(使用 hebo 进行参数优化，因为一次优化的时间消耗大，因此将优化结果存了下来)
lgbParams = {'learning_rate': 0.010035982594961306, 'subsample': 0.35028462644730735, 'colsample_bytree': 0.3796965007412236, 'reg_lambda': 0.0006865914755758266, 'reg_alpha': 0.0026163019185854106, 'num_leaves': 189, 'max_depth': 37, 'n_estimators': 1965, 'objective': 'mse', 'min_child_samples': 25}
xgbParams = {'learning_rate': 0.010211967985703159, 'subsample': 0.30001115463199896, 'colsample_bytree': 0.48460986059028555, 'lambda': 0.032780563044634864, 'alpha': 0.06981554843421037, 'max_depth': 26, 'min_child_weight': 10, 'n_estimators': 1845}
catParams = {'learning_rate': 0.08491056514750207, 'l2_leaf_reg': 1, 'max_depth': 8, 'n_estimators': 2000, 'min_child_samples': 25, 'loss_function': 'RMSE', 'verbose': False, 'task_type': 'CPU'}

from xgboost import plot_importance
from numpy import sort
from sklearn.feature_selection import SelectFromModel

# Fit model using each importance as a threshold
model_XGB = XGBRegressor(**xgbParams, importance_type="weight").fit(tr_x_num, tr_y["成交周期"])
thresholds = sort(model_XGB.feature_importances_)
f_i = dict(zip(numerical, model_XGB.feature_importances_))
f_i = sorted(f_i.items(), key=lambda x: x[1])
print(f_i)

fig,ax = plt.subplots(figsize=(15,15))
```

```
plot_importance(model_XGB, importance_type="weight", ax=ax)
plt.savefig(root2 / "feature_importance_q2.jpg")
plt.show()
```