二手车交易数据综合分析建模案例

本案例涉及到数据类型转换，缺失值处理，异常值处理等常见的数据预处理方法，涉及到二手车业务的相关主题分析、数据探索分析以及预测建模分析，涵盖了 Python 语言中的 Numpy，pandas，matplotlib，seaborn，库，以及第三方库 sklearn、xgboost、lightgbm 机器学习库。通过本案例学生将学会常见的数据分析流程以及预测建模方法的应用以及可视化分析。

本案例适合学生在学完《大数据分析与挖掘》或相关课程后的配套实践，也可以作为学生在学完大数据专业课程后的综合实践，适合应用型本科以及高职院校大数据，应用统计专业学生学习。

难易程度: ★★★★★　　复杂程度: ★★★★

1 案例简介

本节介绍案例目的，适用对象，时间安排，预备知识，硬件要求，软件工具，数据集，案例任务,案例背景，基本思路等内容。

1.1案例目的

通过实践本次案例，学生可以达到以下目的:

- 学会应用 Numpy 库以及常见的操作方法
- 学会应用 pandas 库以及常见的操作方法
- 学会通过 matplotlib 库、seaborn 库绘制柱状图等常见可视化图形
- 学会通过可视化图形进行可视化分析
- 学会数据建模原理及方法的应用
- 学会数据挖掘原理及常见的回归分析方法（GBDT 算法、XGBoost 算法、Lightgbm 算法）
- 学会通过 sklearn 库应用到回归分析
- 熟悉数据类型转换，异常值处理，缺失值处理等常见的数据预处理方法
- 学会探索性分析数据的要素以及方法。
- 了解数据预处理的前提以及常见形式

- 了解数据特征提取的常见形式与方法
- 学会对分析数据进行特征筛选
- 熟悉模型的调参方法（网格搜索法）及模型调参过程

1.2适用对象

- 高校（高职）教师
- 高校（高职）学生
- 大数据学习者（有一定的大数据分析基础）
- 数据挖掘分析者

1.3时间安排

本案例适合学生在学完《大数据分析与挖掘》或相关课程后的配套的“大作业”实践，也可以作为学生在学完大数据专业课程后的综合实践案例之一，建议放在第 6 或第 7 学期，也可以作为学生寒暑假期大数据实习基础案例。学时 12 学时（1.5 天）。

1.4 预备知识

需要学生熟悉 Python 语言，基本语法，常见语义等；特别是常见库中的 Numpy，pandas，matplotlib，seaborn 的方法应用；熟练掌握 Jupyter notbook 的操作方法；熟悉如何安装第三方库，熟悉第三方库(sklearn 机器学习库)方法应用；掌握一定的数据挖掘方法，特别是回归分析方法的原理与应用，掌握回归分析的常见算法；了解常见的数据预处理的时机与方法；熟悉常见的数据建模方法，有一定的数据建模应用分析能力；需要了解一定的人工智能基础知识。

1.5 硬件要求

本案例单机要完成，无需要联网也可以操作（如果将数据集部署至私有云，需要局域网访问数据集）。单机上比较流畅完成本案例，建议计算机硬盘配备 500G 以上硬盘，4G 以上内存。

1.6 软件工具

Anaconda3（64-bit）+Jupyter notbook

Python 3.7

1.7 数据集

本数据集来自 eBay 网上刊登的所有二手车售卖信息爬取数据结果。其中包括广告爬取时间、汽车交易名称、销售方、报价类型、交易价格等 20 个字段,371539 条记录，总计大小 65.2M。

1.8 案例任务

本案例需要完成以下实验任务

- 安装 Anaconda3（64-bit）+Jupyter notbook
- 安装 Python 3.7
- 安装第三方库 sklearn、xgboost、lightgbm
- 数据准备及认识数据
- 实现数据加载
- 实现缺失值分析处理
- 实现异常值分析处理
- 实现列统计值处理
- 设置数据清洗条件
- 实现数据变换处理
- 实现特征分析处理
- 实现数据可视化及结果分析
- 实现特征筛选及模型构建
- 实现模型融合及模型评估

1.9 案例背景

近年来，随着我国经济持续快速增长，国内汽车保有量也在稳步增长。由于消费观念的转变，越来越多的人选择购买二手车作为出行工具，二手车市场需求不断增加，规

模也在不断扩大。由于"互联网+"战略的快速实施，我国二手车交易市场进入新的发展阶段，各项数据均表明我国二手车市场发展势头良好，拥有非常广阔的发展空间。然而，与国外成熟的二手车市场相比，我国二手车市场起步较晚，仍然存在一些问题。在二手车交易中，无论对二手车购买用户还是经销商来说，对二手车的保值率进行合理的估值都是至关重要的。因此，快速并且准确地评估二手车的保值率，不仅能加速二手车市场流通的速度，还能保证买卖双方利益达到最大化。目前二手车的估价还主要是由评估师根据自身的经验进行，评估的随意性较大。利用数据挖掘或者其他方法来建立二手车相关评估模型的研究还处在初步探索阶段，至今还没能找到一种操作方便，准确率高的二手车评估模型。本文将借鉴国外车辆价格评估的经验，采用数据挖掘方法预测二手车保值率，建立合理的二手车保值率预测模型。

1.10 案例目标

1.数据分析

通过分析数据，帮助顾客了解二手车销售信息，帮助选择合适的二手车品牌。

（1）最近 20 年不同品牌二手车辆销售数量。

（2）二手汽车售价分布情况：主要分为总体二手车售价分布，和不同品牌二手车售价分布。

（3）汽车行驶距离分布：根据网络还有和有经验的司机了解，车辆行驶距离在 5 万到 15 万公里之间，是汽车性能最好的时候。

（4）车辆使用年限：用网上爬取数据时间的年月减去汽车注册的年月得到车辆使用年限。

（5）车辆有损坏还没修复：车辆有无损坏情况是顾客非常关心的事情。

2.预测建模

（1）建立特征工程，围绕“价格”目标属性，建立关于影响价格的主要特征。

（2）建立二手车价值损失模型。通过该模型了解汽车每年的保值率是多少，从而比较客观估算二手车价格。

1.11 基本思路

1.数据分析:

- 其中在数据分析中的第一个业务问题：最近 20 年不同品牌二手车辆销售数量。

主要关注的字段是"汽车首次注册的年份"，"品牌"，分析最近 20 年不同品牌的二手车辆销售数量;

- 在分析第二个业务问题：二手汽车售价分布情况。主要关注的字段是"价格"，"品牌"，计算了解全体二手车平均售价，以及其售价的四分位数，并考虑不同品牌二手车的平均售价;
- 在分析第三个业务问题：汽车行驶距离分布。主要关注的字段"已经行驶的里程数"，计算得到全体二手车平均行驶距离，并计算得到行驶距离在 5 到 15 万公里车辆的数量;
- 在分析第四个业务问题：车辆使用年限。主要关注的字段"车辆首次注册年份","车辆首次注册的月份","广告抓取的时间",定义爬取数据的年月减去注册年月，记为车辆使用月数;
- 在分析第五个业务问题：有无损坏。主要关注的字段"车辆有损坏还没修复"，评估车辆损坏维修情况。

2.预测建模:

- 为了达到数据建模所用的高质量的数据，需要进行数据清洗，从重复的、空值中清除数据并为列选择合理的范围可视化分析-了解数据是如何跨类别分布的特征工程-查看各特征对“价格”的影响大小，并进行关联分析，提取部分特征并对属性进行必要标准化处理，处理结果做为建模工程的前提工作;
- 数据建模工程：建立二手车价值损失模型。通过该模型了解汽车每年的保值率是多少，从而比较客观估算二手车价格。在参数设定方面将使用网格搜索方式设置最优参数，训练 GBDT 模型、XGBoost 模型、LightGBM 模型,最后通过模型融合 stacking 得到最终的预测结果。

2.实验步骤

2.1 实验步骤概述

步骤零：实验环境准备

步骤一：数据准备

步骤二：数据探索分析

步骤三：数据预处理

步骤四：特征工程

步骤五：建模调参

步骤六：模型融合

下面给出每个步骤所需要的具体实验内容。

步骤零：（1）安装 Anaconda3（64-bit）+Jupyter notbook

（2）安装 Python 3。7

（3）安装第三方库

【备注】：本手册默认环境已经预制，如果需要安装请具体参考软件安装手册。

步骤一：（1）数据准备

（2）数据解读

（3）新建 notbook 文件

（4）数据读取

步骤二：（1）查看数据中各字段的数据类型、空值情况

（2）查看数据中各字段不同取值个数情况

（3）查看数据的描述性信息

步骤三：（1）数据类型转换

（2）缺失值处理

（3）异常值处理

步骤四：（1）衍生变量

（2）数据分析

（3）特征筛选

步骤五：（1）数据集划分

（2）建模过程

步骤六：（1）模型融合

2.2 步骤一：数据准备

2.2.1 数据解读

数据集名称：autos.csv （13.5MB）

数据来源：爬取来自 eBay 网上刊登的所有二手车售卖信息。

数据描述：数据集包括广告爬取时间、汽车交易名称、销售方、报价类型、交易价格等 20 个字段,371539 条记录，总计大小 65.2M。(具体见数据说明)。

数据说明:

dateCrawled :这个广告第一次被抓取时间，所有的字段值都从这个日期开始

name :汽车交易名称

seller：销售方

offerType：报价类型

price：二手车交易价格（预测目标）

abtest:：AB 测试

vehicleType：车辆类型

yearOfRegistration :车辆首次注册年份

gearbox：变速箱

powerPS：汽车在 PS 中的功率

model：车型编码

kilometer：已经行驶的里程数

monthOfRegistration：车辆首次注册的月份

fuelType：燃料类型

brand：汽车品牌

notRepairedDamage :车辆有损坏还没修复

dateCreated :在 ebay 首次创建广告的时间

nrOfPictures :广告中的图片数量

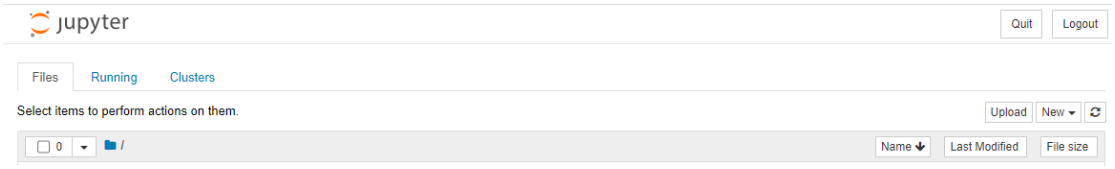
postalCode：邮政编码

lastSeenOnline :当爬虫最后在网上看到这个广告的时候

2.2.2 进入 jupyter notebook

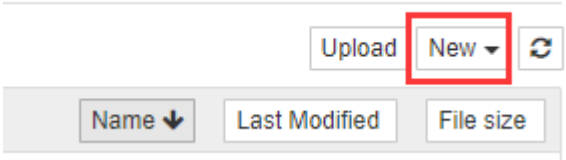
(1):打开 Anaconda Prompt (anaconda), 切换到相应的路径下, 然后输入 jupyter notebook 命令打开 jupyter notebook，结果如下

（2）:进入 Jupyter 的 Home 界面，结果如下

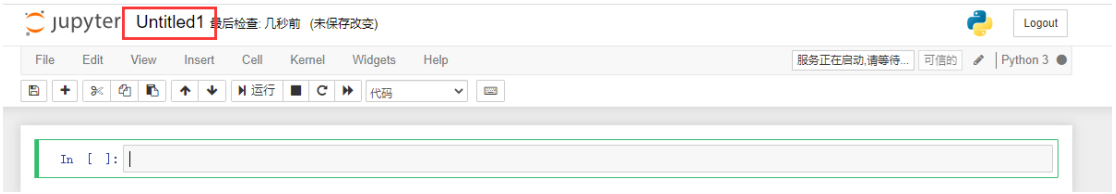


2.2.3 新建 notbook 文件

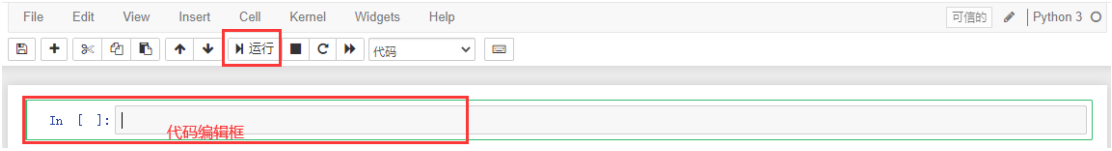
（1）在 Jupyter notbook 界面，选择“new-Python3”（如下图）。



（2）创建完成后如下图，点击“Untitled1”重命名文件名为“二手车交易数据综合分析建模”。



（3）接下来就可以编写代码，编写代码框如下图，编写完成后按“运行”按钮运行。



【注】编写代码前先切换工作目录，之后数据加载与保存就不需要加绝对路径了。

2.2.4 数据读取

首先切换工作路径，然后读取数据，对于原始数据中的列名全部是英文的，进行重命

名，改为相应的中文，方便后期对数据的理解。

代码如下:

```
# 加载所需模块
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
import random
import scipy.stats as st
from sklearn.preprocessing import LabelEncoder
from sklearn import linear_model
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV,cross_val_score
import lightgbm as lgb
import xgboost as xgb
from sklearn.metrics import r2_score,mean_squared_error
import os
%matplotlib inline
os.chdir("E:\\项目\\Kleinanzeigen 二手车数据集(1026)") #切换工作目录
df = pd.read_csv('autos.csv', sep=',', quotechar='"',header=0, encoding='cp1252', engine='python',error_bad_lines=False) #数据读取，第一行数据为表头，编码为 cp1252
df.head() #查看前 5 行数据
colNameDict={'dateCrawled':'广告抓取时间','name':'汽车交易名称','seller':'销售方','offerType':'报价类型','price':'交易价格','abtest':'AB 测试','vehicleType':'车辆类型','yearOfRegistration':'车辆首次注册年份','gearbox':'汽车变速箱','powerPS':'发动机功率','model':'车型编码','kilometer':'已经行驶的里程数','monthOfRegistration':'车辆首次注册的月份','fuelType':'燃料类型','brand':'汽车品牌','notRepairedDamage':'车辆有损坏还没修复','dateCreated':'广告创建时间','nrOfPictures':'广告中的图片数量','postalCode':'邮政编码
```

```
','lastSeen':'最后一次浏览日期'} #定义列名修改字典
df.rename(columns=colNameDict,inplace=True) #列名重命名
df.head() #读取前 5 行数据
```

重命名后部分结果如下表 1:

表 1：重命名后数据结果

	广告抓取时间	汽车交易名称	销售方	报价类型	交易价格	AB测试	车辆类型	车辆首次注册年份	汽车变速箱	发动机功率	车型编码	已经行驶的里程数	车辆首次注册的月份	燃料类型	汽车品牌	车辆有损坏还没修复	广告创建时间
0	2016-03-24 11:52:17	Golf_3_1.6	privat	Angebot	480	test	NaN	1993	manuell	0	golf	150000	0	benzin	volkswagen	NaN	2016-03-2 00:00:0
1	2016-03-24 10:58:45	A5_Sportback_2.7_Tdi	privat	Angebot	18300	test	coupe	2011	manuell	190	NaN	125000	5	diesel	audi	ja	2016-03-2 00:00:0
2	2016-03-14 12:52:21	Jeep_Grand_Cherokee_"Overland"	privat	Angebot	9800	test	suv	2004	automatik	163	grand	125000	8	diesel	jeep	NaN	2016-03-1 00:00:0
3	2016-03-17 16:54:04	GOLF_4_1_4__3TÜRER	privat	Angebot	1500	test	kleinwagen	2001	manuell	75	golf	150000	6	benzin	volkswagen	nein	2016-03-1 00:00:0
4	2016-03-31 17:25:20	Skoda_Fabia_1.4_TDI_PD_Classic	privat	Angebot	3600	test	kleinwagen	2008	manuell	69	fabia	90000	7	diesel	skoda	nein	2016-03-3 00:00:0

2.3 步骤二：数据探索性分析

查看数据的整体情况，包括每个变量的数据类型、缺失值情况，从而在后续进行处理；查看每个变量的取值个数，取值单一的变量对于分析没有意义，直接剔除；最后查看所有变量的描述性统计值，判断变量的大致分布情况，发现异常值。

代码如下:

```
df.info() # 查看数据中各字段的数据类型、空值情况
df.isnull().any() #判断各字段是否存在空值
df.nunique() # 查看数据中各字段不同取值个数情况
df.describe() #查看数据中数值型变量的描述性统计量（记录数、平均值、标准差、最大值、最小值和四分位数）
df.describe(include=['O']) #类别型变量的一些描述信息（计数、不重复取值个数、众数、众数出现的频次）
```

部分结果如下表 2、表 3:

表 2：数值型变量的描述性统计量

	交易价格	车辆首次注册年份	发动机功率	已经行驶的里程数	车辆首次注册的月份	广告中的图片数量	邮政编码
count	3.714570e+05	371457.000000	371457.000000	371457.000000	371457.000000	371457.0	371457.000000
mean	1.729780e+04	2004.578603	115.556221	125617.541196	5.734610	0.0	50822.358650
std	3.588297e+06	92.875411	192.155494	40112.798219	3.712313	0.0	25799.298421
min	0.000000e+00	1000.000000	0.000000	5000.000000	0.000000	0.0	1067.000000
25%	1.150000e+03	1999.000000	70.000000	125000.000000	3.000000	0.0	30459.000000
50%	2.950000e+03	2003.000000	105.000000	150000.000000	6.000000	0.0	49610.000000
75%	7.200000e+03	2008.000000	150.000000	150000.000000	9.000000	0.0	71549.000000
max	2.147484e+09	9999.000000	20000.000000	150000.000000	12.000000	0.0	99998.000000

表 3：类别型变量的描述性统计量

	广告抓取时间	汽车交易名称	销售方	报价类型	AB测试	车辆类型	汽车变速箱	车型编码	燃料类型	汽车品牌	车辆有损坏还没修复	广告创建时间	最后一次浏览日期
count	371457	371457	371457	371457	371457	333600	351256	350980	338084	371457	299411	371457	371457
unique	280461	233464	2	2	2	8	2	251	7	40	2	114	182771
top	2016-03-24 14:49:47	Ford_Fiesta	privat	Angebot	test	limousine	manuell	golf	benzin	volkswagen	nein	2016-04-03 00:00:00	2016-04-06 13:45:54
freq	7	657	371454	371445	192553	95879	274163	30066	223815	79621	263139	14450	17

- 结果显示“广告中的图片数量”字段取值都是 0，在后续进行数据预处理时直接将其剔除；
- 字段“车辆首次注册年份”显示结果最大取值为 9999，超出了当前时间，与事实不符，后续进行数据预处理作为异常值进行处理；
- 字段“发动机功率”的最小值为 0 属于异常值，最大值与上四分位数的差异很大，之后需要对该字段取值进行异常值分析，进而判定异常值情况；
- 结果显示“广告中的图片数量”字段取值只有 1 个，在后续进行数据预处理时直接将其剔除。

2.4 步骤三：数据预处理

2.4.1 数据类型转换

根据探索性分析中得到的结果，数据中存在部分字段的数据类型不对，将这些字段的数据类型转换为正确的数据类型后续才能更好的使用这些数据进行分析和建模。

代码如下：

```
df["广告抓取时间"] = df["广告抓取时间"].astype('datetime64') #将“广告登出时间”设置为时间类型数据
df["广告创建时间"] = df["广告创建时间"].astype('datetime64') #将“广告创建时间”设置为
```

```
时间类型数据
df["邮政编码"] = df["邮政编码"].astype('object') #将“邮政编码”设置为类别型数据
df["最后一次浏览日期"] = df["最后一次浏览日期"].astype('datetime64') #将“最后一次浏览日期”设置为时间类型数据
df.dtypes #查看转换后各字段的数据类型
```

2.4.2 缺失值处理

根据探索性分析中得到的结果，车辆类型、汽车变速箱、车型编码、燃料类型、车辆有损坏还没修复存在缺失值，需要对这些字段进行缺失值处理。

（1）对于对缺失值比例大于等于 0.9 的变量直接做剔除；

（2）对于缺失值比例小于 0.9 的变量，考虑做缺失值填补；

（3）对于缺失值比例大于 0.65 的行，即 65%以上的字段值都缺失的记录，也直接剔除；

代码如下：

```
# 给定样本量的非空阈值和特征非空阈值，剔除不满足的样本和变量
def DropNull(df,cols):
    # 剔除缺失值比率大于 65%的行
    df.dropna(thresh = len(df.T) * 0.35,axis = 0,inplace = True)
    #drop_list 不满足条件列表
    drop_list = []
    #对变量单一值进行检测，比例大于等于 0.95，放入不满足条件列表，最后扔掉
    for col in cols:
        sample_percent = df[col].isnull().sum(axis=0)/float(len(df))
        if sample_percent > 0.9:
            drop_list.append(col)
    df.drop(drop_list,axis = 1,inplace = True)
    return df
df_autos = DropNull(df = df,cols = df.columns) #依据缺失值剔除不满足条件的样本和变量
```

```
df_autos.info() #查看依据缺失值剔除数据后的结果
for col in ["车辆类型","汽车变速箱","车型编码","燃料类型","车辆有损坏还没修复"]:
    df_autos[col].fillna(df[col].mode()[0], inplace=True)
df_autos.isnull().any() #  查看缺失值填补之后变量的缺失值情况
```

实验结果如下表 4：

表 4：缺失值剔除后的结果

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 371457 entries, 0 to 371456
Data columns (total 20 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   广告抓取时间      371457 non-null  datetime64[ns]
 1   汽车交易名称      371457 non-null  object
 2   销售方         371457 non-null  object
 3   报价类型        371457 non-null  object
 4   交易价格        371457 non-null  int64
 5   AB测试        371457 non-null  object
 6   车辆类型        333600 non-null  object
 7   车辆首次注册年份    371457 non-null  int64
 8   汽车变速箱       351256 non-null  object
 9   发动机功率       371457 non-null  int64
 10  车型编码        350980 non-null  object
 11  已经行驶的里程数    371457 non-null  int64
 12  车辆首次注册的月份   371457 non-null  int64
 13  燃料类型        338084 non-null  object
 14  汽车品牌        371457 non-null  object
 15  车辆有损坏还没修复   299411 non-null  object
 16  广告创建时间      371457 non-null  datetime64[ns]
 17  广告中的图片数量    371457 non-null  int64
 18  邮政编码        371457 non-null  object
 19  最后一次浏览日期    371457 non-null  datetime64[ns]
dtypes: datetime64[ns](3), int64(6), object(11)
memory usage: 59.5+ MB
```

结果显示不存在缺失值比例大于等于 0.9 的变量，也不存在缺失值比例大于 0.65 的记录，所以没有数据依据缺失值规则被剔除。所以进一步对缺失值进行填补，从而提升数据质量，便于后续分析和建模；由于存在缺失值的几个字段都是类别型变量，这边采用众数进行缺失值填补。

实验结果如下表 5：

表 5：缺失值填补后的结果

```
广告抓取时间        False
汽车交易名称        False
销售方            False
报价类型           False
交易价格           False
AB测试           False
车辆类型           False
车辆首次注册年份      False
汽车变速箱          False
发动机功率          False
车型编码           False
已经行驶的里程数      False
车辆首次注册的月份     False
燃料类型           False
汽车品牌           False
车辆有损坏还没修复     False
广告创建时间        False
广告中的图片数量      False
邮政编码           False
最后一次浏览日期      False
dtype: bool
```

2.4.3 异常值处理

根据探索性分析中得到的结果，“车辆首次注册年份”、“车辆首次注册的月份”、“交易价格”存在异常值，需要对这些字段进行异常值处理。

（1）“车辆首次注册年份”存在异常值，但是出现的次数不多，所以选取最近的 20 年数据进行研究,但是结果显示 2019 年的数据量相对较少，所以更改为（1999-2018）；

（2）“车辆首次注册的月份”虽然存在取值为 0 的记录，但是由于数量较大，所以不能直接删除，将 0 随机替换为 1-12 月份；

（3）“交易价格”存在为 0 的取值，很显然不可能，鉴于取值为 0 的记录数占比不大直接将这部分数据删除。

代码如下：

```
# 车辆首次注册年份异常值检测
df_autos["车辆首次注册年份"].value_counts() # 每一个年份出现的次数
# 筛选近 20 年，原本为 2000-2019，但是 2019 年二手车数据量太少，所以更改时间段为 1999-2018 年
reg_20yaers = sorted(list(df_autos.loc[df_autos["车辆首次注册年份"]<= 2020,"车辆首次注
```

```
册年份"].unique()),reverse=True)[1:21]
reduced_autos = df_autos[df_autos["车辆首次注册年份"].isin(reg_20yaers)] # 筛选 20 年的二手车数据
reduced_autos = reduced_autos.sort_values(by = "车辆首次注册年份", ascending = True) # 按注册年份排序
reduced_autos = reduced_autos.reset_index(drop = True) # 数据重新创建索引
reduced_autos.info() # 查看选取时间段后二手车数据情况
reduced_autos["车辆首次注册年份"].value_counts() # 查看每一年注册的二手车数量
reduced_autos["车辆首次注册的月份"].value_counts() # 查看每个月注册的二手车数量
reduced_autos.loc[reduced_autos["车辆首次注册的月份"] == 0,"车辆首次注册的月份"] = random.randint(1,12) # 月份为 0 的记录赋值为 1-12 的随机数
reduced_autos['车辆首次注册的月份'].value_counts() # 查看异常值处理后每个月注册的二手车数量
reduced_autos['车辆首次注册年份'].value_counts() # 查看异常值处理后每年注册的二手车数量
reduced_autos["交易价格"].value_counts() # 查看每个交易价格取值对应的二手车数量
reduced_autos = reduced_autos[reduced_autos["交易价格"] > 0] # 直接删除交易价格为 0 的记录
```

结果如下表 6、表 7:

表 6：异常值处理后车辆首次注册的月份的结果

```
10    46079
3     28866
6     26155
4     24493
5     24491
7     22153
11    20753
9     20481
12    20191
1     19426
8     19254
2     18134
Name: 车辆首次注册的月份, dtype: int64
```

表 7：异常值处理后车辆首次注册年份的结果

```
2000    24543
1999    22759
2005    22314
2006    20225
2001    20213
2003    19871
2004    19743
2002    19187
2007    17673
2008    16174
2009    15606
2010    12351
2011    12066
2017    10543
2016     9858
2012     9417
2013     6157
2014     4802
2018     3992
2015     2982
Name: 车辆首次注册年份, dtype: int64
```

2.5 步骤四：特征工程

2.5.1 衍生变量

对于数据中原有的变量，有的不便于直接分析，比如：广告抓取时间、最后一次浏览日期等时间类型的变量；还有一部分变量可以通过交叉构造出新的变量，对后续的分析意义更大，所以在进行建模之前先生成衍生变量。

（1）时间类型的变量“广告抓取时间”、“广告创建时间”、“最后一次浏览日期”，生成对应变量的年、月，以及年+月；

（2）通过“广告抓取时间”和“车辆首次注册年份”、“车辆首次注册的月份”，计算出汽车使用月数，结果显示存在存在广告抓取在车辆首次注册月份之前的数据，这种数据肯定是在数据抓取时信息有误，但是由于数据量高达 22554，所以这部分数据不能直接删除，将这部分数据的取值设为 0；

（3）从“邮政编码”提取相对应的省份和城市，由于邮政编码长度存在 4 和 5 两种，都是国外的邮政编码，对于长度为 4 的，截取第一位为省份，后 3 位为城市，长度为 5 的，截取前两位为省份，后 3 位为城市；

（4）数据分箱：对“发动机功率”分箱，分箱的原因如下：

- 离散后稀疏向量内积乘法运算速度更快，计算结果也方便存储，容易扩展；
- 离散后的特征对异常值更具鲁棒性，如 age>30 为 1 否则为 0，对于年龄为 200 的也不会对模型造成很大的干扰；
- 离散后特征可以进行特征交叉，提升表达能力，由 M+N 个变量编程 M*N 个变量，进一步引入非线形，提升了表达能力；
- 特征离散后模型更稳定，如用户年龄区间，不会因为用户年龄长了一岁就变化

代码如下：

```
reduced_autos['广告抓取年份'] = reduced_autos['广告抓取时间'].dt.year # 创建衍生变量“广告抓取年份”
reduced_autos['广告抓取月份'] = reduced_autos['广告抓取时间'].dt.month # 创建衍生变量“广告抓取月份”
reduced_autos['广告抓取年月'] = reduced_autos['广告抓取时间'].apply(lambda x : str(x)[0:7]) # 创建衍生变量“广告抓取年月”
reduced_autos['广告创建年份'] = reduced_autos['广告创建时间'].dt.year # 创建衍生变量“广告创建年份”
reduced_autos['广告创建月份'] = reduced_autos['广告创建时间'].dt.month # 创建衍生变量“广告创建月份”
reduced_autos['广告创建年月'] = reduced_autos['广告创建时间'].apply(lambda x : str(x)[0:7]) # 创建衍生变量“广告创建年月”
reduced_autos['最后一次浏览年份'] = reduced_autos['最后一次浏览日期'].dt.year # 创建衍生变量“最后一次浏览年份”
reduced_autos['最后一次浏览月份'] = reduced_autos['最后一次浏览日期'].dt.month # 创建衍生变量“最后一次浏览月份”
reduced_autos['最后一次浏览年月'] = reduced_autos['最后一次浏览日期'].apply(lambda x : str(x)[0:7]) # 创建衍生变量“最后一次浏览年月”
reduced_autos.head() # 查看结果
reduced_autos['汽车使用的月数'] = (reduced_autos['广告抓取时间'].dt.year - reduced_autos['车辆首次注册年份']) * 12 + (reduced_autos['广告抓取时间'].dt.month - reduced_autos['车辆首次注册的月份']) * 1  # 创建衍生变量“汽车使用的月数”
```

```
reduced_autos.loc[reduced_autos['汽车使用的月数'] < 0,'汽车使用的月数'] = 0
reduced_autos['邮编省份'] = reduced_autos['邮政编码'].apply(lambda x : str(x)[0:1] if len(str(x)) == 4 else str(x)[0:2]) #  创建衍生变量“邮编省份”
reduced_autos['邮编城市'] = reduced_autos['邮政编码'].apply(lambda x : str(x)[3:]) #  创建衍生变量“邮编城市”
#  创建衍生变量“发动机功率分箱”
bins = reduced_autos['发动机功率'].quantile(q = np.linspace(0,1,31),interpolation = 'nearest')  #  设置分箱的切分点
bins = list(bins.unique()) #  对切分点去重
bins[0] = -1 #设置分箱的初始值必须小于数据的最小值，要不最小值分箱后取值为 NaN
labels = [i+1 for i in range(len(bins)-1)] #  设置分箱后的标签值
reduced_autos['发动机功率分箱'] = pd.cut(reduced_autos['发动机功率'],bins,labels = labels) #对“发动机功率”分箱
reduced_autos[['发动机功率','发动机功率分箱']] #  查看分箱结果
#  对衍生变量进行数据类型转换
reduced_autos["车辆首次注册年份"] = reduced_autos["车辆首次注册年份"].astype('object') #将“车辆首次注册年份”设置为类别型数据
reduced_autos["车辆首次注册的月份"] = reduced_autos["车辆首次注册的月份"].astype('object') #将“广告抓取年份”设置为类别型数据
reduced_autos["广告抓取年份"] = reduced_autos["广告抓取年份"].astype('object') #将“广告抓取年份”设置为类别型数据
reduced_autos["广告抓取月份"] = reduced_autos["广告抓取月份"].astype('object') #将“广告抓取月份”设置为类别型数据
reduced_autos["广告创建年份"] = reduced_autos["广告创建年份"].astype('object') #将“广告创建年份”设置为类别型数据
reduced_autos["广告创建月份"] = reduced_autos["广告创建月份"].astype('object') #将“广告抓取年份”设置为类别型数据
reduced_autos["最后一次浏览年份"] = reduced_autos["最后一次浏览年份"].astype('object') #将“最后一次浏览年份”设置为类别型数据
```

```
reduced_autos["最后一次浏览月份"] = reduced_autos["最后一次浏览月份
"].astype('object') #将“最后一次浏览月份”设置为类别型数据
```

结果如下表 8:

表 8：衍生变量创建后的结果表

汽车交易名称	销售方	报价类型	交易价格	AB测试	车辆类型	车辆首次注册年份	汽车变速箱	发动机功率	...	广告创建年份	广告创建月份	广告创建年月	最后一次浏览年份	最后一次浏览月份	最后一次浏览年月	汽车使用的月数	邮编省份	邮编城市	发动机功率分箱
BMW_316i_Sport_Edition	privat	Angebot	1000	control	coupe	1999	manuell	102	...	2016	3	2016-03	2016	3	2016-03	198	71	63	10
Chevrolet_Silverado	privat	Angebot	9800	test	suv	1999	automatik	200	...	2016	3	2016-03	2016	4	2016-04	204	35	96	24
Ford_Focus_Turnier_Ghia	privat	Angebot	2200	test	kombi	1999	manuell	116	...	2016	3	2016-03	2016	4	2016-04	201	55	77	13
Ford_Fokus_500_€_fast_geschenkt	privat	Angebot	500	control	kleinwagen	1999	manuell	0	...	2016	3	2016-03	2016	3	2016-03	198	92	33	1
_Micra_1.0_Style_unfallfrei_AUTOMATIKGET...	privat	Angebot	1750	control	kleinwagen	1999	automatik	54	...	2016	3	2016-03	2016	4	2016-04	204	19	89	2

2.5.2 数据分析

通过分析数据，帮助顾客了解二手车销售信息，帮助选择合适的二手车品牌。

（1）最近 20 年不同品牌二手车辆销售数量。

（2）二手汽车售价分布情况：主要分为总体二手车售价分布，和不同品牌二手车售价分布。

（3）汽车行驶距离分布：根据网络还有和有经验的司机了解，车辆行驶距离在 5 万到 15 万公里之间，是汽车性能最好的时候。

（4）车辆使用年限：用网上爬取数据时间的年月减去汽车注册的年月得到车辆使用年限。

（5）车辆有损坏还没修复：车辆有无损坏情况是顾客非常关心的事情。

代码如下:

```
# **********分析最近 20 年不同品牌二手车辆销售数量**********
#  最近 20 年销售的二手车品牌数量和品牌数量
brand_num = reduced_autos.drop_duplicates(subset = ["汽车品牌"])
print("汽车品牌数量：{}种".format(brand_num.shape[0]))
```

```
print("二手车数量：{}辆".format(reduced_autos.shape[0]))
# 最近 20 年不同品牌二手车辆销售数量，按二手车数量进行升序排序，再重置索引值
reduced_autos.groupby("汽车品牌")['汽车交易名称'].count().reset_index(name = '二手车数量').sort_values(by = '二手车数量').reset_index()
# **********二手汽车售价分布情况**********
# 二手车平均售价
price_mean = reduced_autos['交易价格'].mean()
print("二手车平均售价：\n{}\n".format(price_mean))
# 二手车价格四分位数
price_quantile = reduced_autos['交易价格'].quantile([0.25,0.5,0.75])
print("二手车价格四分位数：\n{}\n".format(price_quantile))
# 二手车价格标准差
price_std = reduced_autos['交易价格'].std()
print("二手车价格标准差：\n{}".format(price_std))
brand_group = reduced_autos.groupby("汽车品牌")['交易价格'].mean().reset_index(name = '交易平均价格') # 每个品牌交易平均价格
brand_group['交易价格 Q1'] = list(reduced_autos.groupby("汽车品牌")['交易价格'].quantile(0.25)) # 每个品牌交易价格下四分位数
brand_group['交易价格 Q2'] = list(reduced_autos.groupby("汽车品牌")['交易价格'].quantile(0.5)) # 每个品牌交易价格中位数
brand_group['交易价格 Q3'] = list(reduced_autos.groupby("汽车品牌")['交易价格'].quantile(0.75)) # 每个品牌交易价格上四分位数
brand_group['交易价格标准差'] = list(reduced_autos.groupby("汽车品牌")['交易价格'].std()) # 每个品牌交易价格标准差
brand_group.sort_values(by = '交易价格标准差', inplace = True) # 依据交易价格标准差对数据进行排序
# 查看交易价格的具体频数
plt.hist(reduced_autos['交易价格'], orientation = 'vertical',histtype = 'bar', color ='red')
# log 变换之后的分布较均匀，可以进行 log 变换进行预测，这也是预测问题常用的 trick
```

```
plt.hist(np.log(reduced_autos['交易价格']), orientation = 'vertical',histtype = 'bar', color
='red')
# **********分析汽车行驶距离分布**********
# 汽车行驶距离分布
reduced_autos['已经行驶的里程数'].value_counts()
# 查看汽车行驶距离的具体频数
plt.hist(reduced_autos['已经行驶的里程数'], orientation = 'vertical',histtype = 'bar', color
='red')
# 汽车行驶平均距离
kilometer_mean = reduced_autos['已经行驶的里程数'].mean()
print("汽车行驶平均距离：\n{}\n".format(kilometer_mean))
# 汽车行驶距离四分位数
kilometer_quantile = reduced_autos['已经行驶的里程数'].quantile([0.25,0.5,0.75])
print("汽车行驶距离四分位数：\n{}\n".format(kilometer_quantile))
# 查看不同品牌二手车的平均行驶距离、四分位数以及标准差
kilometer_group = reduced_autos.groupby("汽车品牌")['已经行驶的里程数
'].mean().reset_index(name = '汽车行驶平均距离') # 每个品牌汽车行驶平均距离
kilometer_group['汽车行驶距离 Q1'] = list(reduced_autos.groupby("汽车品牌")['已经行驶
的里程数'].quantile(0.25)) # 每个品牌汽车行驶距离下四分位数
kilometer_group['汽车行驶距离 Q2'] = list(reduced_autos.groupby("汽车品牌")['已经行驶
的里程数'].quantile(0.5)) # 每个品牌交汽车行驶距离中位数
kilometer_group['汽车行驶距离 Q3'] = list(reduced_autos.groupby("汽车品牌")['已经行驶
的里程数'].quantile(0.75)) # 每个品牌汽车行驶距离上四分位数
kilometer_group['汽车行驶距离标准差'] = list(reduced_autos.groupby("汽车品牌")['已经
行驶的里程数'].std()) # 每个品牌汽车行驶距离标准差
kilometer_group.sort_values(by = '汽车行驶距离标准差', inplace = True) # 依据汽车行驶
距离标准差对数据进行排序
kilometer_group
# 行驶距离在 5-15 万公里范围内的二手车数量
```

```
autos_5_15km = reduced_autos.loc[(reduced_autos['已经行驶的里程数'] <= 150000) &
(reduced_autos['已经行驶的里程数'] >= 50000)]
cnt = autos_5_15km['汽车交易名称'].count()
print("行驶距离在 5 万到 15 万公里的二手车数量：{}".format(cnt))
print("行驶距离在 5 万到 15 万公里的二手车交易平均价格:
{}".format(price_mean_5_15km))
# 各个品牌的行驶距离在 5 万到 15 万公里的二手车数量与平均交易价格
kilometer_group_price = autos_5_15km.groupby("汽车品牌")['汽车交易名称
'].count().reset_index(name = '二手车数量') # 每个品牌的二手车数量
kilometer_group_price["平均交易价格"] = list(autos_5_15km.groupby("汽车品牌")['交易价
格'].mean()) # 每个品牌的二手车平均交易价格
kilometer_group_price.sort_values(by = "平均交易价格",inplace = True)
kilometer_group_price
# 作出每个品牌平均交易价格和数量的散点图
plt.rcParams['font.sans-serif']=['SimHei'] #设置默认字体
plt.rcParams['axes.unicode_minus'] = False # 显示负数
plt.xlabel('平均交易价格')
plt.ylabel('二手车数量')
x = kilometer_group_price['平均交易价格']
y = kilometer_group_price['二手车数量']
plt.scatter(x, y)
# 将上述品牌相关的各个指标关联
brand_all = pd.concat([kilometer_group_price,
kilometer_group.iloc[:,1:],brand_group.iloc[:,2:]],axis = 1,keys = '汽车品牌')
# **********分析车辆使用年限**********
# 车辆使用年限，查看二手车使用年限分布情况
plt.hist(reduced_autos['汽车使用的月数'], orientation = 'vertical',histtype = 'bar', color
='red')
# **********分析车辆有损坏还没修复**********
```

```
# 车辆有损坏还没修复
reduced_autos['车辆有损坏还没修复'].value_counts() # 车辆有损坏还没修复取值分布情况
reduced_autos.drop("车辆有损坏还没修复", axis = 1, inplace=True) # 剔除变量“车辆有损坏还没修复”
```

部分结果如下：

表 9：每个品牌相关指标数据

	汽车品牌	二手车数量	平均交易价格	汽车行驶平均距离	汽车行驶距离Q1	汽车行驶距离Q2	汽车行驶距离Q3	汽车行驶距离标准差	交易价格Q1	交易价格Q2	交易价格Q3	交易价格标准差
0	alfa_romeo	1883	3483.579395	131118.791603	125000.0	150000.0	150000.0	33865.395017	1150.0	2399.0	5099.0	1.678595e+06
1	audi	23544	14311.446058	126074.844805	125000.0	150000.0	150000.0	40645.612431	3850.0	7999.0	14500.0	8.826113e+05
2	bmw	28679	10120.484013	130108.699249	125000.0	150000.0	150000.0	36285.879536	3800.0	7750.0	13387.0	2.063474e+05
3	chevrolet	1182	5407.549915	96884.949349	60000.0	90000.0	150000.0	44463.693762	2200.0	3800.0	8000.0	6.665260e+03
4	chrysler	1063	4386.175917	134595.086442	125000.0	150000.0	150000.0	31798.615140	1500.0	2950.0	6225.0	4.488025e+03
5	citroen	4195	3476.013826	120740.946456	90000.0	150000.0	150000.0	40012.092460	1400.0	2700.0	4800.0	4.072097e+05
6	dacia	681	4930.701909	86276.715411	50000.0	80000.0	125000.0	45171.419350	2800.0	4900.0	8500.0	3.662096e+03
7	daewoo	448	1097.810268	123478.260870	100000.0	125000.0	150000.0	33214.978857	600.0	1000.0	1450.0	6.388253e+02
8	daihatsu	605	1853.689256	118215.408805	90000.0	125000.0	150000.0	37560.439252	650.0	1250.0	2700.0	1.986529e+03
9	fiat	6984	6119.010023	116713.785501	90000.0	150000.0	150000.0	42719.984304	900.0	1900.0	4190.0	1.987897e+05

图 1：每个品牌的交易数量与平均交易价格散点图

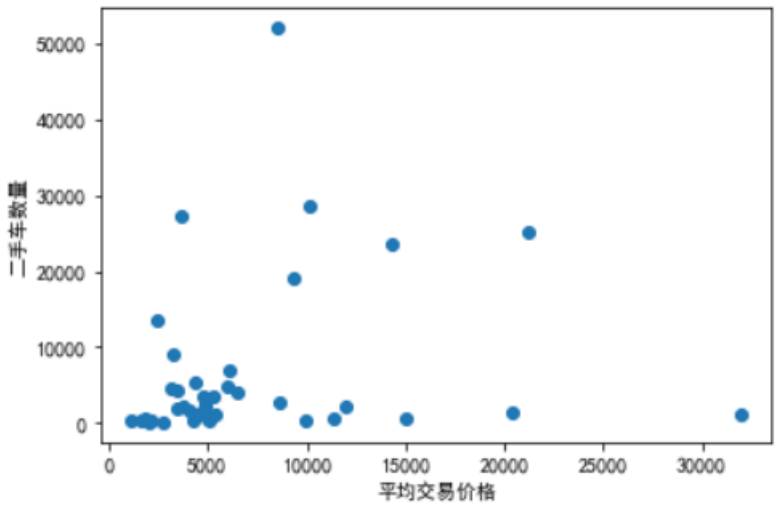


图 2：二手车交易价格柱状图

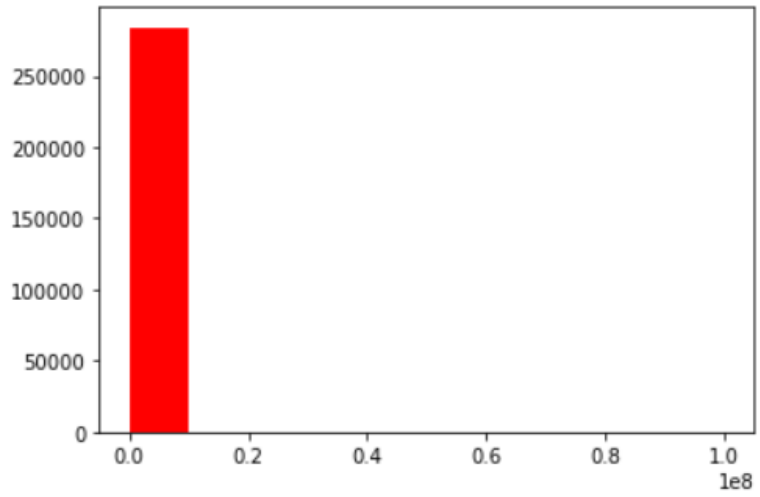


图 3：对数变换后的二手车交易价格柱状图

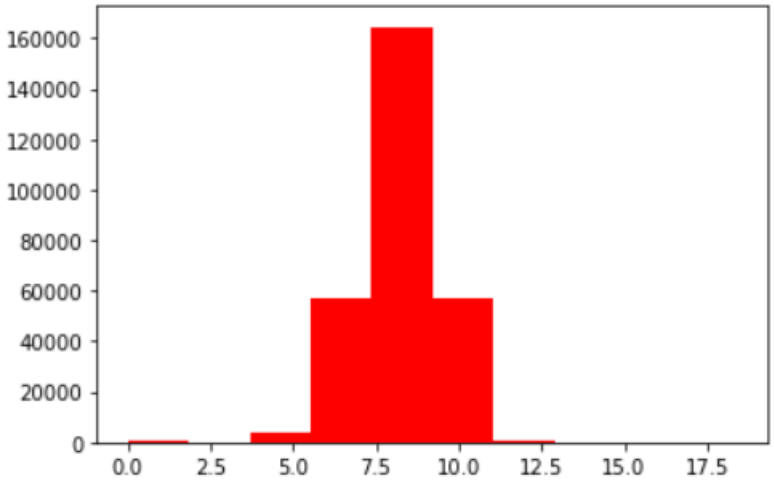


图 4：汽车行驶距离柱状图

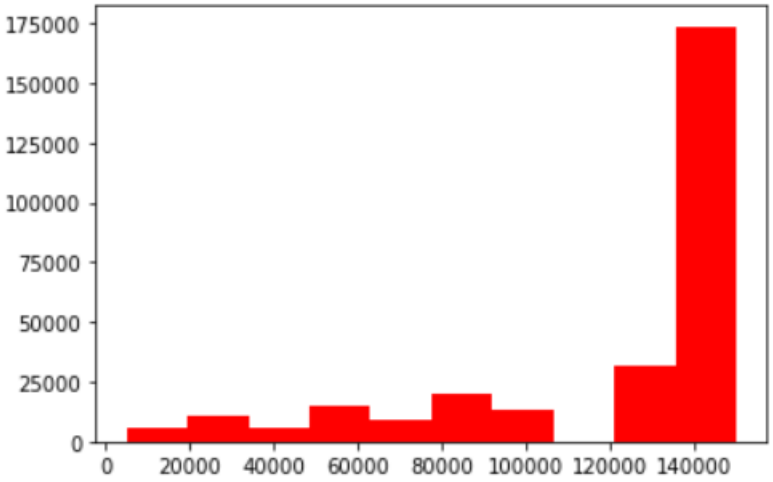
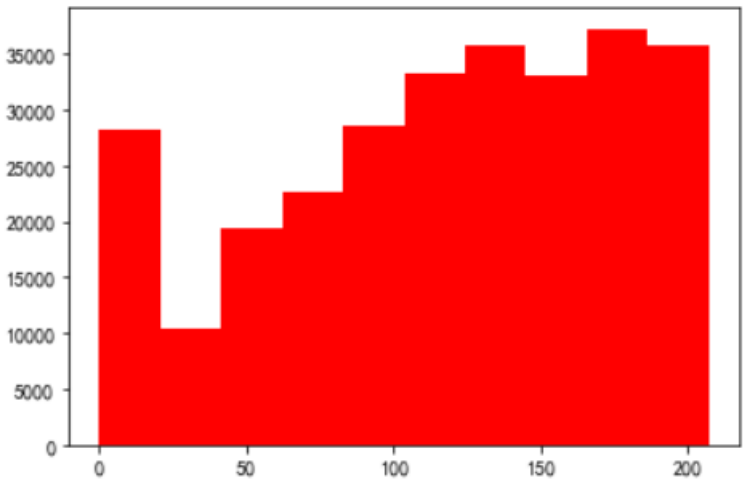


图 5：二手车使用年限柱状图



数据分析结论:

1）在经过数据清洗后，整体二手车销售数量为 284047 辆，共 40 种不同品牌的汽车，二手车市场广阔；

2）市场上二手车平均价格为 11198，但是标准差很高，意味着体征价格浮动非常大，不同品牌车辆也同样，价格浮动变化非常大，购买车辆的时候需要注意；另外查看二手车交易价格分布情况，呈现长尾分布，且为右偏，对交易价格进行对数变换后为正态分布；

3）汽车行驶的距离整体呈左偏分布，50%以上的二手车行驶距离为 15 万公里，50%二手车的行驶距离都在 15 万公里以下，但是浮动依旧也很大。根据经验，行驶距离在 5 到 15 万公里的汽车是性能最佳的时候。有 262324 辆车的行驶距离在这个范围内，这些车的平均售价为 8945，远低于整体平均售价。但是整体便宜不代表每个品牌的车都便宜，经过对比计算，有范围内的部分车辆价格要高于该品牌整体的平均价。另外，各品牌二手车数量与平均价格的散点图显示两者没有一定的相关性，二手车的销量不是由其交易价格决定的，更多的存在品牌效应；

4）经过数据清洗后，柱状图显示大部分车辆使用年限都是在 120 个月以上，即 10 年以上，在 10 年到 12 年之间的二手车数量最多；

5）“车辆有损坏还没修复”变量的取值为 nain 和 ja，这是对应的德语，翻译为中文意思为“不是”和“是”，结果显示 91%以上的二手车没有未修复的损坏，变量的大部分取值都一样，所以该变量不用来构建模型，直接删除。

2.5.3 特征筛选

经过之前的处理后，数据现有 31 个特征，需要进一步分析这 31 个特征的显著性，筛选出对“交易价格”有影响的特征。

（1）变量“广告抓取时间”、“广告创建时间”和“最后一次浏览日期”生成衍生变量后，使用衍生变量进行后续的建模，原有的变量直接剔除；

（2）变量“广告中的图片数量”、“广告抓取年份”和“最后一次浏览年份”为单一取值，直接剔除；

（3）数值型变量进行相关性分析，结果显示各变量间不存在多重共线性，且各数值变量与“交易价格”线性相关性很弱，但是进一步探究与“交易价格”进行对数变换后的相关性，结果显示各数值型变量与对数“交易价格”相关性较强，所以数值型变量全部保留，对预测变量“交易价格”进行对数变换；

（4）类别型变量，考查其单一取值比例是否达到 90%以上，如果是，则该变量对预测结果不显著，直接将其剔除。

代码如下：

```
# 剔除变量“广告抓取时间”、“广告创建时间”和“最后一次浏览日期”
feature_cols = [col for col in reduced_autos.columns if col not in ['广告抓取时间','广告创建时间','最后一次浏览日期']]
reduced_autos = reduced_autos[feature_cols]
# 变量“广告中的图片数量”、“广告抓取年份”和“最后一次浏览年份”为单一取值，直接剔除
feature_cols = [col for col in reduced_autos.columns if col not in ['广告中的图片数量','广告抓取年份','最后一次浏览年份']]
reduced_autos = reduced_autos[feature_cols]
reduced_autos.info()
# 特征分为类别特征和数字特征


# 数字特征
numeric_features = list(reduced_autos.select_dtypes(include=[np.number]).columns)
print(numeric_features)
```

```
# 类型特征
categorical_features = list(reduced_autos.select_dtypes(exclude=[np.number]).columns)
print(categorical_features)
# 数字特征与预测变量“交易价格”的相关性分析
price_numeric = reduced_autos[numeric_features]
correlation1 = price_numeric.corr() # 相关系数计算
print(correlation1['交易价格'].sort_values(ascending = False),'\n')
# 数字特征与预测变量“交易价格”对数变换后的相关性分析
price_numeric = reduced_autos[numeric_features]
price_numeric['交易价格'] = np.log(price_numeric['交易价格'])
correlation2 = price_numeric.corr() # 相关系数计算
print(correlation2['交易价格'].sort_values(ascending = False),'\n')
# 相关性可视化
f , ax = plt.subplots(figsize = (18, 8)) # 设置显示图片大小
plt.rcParams['font.sans-serif']=['SimHei'] #设置默认字体
plt.rcParams['axes.unicode_minus'] = False # 显示负数
plt.subplot(121)
plt.title('Correlation of Numeric Features with Price',y=1,size=12) # 标题设置
sns.heatmap(correlation1, annot=True, vmax=1, square=True, cmap="Reds") # 画热力图
plt.subplot(122)
plt.title('Correlation of Numeric Features with Log(Price)',y=1,size=12) # 标题设置
sns.heatmap(correlation2, annot=True, vmax=1, square=True, cmap="Reds") # 画热力图
# 对预测变量“交易价格”进行对数变换
reduced_autos['交易价格'] = np.log(reduced_autos['交易价格'])
# 对类别型变量进行编码,将原本的取值全部转换为数值
le = LabelEncoder()
for col in categorical_features:
    reduced_autos[col] = le.fit_transform(reduced_autos[col])
# 类别型变量筛选
```

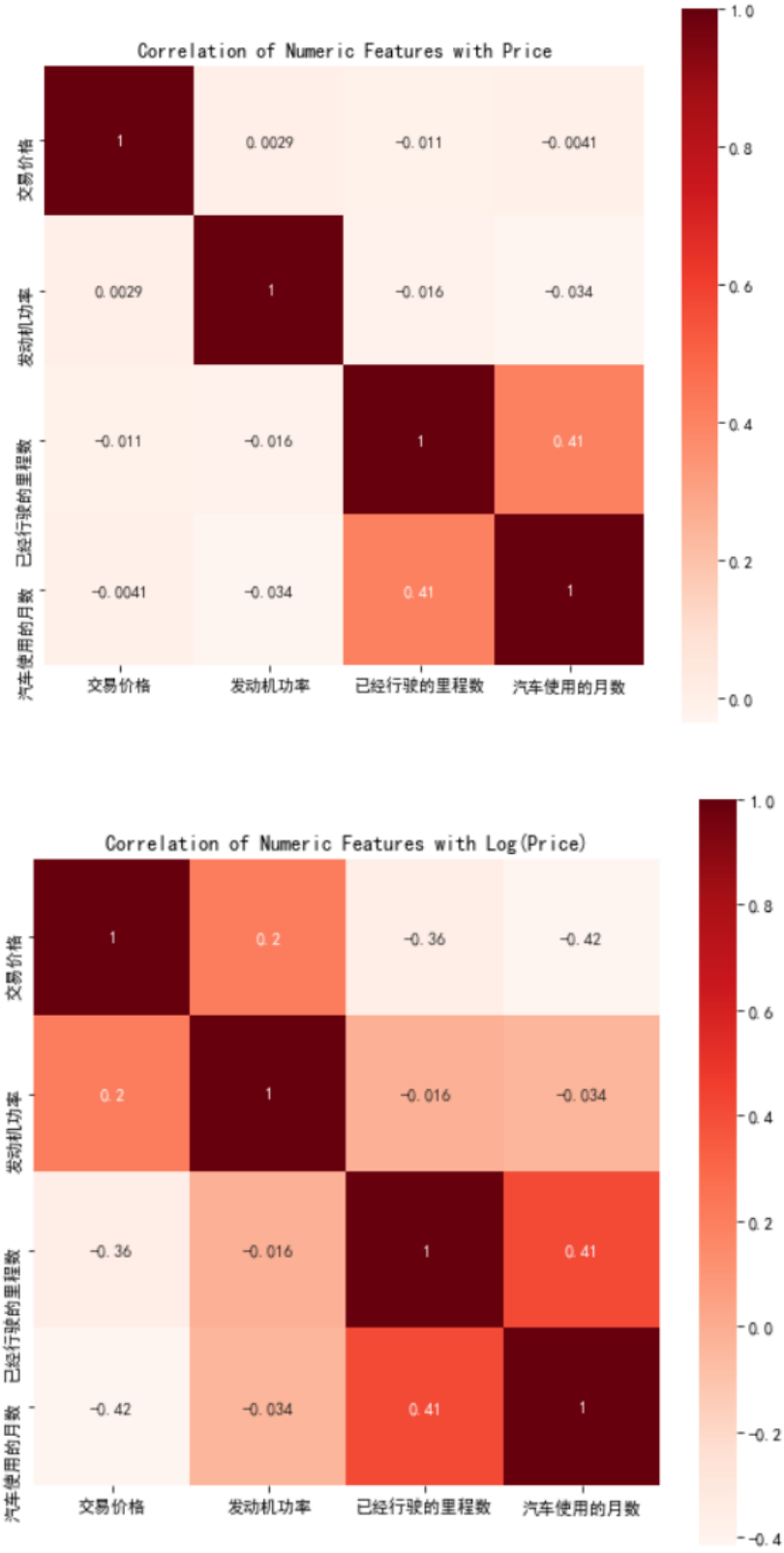
```
# 定义函数，剔除单一取值比例大于 0.9 的变量
def DropNotUsefulVar(df):
    #drop_list 不满足条件列表
    drop_list=[]
    #对变量单一值进行检测，比例大于等于 0.9，放入不满足条件列表，最后扔掉
    for col in df.iloc[:,:-1].columns:
        percent = df[col].value_counts().max()/float(len(df))
        if percent >= 0.9:
            print( ("变量'{}'不显著,将其剔除".format(col)))
            drop_list.append(col)
    df.drop(drop_list,axis = 1,inplace = True)
    return df
categorical_features = DropNotUsefulVar(reduced_autos[categorical_features]).columns #
剔除不显著的类别型变量
# 特征筛选结果
res_cols = list(numeric_features) + list(categorical_features) # 剩余的变量
reduced_autos= reduced_autos[res_cols] # 最终用来建模的数据
```

部分结果如下：

表 10：特征筛选后的数据

	交易价格	发动机功率	已经行驶的里程数	汽车使用的月数	汽车交易名称	AB测试	车辆类型	车辆首次注册年份	汽车变速箱	车型编码	...	邮政编码	广告抓取月份	广告抓取年月	广告创建月份	广告创建年月	最后一次浏览月份	最后一次浏览年月	邮编省份	邮编城市	发动机功率分箱
0	6.907755	102	150000	198	20568	0	3	0	1	10	...	4906	0	0	2	9	0	0	64	70	9
1	9.190138	200	150000	204	36734	1	7	0	0	39	...	2482	0	0	2	9	1	1	27	106	23
2	7.696213	116	150000	201	52646	1	5	0	1	104	...	3855	0	0	2	9	1	1	47	85	12
3	6.214608	0	150000	198	53016	0	4	0	1	104	...	7231	0	0	2	9	0	0	87	37	0
4	7.467371	54	125000	204	90758	0	4	0	0	152	...	1126	0	0	2	9	1	1	9	98	1
5	7.377759	165	150000	201	10376	1	5	0	1	28	...	1916	0	0	2	9	0	0	18	5	20
6	6.214608	120	150000	199	4581	1	6	0	1	3	...	3357	0	0	2	9	0	0	41	48	13
7	6.214608	131	150000	196	54895	0	6	0	1	155	...	955	0	0	2	9	1	1	6	83	15
8	6.802395	75	150000	203	157302	1	6	0	1	117	...	1186	0	0	2	9	0	0	12	81	4
9	7.003065	106	125000	199	63569	1	6	0	1	39	...	2483	0	0	2	9	1	1	27	108	10

图 6：数值型特征与“交易价格”、对数变换后的“交易价格”热力图



特征筛选结果：

- 生成衍生变量后，剔除原有的变量“广告抓取时间”、“广告创建时间”和“最后一次

浏览日期”；

- 所有的数值型特征全部保留，将预测变量“交易价格”进行对数变换；
- 类别型特征剔除：'销售方'、'报价类型'、'广告创建年份'、“广告中的图片数量”、“广告抓取年份”和“最后一次浏览年份”；
- 总共剔除特征 9 个，最终保留特征 22 个，且变量取值全部数字化。

2.6 步骤五：建模调参

2.6.1 数据集划分

将数据集按照 3:1 的比例划分为训练集和测试集，训练集用来进行模型训练，用测试集验证模型效果。

代码如下：

```
X = reduced_autos.iloc[:, reduced_autos.columns != "交易价格"] # 特征数据

y = reduced_autos['交易价格'] # 预测数据

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1,test_size=0.25) # 划分训练集和测试集
```

2.6.2 建模过程

利用训练集进行模型训练，验证集进行模型验证，利用网格搜索法选择模型最佳参数，可决系数 R^2 进行模型评估；

分别构建 GBDT 模型、XGBoost 模型和 LightGBM 模型，通过交叉验证和网格搜索得到模型最优参数，然后用测试集验证模型预测效果。

代码如下：

```
# 模型构建


# 线性回归模型
def build_model_lr(x_train,y_train):
    reg_model = linear_model.LinearRegression()
```

```
    reg_model.fit(x_train,y_train)
    return reg_model


# GBDT 模型
def build_model_gbdt(x_train,y_train):
    estimator =GradientBoostingRegressor(loss='ls',subsample= 0.85,max_depth= 5)
    param_grid = {
                'learning_rate': [0.05,0.08,0.1,0.2],
                'n_estimators':[50,80,100]
                }
    gbdt = GridSearchCV(estimator, param_grid,cv=5)
    gbdt.fit(x_train,y_train)
    print(gbdt.best_params_)
    print(gbdt.best_estimator_ )
    bst = gbdt.best_estimator_
    return (bst,gbdt)


# XGBoost 模型
def build_model_xgb(x_train,y_train):
    estimator = xgb.XGBRegressor(gamma=0, subsample=0.8,colsample_bytree=0.9,
max_depth=5)
    param_grid = {
                'learning_rate': [0.05,0.08,0.1,0.2],
                'n_estimators':[50,80,100]
    }
    xgbt = GridSearchCV(estimator, param_grid,cv=5)
    xgbt.fit(x_train, y_train)
    print(xgbt.best_params_)
    print(xgbt.best_estimator_ )
```

```
    bst = xgbt.best_estimator_
    return (bst,xgb)


# LightGBM 模型
def build_model_lgb(x_train,y_train):
    estimator = lgb.LGBMRegressor(num_leaves=63)
    param_grid = {
        'learning_rate': [0.01, 0.05, 0.1],
        'n_estimators':[50,80,100]
    }
    gbm = GridSearchCV(estimator, param_grid,cv=5)
    gbm.fit(x_train, y_train)
    print(gbm.best_params_)
    print(gbm.best_estimator_ )
    bst = gbm.best_estimator_
    return (bst,gbm)
#  调参，通过网格搜索选择模型最优参数


# GBDT 模型
print('Predict GBDT...')
model_gbdt = build_model_gbdt(X_train,y_train)
bst = model_gbdt[0]
pre_gbdt = bst.predict(X_test)


# XGBoost 模型
print('predict XGB...')
model_xgb = build_model_xgb(X_train,y_train)
bst = model_xgb[0]
pre_xgb = bst.predict(X_test)
```

```
# LightGBM 模型
print('predict lgb...')
model_lgb = build_model_lgb(X_train,y_train)
bst = model_lgb[0]
pre_lgb = bst.predict(X_test)
# 模型评估

# 计算各预测模型对应的 MSE
gbdt_mse = mean_squared_error(y_test,pre_gbdt) # GBDT 模型
xgb_mse = mean_squared_error(y_test,pre_xgb) # XGBoost 模型
lgb_mse = mean_squared_error(y_test,pre_lgb) # LightGBM 模型
print("GBDT 模型的 MSE:{0}\nXGBoost 模型的 MSE:{1}\nLightGBM 模型的 MSE: {2}".
format(gbdt_mse,xgb_mse,lgb_mse))
# 计算各预测模型对应的 R 可决系数 R2
gbdt_r2 = r2_score(y_test,pre_gbdt) # GBDT 模型
xgb_r2 = r2_score(y_test,pre_xgb) # XGBoost 模型
lgb_r2 = r2_score(y_test,pre_lgb) # LightGBM 模型
print("GBDT 模型的 R2:{0}\nXGBoost 模型的 R2:{1}\nLightGBM 模型的 R2:{2}". format
(gbdt_r2,xgb_r2,lgb_r2))
```

模型预测结果如下：

```
GBDT 模型的 MSE:0.3697413749447733
XGBoost 模型的 MSE:0.3689953082911932
LightGBM 模型的 MSE:0.3640731477922518

GBDT 模型的 R2:0.7450901401985708
XGBoost 模型的 R2:0.7456044990422221
LightGBM 模型的 R2:0.748997971690212
```

GBDT 模型、XGBoost 模型和 LightGBM 模型的预测结果对应的 MSE 都在 0.37 以下，对应的模型的 R2 值都在 0.74 以上，三个模型的拟合效果都较好。

2.7 步骤六：模型融合

模型融合目标：对于完成调参的 GBDT 模型、XGBoost 模型和 LightGBM 模型进行模型融合，采用 stacking 融合方法；

stacking 融合：构建多层模型，并利用 GBDT 模型、XGBoost 模型和 LightGBM 模型预测结果再拟合线性回归模型进行预测。

代码如下：

```
## 第一层，得到每个模型的预测结果
train_lgb_pred = model_lgb[0].predict(X_train) # LightGBM 模型训练集预测结果
train_xgb_pred = model_xgb[0].predict(X_train) # XGBoost 模型训练集预测结果
train_gbdt_pred = model_gbdt[0].predict(X_train) # GBDT 模型训练集预测结果

Strak_X_train = pd.DataFrame()
Strak_X_train['Method_1'] = train_lgb_pred
Strak_X_train['Method_2'] = train_xgb_pred
Strak_X_train['Method_3'] = train_gbdt_pred

Strak_X_test = pd.DataFrame()
Strak_X_test['Method_1'] = pre_lgb # LightGBM 模型测试集预测结果
Strak_X_test['Method_2'] = pre_xgb # XGBoost 模型测试集预测结果
Strak_X_test['Method_3'] = pre_gbdt # GBDT 模型测试集预测结果

# 第二层，使用第一层的预测结果用来训练线性回归模型，得到最终的预测结果
model_lr_Stacking = build_model_lr(Strak_X_train,y_train)
# 训练集
train_pre_Stacking = model_lr_Stacking.predict(Strak_X_train)
print('MSE of train_Stacking-LR:',mean_squared_error(y_train,train_pre_Stacking))
print('R2 of train_Stacking-LR:',r2_score(y_train,train_pre_Stacking))

# 测试集
```

```
print('Predict Stacking-LR...')
test_pre_Stacking = model_lr_Stacking.predict(Strak_X_test)
print('MSE of test_Stacking-LR:',mean_squared_error(y_test,test_pre_Stacking))
print('R2 of test_Stacking-LR:',r2_score(y_test,test_pre_Stacking))
```

模型融合预测结果：

```
MSE of train_Stacking-LR: 0.34610209975455686
R2 of train_Stacking-LR: 0.765998794866037

MSE of test_Stacking-LR: 0.36283327171351637
R2 of test_Stacking-LR: 0.7498527763153335
```

采用 stacking 融合方法后，预测效果较比与原本的三个模型都有所提升。

【思考题】：

本案例中对类别型特征采用 LabelEncode 方法数值化，同学们可以采用试试 One-Hot 方法进行数值化，比较两种方法进行数据处理后，GBDT、XGBoost 和 LightGBM 模型预测效果的差异，哪种方法预测效果更好？通过比较思考为什么本案例选取 LabelEncode 方法。本思考题可以做为课堂讨论题。