2020

Datawhale 数据挖掘学习路径

作者：ML67，AI 蜗牛车，阿泽，小雨姑娘

本文档通过理论+竞赛的形式从 0 到 1 梳理数据挖掘学习路径

https://www.datawhale.club/

github：datawhalechina

公众号：Datawhale

当前版本：v1.0

日期：2020 年 3 月 17 日

- **版本纪录**

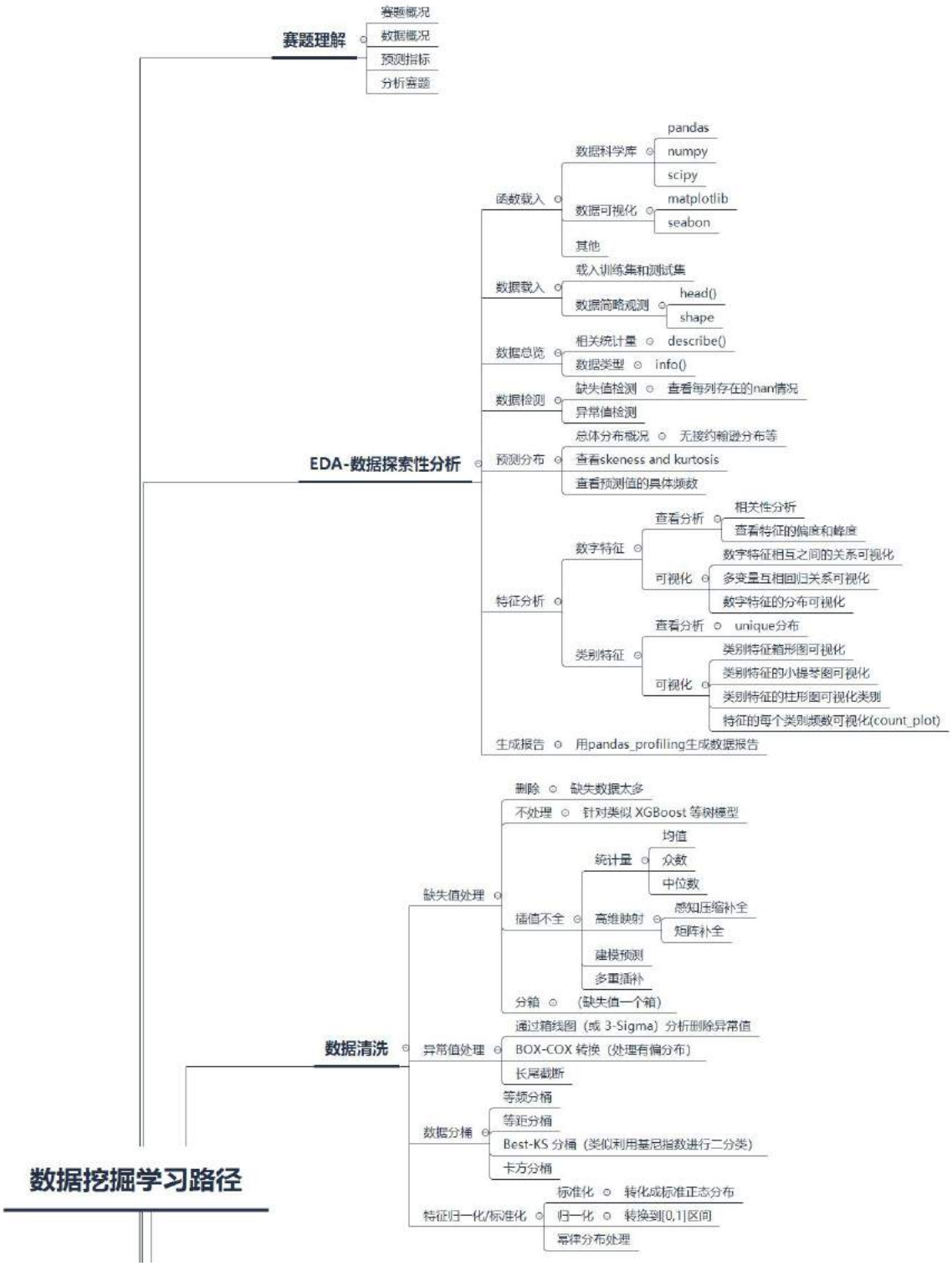
版本号	版本时间	修订内容
V1.0	2020 年 3 月 17 日	初稿版本

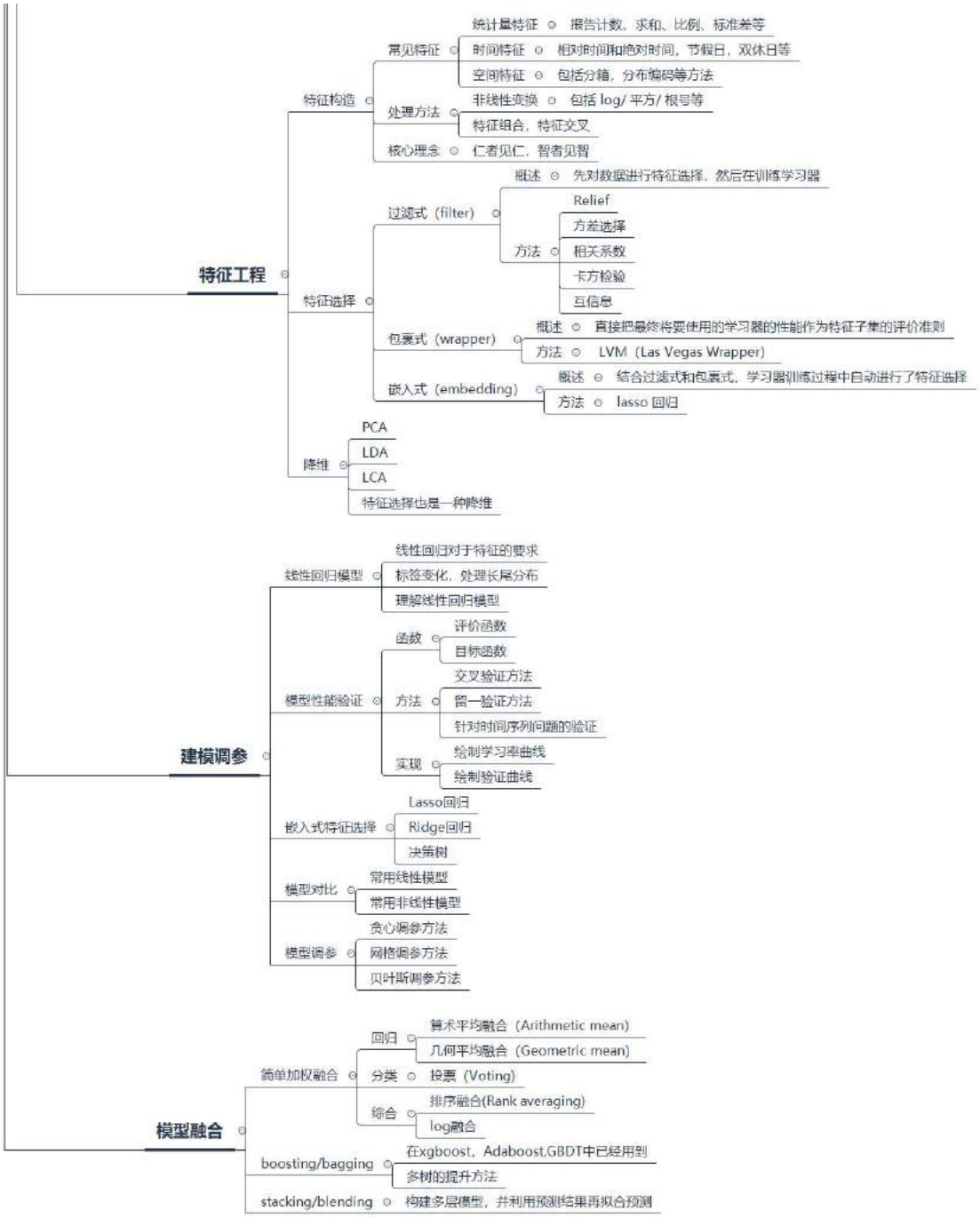
- **开源贡献：**

内容设计	
Datawhale 零基础入门数据挖掘-Baseline	ML67
Datawhale 零基础入门数据挖掘-Task1 赛题理解	AI 蜗牛车
Datawhale 零基础入门数据挖掘-Task2 数据分析	AI 蜗牛车
Datawhale 零基础入门数据挖掘-Task3 特征工程	阿泽
Datawhale 零基础入门数据挖掘-Task4 建模调参	小雨姑娘
Datawhale 零基础入门数据挖掘-Task5 模型融合	ML67
成员信息	
■ ML67 华中科技大学研究生在读 github：https://github.com/mlw67 ■ AI 蜗牛车 东南大学硕士 公众号： AI 蜗牛车 知乎： https://www.zhihu.com/people/seu-aigua-niu-che ■ 阿泽 复旦大学计算机硕士 知乎：阿泽 https://www.zhihu.com/people/is-aze ■ 小雨姑娘 数据挖掘爱好者，多次获得比赛 TOP 名次。 知乎：小雨姑娘的机器学习笔记：https://zhuanlan.zhihu.com/mlbasic	

rr p r r o r

零基础入门数据挖掘学习路径




特征工程
特征构造
常见特征
统计量特征
报告计数、求和、比例、标准差等
时间特征
相对时间和绝对时间，节假日，双休日等
空间特征
包括分箱，分布编码等方法
处理方法
非线性变换
包括 log/ 平方/ 根号等
特征组合，特征交叉
核心理念
仁者见仁，智者见智
特征选择
过滤式（filter）
概述
先对数据进行特征选择，然后在训练学习器
方法
Relief
方差选择
相关系数
卡方检验
互信息
包裹式（wrapper）
概述
直接把最终将要使用的学习器的性能作为特征子集的评价准则
方法
LVM（Las Vegas Wrapper）
嵌入式（embedding）
概述
结合过滤式和包裹式，学习器训练过程中自动进行了特征选择
方法
lasso 回归
降维
PCA
LDA
LCA
特征选择也是一种降维
建模调参
线性回归模型
线性回归对于特征的要求
标签变化，处理长尾分布
理解线性回归模型
模型性能验证
函数
评价函数
目标函数
方法
交叉验证方法
留一验证方法
针对时间序列问题的验证
实现
绘制学习率曲线
绘制验证曲线
嵌入式特征选择
Lasso回归
Ridge回归
决策树
模型对比
常用线性模型
常用非线性模型
模型调参
贪心调参方法
网格调参方法
贝叶斯调参方法
模型融合
简单加权融合
回归
算术平均融合（Arithmetic mean）
几何平均融合（Geometric mean）
分类
投票（Voting）
综合
排序融合(Rank averaging)
log融合
boosting/bagging
在xgboost，Adaboost,GBDT中已经用到
多树的提升方法
stacking/blending
构建多层模型，并利用预测结果再拟合预测

Datawhale 零基础入门数据挖掘-Baseline

Baseline-v1.0 版

Tip:这是一个最初始baseline版本,抛砖引玉,为大家提供一个基本Baseline和一个竞赛流程的基本介绍，欢迎大家多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

In [5]:

```
# 查看数据文件目录  list datalab files
!ls datalab/
```

231784

Step 1:导入函数工具箱

In [6]:

```
## 基础工具
import numpy as np
import pandas as pd
import warnings
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.special import jn
from IPython.display import display, clear_output
import time

warnings.filterwarnings('ignore')
%matplotlib inline

## 模型预测的
from sklearn import linear_model
from sklearn import preprocessing
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor

## 数据降维处理的
from sklearn.decomposition import PCA,FastICA,FactorAnalysis,SparsePCA

import lightgbm as lgb
import xgboost as xgb

## 参数搜索和评价的
from sklearn.model_selection import GridSearchCV,cross_val_score,StratifiedKFold,train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Step 2:数据读取

In [14]:

```
## 通过Pandas对于数据进行读取 (pandas是一个很友好的数据读取函数库)
Train_data = pd.read_csv('datalab/231784/used_car_train_20200313.csv', sep=' ')
TestA_data = pd.read_csv('datalab/231784/used_car_testA_20200313.csv', sep=' ')

## 输出数据的大小信息
print('Train data shape:',Train_data.shape)
print('TestA data shape:',TestA_data.shape)
```

```
Train data shape: (150000, 31)
TestA data shape: (50000, 30)
```

1) 数据简要浏览

In [15]:

```
## 通过.head() 简要浏览读取数据的形式
Train_data.head()
```

Out[15]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilometer	...
0	0	736	20040402	30.0	6	1.0	0.0	0.0	60	12.5	...
1	1	2262	20030301	40.0	1	2.0	0.0	0.0	0	15.0	...
2	2	14874	20040403	115.0	15	1.0	0.0	0.0	163	12.5	...
3	3	71865	19960908	109.0	10	0.0	0.0	1.0	193	15.0	...
4	4	111080	20120103	110.0	5	1.0	0.0	0.0	68	5.0	...

5 rows × 31 columns

2) 数据信息查看

In [16]:

```
## 通过 .info() 简要可以看到对应一些数据列名，以及NAN缺失信息
Train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 31 columns):
SaleID               150000 non-null int64
name                 150000 non-null int64
regDate              150000 non-null int64
model                149999 non-null float64
brand                150000 non-null int64
bodyType             145494 non-null float64
fuelType             141320 non-null float64
gearbox              144019 non-null float64
power                150000 non-null int64
kilometer            150000 non-null float64
notRepairedDamage    150000 non-null object
regionCode           150000 non-null int64
seller               150000 non-null int64
offerType            150000 non-null int64
creatDate            150000 non-null int64
price                150000 non-null int64
v_0                  150000 non-null float64
v_1                  150000 non-null float64
v_2                  150000 non-null float64
v_3                  150000 non-null float64
v_4                  150000 non-null float64
v_5                  150000 non-null float64
v_6                  150000 non-null float64
v_7                  150000 non-null float64
v_8                  150000 non-null float64
v_9                  150000 non-null float64
v_10                 150000 non-null float64
v_11                 150000 non-null float64
v_12                 150000 non-null float64
v_13                 150000 non-null float64
v_14                 150000 non-null float64
dtypes: float64(20), int64(10), object(1)
memory usage: 35.5+ MB
```

In [17]:

```
## 通过 .columns 查看列名
Train_data.columns
```

Out[17]:

```
Index(['SaleID', 'name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType',
       'gearbox', 'power', 'kilometer', 'notRepairedDamage', 'regionCode',
       'seller', 'offerType', 'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3',
       'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12',
       'v_13', 'v_14'],
      dtype='object')
```

In [18]:

```
TestA_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 30 columns):
SaleID               50000 non-null int64
name                 50000 non-null int64
regDate              50000 non-null int64
model                50000 non-null float64
brand                50000 non-null int64
bodyType             48587 non-null float64
fuelType             47107 non-null float64
gearbox              48090 non-null float64
power                50000 non-null int64
kilometer            50000 non-null float64
notRepairedDamage    50000 non-null object
regionCode           50000 non-null int64
seller               50000 non-null int64
offerType            50000 non-null int64
creatDate            50000 non-null int64
v_0                  50000 non-null float64
v_1                  50000 non-null float64
v_2                  50000 non-null float64
v_3                  50000 non-null float64
v_4                  50000 non-null float64
v_5                  50000 non-null float64
v_6                  50000 non-null float64
v_7                  50000 non-null float64
v_8                  50000 non-null float64
v_9                  50000 non-null float64
v_10                 50000 non-null float64
v_11                 50000 non-null float64
v_12                 50000 non-null float64
v_13                 50000 non-null float64
v_14                 50000 non-null float64
dtypes: float64(20), int64(9), object(1)
memory usage: 11.4+ MB
```

3) 数据统计信息浏览

In [19]:

```
## 通过 .describe() 可以查看数值特征列的一些统计信息
Train_data.describe()
```

Out[19]:

	SaleID	name	regDate	model	brand	bodyTyp
count	150000.000000	150000.000000	1.500000e+05	149999.000000	150000.000000	145494.0000(
mean	74999.500000	68349.172873	2.003417e+07	47.129021	8.052733	1.7923(
std	43301.414527	61103.875095	5.364988e+04	49.536040	7.864956	1.7606-
min	0.000000	0.000000	1.991000e+07	0.000000	0.000000	0.0000(
25%	37499.750000	11156.000000	1.999091e+07	10.000000	1.000000	0.0000(
50%	74999.500000	51638.000000	2.003091e+07	30.000000	6.000000	1.0000(
75%	112499.250000	118841.250000	2.007111e+07	66.000000	13.000000	3.0000(
max	149999.000000	196812.000000	2.015121e+07	247.000000	39.000000	7.0000(

8 rows × 30 columns

In [20]:

```
TestA_data.describe()
```

Out[20]:

	SaleID	name	regDate	model	brand	bodyType
count	50000.000000	50000.000000	5.000000e+04	50000.000000	50000.000000	48587.000000
mean	174999.500000	68542.223280	2.003393e+07	46.844520	8.056240	1.782185
std	14433.901067	61052.808133	5.368870e+04	49.469548	7.819477	1.760736
min	150000.000000	0.000000	1.991000e+07	0.000000	0.000000	0.000000
25%	162499.750000	11203.500000	1.999091e+07	10.000000	1.000000	0.000000
50%	174999.500000	52248.500000	2.003091e+07	29.000000	6.000000	1.000000
75%	187499.250000	118856.500000	2.007110e+07	65.000000	13.000000	3.000000
max	199999.000000	196805.000000	2.015121e+07	246.000000	39.000000	7.000000

8 rows × 29 columns

Step 3:特征与标签构建

1) 提取数值类型特征列名

In [21]:

```
numerical_cols = Train_data.select_dtypes(exclude = 'object').columns
print(numerical_cols)
```

```
Index(['SaleID', 'name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType',
       'gearbox', 'power', 'kilometer', 'regionCode', 'seller', 'offerType',
       'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6',
       'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13', 'v_14'],
      dtype='object')
```

In [22]:

```
categorical_cols = Train_data.select_dtypes(include = 'object').columns
print(categorical_cols)
```

```
Index(['notRepairedDamage'], dtype='object')
```

2) 构建训练和测试样本

In [23]:

```
## 选择特征列
feature_cols = [col for col in numerical_cols if col not in ['SaleID','name','regDate','creatDate'
feature_cols = [col for col in feature_cols if 'Type' not in col]

## 提前特征列，标签列构造训练样本和测试样本
X_data = Train_data[feature_cols]
Y_data = Train_data['price']

X_test  = TestA_data[feature_cols]

print('X train shape:',X_data.shape)
print('X test shape:',X_test.shape)
```

```
X train shape: (150000, 18)
X test shape: (50000, 18)
```

In [24]:

```
## 定义了一个统计函数，方便后续信息统计
def Sta_inf(data):
    print('_min',np.min(data))
    print('_max:',np.max(data))
    print('_mean',np.mean(data))
    print('_ptp',np.ptp(data))
    print('_std',np.std(data))
    print('_var',np.var(data))
```

3) 统计标签的基本分布信息

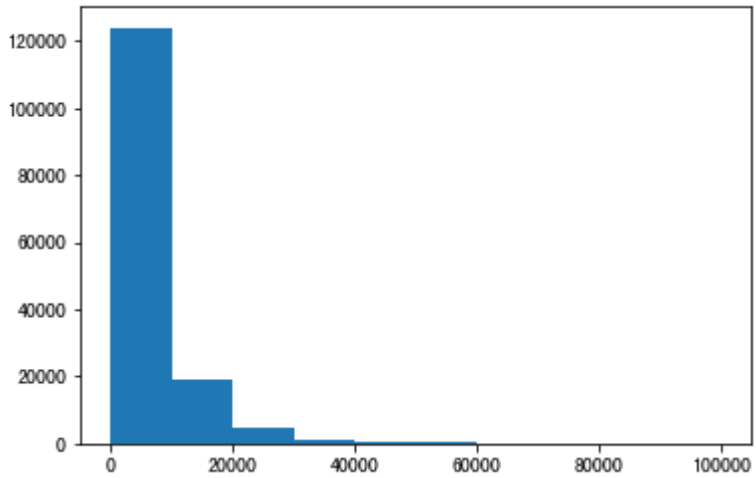
In [25]:

```
print('Sta of label:')
Sta_inf(Y_data)
```

```
Sta of label:
_min 11
_max: 99999
_mean 5923.32733333
_ptp 99988
_std 7501.97346988
_var 56279605.9427
```

In [26]:

```
## 绘制标签的统计图，查看标签分布
plt.hist(Y_data)
plt.show()
plt.close()
```



4) 缺省值用-1填补

In [27]:

```
X_data = X_data.fillna(-1)
X_test = X_test.fillna(-1)
```

Step 4:模型训练与预测

1) 利用xgb进行五折交叉验证查看模型的参数效果

In [32]:

```
## xgb-Model
xgr = xgb.XGBRegressor(n_estimators=120, learning_rate=0.1, gamma=0, subsample=0.8,\
        colsample_bytree=0.9, max_depth=7) #,objective ='reg:squarederror'

scores_train = []
scores = []

## 5折交叉验证方式
sk=StratifiedKFold(n_splits=5,shuffle=True,random_state=0)
for train_ind,val_ind in sk.split(X_data,Y_data):

    train_x=X_data.iloc[train_ind].values
    train_y=Y_data.iloc[train_ind]
    val_x=X_data.iloc[val_ind].values
    val_y=Y_data.iloc[val_ind]

    xgr.fit(train_x,train_y)
    pred_train_xgb=xgr.predict(train_x)
    pred_xgb=xgr.predict(val_x)

    score_train = mean_absolute_error(train_y,pred_train_xgb)
    scores_train.append(score_train)
    score = mean_absolute_error(val_y,pred_xgb)
    scores.append(score)

print('Train mae:',np.mean(score_train))
print('Val mae',np.mean(scores))
```

```
Train mae: 628.086664863
Val mae 715.990013454
```

2） 定义xgb和lgb模型函数

In [35]:

```
def build_model_xgb(x_train,y_train):
    model = xgb.XGBRegressor(n_estimators=150, learning_rate=0.1, gamma=0, subsample=0.8,\
        colsample_bytree=0.9, max_depth=7) #, objective ='reg:squarederror'
    model.fit(x_train, y_train)
    return model

def build_model_lgb(x_train,y_train):
    estimator = lgb.LGBMRegressor(num_leaves=127,n_estimators = 150)
    param_grid = {
        'learning_rate': [0.01, 0.05, 0.1, 0.2],
    }
    gbm = GridSearchCV(estimator, param_grid)
    gbm.fit(x_train, y_train)
    return gbm
```

3）切分数据集（Train,Val）进行模型训练，评价和预测

In [36]:

```
## Split data with val
x_train,x_val,y_train,y_val = train_test_split(X_data,Y_data,test_size=0.3)
```

In [37]:

```
print('Train lgb...')
model_lgb = build_model_lgb(x_train,y_train)
val_lgb = model_lgb.predict(x_val)
MAE_lgb = mean_absolute_error(y_val,val_lgb)
print('MAE of val with lgb:',MAE_lgb)

print('Predict lgb...')
model_lgb_pre = build_model_lgb(X_data,Y_data)
subA_lgb = model_lgb_pre.predict(X_test)
print('Sta of Predict lgb:')
Sta_inf(subA_lgb)
```

```
Train lgb...
MAE of val with lgb: 689.084070621
Predict lgb...
Sta of Predict lgb:
_min -519.150259864
_max: 88575.1087721
_mean 5922.98242599
_ptp 89094.259032
_std 7377.29714126
_var 54424513.1104
```

In [38]:

```
print('Train xgb...')
model_xgb = build_model_xgb(x_train,y_train)
val_xgb = model_xgb.predict(x_val)
MAE_xgb = mean_absolute_error(y_val,val_xgb)
print('MAE of val with xgb:',MAE_xgb)

print('Predict xgb...')
model_xgb_pre = build_model_xgb(X_data,Y_data)
subA_xgb = model_xgb_pre.predict(X_test)
print('Sta of Predict xgb:')
Sta_inf(subA_xgb)
```

```
Train xgb...
MAE of val with xgb: 715.37757816
Predict xgb...
Sta of Predict xgb:
_min -165.479
_max: 90051.8
_mean 5922.9
_ptp 90217.3
_std 7361.13
_var 5.41862e+07
```

4）进行两模型的结果加权融合

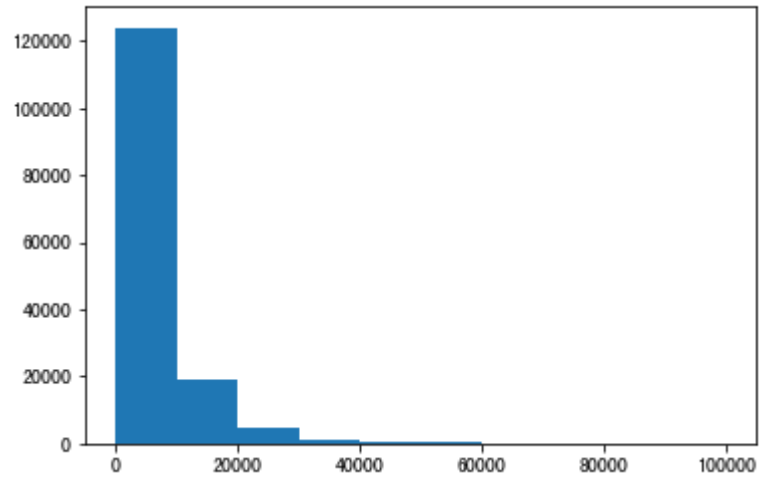
In [39]:

```
## 这里我们采取了简单的加权融合的方式
val_Weighted = (1-MAE_lgb/(MAE_xgb+MAE_lgb))*val_lgb+(1-MAE_xgb/(MAE_xgb+MAE_lgb))*val_xgb
val_Weighted[val_Weighted<0]=10 # 由于我们发现预测的最小值有负数，而真实情况下，price为负是不存在的
print('MAE of val with Weighted ensemble:',mean_absolute_error(y_val,val_Weighted))
```

```
MAE of val with Weighted ensemble: 687.275745703
```

In [40]:

```
sub_Weighted = (1-MAE_lgb/(MAE_xgb+MAE_lgb))*subA_lgb+(1-MAE_xgb/(MAE_xgb+MAE_lgb))*subA_xgb

## 查看预测值的统计进行
plt.hist(Y_data)
plt.show()
plt.close()
```



5）输出结果

In [41]:

```
sub = pd.DataFrame()
sub['SaleID'] = X_test.SaleID
sub['price'] = sub_Weighted
sub.to_csv('./sub_Weighted.csv',index=False)
```

In [42]:

```
sub.head()
```

Out[42]:

	SaleID	price
0	0	39533.727414
1	1	386.081960
2	2	7791.974571
3	3	11835.211966
4	4	585.420407

Baseline END.

--- By: ML67

Email: maolinw67@163.com
PS: 华中科技大学研究生，长期混迹Tianchi等，希望和大家多多交流。
github: https://github.com/mlw67 （近期会做一些书籍推导和代码的整理）

--- By: AI蜗牛车

PS：东南大学研究生，研究方向主要是时空序列预测和时间序列数据挖掘
公众号： AI蜗牛车
知乎： https://www.zhihu.com/people/seu-aigua-niu-che
github: https://github.com/chehongshu

--- By: 阿泽

PS：复旦大学计算机研究生
知乎：阿泽 https://www.zhihu.com/people/is-aze（主要面向初学者的知识整理）

--- By: 小雨姑娘

PS：数据挖掘爱好者，多次获得比赛TOP名次。
知乎：小雨姑娘的机器学习笔记：https://zhuanlan.zhihu.com/mlbasic

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

Datawhale 零基础入门数据挖掘-Task1 赛题理解

一、 赛题理解

Tip:此部分为零基础入门数据挖掘的 Task1 赛题理解 部分，为大家入门数据挖掘比赛提供一个基本的赛题入门讲解，欢迎后续大家多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

1.1 学习目标

- 理解赛题数据和目标，清楚评分体系。
- 完成相应报名，下载数据和结果提交打卡（可提交示例结果），熟悉比赛流程

1.2 了解赛题

- 赛题概况
- 数据概况
- 预测指标
- 分析赛题

1.2.1 赛题概况

比赛要求参赛选手根据给定的数据集，建立模型，二手汽车的交易价格。

来自 Ebay Kleinanzeigen 报废的二手车，数量超过 370,000，包含 20 列变量信息，为了保证 比赛的公平性，将会从中抽取 10 万条作为训练集，5 万条作为测试集 A，5 万条作为测试集 B。同时会对名称、车辆类型、变速箱、model、燃油类型、品牌、公里数、价格等信息进行 脱敏。

通过这道赛题来引导大家走进 AI 数据竞赛的世界，主要针对于于竞赛新人进行自我练 习、自我提高。

1.2.2 数据概况

一般而言，对于数据在比赛界面都有对应的数据概况介绍（匿名特征除外），说明列的性质特征。了解列的性质会有助于我们对于数据的理解和后续分析。 Tip:匿名特征，就是未告知数据列所属的性质的特征列。

---

train.csv

- name - 汽车编码
- regDate - 汽车注册时间
- model - 车型编码

- brand - 品牌
- bodyType - 车身类型
- fuelType - 燃油类型
- gearbox - 变速箱
- power - 汽车功率
- kilometer - 汽车行驶公里
- notRepairedDamage - 汽车有尚未修复的损坏
- regionCode - 看车地区编码
- seller - 销售方
- offerType - 报价类型
- creatDate - 广告发布时间
- price - 汽车价格
- v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13','v_14'（根据汽车的评论、标签等大量信息得到的embedding向量）【人工构造 匿名特征】

数字全都脱敏处理，都为label encoding形式，即数字形式

1.2.3 预测指标

本赛题的评价标准为MAE(Mean Absolute Error):

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

其中y_i代表第i个样本的真实值，其中$\hat{y}_i$代表第i个样本的预测值。

一般问题评价指标说明:

什么是评估指标：

> 评估指标即是我们对于一个模型效果的数值型量化。（有点类似与对于一个商品评价打分，而这是针对于模型效果和理想效果之间的一个打分）

一般来说分类和回归问题的评价指标有如下一些形式：

分类算法常见的评估指标如下：

- 对于二类分类器/分类算法，评价指标主要有accuracy， [Precision，Recall，F-score，Pr曲线]，ROC-AUC曲线。
- 对于多类分类器/分类算法，评价指标主要有accuracy， [宏平均和微平均，F-score]。

对于回归预测类常见的评估指标如下:

- 平均绝对误差（Mean Absolute Error，MAE），均方误差（Mean Squared Error，MSE），平均绝对百分误差（Mean Absolute Percentage Error，MAPE），均方根误差（Root Mean Squared Error）， R2（R-Square）

平均绝对误差 平均绝对误差（Mean Absolute Error，MAE）:平均绝对误差，其能更好地反映预测值与真实值误差的实际情况，其计算公式如下：

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

均方误差 均方误差（Mean Squared Error，MSE）,均方误差,其计算公式为:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$

R2（R-Square）的公式为： 残差平方和:

$$SS_{res} = \sum\left(y_i - \hat{y}_i\right)^2$$

总平均值:

$$SS_{tot} = \sum\left(y_i - \bar{y}_i\right)^2$$

其中$\bar{y}$表示y的平均值 得到R^2表达式为:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum\left(y_i - \hat{y}_i\right)^2}{\sum\left(y_i - \bar{y}\right)^2}$$

R^2用于度量因变量的变异中可由自变量解释部分所占的比例，取值范围是 0~1，R^2越接近1,表明回归平方和占总平方和的比例越大,回归线与各观测点越接近，用x的变化来解释y值变化的部分就越多,回归的拟合程度就越好。所以R^2也称为拟合优度（Goodness of Fit）的统计量。

y_i表示真实值，$\hat{y}_i$表示预测值，$\bar{y}_i$表示样本均值。得分越高拟合效果越好。

1.2.4. 分析赛题

1. 此题为传统的数据挖掘问题，通过数据科学以及机器学习深度学习的办法来进行建模得到结果。
2. 此题是一个典型的回归问题。
3. 主要应用xgb、lgb、catboost，以及pandas、numpy、matplotlib、seabon、sklearn、keras等等数据挖掘常用库或者框架来进行数据挖掘任务。
4. 通过EDA来挖掘数据的联系和自我熟悉数据。

1.3 代码示例

本部分为对于数据读取和指标评价的示例。

1.3.1 数据读取pandas

In [2]:

```
import pandas as pd
import numpy as np

path = './data/'
## 1) 载入训练集和测试集;
Train_data = pd.read_csv(path+'train.csv', sep=' ')
Test_data = pd.read_csv(path+'testA.csv', sep=' ')
print('Train data shape:',Train_data.shape)
print('TestA data shape:',Test_data.shape)
```

```
Train data shape: (150000, 31)
TestA data shape: (50000, 30)
```

In [3]:

```
Train_data.head()
```

Out[3]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilometer	...
0	0	736	20040402	30.0	6	1.0	0.0	0.0	60	12.5	...
1	1	2262	20030301	40.0	1	2.0	0.0	0.0	0	15.0	...
2	2	14874	20040403	115.0	15	1.0	0.0	0.0	163	12.5	...
3	3	71865	19960908	109.0	10	0.0	0.0	1.0	193	15.0	...
4	4	111080	20120103	110.0	5	1.0	0.0	0.0	68	5.0	...

5 rows × 31 columns

1.3.2 分类指标评价计算示例

In [4]:

```
## accuracy
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = [0, 1, 0, 1]
y_true = [0, 1, 1, 1]
print('ACC:',accuracy_score(y_true, y_pred))
```

```
ACC: 0.75
```

In [5]:

```
## Precision,Recall,F1-score
from sklearn import metrics
y_pred = [0, 1, 0, 0]
y_true = [0, 1, 0, 1]
print('Precision',metrics.precision_score(y_true, y_pred))
print('Recall',metrics.recall_score(y_true, y_pred))
print('F1-score:',metrics.f1_score(y_true, y_pred))
```

```
Precision 1.0
Recall 0.5
F1-score: 0.6666666666666666
```

In [6]:

```
## AUC
import numpy as np
from sklearn.metrics import roc_auc_score
y_true = np.array([0, 0, 1, 1])
y_scores = np.array([0.1, 0.4, 0.35, 0.8])
print('AUC socre:',roc_auc_score(y_true, y_scores))
```

```
AUC socre: 0.75
```

1.3.3 回归指标评价计算示例

In [8]:

```
# coding=utf-8
import numpy as np
from sklearn import metrics

# MAPE需要自己实现
def mape(y_true, y_pred):
    return np.mean(np.abs((y_pred - y_true) / y_true))

y_true = np.array([1.0, 5.0, 4.0, 3.0, 2.0, 5.0, -3.0])
y_pred = np.array([1.0, 4.5, 3.8, 3.2, 3.0, 4.8, -2.2])

# MSE
print('MSE:',metrics.mean_squared_error(y_true, y_pred))
# RMSE
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_true, y_pred)))
# MAE
print('MAE:',metrics.mean_absolute_error(y_true, y_pred))
# MAPE
print('MAPE:',mape(y_true, y_pred))
```

```
MSE: 0.2871428571428571
RMSE: 0.5358571238146014
MAE: 0.4142857142857143
MAPE: 0.1461904761904762
```

In [10]:

```
## R2-score
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
print('R2-score:',r2_score(y_true, y_pred))
```

```
R2-score: 0.9486081370449679
```

1.4 经验总结

作为切入一道赛题的基础，赛题理解是极其重要的，对于赛题的理解甚至会影响后续的特征工程构建以及模型的选择，最主要是会影响后续发展工作的方向，比如挖掘特征的方向或者存在问题解决问题的方向，对了赛题背后的思想以及赛题业务逻辑的清晰，也很有利于花费更少时间构建更为有效的特征模型，赛题理解要达到的地步是什么呢，把一道赛题转化为一种宏观理解的解决思路。 以下将从多方面对于此进行说明：

- 1） 赛题理解究竟是理解什么： 理解赛题是不是把一道赛题的背景介绍读一遍就OK了呢？并不是的，理解赛题其实也是从直观上梳理问题，分析问题是否可行的方法，有多少可行度，赛题做的价值大不大，理清一道赛题要从背后的赛题背景引发的赛题任务理解其中的任务逻辑，可能对于赛题有意义的外在数据有哪些，并对于赛题数据有一个初步了解，知道现在和任务的相关数据有哪些，其中数据之间的关联逻辑是什么样的。 对于不同的问题，在处理方式上的差异是很大的。如果用简短的话来说，并且在比赛的角度或者做工程的角度，就是该赛题符合的问题是什么问题，大概要去用哪些指标，哪些指标是否会做到线上线下的一致性，是否有效的利于我们进一步的探索更高线上分数的线下验证方法，在业务上，你是否对很多原始特征有很深刻的了解，并且可以通过EDA来寻求他们直接的关系，最后构造出满意的特征。

- 2） 有了赛题理解后能做什么： 在对于赛题有了一定的了解后，分析清楚了问题的类型性质和对于数据理解的这一基础上，是不是赛题理解就做完了呢? 并不是的，就像摸清了敌情后，我们至少就要有一些相应的理解分析，比如这题的难点可能在哪里，关键点可能在哪里，哪些地方可以挖掘更好的特征，用什么样得线下验证方式更为稳定，出现了过拟合或者其他问题，估摸可以用什么方法去解决这些问题，哪些数据是可靠的，哪些数据是需要精密的处理的，哪部分数据应该是关键数据（背景的业务逻辑下，比如CTR的题，一个寻常顾客大体会有怎么样的购买行为逻辑规律，或者风电那种题，如果机组比较邻近，相关一些风速，转速特征是否会很近似）。这时是在一个宏观的大体下分析的，有助于摸清整个题的思路脉络，以及后续的分析方向。
- 3） 赛题理解的-评价指标： 为什么要把这部分单独拿出来呢，因为这部分会涉及后续模型预测中两个很重要的问题： 1． 本地模型的验证方式，很多情况下，线上验证是有一定的时间和次数限制的，所以在比赛中构建一个合理的本地的验证集和验证的评价指标是很关键的步骤，能有效的节省很多时间。 2． 不同的指标对于同样的预测结果是具有误差敏感的差异性的，比如AUC，logloss, MAE，RSME，或者一些特定的评价函数。是会有很大可能会影响后续一些预测的侧重点。
- 4） 赛题背景中可能潜在隐藏的条件： 其实赛题中有些说明是很有利益-都可以在后续答辩中以及问题思考中所体现出来的，比如高效性要求，比如对于数据异常的识别处理，比如工序流程的差异性，比如模型运行的时间，比模型的鲁棒性，有些的意识是可以贯穿问题思考，特征，模型以及后续处理的，也有些会对于特征构建或者选择模型上有很大益处，反过来如果在模型预测效果不好，其实有时也要反过来思考，是不是赛题背景有没有哪方面理解不清晰或者什么其中的问题没考虑到。

Task1 赛题理解 END.

--- By: AI蜗牛车

PS：东南大学研究生，研究方向主要是时空序列预测和时间序列数据挖掘
公众号： AI蜗牛车
知乎： https://www.zhihu.com/people/seu-aigua-niu-che
github: https://github.com/chehongshu

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

Datawhale 零基础入门数据挖掘-Task2 数据分析

二、 EDA-数据探索性分析

Tip:此部分为零基础入门数据挖掘的 Task2 EDA-数据探索性分析 部分，带你来了解数据，熟悉数据，和数据做朋友，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

2.1 EDA目标

- EDA的价值主要在于熟悉数据集，了解数据集，对数据集进行验证来确定所获得数据集可以用于接下来的机器学习或者深度学习使用。
- 当了解了数据集之后我们下一步就是要去了解变量间的相互关系以及变量与预测值之间的存在关系。
- 引导数据科学从业者进行数据处理以及特征工程的步骤,使数据集的结构和特征集让接下来的预测问题更加可靠。
- 完成对于数据的探索性分析，并对于数据进行一些图表或者文字总结并打卡。

2.2 内容介绍

1. 载入各种数据科学以及可视化库:
    - 数据科学库 pandas、numpy、scipy；
    - 可视化库 matplotlib、seabon；
    - 其他；
2. 载入数据：
    - 载入训练集和测试集；
    - 简略观察数据(head()+shape)；
3. 数据总览:
    - 通过describe()来熟悉数据的相关统计量
    - 通过info()来熟悉数据类型
4. 判断数据缺失和异常
    - 查看每列的存在nan情况
    - 异常值检测
5. 了解预测值的分布
    - 总体分布概况（无界约翰逊分布等）
    - 查看skewness and kurtosis
    - 查看预测值的具体频数
6. 特征分为类别特征和数字特征，并对类别特征查看unique分布
7. 数字特征分析
    - 相关性分析
    - 查看几个特征得 偏度和峰值
    - 每个数字特征得分布可视化
    - 数字特征相互之间的关系可视化

    - 多变量互相回归关系可视化
8. 类型特征分析
    - unique分布
    - 类别特征箱形图可视化
    - 类别特征的小提琴图可视化
    - 类别特征的柱形图可视化类别
    - 特征的每个类别频数可视化(count_plot)
9. 用pandas_profiling生成数据报告

2.3 代码示例

2.3.1 载入各种数据科学以及可视化库

以下库都是pip install 安装， 有特殊情况我会单独说明 例如 pip install pandas -i https://pypi.tuna.tsinghua.edu.cn/simple (https://pypi.tuna.tsinghua.edu.cn/simple)

In [1]:

```
#coding:utf-8
#导入warnings包，利用过滤器来实现忽略警告语句。
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

2.3.2 载入数据

In [6]:

```
## 1) 载入训练集和测试集;
Train_data = pd.read_csv('train.csv', sep=' ')
Test_data = pd.read_csv('testA.csv', sep=' ')
```

所有特征集均脱敏处理(方便大家观看)

- name - 汽车编码
- regDate - 汽车注册时间
- model - 车型编码
- brand - 品牌
- bodyType - 车身类型
- fuelType - 燃油类型
- gearbox - 变速箱
- power - 汽车功率
- kilometer - 汽车行驶公里
- notRepairedDamage - 汽车有尚未修复的损坏
- regionCode - 看车地区编码

- seller - 销售方
- offerType - 报价类型
- creatDate - 广告发布时间
- price - 汽车价格
- v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13','v_14'（根据汽车的评论、标签等大量信息得到的embedding向量）【人工构造 匿名特征】

In [9]:

```
## 2) 简略观察数据(head()+shape)
Train_data.head().append(Train_data.tail())
```

Out[9]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilome
0	0	736	20040402	30.0	6	1.0	0.0	0.0	60	1
1	1	2262	20030301	40.0	1	2.0	0.0	0.0	0	1
2	2	14874	20040403	115.0	15	1.0	0.0	0.0	163	1
3	3	71865	19960908	109.0	10	0.0	0.0	1.0	193	1
4	4	111080	20120103	110.0	5	1.0	0.0	0.0	68	
149995	149995	163978	20000607	121.0	10	4.0	0.0	1.0	163	1
149996	149996	184535	20091102	116.0	11	0.0	0.0	0.0	125	1
149997	149997	147587	20101003	60.0	11	1.0	1.0	0.0	90	
149998	149998	45907	20060312	34.0	10	3.0	1.0	0.0	156	1
149999	149999	177672	19990204	19.0	28	6.0	0.0	1.0	193	1

10 rows × 31 columns

In [10]:

```
Train_data.shape
```

Out[10]:

```
(150000, 31)
```

In [11]:

```
Test_data.head().append(Test_data.tail())
```

Out[11]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilomet
0	150000	66932	20111212	222.0	4	5.0	1.0	1.0	313	15
1	150001	174960	19990211	19.0	21	0.0	0.0	0.0	75	12
2	150002	5356	20090304	82.0	21	0.0	0.0	0.0	109	7
3	150003	50688	20100405	0.0	0	0.0	0.0	1.0	160	7
4	150004	161428	19970703	26.0	14	2.0	0.0	0.0	75	15
49995	199995	20903	19960503	4.0	4	4.0	0.0	0.0	116	15
49996	199996	708	19991011	0.0	0	0.0	0.0	0.0	75	15
49997	199997	6693	20040412	49.0	1	0.0	1.0	1.0	224	15
49998	199998	96900	20020008	27.0	1	0.0	0.0	1.0	334	15
49999	199999	193384	20041109	166.0	6	1.0	NaN	1.0	68	9

10 rows × 30 columns

In [12]:

```
Test_data.shape
```

Out[12]:

```
(50000, 30)
```

要养成看数据集的head()以及shape的习惯，这会让你每一步更放心，导致接下里的连串的错误, 如果对自己的pandas等操作不放心，建议执行一步看一下，这样会有效的方便你进行理解函数并进行操作

2.3.3 总览数据概况

1. describe种有每列的统计量，个数count、平均值mean、方差std、最小值min、中位数25% 50% 75% 、以及最大值 看这个信息主要是瞬间掌握数据的大概的范围以及每个值的异常值的判断，比如有的时候会发现999 9999 -1 等值这些其实都是nan的另外一种表达方式，有的时候需要注意下
2. info 通过info来了解数据每列的type，有助于了解是否存在除了nan以外的特殊符号异常

In [13]:

```
## 1) 通过describe()来熟悉数据的相关统计量
Train_data.describe()
```

Out[13]:

	SaleID	name	regDate	model	brand	bodyTyp
count	150000.000000	150000.000000	1.500000e+05	149999.000000	150000.000000	145494.0000(
mean	74999.500000	68349.172873	2.003417e+07	47.129021	8.052733	1.7923(
std	43301.414527	61103.875095	5.364988e+04	49.536040	7.864956	1.7606-
min	0.000000	0.000000	1.991000e+07	0.000000	0.000000	0.0000(
25%	37499.750000	11156.000000	1.999091e+07	10.000000	1.000000	0.0000(
50%	74999.500000	51638.000000	2.003091e+07	30.000000	6.000000	1.0000(
75%	112499.250000	118841.250000	2.007111e+07	66.000000	13.000000	3.0000(
max	149999.000000	196812.000000	2.015121e+07	247.000000	39.000000	7.0000(

8 rows × 30 columns

In [14]:

```
Test_data.describe()
```

Out[14]:

	SaleID	name	regDate	model	brand	bodyType
count	50000.000000	50000.000000	5.000000e+04	50000.000000	50000.000000	48587.000000
mean	174999.500000	68542.223280	2.003393e+07	46.844520	8.056240	1.782185
std	14433.901067	61052.808133	5.368870e+04	49.469548	7.819477	1.760736
min	150000.000000	0.000000	1.991000e+07	0.000000	0.000000	0.000000
25%	162499.750000	11203.500000	1.999091e+07	10.000000	1.000000	0.000000
50%	174999.500000	52248.500000	2.003091e+07	29.000000	6.000000	1.000000
75%	187499.250000	118856.500000	2.007110e+07	65.000000	13.000000	3.000000
max	199999.000000	196805.000000	2.015121e+07	246.000000	39.000000	7.000000

8 rows × 29 columns

In [15]:

```
## 2) 通过info()来熟悉数据类型
Train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 31 columns):
SaleID               150000 non-null int64
name                 150000 non-null int64
regDate              150000 non-null int64
model                149999 non-null float64
brand                150000 non-null int64
bodyType             145494 non-null float64
fuelType             141320 non-null float64
gearbox              144019 non-null float64
power                150000 non-null int64
kilometer            150000 non-null float64
notRepairedDamage    150000 non-null object
regionCode           150000 non-null int64
seller               150000 non-null int64
offerType            150000 non-null int64
creatDate            150000 non-null int64
price                150000 non-null int64
v_0                  150000 non-null float64
v_1                  150000 non-null float64
v_2                  150000 non-null float64
v_3                  150000 non-null float64
v_4                  150000 non-null float64
v_5                  150000 non-null float64
v_6                  150000 non-null float64
v_7                  150000 non-null float64
v_8                  150000 non-null float64
v_9                  150000 non-null float64
v_10                 150000 non-null float64
v_11                 150000 non-null float64
v_12                 150000 non-null float64
v_13                 150000 non-null float64
v_14                 150000 non-null float64
dtypes: float64(20), int64(10), object(1)
memory usage: 35.5+ MB
```

In [16]:

```
Test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 30 columns):
SaleID               50000 non-null int64
name                 50000 non-null int64
regDate              50000 non-null int64
model                50000 non-null float64
brand                50000 non-null int64
bodyType             48587 non-null float64
fuelType             47107 non-null float64
gearbox              48090 non-null float64
power                50000 non-null int64
kilometer            50000 non-null float64
notRepairedDamage    50000 non-null object
regionCode           50000 non-null int64
seller               50000 non-null int64
offerType            50000 non-null int64
creatDate            50000 non-null int64
v_0                  50000 non-null float64
v_1                  50000 non-null float64
v_2                  50000 non-null float64
v_3                  50000 non-null float64
v_4                  50000 non-null float64
v_5                  50000 non-null float64
v_6                  50000 non-null float64
v_7                  50000 non-null float64
v_8                  50000 non-null float64
v_9                  50000 non-null float64
v_10                 50000 non-null float64
v_11                 50000 non-null float64
v_12                 50000 non-null float64
v_13                 50000 non-null float64
v_14                 50000 non-null float64
dtypes: float64(20), int64(9), object(1)
memory usage: 11.4+ MB
```

2.3.4 判断数据缺失和异常

In [17]:

```
## 1）查看每列的存在nan情况
Train_data.isnull().sum()
```

Out[17]:

```
SaleID               0
name                 0
regDate              0
model                1
brand                0
bodyType          4506
fuelType          8680
gearbox           5981
power                0
kilometer            0
notRepairedDamage    0
regionCode           0
seller               0
offerType            0
creatDate            0
price                0
v_0                  0
v_1                  0
v_2                  0
v_3                  0
v_4                  0
v_5                  0
v_6                  0
v_7                  0
v_8                  0
v_9                  0
v_10                 0
v_11                 0
v_12                 0
v_13                 0
v_14                 0
dtype: int64
```

In [18]:

```
Test_data.isnull().sum()
```

Out[18]:

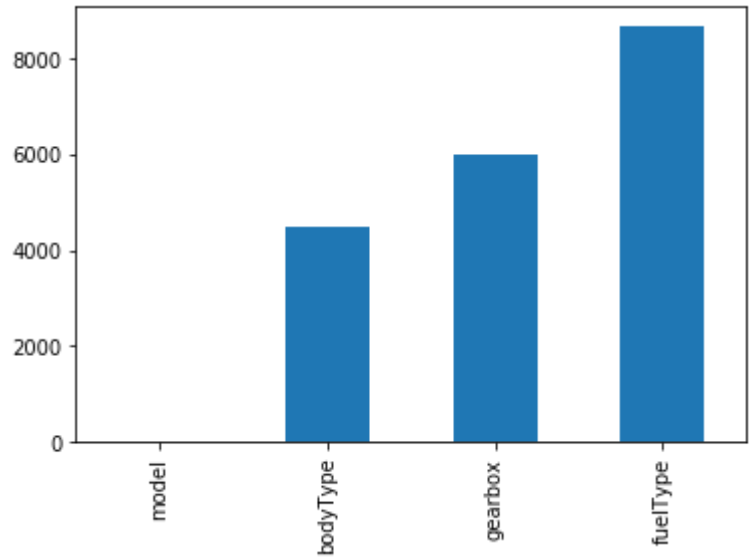
```
SaleID               0
name                 0
regDate              0
model                0
brand                0
bodyType          1413
fuelType          2893
gearbox           1910
power                0
kilometer            0
notRepairedDamage    0
regionCode           0
seller               0
offerType            0
creatDate            0
v_0                  0
v_1                  0
v_2                  0
v_3                  0
v_4                  0
v_5                  0
v_6                  0
v_7                  0
v_8                  0
v_9                  0
v_10                 0
v_11                 0
v_12                 0
v_13                 0
v_14                 0
dtype: int64
```

In [19]:

```
# nan可视化
missing = Train_data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b156dba8>



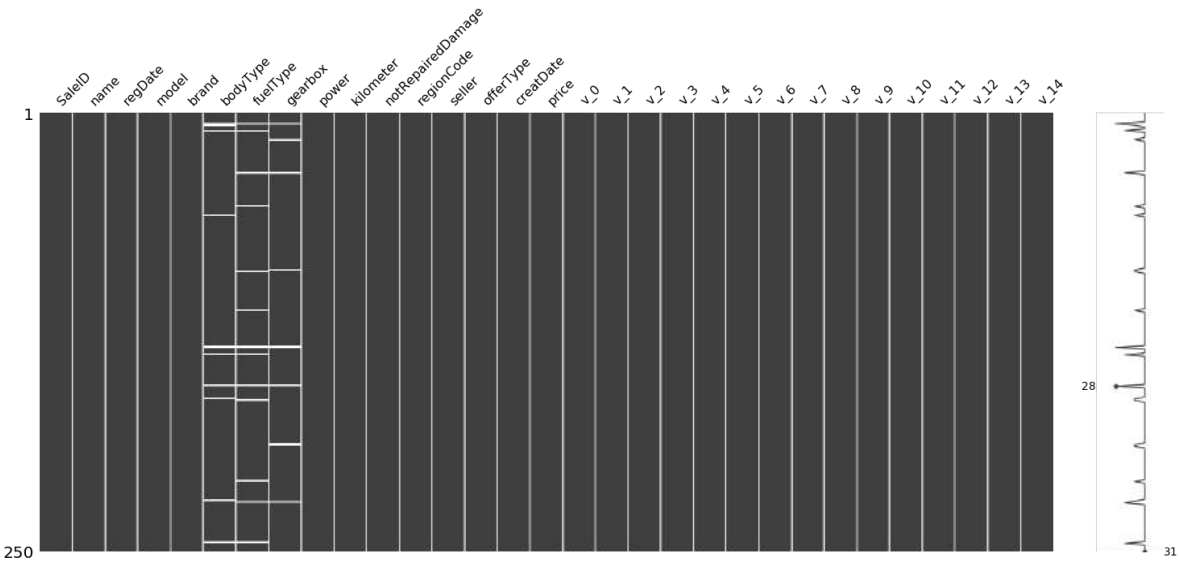
通过以上两句可以很直观的了解哪些列存在 “nan”, 并可以把nan的个数打印，主要的目的在于 nan存在的个数是否真的很大，如果很小一般选择填充，如果使用lgb等树模型可以直接空缺，让树自己去优化，但如果nan存在的过多、可以考虑删掉

In [20]:

```
# 可视化看下缺省值
msno.matrix(Train_data.sample(250))
```

Out[20]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b16e75f8>

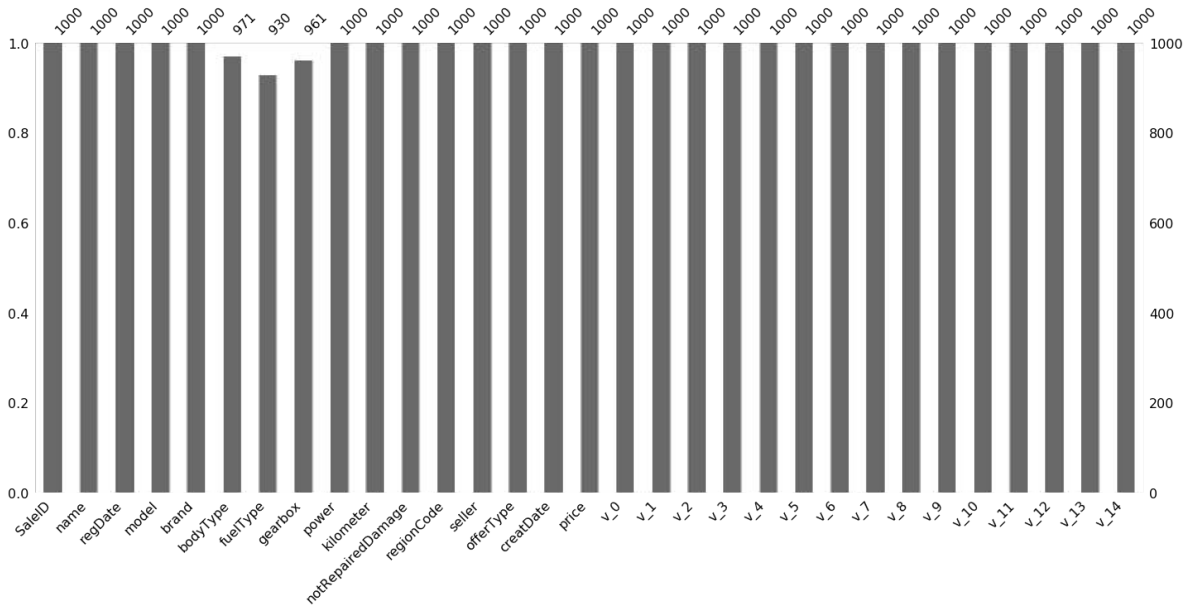


In [21]:

```
msno.bar(Train_data.sample(1000))
```

Out[21]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b1936f98>

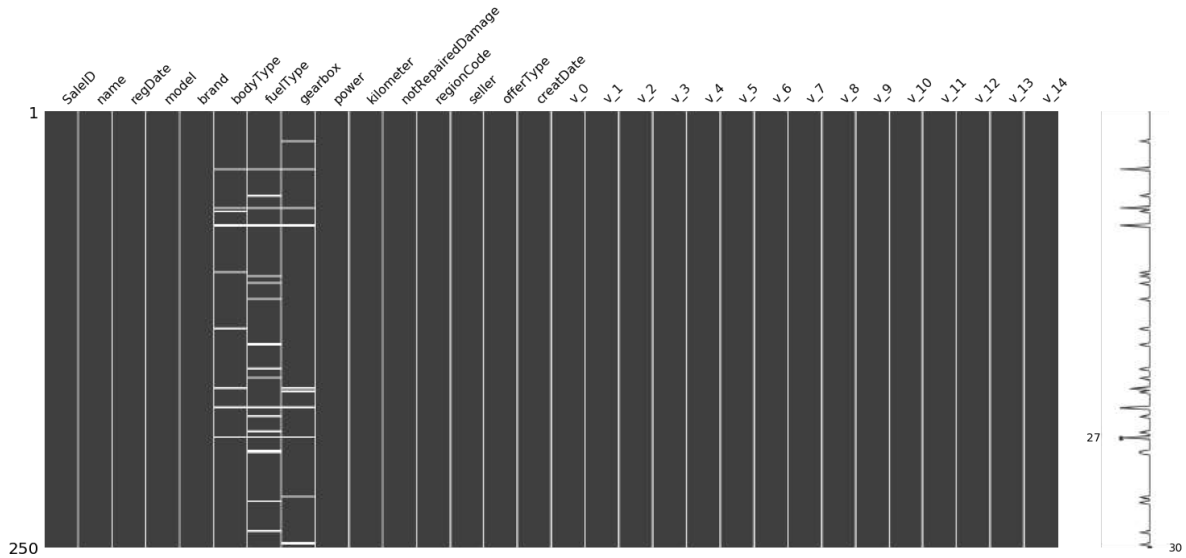


In [22]:

```
# 可视化看下缺省值
msno.matrix(Test_data.sample(250))
```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b1b4ba20>

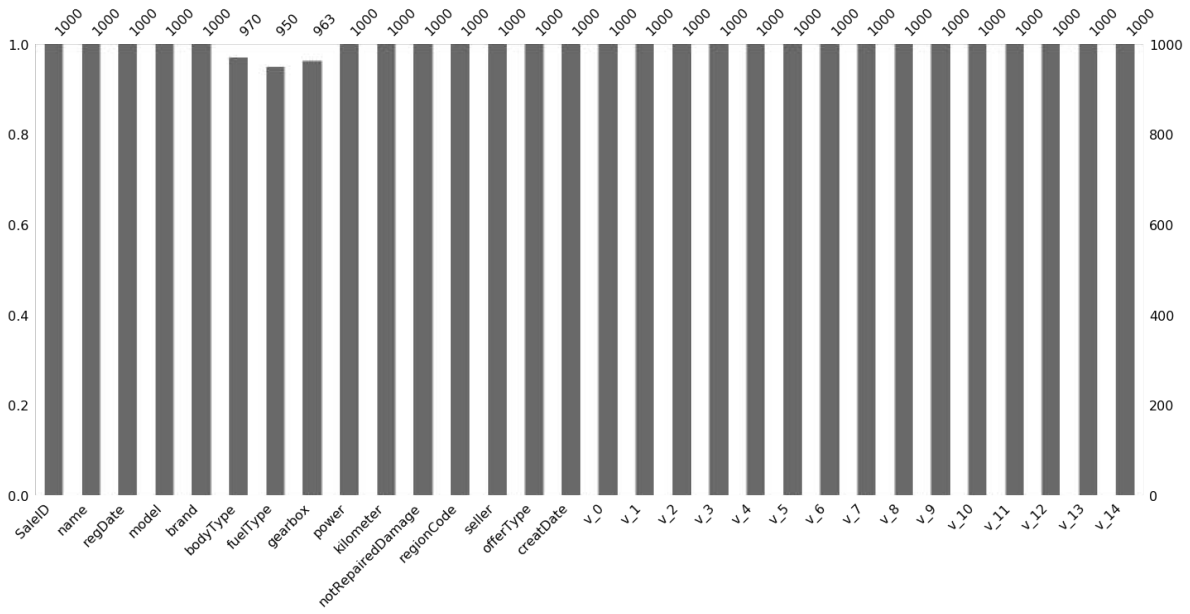


In [23]:

```
msno.bar(Test_data.sample(1000))
```

Out[23]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b1bc5978>



测试集的缺省和训练集的差不多情况，可视化有四列有缺省，notRepairedDamage缺省得最多

In [24]:

```
## 2) 查看异常值检测
```

In [25]:

```
Train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 31 columns):
SaleID               150000 non-null int64
name                 150000 non-null int64
regDate              150000 non-null int64
model                149999 non-null float64
brand                150000 non-null int64
bodyType             145494 non-null float64
fuelType             141320 non-null float64
gearbox              144019 non-null float64
power                150000 non-null int64
kilometer            150000 non-null float64
notRepairedDamage    150000 non-null object
regionCode           150000 non-null int64
seller               150000 non-null int64
offerType            150000 non-null int64
creatDate            150000 non-null int64
price                150000 non-null int64
v_0                  150000 non-null float64
v_1                  150000 non-null float64
v_2                  150000 non-null float64
v_3                  150000 non-null float64
v_4                  150000 non-null float64
v_5                  150000 non-null float64
v_6                  150000 non-null float64
v_7                  150000 non-null float64
v_8                  150000 non-null float64
v_9                  150000 non-null float64
v_10                 150000 non-null float64
v_11                 150000 non-null float64
v_12                 150000 non-null float64
v_13                 150000 non-null float64
v_14                 150000 non-null float64
dtypes: float64(20), int64(10), object(1)
memory usage: 35.5+ MB
```

可以发现除了notRepairedDamage 为object类型其他都为数字 这里我们把他的几个不同的值都进行显示就知道了

In [26]:

```
Train_data['notRepairedDamage'].value_counts()
```

Out[26]:

```
0.0    111361
-       24324
1.0     14315
Name: notRepairedDamage, dtype: int64
```

可以看出来‘-’也为空缺值，因为很多模型对nan有直接的处理，这里我们先不做处理，先替换成nan

In [27]:

```
Train_data['notRepairedDamage'].replace('-', np.nan, inplace=True)
```

In [28]:

```
Train_data['notRepairedDamage'].value_counts()
```

Out[28]:

```
0.0    111361
1.0     14315
Name: notRepairedDamage, dtype: int64
```

In [29]:

```
Train_data.isnull().sum()
```

Out[29]:

```
SaleID                0
name                  0
regDate               0
model                 1
brand                 0
bodyType           4506
fuelType           8680
gearbox            5981
power                 0
kilometer             0
notRepairedDamage    24324
regionCode            0
seller                0
offerType             0
creatDate             0
price                 0
v_0                   0
v_1                   0
v_2                   0
v_3                   0
v_4                   0
v_5                   0
v_6                   0
v_7                   0
v_8                   0
v_9                   0
v_10                  0
v_11                  0
v_12                  0
v_13                  0
v_14                  0
dtype: int64
```

In [30]:

```
Test_data['notRepairedDamage'].value_counts()
```

Out[30]:

```
0.0    37249
-       8031
1.0     4720
Name: notRepairedDamage, dtype: int64
```

In [31]:

```
Test_data['notRepairedDamage'].replace('-', np.nan, inplace=True)
```

以下两个类别特征严重倾斜，一般不会对预测有什么帮助，故这边先删掉，当然你也可以继续挖掘，但是一般意义不大

In [32]:

```
Train_data["seller"].value_counts()
```

Out[32]:

```
0    149999
1         1
Name: seller, dtype: int64
```

In [33]:

```
Train_data["offerType"].value_counts()
```

Out[33]:

```
0    150000
Name: offerType, dtype: int64
```

In [34]:

```
del Train_data["seller"]
del Train_data["offerType"]
del Test_data["seller"]
del Test_data["offerType"]
```

2.3.5 了解预测值的分布

In [35]:

```
Train_data['price']
```

Out[35]:

```
0          1850
1          3600
2          6222
3          2400
4          5200
           ...
149995     5900
149996     9500
149997     7500
149998     4999
149999     4700
Name: price, Length: 150000, dtype: int64
```

In [36]:

```
Train_data['price'].value_counts()
```

Out[36]:

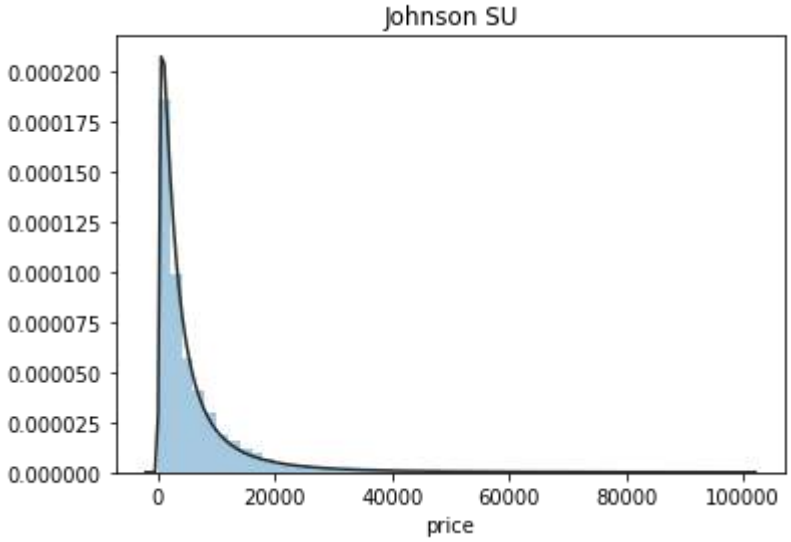
```
500       2337
1500      2158
1200      1922
1000      1850
2500      1821
          ...
25321        1
8886         1
8801         1
37920        1
8188         1
Name: price, Length: 3763, dtype: int64
```

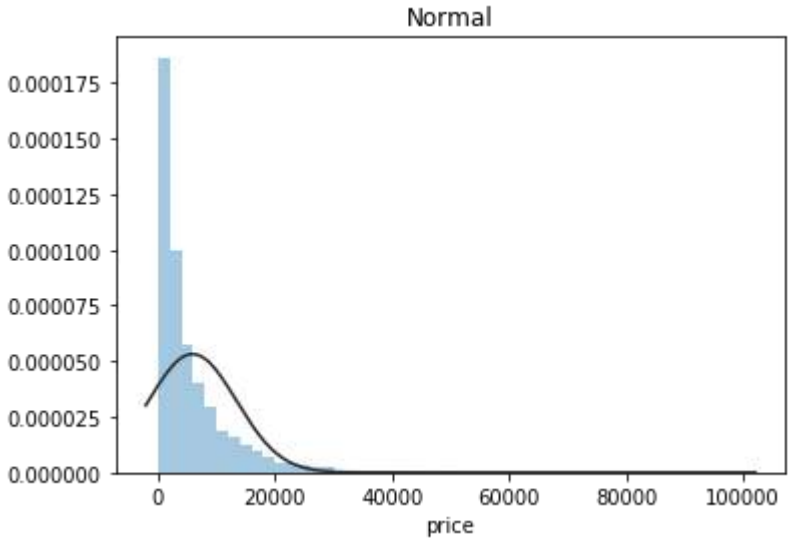
In [37]:

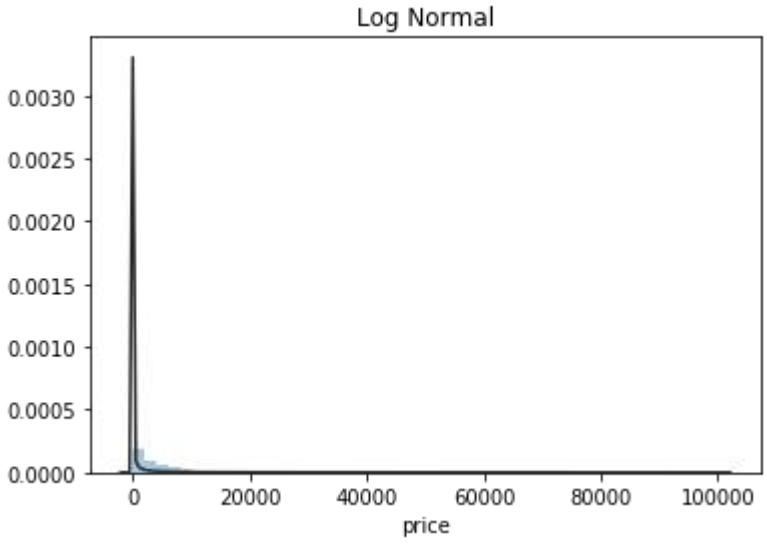
```
## 1）总体分布概况（无界约翰逊分布等）
import scipy.stats as st
y = Train_data['price']
plt.figure(1); plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)
plt.figure(2); plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)
plt.figure(3); plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```

Out[37]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b1ee82b0>





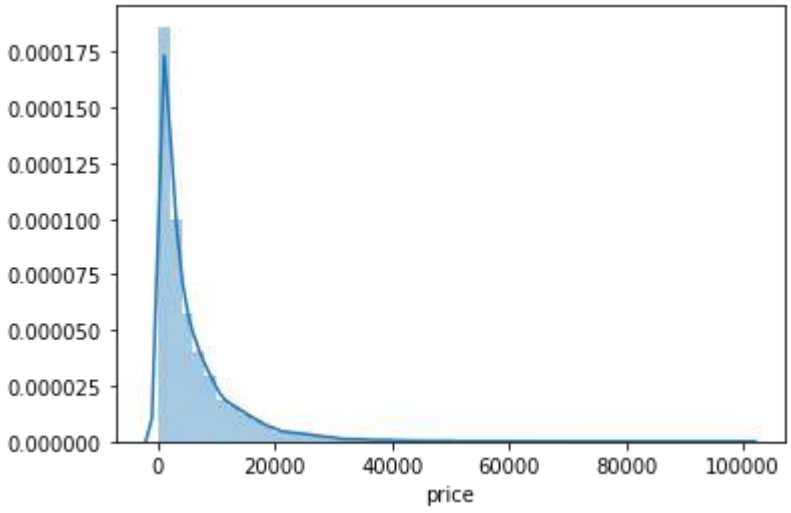


价格不服从正态分布，所以在进行回归之前，它必须进行转换。虽然对数变换做得很好，但最佳拟合是无界约翰逊分布

In [38]:

```
## 2) 查看skewness and kurtosis
sns.distplot(Train_data['price']);
print("Skewness: %f" % Train_data['price'].skew())
print("Kurtosis: %f" % Train_data['price'].kurt())
```

```
Skewness: 3.346487
Kurtosis: 18.995183
```



In [39]:

```
Train_data.skew(), Train_data.kurt()
```

Out[39]:

```
(SaleID               6.017846e-17
 name                 5.576058e-01
 regDate              2.849508e-02
 model                1.484388e+00
 brand                1.150760e+00
 bodyType             9.915299e-01
 fuelType             1.595486e+00
 gearbox              1.317514e+00
 power                6.586318e+01
 kilometer           -1.525921e+00
 notRepairedDamage    2.430640e+00
 regionCode           6.888812e-01
 creatDate           -7.901331e+01
 price                3.346487e+00
 v_0                 -1.316712e+00
 v_1                  3.594543e-01
 v_2                  4.842556e+00
 v_3                  1.062920e-01
 v_4                  3.679890e-01
 v_5                 -4.737094e+00
 v_6                  3.680730e-01
 v_7                  5.130233e+00
 v_8                  2.046133e-01
 v_9                  4.195007e-01
 v_10                 2.522046e-02
 v_11                 3.029146e+00
 v_12                 3.653576e-01
 v_13                 2.679152e-01
 v_14                -1.186355e+00
 dtype: float64, SaleID               -1.200000
 name                   -1.039945
 regDate                -0.697308
 model                   1.740483
 brand                   1.076201
 bodyType                0.206937
 fuelType                5.880049
 gearbox                -0.264161
 power                5733.451054
 kilometer               1.141934
 notRepairedDamage       3.908072
 regionCode             -0.340832
 creatDate            6881.080328
 price                  18.995183
 v_0                     3.993841
 v_1                    -1.753017
 v_2                    23.860591
 v_3                    -0.418006
 v_4                    -0.197295
 v_5                    22.934081
 v_6                    -1.742567
 v_7                    25.845489
 v_8                    -0.636225
 v_9                    -0.321491
 v_10                   -0.577935
 v_11                   12.568731
```

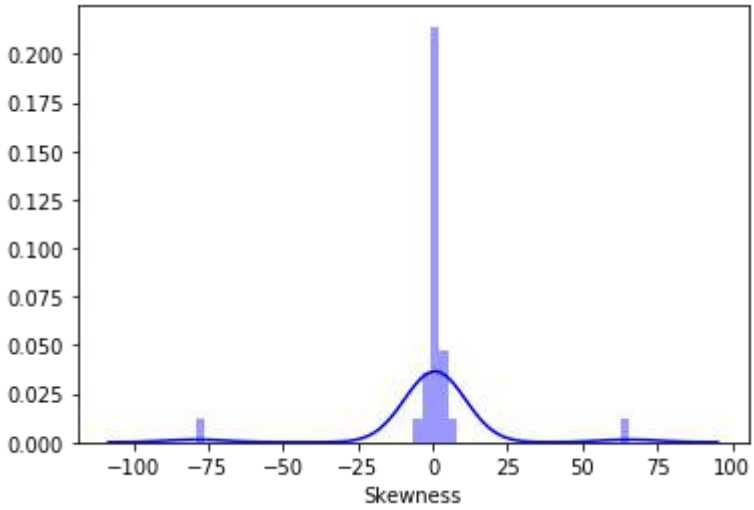
```
v_12                    0.268937
v_13                   -0.438274
v_14                    2.393526
dtype: float64)
```

In [40]:

```
sns.distplot(Train_data.skew(),color='blue',axlabel ='Skewness')
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x111b484ab38>
```

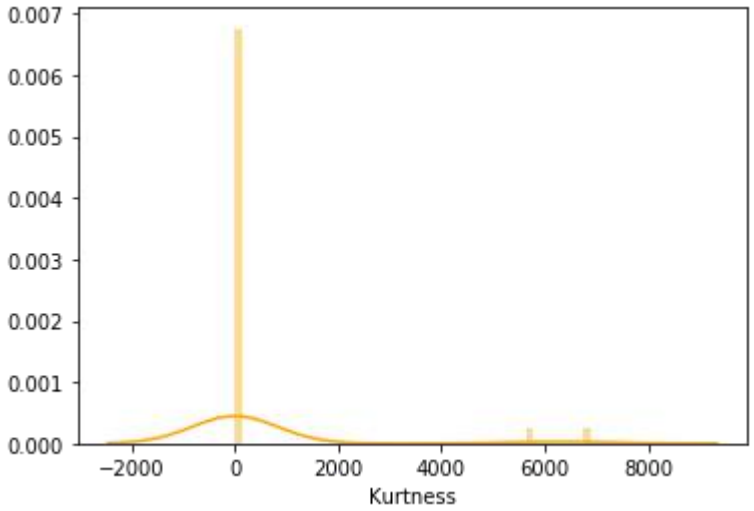


In [41]:

```
sns.distplot(Train_data.kurt(),color='orange',axlabel ='Kurtness')
```

Out[41]:

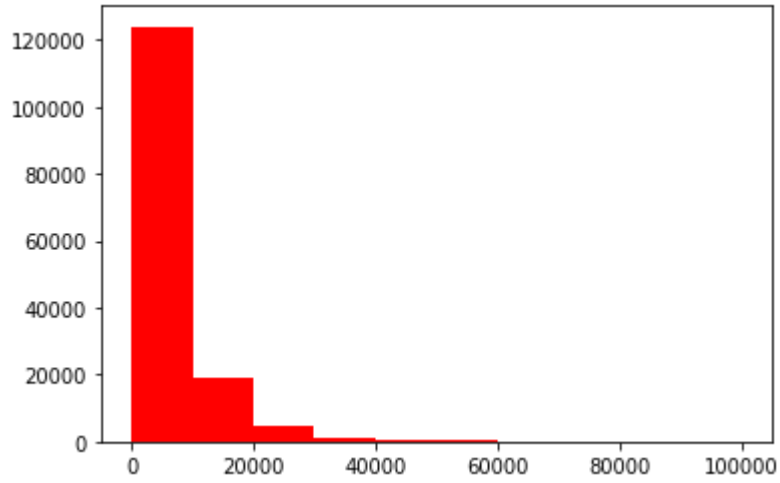
```
<matplotlib.axes._subplots.AxesSubplot at 0x111b4953940>
```



skew、kurt说明参考https://www.cnblogs.com/wyy1480/p/10474046.html (https://www.cnblogs.com/wyy1480/p/10474046.html)

In [42]:

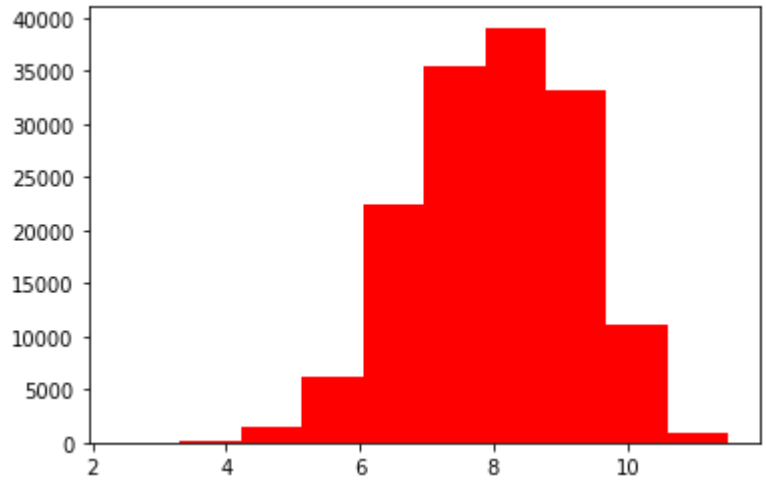
```
## 3) 查看预测值的具体频数
plt.hist(Train_data['price'], orientation = 'vertical',histtype = 'bar', color ='red')
plt.show()
```



查看频数, 大于20000得值极少，其实这里也可以把这些当作特殊得值（异常值）直接用填充或者删掉，再前面进行

In [43]:

```
# log变换 z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(Train_data['price']), orientation = 'vertical',histtype = 'bar', color ='red')
plt.show()
```



2.3.6 特征分为类别特征和数字特征，并对类别特征查看unique分布

数据类型

列

- name - 汽车编码
- regDate - 汽车注册时间
- model - 车型编码
- brand - 品牌
- bodyType - 车身类型
- fuelType - 燃油类型
- gearbox - 变速箱
- power - 汽车功率
- kilometer - 汽车行驶公里
- notRepairedDamage - 汽车有尚未修复的损坏
- regionCode - 看车地区编码
- seller - 销售方 【以删】
- offerType - 报价类型 【以删】
- creatDate - 广告发布时间
- price - 汽车价格
- v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13','v_14' （根据汽车的评论、标签等大量信息得到的embedding向量）【人工构造 匿名特征】

In [44]:

```
# 分离label即预测值
Y_train = Train_data['price']
```

In [45]:

```
# 这个区别方式适用于没有直接label coding的数据
# 这里不适用，需要人为根据实际含义来区分
# 数字特征
# numeric_features = Train_data.select_dtypes(include=[np.number])
# numeric_features.columns
# # 类型特征
# categorical_features = Train_data.select_dtypes(include=[np.object])
# categorical_features.columns
```

In [46]:

```
numeric_features = ['power', 'kilometer', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v

categorical_features = ['name', 'model', 'brand', 'bodyType', 'fuelType', 'gearbox', 'notRepairedDam
```

In [47]:

```
# 特征nunique分布
for cat_fea in categorical_features:
    print(cat_fea + "的特征分布如下：")
    print("{}特征有个{}不同的值".format(cat_fea, Train_data[cat_fea].nunique()))
    print(Train_data[cat_fea].value_counts())
```

```
name的特征分布如下：
name特征有个99662不同的值
708       282
387       282
55        280
1541      263
203       233
         ...
5074        1
7123        1
11221       1
13270       1
174485      1
Name: name, Length: 99662, dtype: int64
model的特征分布如下：
model特征有个248不同的值
0.0      11762
19.0      9573
4.0       8445
```

In [50]:

```
# 特征nunique分布
for cat_fea in categorical_features:
    print(cat_fea + "的特征分布如下：")
    print("{}特征有个{}不同的值".format(cat_fea, Test_data[cat_fea].nunique()))
    print(Test_data[cat_fea].value_counts())
```

```
name的特征分布如下：
name特征有个37453不同的值
55       97
708      96
387      95
1541     88
713      74
         ..
22270     1
89855     1
42752     1
48899     1
11808     1
Name: name, Length: 37453, dtype: int64
model的特征分布如下：
model特征有个247不同的值
0.0      3896
19.0     3245
4.0      3007
```

2.3.7 数字特征分析

In [54]:

```
numeric_features.append('price')
```

In [55]:

```
numeric_features
```

Out[55]:

```
['power',
 'kilometer',
 'v_0',
 'v_1',
 'v_2',
 'v_3',
 'v_4',
 'v_5',
 'v_6',
 'v_7',
 'v_8',
 'v_9',
 'v_10',
 'v_11',
 'v_12',
 'v_13',
 'v_14',
 'price']
```

In [56]:

```
Train_data.head()
```

Out[56]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilometer	...
0	0	736	20040402	30.0	6	1.0	0.0	0.0	60	12.5	...
1	1	2262	20030301	40.0	1	2.0	0.0	0.0	0	15.0	...
2	2	14874	20040403	115.0	15	1.0	0.0	0.0	163	12.5	...
3	3	71865	19960908	109.0	10	0.0	0.0	1.0	193	15.0	...
4	4	111080	20120103	110.0	5	1.0	0.0	0.0	68	5.0	...

5 rows × 29 columns

In [57]:

```
## 1）相关性分析
price_numeric = Train_data[numeric_features]
correlation = price_numeric.corr()
print(correlation['price'].sort_values(ascending = False),'\n')
```

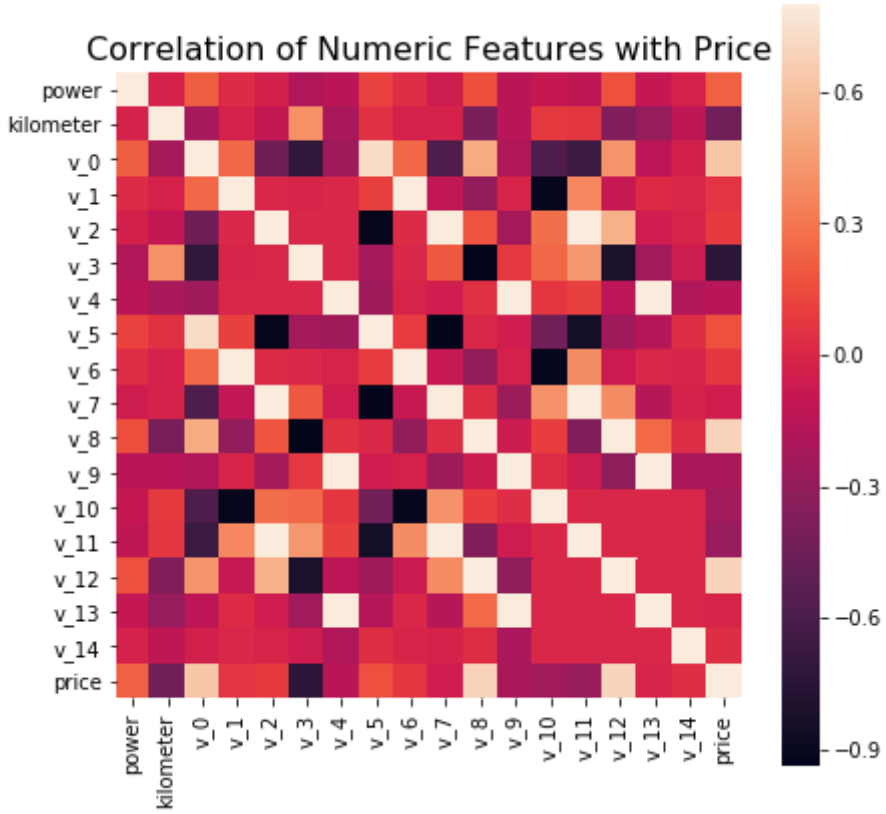
```
price         1.000000
v_12          0.692823
v_8           0.685798
v_0           0.628397
power         0.219834
v_5           0.164317
v_2           0.085322
v_6           0.068970
v_1           0.060914
v_14          0.035911
v_13         -0.013993
v_7          -0.053024
v_4          -0.147085
v_9          -0.206205
v_10         -0.246175
v_11         -0.275320
kilometer    -0.440519
v_3          -0.730946
Name: price, dtype: float64
```

In [58]:

```
f , ax = plt.subplots(figsize = (7, 7))

plt.title('Correlation of Numeric Features with Price',y=1,size=16)

sns.heatmap(correlation,square = True,  vmax=0.8)
```

Out[58]:

<matplotlib.axes._subplots.AxesSubplot at 0x111b6ed5b70>



In [59]:

```
del price_numeric['price']
```

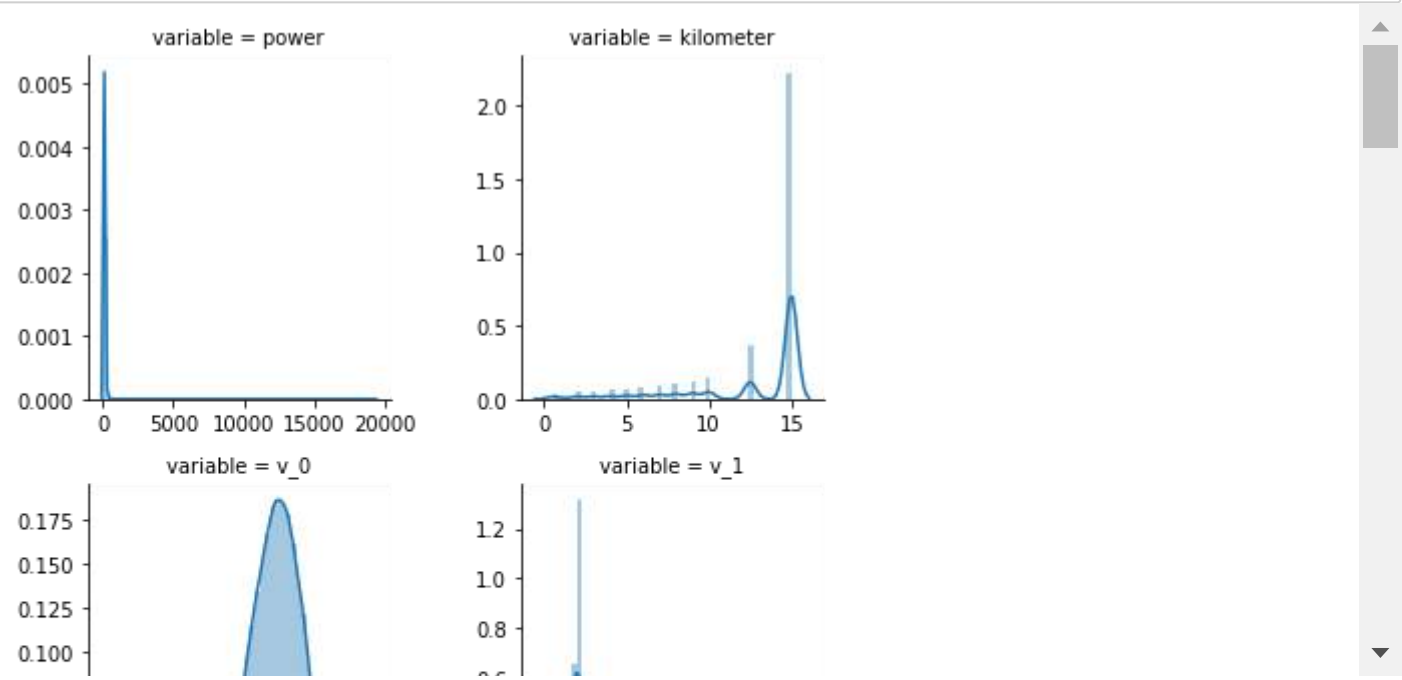
In [47]:

```
## 2) 查看几个特征得 偏度和峰值
for col in numeric_features:
    print('{:15}'.format(col),
          'Skewness: {:05.2f}'.format(Train_data[col].skew()) ,
          '   ' ,
          'Kurtosis: {:06.2f}'.format(Train_data[col].kurt())
         )
```

```
power           Skewness: 65.86     Kurtosis: 5733.45
kilometer       Skewness: -1.53     Kurtosis: 001.14
v_0             Skewness: -1.32     Kurtosis: 003.99
v_1             Skewness: 00.36     Kurtosis: -01.75
v_2             Skewness: 04.84     Kurtosis: 023.86
v_3             Skewness: 00.11     Kurtosis: -00.42
v_4             Skewness: 00.37     Kurtosis: -00.20
v_5             Skewness: -4.74     Kurtosis: 022.93
v_6             Skewness: 00.37     Kurtosis: -01.74
v_7             Skewness: 05.13     Kurtosis: 025.85
v_8             Skewness: 00.20     Kurtosis: -00.64
v_9             Skewness: 00.42     Kurtosis: -00.32
v_10            Skewness: 00.03     Kurtosis: -00.58
v_11            Skewness: 03.03     Kurtosis: 012.57
v_12            Skewness: 00.37     Kurtosis: 000.27
v_13            Skewness: 00.27     Kurtosis: -00.44
v_14            Skewness: -1.19     Kurtosis: 002.39
price           Skewness: 03.35     Kurtosis: 019.00
```

In [48]:

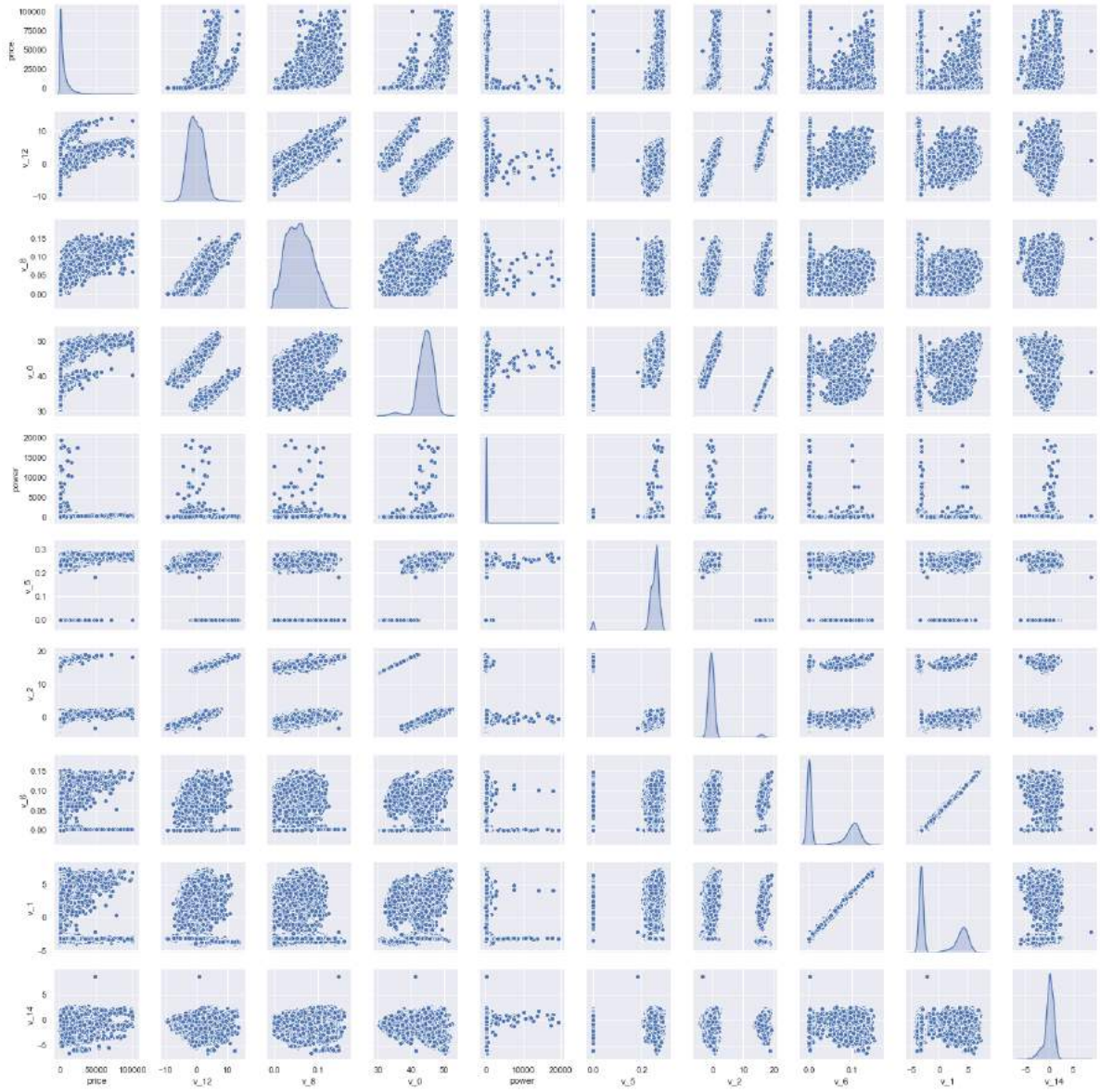
```
## 3) 每个数字特征得分布可视化
f = pd.melt(Train_data, value_vars=numeric_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False)
g = g.map(sns.distplot, "value")
```



可以看出匿名特征相对分布均匀

In [52]:

```
## 4) 数字特征相互之间的关系可视化
sns.set()
columns = ['price', 'v_12', 'v_8' , 'v_0', 'power', 'v_5',  'v_2', 'v_6', 'v_1', 'v_14']
sns.pairplot(Train_data[columns],size = 2 ,kind ='scatter',diag_kind='kde')
plt.show()
```



In [60]:

```
Train_data.columns
```

Out[60]:

```
Index(['SaleID', 'name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType',
       'gearbox', 'power', 'kilometer', 'notRepairedDamage', 'regionCode',
       'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6',
       'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13', 'v_14'],
      dtype='object')
```

In [62]:

```
Y_train
```

Out[62]:

```
0          1850
1          3600
2          6222
3          2400
4          5200
           ...
149995     5900
149996     9500
149997     7500
149998     4999
149999     4700
Name: price, Length: 150000, dtype: int64
```

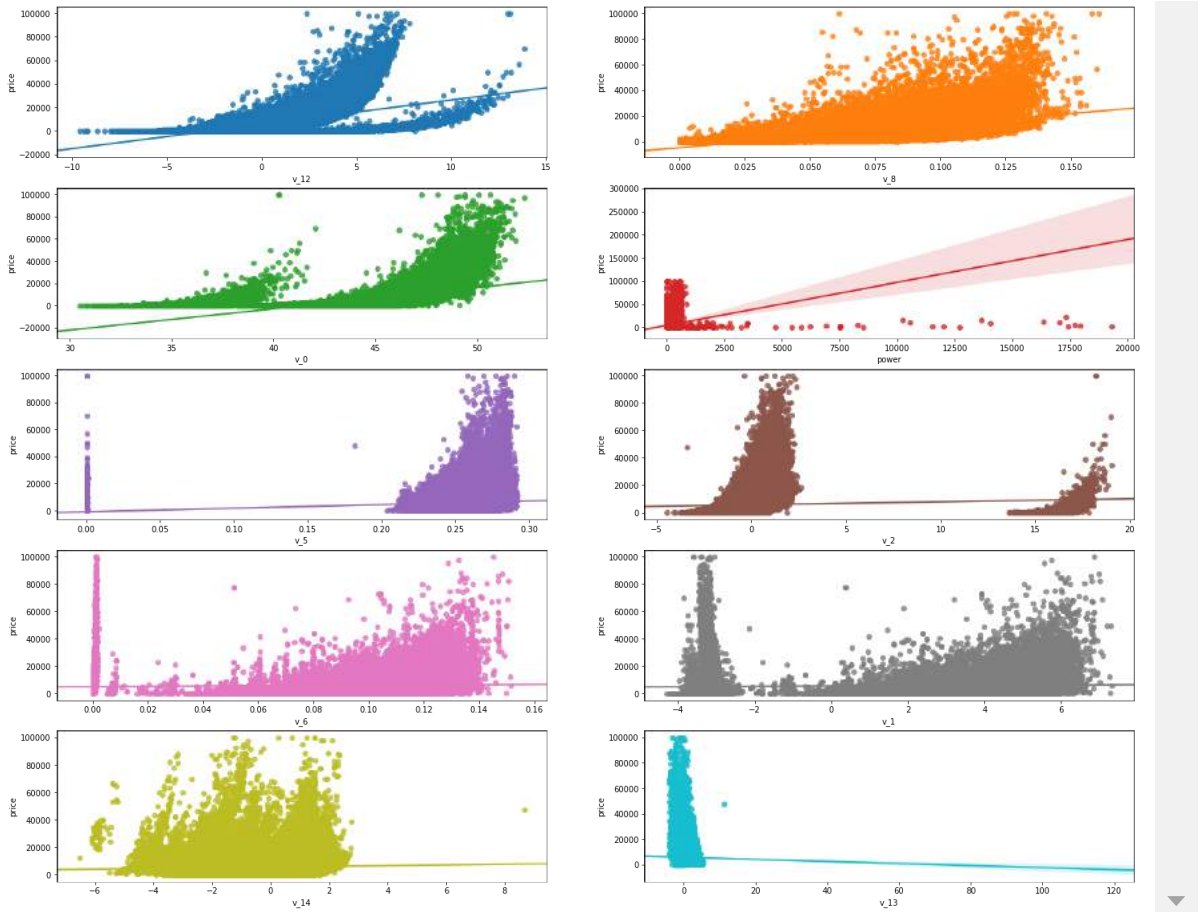
此处是多变量之间的关系可视化，可视化更多学习可参考很不错的文章
https://www.jianshu.com/p/6e18d21a4cad (https://www.jianshu.com/p/6e18d21a4cad)

In [63]:

```
## 5）多变量互相回归关系可视化
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6), (ax7, ax8), (ax9, ax10)) = plt.subplots(nrows=5, ncols=2,
# ['v_12', 'v_8' , 'v_0', 'power', 'v_5',  'v_2', 'v_6', 'v_1', 'v_14']
v_12_scatter_plot = pd.concat([Y_train,Train_data['v_12']],axis = 1)
sns.regplot(x='v_12',y = 'price', data = v_12_scatter_plot,scatter= True, fit_reg=True, ax=ax1)

v_8_scatter_plot = pd.concat([Y_train,Train_data['v_8']],axis = 1)
sns.regplot(x='v_8',y = 'price',data = v_8_scatter_plot,scatter= True, fit_reg=True, ax=ax2)

v_0_scatter_plot = pd.concat([Y_train,Train_data['v_0']],axis = 1)
sns.regplot(x='v_0',y = 'price',data = v_0_scatter_plot,scatter= True, fit_reg=True, ax=ax3)

power_scatter_plot = pd.concat([Y_train,Train_data['power']],axis = 1)
sns.regplot(x='power',y = 'price',data = power_scatter_plot,scatter= True, fit_reg=True, ax=ax4)

v_5_scatter_plot = pd.concat([Y_train,Train_data['v_5']],axis = 1)
sns.regplot(x='v_5',y = 'price',data = v_5_scatter_plot,scatter= True, fit_reg=True, ax=ax5)

v_2_scatter_plot = pd.concat([Y_train,Train_data['v_2']],axis = 1)
sns.regplot(x='v_2',y = 'price',data = v_2_scatter_plot,scatter= True, fit_reg=True, ax=ax6)

v_6_scatter_plot = pd.concat([Y_train,Train_data['v_6']],axis = 1)
sns.regplot(x='v_6',y = 'price',data = v_6_scatter_plot,scatter= True, fit_reg=True, ax=ax7)

v_1_scatter_plot = pd.concat([Y_train,Train_data['v_1']],axis = 1)
sns.regplot(x='v_1',y = 'price',data = v_1_scatter_plot,scatter= True, fit_reg=True, ax=ax8)

v_14_scatter_plot = pd.concat([Y_train,Train_data['v_14']],axis = 1)
sns.regplot(x='v_14',y = 'price',data = v_14_scatter_plot,scatter= True, fit_reg=True, ax=ax9)

v_13_scatter_plot = pd.concat([Y_train,Train_data['v_13']],axis = 1)
sns.regplot(x='v_13',y = 'price',data = v_13_scatter_plot,scatter= True, fit_reg=True, ax=ax10)
```

Out[63]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x111b47b2b38>
```



2.3.8 类别特征分析

In [55]:

```
## 1) unique分布
for fea in categorical_features:
    print(Train_data[fea].nunique())
```

```
99662
248
40
8
7
2
2
7905
```

In [56]:

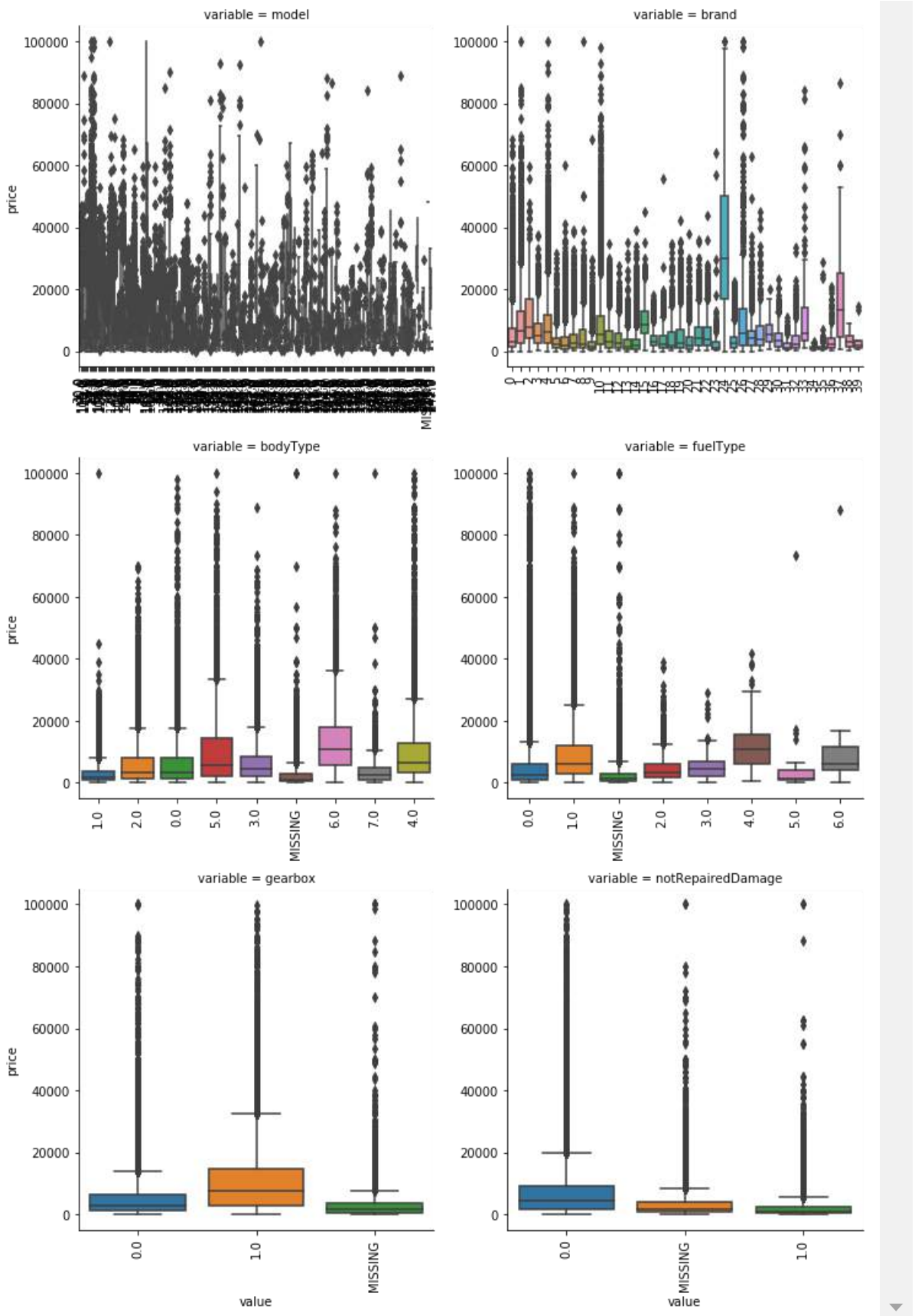
```
categorical_features
```

Out[56]:

```
['name',
 'model',
 'brand',
 'bodyType',
 'fuelType',
 'gearbox',
 'notRepairedDamage',
 'regionCode']
```

In [64]:

```
## 2) 类别特征箱形图可视化

# 因为 name和 regionCode的类别太稀疏了，这里我们把不稀疏的几类画一下
categorical_features = ['model',
 'brand',
 'bodyType',
 'fuelType',
 'gearbox',
 'notRepairedDamage']
for c in categorical_features:
    Train_data[c] = Train_data[c].astype('category')
    if Train_data[c].isnull().any():
        Train_data[c] = Train_data[c].cat.add_categories(['MISSING'])
        Train_data[c] = Train_data[c].fillna('MISSING')

def boxplot(x, y, **kwargs):
    sns.boxplot(x=x, y=y)
    x=plt.xticks(rotation=90)

f = pd.melt(Train_data, id_vars=['price'], value_vars=categorical_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False, size=5)
g = g.map(boxplot, "value", "price")
```


variable = model
variable = brand
variable = bodyType
variable = fuelType
variable = gearbox
variable = notRepairedDamage
price
value
MISSING

In [65]:

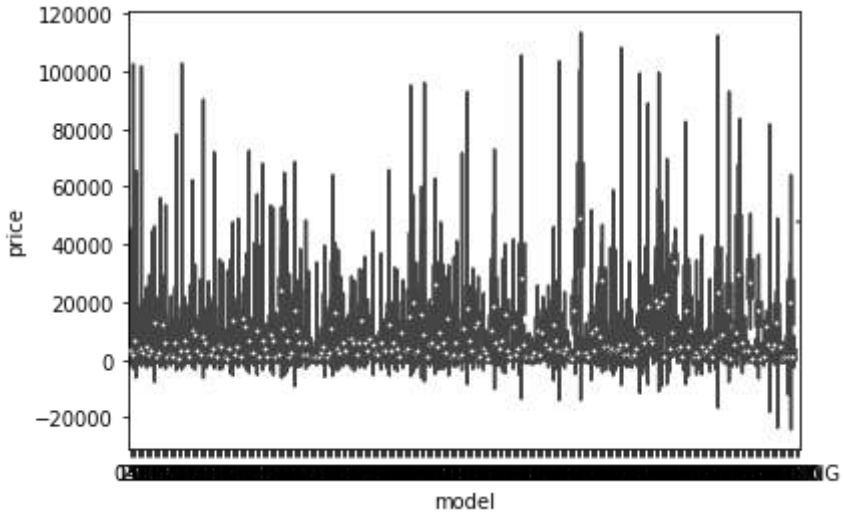
```
Train_data.columns
```

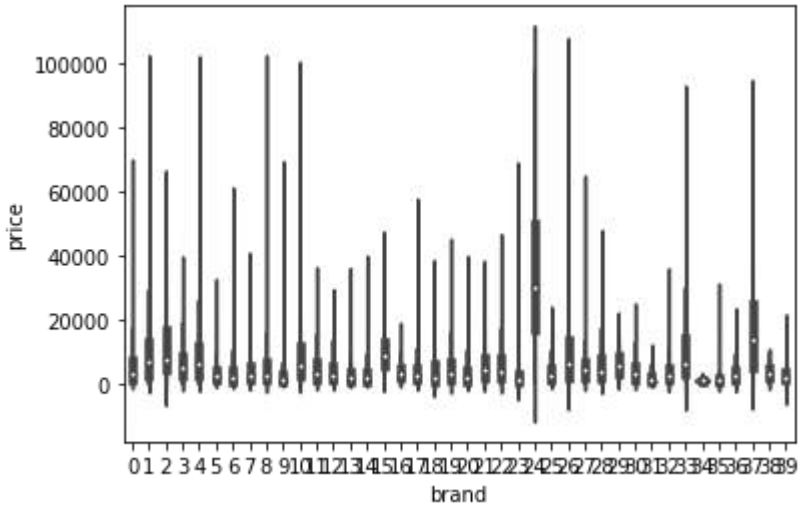
Out[65]:

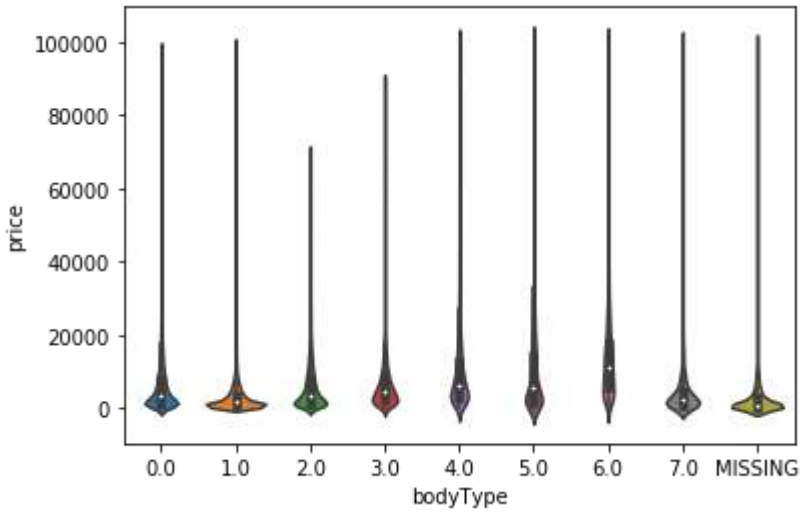
```
Index(['SaleID', 'name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType',
       'gearbox', 'power', 'kilometer', 'notRepairedDamage', 'regionCode',
       'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6',
       'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13', 'v_14'],
      dtype='object')
```

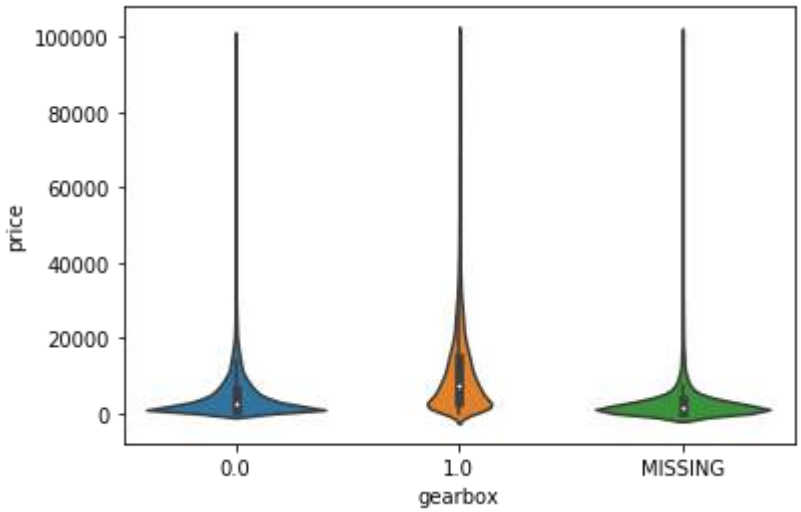
In [66]:

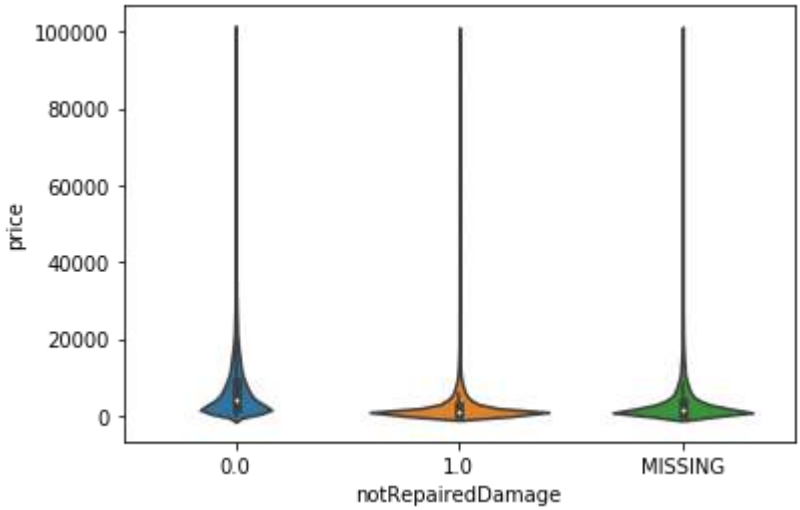
```
## 3）类别特征的小提琴图可视化
catg_list = categorical_features
target = 'price'
for catg in catg_list :
    sns.violinplot(x=catg, y=target, data=Train_data)
    plt.show()
```









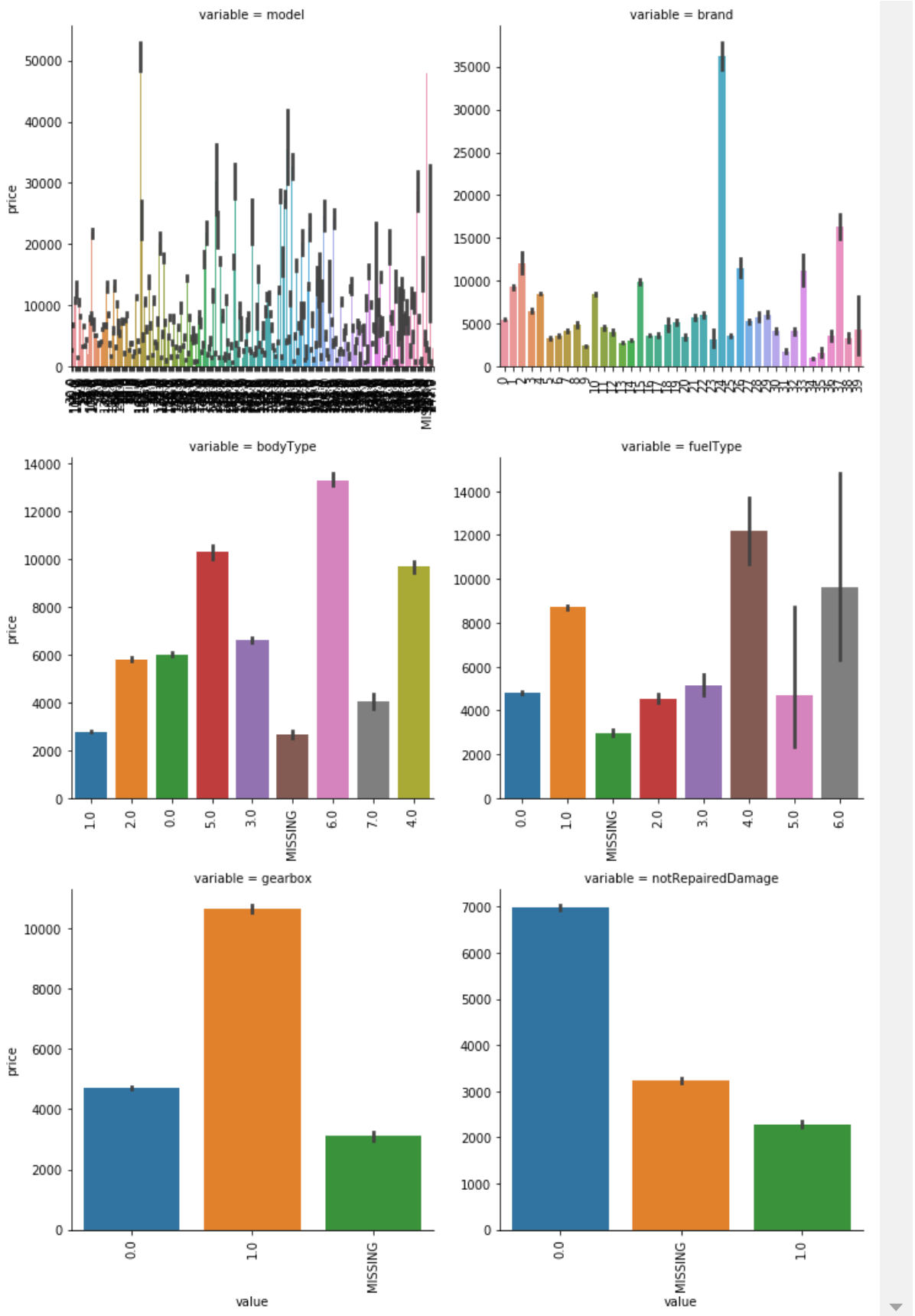


In [67]:

```
categorical_features = ['model',
 'brand',
 'bodyType',
 'fuelType',
 'gearbox',
 'notRepairedDamage']
```

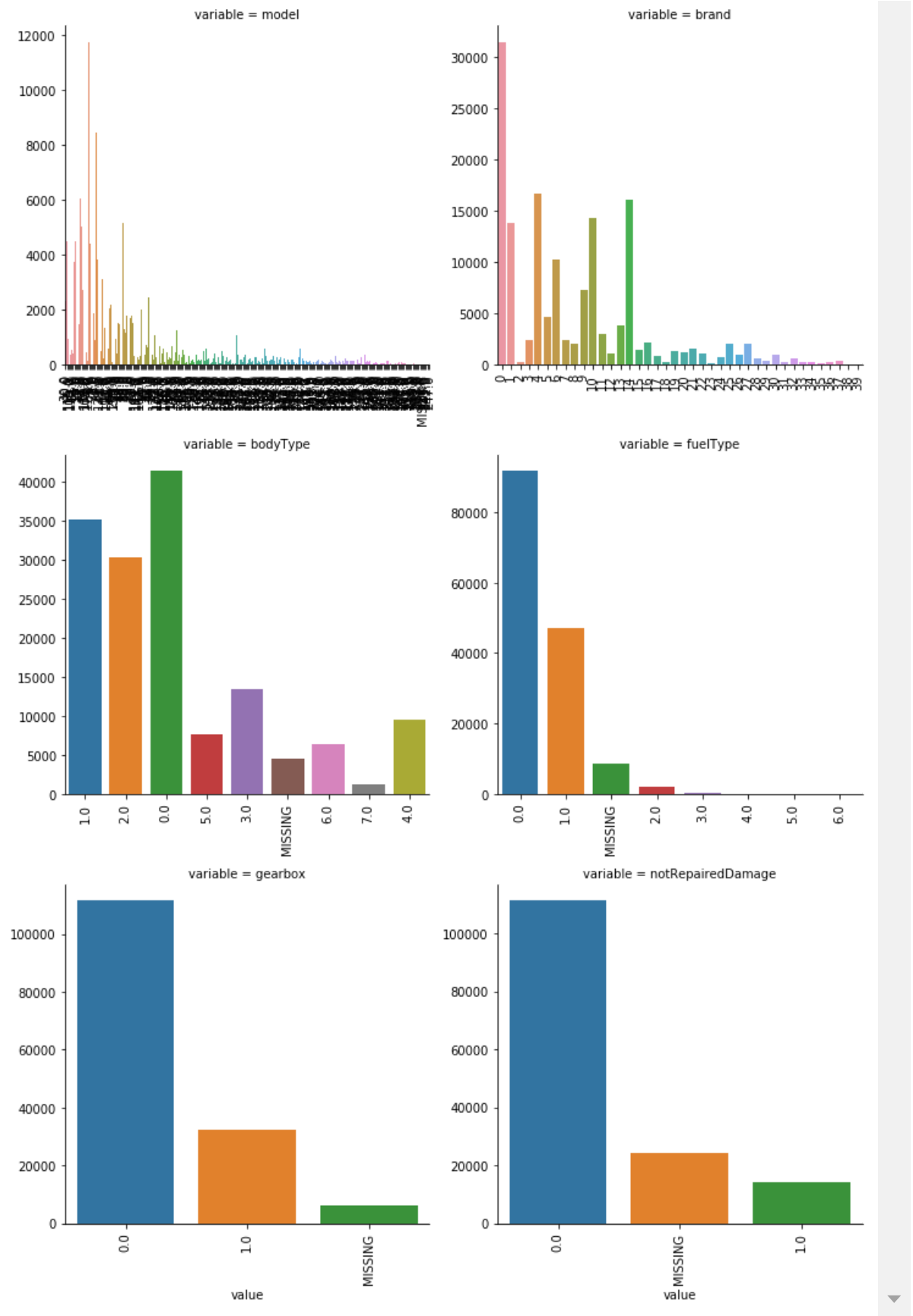
In [68]:

```
## 4）类别特征的柱形图可视化
def bar_plot(x, y, **kwargs):
    sns.barplot(x=x, y=y)
    x=plt.xticks(rotation=90)

f = pd.melt(Train_data, id_vars=['price'], value_vars=categorical_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False, size=5)
g = g.map(bar_plot, "value", "price")
```


variable = model
variable = brand
variable = bodyType
variable = fuelType
variable = gearbox
variable = notRepairedDamage
price
value
MISSING

In [69]:

```
##  5）类别特征的每个类别频数可视化(count_plot)
def count_plot(x,  **kwargs):
    sns.countplot(x=x)
    x=plt.xticks(rotation=90)

f = pd.melt(Train_data,  value_vars=categorical_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False, size=5)
g = g.map(count_plot, "value")
```



2.3.9 用pandas_profiling生成数据报告

用pandas_profiling生成一个较为全面的可视化和数据报告(较为简单、方便) 最终打开html文件即可

In [63]:

```
import pandas_profiling
```

In [64]:

```
pfr = pandas_profiling.ProfileReport(Train_data)
pfr.to_file("./example.html")
```

```
HBox(children=(FloatProgress(value=0.0, description='variables', max=29.0, style=Pro
gressStyle(description_wid…
```

```
HBox(children=(FloatProgress(value=0.0, description='correlations', max=6.0, style=P
rogressStyle(description_w…
```

```
HBox(children=(FloatProgress(value=0.0, description='interactions [continuous]', max
=729.0, style=ProgressStyl…
```

```
HBox(children=(FloatProgress(value=0.0, description='table', max=1.0, style=Progress
Style(description_width='i…
```

```
HBox(children=(FloatProgress(value=0.0, description='missing', max=4.0, style=Progre
ssStyle(description_width=…
```

```
HBox(children=(FloatProgress(value=0.0, description='warnings', max=3.0, style=Progr
essStyle(description_width…
```

```
HBox(children=(FloatProgress(value=0.0, description='package', max=1.0, style=Progre
ssStyle(description_width=…
```

```
HBox(children=(FloatProgress(value=0.0, description='build report structure', max=1.
0, style=ProgressStyle(des…
```

2.4 经验总结

所给出的EDA步骤为广为普遍的步骤，在实际的不管是工程还是比赛过程中，这只是最开始的一步，也是最基本的一步。

接下来一般要结合模型的效果以及特征工程等来分析数据的实际建模情况，根据自己的一些理解，查阅文献，对实际问题做出判断和深入的理解。

最后不断进行EDA与数据处理和挖掘，来到达更好的数据结构和分布以及较为强势相关的特征

数据探索在机器学习中我们一般称为EDA（Exploratory Data Analysis）：

> 是指对已有的数据（特别是调查或观察得来的原始数据）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的一种数据分析方法。

数据探索有利于我们发现数据的一些特性，数据之间的关联性，对于后续的特征构建是很有帮助的。

1. 对于数据的初步分析（直接查看数据，或.sum(), .mean()，.descirbe()等统计函数）可以从：样本数量，训练集数量，是否有时间特征，是否是时许问题，特征所表示的含义（非匿名特征），特征类型（字符类似，int，float，time），特征的缺失情况（注意缺失的在数据中的表现形式，有些是空的有些是”NAN”符号等），特征的均值方差情况。
2. 分析记录某些特征值缺失占比30%以上样本的缺失处理，有助于后续的模型验证和调节，分析特征应该是填充（填充方式是什么，均值填充，0填充，众数填充等），还是舍去，还是先做样本分类用不同的特征模型去预测。
3. 对于异常值做专门的分析，分析特征异常的label是否为异常值（或者偏离均值较远或者事特殊符号）,异常值是否应该剔除，还是用正常值填充，是记录异常，还是机器本身异常等。
4. 对于Label做专门的分析，分析标签的分布情况等。
5. 进步分析可以通过对特征作图，特征和label联合做图（统计图，离散图），直观了解特征的分布情况，通过这一步也可以发现数据之中的一些异常值等，通过箱型图分析一些特征值的偏离情况，对于特征和特征联合作图，对于特征和label联合作图，分析其中的一些关联性。

Task2 EDA数据分析 END.

--- By: AI蜗牛车

PS：东南大学研究生，研究方向主要是时空序列预测和时间序列数据挖掘
公众号： AI蜗牛车
知乎： https://www.zhihu.com/people/seu-aigua-niu-che
github: https://github.com/chehongshu

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale： （图片！！！）

Datawhale 零基础入门数据挖掘-Task3 特征工程

三、 特征工程目标

Tip:此部分为零基础入门数据挖掘的 Task3 特征工程 部分，带你来了解各种特征工程以及分析方法，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

3.1 特征工程目标

- 对于特征进行进一步分析，并对于数据进行处理
- 完成对于特征工程的分析，并对于数据进行一些图表或者文字总结并打卡。

3.2 内容介绍

常见的特征工程包括：

1. 异常处理：
   - 通过箱线图（或 3-Sigma）分析删除异常值；
   - BOX-COX 转换（处理有偏分布）；
   - 长尾截断；
2. 特征归一化/标准化：
   - 标准化（转换为标准正态分布）；
   - 归一化（抓换到 [0,1] 区间）；
   - 针对幂律分布，可以采用公式：$log(\frac{1+x}{1+median})$
3. 数据分桶：
   - 等频分桶；
   - 等距分桶；
   - Best-KS 分桶（类似利用基尼指数进行二分类）；
   - 卡方分桶；
4. 缺失值处理：
   - 不处理（针对类似 XGBoost 等树模型）；
   - 删除（缺失数据太多）；
   - 插值补全，包括均值/中位数/众数/建模预测/多重插补/压缩感知补全/矩阵补全等；
   - 分箱，缺失值一个箱；
5. 特征构造：
   - 构造统计量特征，报告计数、求和、比例、标准差等；
   - 时间特征，包括相对时间和绝对时间，节假日，双休日等；
   - 地理信息，包括分箱，分布编码等方法；
   - 非线性变换，包括 log/ 平方/ 根号等；
   - 特征组合，特征交叉；
   - 仁者见仁，智者见智。

6. 特征筛选
    - 过滤式（filter）：先对数据进行特征选择，然后在训练学习器，常见的方法有 Relief/方差选择发/相关系数法/卡方检验法/互信息法；
    - 包裹式（wrapper）：直接把最终将要使用的学习器的性能作为特征子集的评价准则，常见方法有 LVM（Las Vegas Wrapper）；
    - 嵌入式（embedding）：结合过滤式和包裹式，学习器训练过程中自动进行了特征选择，常见的有 lasso 回归；
7. 降维
    - PCA/ LDA/ ICA；
    - 特征选择也是一种降维。

3.3 代码示例

3.3.0 导入数据

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from operator import itemgetter

%matplotlib inline
```

In [2]:

```
train = pd.read_csv('train.csv', sep=' ')
test = pd.read_csv('testA.csv', sep=' ')
print(train.shape)
print(test.shape)
```

```
(150000, 30)
(50000, 30)
```

In [3]:

```
train.head()
```

Out[3]:

	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilometer	notRepaire
0	736	20040402	30.0	6	1.0	0.0	0.0	60	12.5	
1	2262	20030301	40.0	1	2.0	0.0	0.0	0	15.0	
2	14874	20040403	115.0	15	1.0	0.0	0.0	163	12.5	
3	71865	19960908	109.0	10	0.0	0.0	1.0	193	15.0	
4	111080	20120103	110.0	5	1.0	0.0	0.0	68	5.0	

5 rows × 30 columns

In [4]:

```
train.columns
```

Out[4]:

```
Index(['name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType', 'gearbox',
       'power', 'kilometer', 'notRepairedDamage', 'regionCode', 'seller',
       'offerType', 'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4',
       'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13',
       'v_14'],
      dtype='object')
```

In [5]:

```
test.columns
```

Out[5]:

```
Index(['name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType', 'gearbox',
       'power', 'kilometer', 'notRepairedDamage', 'regionCode', 'seller',
       'offerType', 'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4',
       'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13',
       'v_14'],
      dtype='object')
```

3.3.1 删除异常值

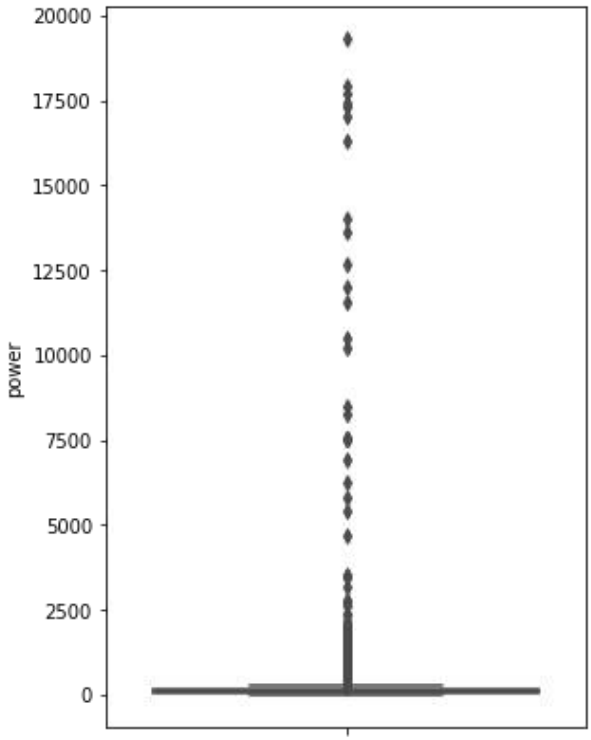
In [6]:

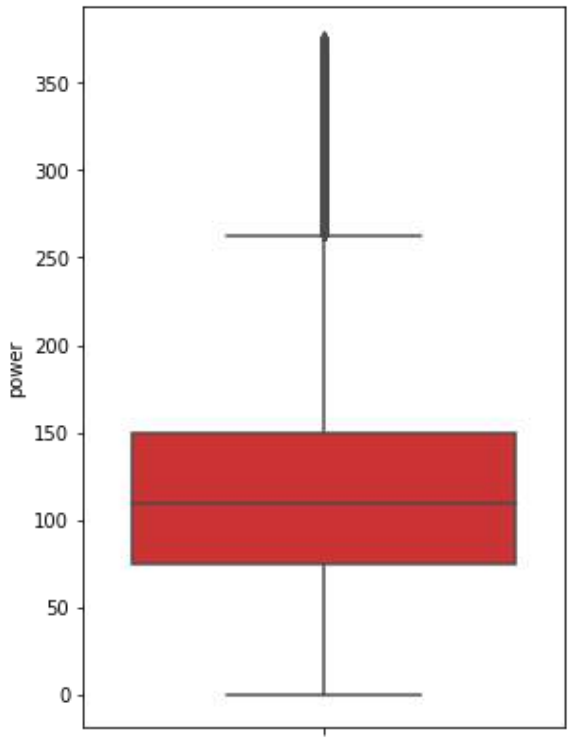
```
# 这里我包装了一个异常值处理的代码，可以随便调用。
def outliers_proc(data, col_name, scale=3):
    """
    用于清洗异常值，默认用 box_plot（scale=3）进行清洗
    :param data: 接收 pandas 数据格式
    :param col_name: pandas 列名
    :param scale: 尺度
    :return:
    """


    def box_plot_outliers(data_ser, box_scale):
        """
        利用箱线图去除异常值
        :param data_ser: 接收 pandas.Series 数据格式
        :param box_scale: 箱线图尺度，
        :return:
        """
        iqr = box_scale * (data_ser.quantile(0.75) - data_ser.quantile(0.25))
        val_low = data_ser.quantile(0.25) - iqr
        val_up = data_ser.quantile(0.75) + iqr
        rule_low = (data_ser < val_low)
        rule_up = (data_ser > val_up)
        return (rule_low, rule_up), (val_low, val_up)

    data_n = data.copy()
    data_series = data_n[col_name]
    rule, value = box_plot_outliers(data_series, box_scale=scale)
    index = np.arange(data_series.shape[0])[rule[0] | rule[1]]
    print("Delete number is: {}".format(len(index)))
    data_n = data_n.drop(index)
    data_n.reset_index(drop=True, inplace=True)
    print("Now column number is: {}".format(data_n.shape[0]))
    index_low = np.arange(data_series.shape[0])[rule[0]]
    outliers = data_series.iloc[index_low]
    print("Description of data less than the lower bound is:")
    print(pd.Series(outliers).describe())
    index_up = np.arange(data_series.shape[0])[rule[1]]
    outliers = data_series.iloc[index_up]
    print("Description of data larger than the upper bound is:")
    print(pd.Series(outliers).describe())

    fig, ax = plt.subplots(1, 2, figsize=(10, 7))
    sns.boxplot(y=data[col_name], data=data, palette="Set1", ax=ax[0])
    sns.boxplot(y=data_n[col_name], data=data_n, palette="Set1", ax=ax[1])
    return data_n
```

In [7]:

```
# 我们可以删掉一些异常数据，以 power 为例。
# 这里删不删同学可以自行判断
# 但是要注意 test 的数据不能删 = = 不能掩耳盗铃是不是

train = outliers_proc(train, 'power', scale=3)
```

```
Delete number is: 963
Now column number is: 149037
Description of data less than the lower bound is:
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: power, dtype: float64
Description of data larger than the upper bound is:
count      963.000000
mean       846.836968
std       1929.418081
min        376.000000
25%        400.000000
50%        436.000000
75%        514.000000
max      19312.000000
Name: power, dtype: float64
```





3.3.2 特征构造

In [8]:

```
# 训练集和测试集放在一起，方便构造特征
train['train']=1
test['train']=0
data = pd.concat([train, test], ignore_index=True, sort=False)
```

In [9]:

```
# 使用时间：data['creatDate'] - data['regDate']，反应汽车使用时间，一般来说价格与使用时间成反比
# 不过要注意，数据里有时间出错的格式，所以我们需要 errors='coerce'
data['used_time'] = (pd.to_datetime(data['creatDate'], format='%Y%m%d', errors='coerce') -
                            pd.to_datetime(data['regDate'], format='%Y%m%d', errors='coerce')).dt.da
```

In [10]:

```
# 看一下空数据，有 15k 个样本的时间是有问题的，我们可以选择删除，也可以选择放着。
# 但是这里不建议删除，因为删除缺失数据占总样本量过大，7.5%
# 我们可以先放着，因为如果我们 XGBoost 之类的决策树，其本身就能处理缺失值，所以可以不用管；
data['used_time'].isnull().sum()
```

Out[10]:

15072

In [11]:

```
# 从邮编中提取城市信息，因为是德国的数据，所以参考德国的邮编，相当于加入了先验知识
data['city'] = data['regionCode'].apply(lambda x : str(x)[:-3])
```

In [12]:

```
# 计算某品牌的销售统计量，同学们还可以计算其他特征的统计量
# 这里要以 train 的数据计算统计量
train_gb = train.groupby("brand")
all_info = {}
for kind, kind_data in train_gb:
    info = {}
    kind_data = kind_data[kind_data['price'] > 0]
    info['brand_amount'] = len(kind_data)
    info['brand_price_max'] = kind_data.price.max()
    info['brand_price_median'] = kind_data.price.median()
    info['brand_price_min'] = kind_data.price.min()
    info['brand_price_sum'] = kind_data.price.sum()
    info['brand_price_std'] = kind_data.price.std()
    info['brand_price_average'] = round(kind_data.price.sum() / (len(kind_data) + 1), 2)
    all_info[kind] = info
brand_fe = pd.DataFrame(all_info).T.reset_index().rename(columns={"index": "brand"})
data = data.merge(brand_fe, how='left', on='brand')
```

In [13]:

```
# 数据分桶 以 power 为例
# 这时候我们的缺失值也进桶了，
# 为什么要做数据分桶呢，原因有很多，= =
# 1. 离散后稀疏向量内积乘法运算速度更快，计算结果也方便存储，容易扩展；
# 2. 离散后的特征对异常值更具鲁棒性，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
# 3. LR 属于广义线性模型，表达能力有限，经过离散化后，每个变量有单独的权重，这相当于引入了非线性，能
# 4. 离散后特征可以进行特征交叉，提升表达能力，由 M+N 个变量编程 M*N 个变量，进一步引入非线形，提升
# 5. 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

# 当然还有很多原因，LightGBM 在改进 XGBoost 时就增加了数据分桶，增强了模型的泛化性

bin = [i*10 for i in range(31)]
data['power_bin'] = pd.cut(data['power'], bin, labels=False)
data[['power_bin', 'power']].head()
```

Out[13]:

	power_bin	power
0	5.0	60
1	NaN	0
2	16.0	163
3	19.0	193
4	6.0	68

In [14]:

```
# 利用好了，就可以删掉原始数据了
data = data.drop(['creatDate', 'regDate', 'regionCode'], axis=1)
```

In [15]:

```
print(data.shape)
data.columns
```

```
(199037, 38)
```

Out[15]:

```
Index(['name', 'model', 'brand', 'bodyType', 'fuelType', 'gearbox', 'power',
       'kilometer', 'notRepairedDamage', 'seller', 'offerType', 'price', 'v_0',
       'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6', 'v_7', 'v_8', 'v_9', 'v_10',
       'v_11', 'v_12', 'v_13', 'v_14', 'train', 'used_time', 'city',
       'brand_amount', 'brand_price_average', 'brand_price_max',
       'brand_price_median', 'brand_price_min', 'brand_price_std',
       'brand_price_sum', 'power_bin'],
      dtype='object')
```

In [16]:

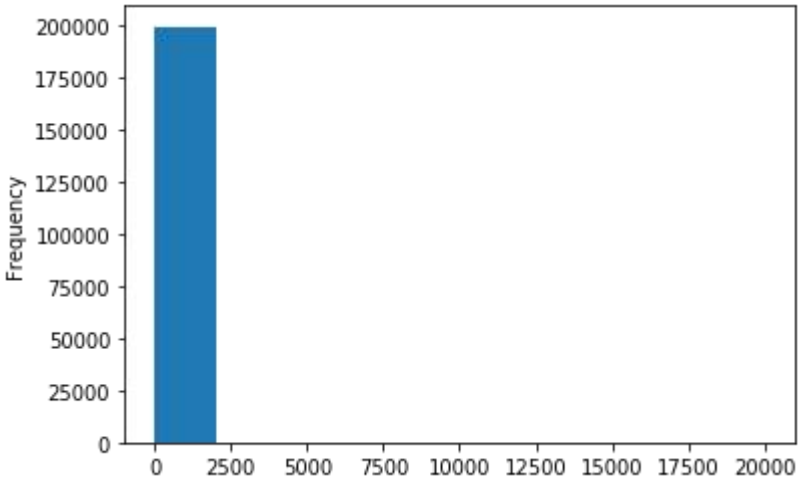
```
# 目前的数据其实已经可以给树模型使用了，所以我们导出一下
data.to_csv('data_for_tree.csv', index=0)
```

In [17]:

```
# 我们可以再构造一份特征给 LR NN 之类的模型用
# 之所以分开构造是因为，不同模型对数据集的要求不同
# 我们看下数据分布：
data['power'].plot.hist()
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x12904e5c0>

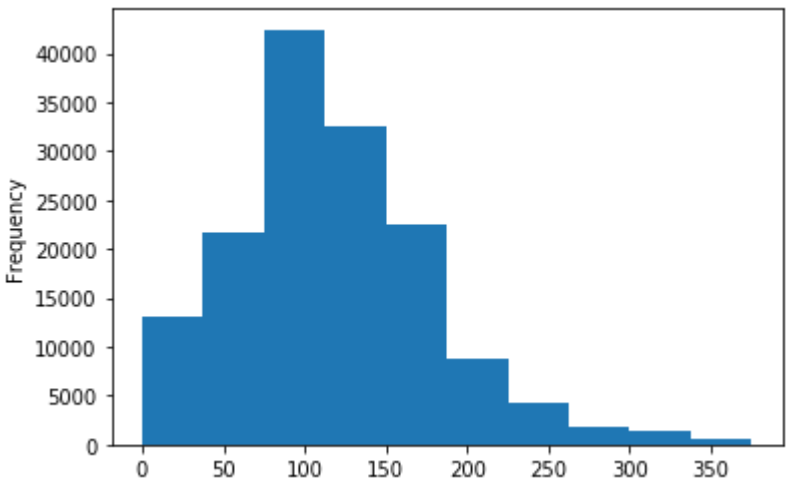


In [18]:

```
# 我们刚刚已经对 train 进行异常值处理了，但是现在还有这么奇怪的分布是因为 test 中的 power 异常值，
# 所以我们其实刚刚 train 中的 power 异常值不删为好，可以用长尾分布截断来代替
train['power'].plot.hist()
```

Out[18]:

<matplotlib.axes._subplots.AxesSubplot at 0x12de6bba8>

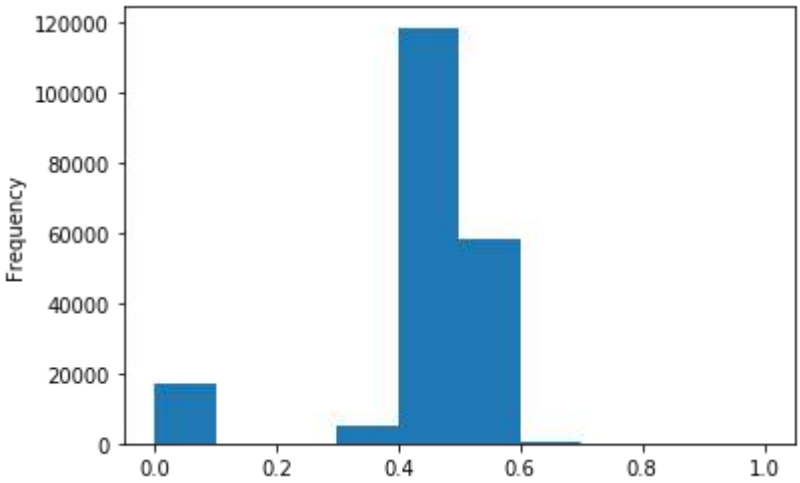


In [19]:

```
# 我们对其取 log，在做归一化
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
data['power'] = np.log(data['power'] + 1)
data['power'] = ((data['power'] - np.min(data['power'])) / (np.max(data['power']) - np.min(data['po
data['power'].plot.hist()
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x129ad5dd8>

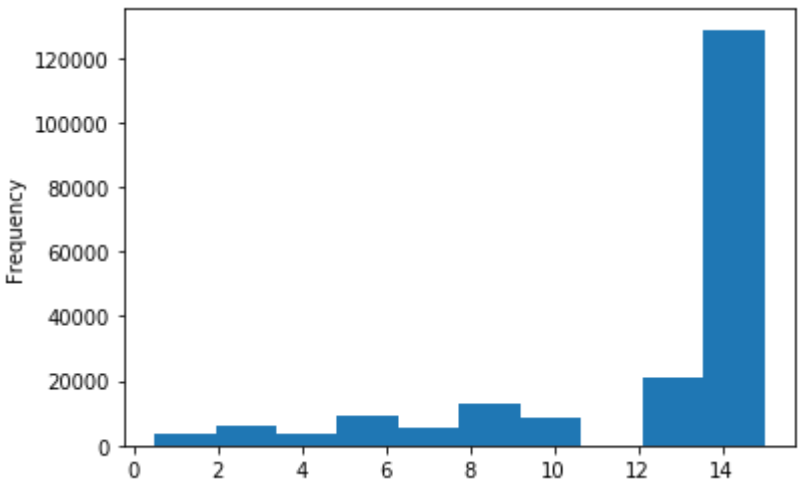


In [20]:

```
# km 的比较正常，应该是已经做过分桶了
data['kilometer'].plot.hist()
```

Out[20]:

<matplotlib.axes._subplots.AxesSubplot at 0x12de58cf8>

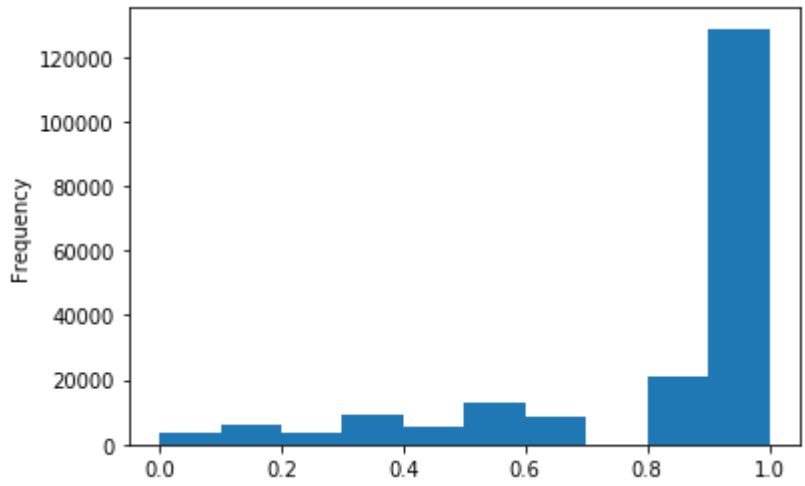


In [21]:

```
# 所以我们可以直接做归一化
data['kilometer'] = ((data['kilometer'] - np.min(data['kilometer'])) /
                     (np.max(data['kilometer']) - np.min(data['kilometer'])))
data['kilometer'].plot.hist()
```

Out[21]:

<matplotlib.axes._subplots.AxesSubplot at 0x128b4fd30>



In [23]:

```
# 除此之外 还有我们刚刚构造的统计量特征:
# 'brand_amount', 'brand_price_average', 'brand_price_max',
# 'brand_price_median', 'brand_price_min', 'brand_price_std',
# 'brand_price_sum'
# 这里不再一一举例分析了，直接做变换，
def max_min(x):
    return (x - np.min(x)) / (np.max(x) - np.min(x))

data['brand_amount'] = ((data['brand_amount'] - np.min(data['brand_amount'])) /
                        (np.max(data['brand_amount']) - np.min(data['brand_amount'])))
data['brand_price_average'] = ((data['brand_price_average'] - np.min(data['brand_price_average'])) /
                               (np.max(data['brand_price_average']) - np.min(data['brand_price_avera
data['brand_price_max'] = ((data['brand_price_max'] - np.min(data['brand_price_max'])) /
                           (np.max(data['brand_price_max']) - np.min(data['brand_price_max'])))
data['brand_price_median'] = ((data['brand_price_median'] - np.min(data['brand_price_median'])) /
                              (np.max(data['brand_price_median']) - np.min(data['brand_price_median'
data['brand_price_min'] = ((data['brand_price_min'] - np.min(data['brand_price_min'])) /
                           (np.max(data['brand_price_min']) - np.min(data['brand_price_min'])))
data['brand_price_std'] = ((data['brand_price_std'] - np.min(data['brand_price_std'])) /
                           (np.max(data['brand_price_std']) - np.min(data['brand_price_std'])))
data['brand_price_sum'] = ((data['brand_price_sum'] - np.min(data['brand_price_sum'])) /
                           (np.max(data['brand_price_sum']) - np.min(data['brand_price_sum'])))
```

In [24]:

```
# 对类别特征进行 OneEncoder
data = pd.get_dummies(data, columns=['model', 'brand', 'bodyType', 'fuelType',
                                     'gearbox', 'notRepairedDamage', 'power_bin'])
```

In [25]:

```
print(data.shape)
data.columns
```

(199037, 369)

Out[25]:

```
Index(['name', 'power', 'kilometer', 'seller', 'offerType', 'price', 'v_0',
       'v_1', 'v_2', 'v_3',
       ...
       'power_bin_20.0', 'power_bin_21.0', 'power_bin_22.0', 'power_bin_23.0',
       'power_bin_24.0', 'power_bin_25.0', 'power_bin_26.0', 'power_bin_27.0',
       'power_bin_28.0', 'power_bin_29.0'],
      dtype='object', length=369)
```

In [26]:

```
# 这份数据可以给 LR 用
data.to_csv('data_for_lr.csv', index=0)
```

3.3.3 特征筛选

1) 过滤式

In [27]:

```
# 相关性分析
print(data['power'].corr(data['price'], method='spearman'))
print(data['kilometer'].corr(data['price'], method='spearman'))
print(data['brand_amount'].corr(data['price'], method='spearman'))
print(data['brand_price_average'].corr(data['price'], method='spearman'))
print(data['brand_price_max'].corr(data['price'], method='spearman'))
print(data['brand_price_median'].corr(data['price'], method='spearman'))
```

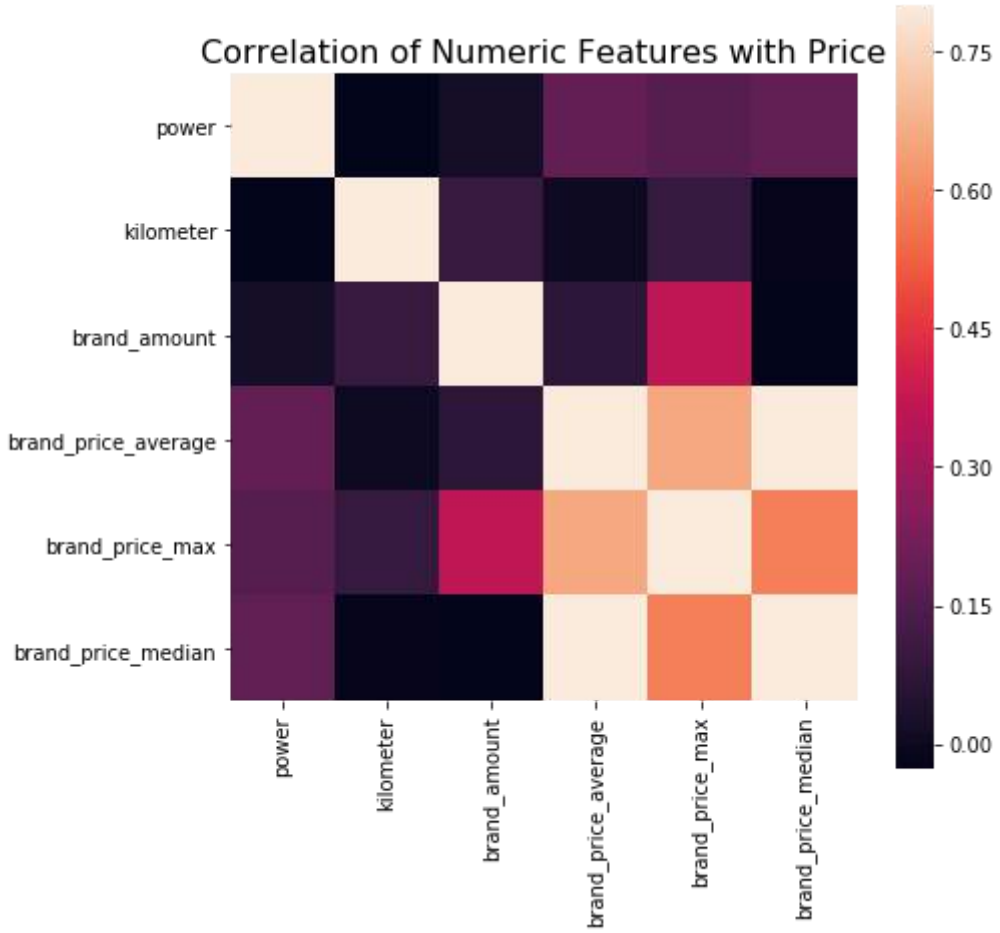
0.5737373458520139
-0.4093147076627742
0.0579639618400197
0.38587089498185884
0.26142364388130207
0.3891431767902722

In [28]:

```
# 当然也可以直接看图
data_numeric = data[['power', 'kilometer', 'brand_amount', 'brand_price_average',
                     'brand_price_max', 'brand_price_median']]
correlation = data_numeric.corr()

f , ax = plt.subplots(figsize = (7, 7))
plt.title('Correlation of Numeric Features with Price',y=1,size=16)
sns.heatmap(correlation,square = True,  vmax=0.8)
```

Out[28]:

<matplotlib.axes._subplots.AxesSubplot at 0x129059470>



2) 包裹式

In [ ]:

```
!pip install mlxtend
```

In [16]:

```
# k_feature 太大会很难跑，没服务器，所以提前 interrupt 了
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LinearRegression
sfs = SFS(LinearRegression(),
           k_features=10,
           forward=True,
           floating=False,
           scoring = 'r2',
           cv = 0)
x = data.drop(['price'], axis=1)
x = x.fillna(0)
y = data['price']
sfs.fit(x, y)
sfs.k_feature_names_
```

```
STOPPING EARLY DUE TO KEYBOARD INTERRUPT...
```

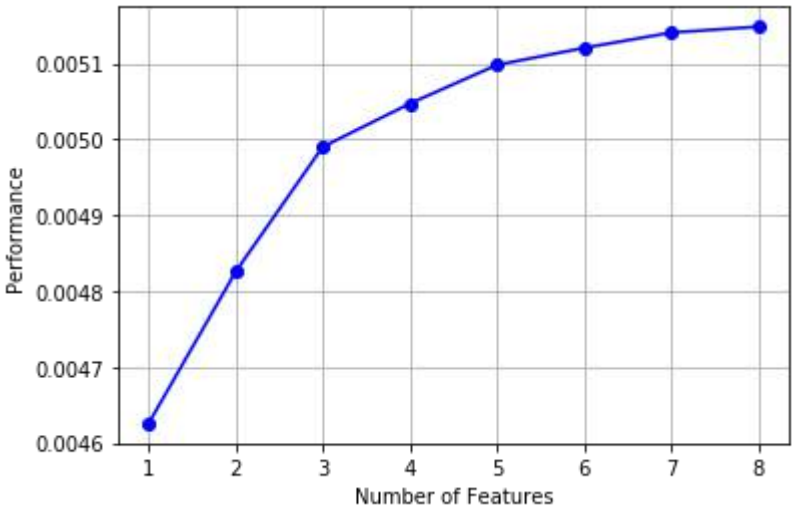
Out[16]:

```
('powerPS_ten',
 'city',
 'brand_price_std',
 'vehicleType_andere',
 'model_145',
 'model_601',
 'fuelType_andere',
 'notRepairedDamage_ja')
```

In [17]:

```
# 画出来，可以看到边际效益
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
import matplotlib.pyplot as plt
fig1 = plot_sfs(sfs.get_metric_dict(), kind='std_dev')
plt.grid()
plt.show()
```

```
/Users/chenze/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:140: Runt
imeWarning: Degrees of freedom <= 0 for slice
  keepdims=keepdims)
/Users/chenze/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:132: Runt
imeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



3) 嵌入式

In [18]:

```
# 下一章介绍，Lasso 回归和决策树可以完成嵌入式特征选择
# 大部分情况下都是用嵌入式做特征筛选
```

3.4 经验总结

特征工程是比赛中最至关重要的的一块，特别的传统的比赛，大家的模型可能都差不多，调参带来的效果增幅是非常有限的，但特征工程的好坏往往会决定了最终的排名和成绩。

特征工程的主要目的还是在于将数据转换为能更好地表示潜在问题的特征，从而提高机器学习的性能。比如，异常值处理是为了去除噪声，填补缺失值可以加入先验知识等。

特征构造也属于特征工程的一部分，其目的是为了增强数据的表达。

有些比赛的特征是匿名特征，这导致我们并不清楚特征相互直接的关联性，这时我们就只有单纯基于特征进行处理，比如装箱，groupby，agg 等这样一些操作进行一些特征统计，此外还可以对特征进行进一步的 log，exp 等变换，或者对多个特征进行四则运算（如上面我们算出的使用时长），多项式组合等然后进行筛选。由于特性的匿名性其实限制了很多对于特征的处理，当然有些时候用 NN 去提取一些特征也会达到意想不到的良好效果。

对于知道特征含义（非匿名）的特征工程，特别是在工业类型比赛中，会基于信号处理，频域提取，丰度，偏度等构建更为有实际意义的特征，这就是结合背景的特征构建，在推荐系统中也是这样的，各种类型点击率统计，各时段统计，加用户属性的统计等等，这样一种特征构建往往要深入分析背后的业务逻辑或者说物理原理，从而才能更好的找到 magic。

当然特征工程其实是和模型结合在一起的，这就是为什么要为 LR NN 做分桶和特征归一化的原因，而对于特征的处理效果和特征重要性等往往要通过模型来验证。

总的来说，特征工程是一个入门简单，但想精通非常难的一件事。

Task 3-特征工程 END.

--- By: 阿泽

PS：复旦大学计算机研究生
知乎：阿泽 https://www.zhihu.com/people/is-aze（主要面向初学者的知识整理）

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

Datawhale 零基础入门数据挖掘-Task4 建模调参

四、建模与调参

Tip:此部分为零基础入门数据挖掘的 Task4 建模调参 部分，带你来了解各种模型以及模型的评价和调参策略，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

5.1 学习目标

- 了解常用的机器学习模型，并掌握机器学习模型的建模与调参流程
- 完成相应学习打卡任务

5.2 内容介绍

1. 线性回归模型：
    - 线性回归对于特征的要求；
    - 处理长尾分布；
    - 理解线性回归模型；
2. 模型性能验证：
    - 评价函数与目标函数；
    - 交叉验证方法；
    - 留一验证方法；
    - 针对时间序列问题的验证；
    - 绘制学习率曲线；
    - 绘制验证曲线；
3. 嵌入式特征选择：
    - Lasso回归；
    - Ridge回归；
    - 决策树；
4. 模型对比：
    - 常用线性模型；
    - 常用非线性模型；
5. 模型调参：
    - 贪心调参方法；
    - 网格调参方法；
    - 贝叶斯调参方法；

5.3 相关原理介绍与推荐

由于相关算法原理篇幅较长，本文推荐了一些博客与教材供初学者们进行学习。

5.3.1 线性回归模型

https://zhuanlan.zhihu.com/p/49480391 (https://zhuanlan.zhihu.com/p/49480391)

5.3.2 决策树模型

https://zhuanlan.zhihu.com/p/65304798 (https://zhuanlan.zhihu.com/p/65304798)

5.3.3 GBDT模型

https://zhuanlan.zhihu.com/p/45145899 (https://zhuanlan.zhihu.com/p/45145899)

5.3.4 XGBoost模型

https://zhuanlan.zhihu.com/p/86816771 (https://zhuanlan.zhihu.com/p/86816771)

5.3.5 LightGBM模型

https://zhuanlan.zhihu.com/p/89360721 (https://zhuanlan.zhihu.com/p/89360721)

5.3.6 推荐教材：

- 《机器学习》 https://book.douban.com/subject/26708119/ (https://book.douban.com/subject/26708119/)
- 《统计学习方法》 https://book.douban.com/subject/10590856/ (https://book.douban.com/subject/10590856/)
- 《Python大战机器学习》 https://book.douban.com/subject/26987890/ (https://book.douban.com/subject/26987890/)
- 《面向机器学习的特征工程》 https://book.douban.com/subject/26826639/ (https://book.douban.com/subject/26826639/)
- 《数据科学家访谈录》 https://book.douban.com/subject/30129410/ (https://book.douban.com/subject/30129410/)

5.4 代码示例

5.4.1 读取数据

In [1]:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

reduce_mem_usage 函数通过调整数据类型，帮助我们减少数据在内存中占用的空间

In [2]:

```
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum()
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum()
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))
    return df
```

In [3]:

```
sample_feature = reduce_mem_usage(pd.read_csv('data_for_tree.csv'))
```

```
Memory usage of dataframe is 60507328.00 MB
Memory usage after optimization is: 15724107.00 MB
Decreased by 74.0%
```

In [4]:

```
continuous_feature_names = [x for x in sample_feature.columns if x not in ['price','brand','model'
```

5.4.2 线性回归 & 五折交叉验证 & 模拟真实业务情况

In [5]:

```
sample_feature = sample_feature.dropna().replace('-', 0).reset_index(drop=True)
sample_feature['notRepairedDamage'] = sample_feature['notRepairedDamage'].astype(np.float32)
train = sample_feature[continuous_feature_names + ['price']]

train_X = train[continuous_feature_names]
train_y = train['price']
```

5.4.2 - 1 简单建模

In [6]:

```
from sklearn.linear_model import LinearRegression
```

In [7]:

```
model = LinearRegression(normalize=True)
```

In [8]:

```
model = model.fit(train_X, train_y)
```

查看训练的线性回归模型的截距（intercept）与权重(coef)

In [9]:

```
'intercept:'+ str(model.intercept_)

sorted(dict(zip(continuous_feature_names, model.coef_)).items(), key=lambda x:x[1], reverse=True)
```

Out[9]:

```
[('v_6', 3342612.384537345),
 ('v_8', 684205.534533214),
 ('v_9', 178967.94192530424),
 ('v_7', 35223.07319016895),
 ('v_5', 21917.550249749802),
 ('v_3', 12782.03250792227),
 ('v_12', 11654.925634146672),
 ('v_13', 9884.194615297649),
 ('v_11', 5519.182176035517),
 ('v_10', 3765.6101415594258),
 ('gearbox', 900.3205339198406),
 ('fuelType', 353.5206495542567),
 ('bodyType', 186.51797317460046),
 ('city', 45.17354204168846),
 ('power', 31.163045441455335),
 ('brand_price_median', 0.535967111869784),
 ('brand_price_std', 0.4346788365040235),
 ('brand_amount', 0.15308295553300566),
```

In [10]:

```
from matplotlib import pyplot as plt
```

In [11]:

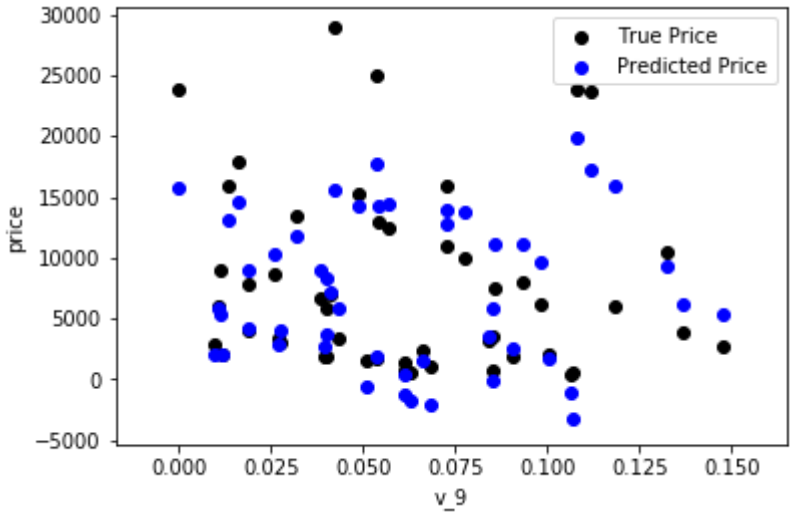
```
subsample_index = np.random.randint(low=0, high=len(train_y), size=50)
```

绘制特征v_9的值与标签的散点图，图片发现模型的预测结果（蓝色点）与真实标签（黑色点）的分布差异较大，且部分预测值出现了小于0的情况，说明我们的模型存在一些问题

In [12]:

```
plt.scatter(train_X['v_9'][subsample_index], train_y[subsample_index], color='black')
plt.scatter(train_X['v_9'][subsample_index], model.predict(train_X.loc[subsample_index]), color='blu
plt.xlabel('v_9')
plt.ylabel('price')
plt.legend(['True Price','Predicted Price'],loc='upper right')
print('The predicted price is obvious different from true price')
plt.show()
```

```
The predicted price is obvious different from true price
```



通过作图我们发现数据的标签（price）呈现长尾分布，不利于我们的建模预测。原因是很多模型都假设数据误差项符合正态分布，而长尾分布的数据违背了这一假设。参考博客：
https://blog.csdn.net/Noob_daniel/article/details/76087829
(https://blog.csdn.net/Noob_daniel/article/details/76087829)

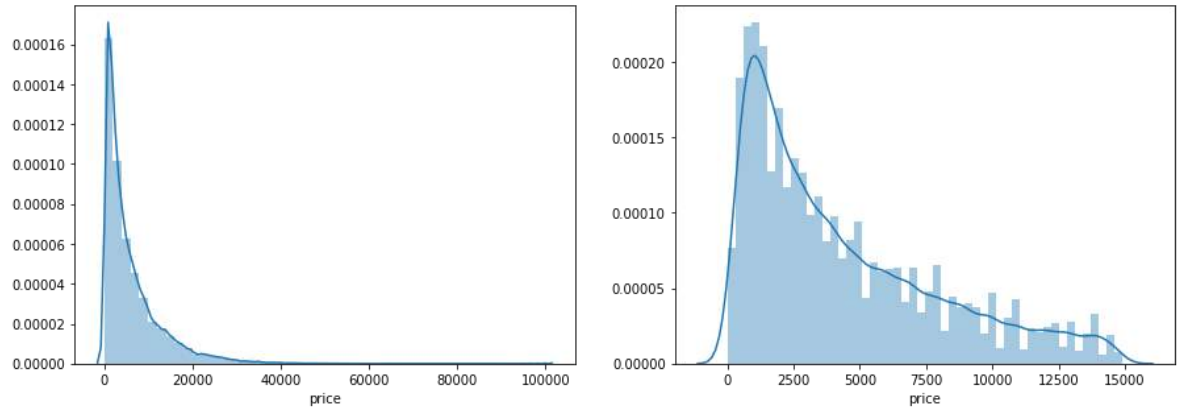
In [13]:

```
import seaborn as sns
print('It is clear to see the price shows a typical exponential distribution')
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.distplot(train_y)
plt.subplot(1,2,2)
sns.distplot(train_y[train_y < np.quantile(train_y, 0.9)])
```

```
It is clear to see the price shows a typical exponential distribution
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b33efb2f98>
```



在这里我们对标签进行了 $log(x+1)$ 变换，使标签贴近于正态分布

In [14]:

```
train_y_ln = np.log(train_y + 1)
```

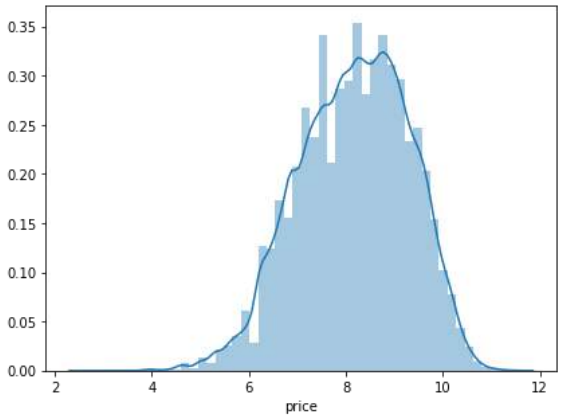
In [15]:

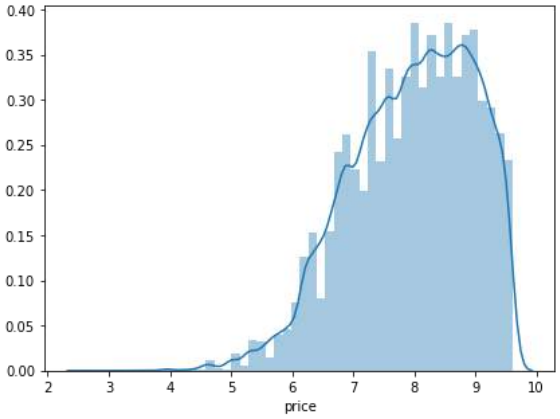
```
import seaborn as sns
print('The transformed price seems like normal distribution')
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.distplot(train_y_ln)
plt.subplot(1,2,2)
sns.distplot(train_y_ln[train_y_ln < np.quantile(train_y_ln, 0.9)])
```

```
The transformed price seems like normal distribution
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b33f077160>
```





In [16]:

```
model = model.fit(train_X, train_y_ln)

print('intercept:'+ str(model.intercept_))
sorted(dict(zip(continuous_feature_names, model.coef_)).items(), key=lambda x:x[1], reverse=True)
```

```
intercept:23.515920686637713
```

Out[16]:

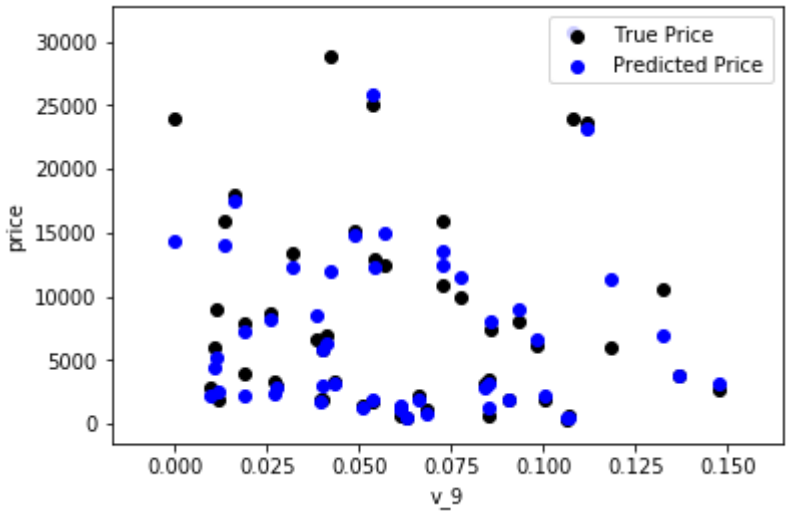
```
[('v_9', 6.043993029165403),
 ('v_12', 2.0357439855551394),
 ('v_11', 1.3607608712255672),
 ('v_1', 1.3079816298861897),
 ('v_13', 1.0788833838535354),
 ('v_3', 0.9895814429387444),
 ('gearbox', 0.009170812023421397),
 ('fuelType', 0.006447089787635784),
 ('bodyType', 0.004815242907679581),
 ('power_bin', 0.003151801949447194),
 ('power', 0.0012550361843629999),
 ('train', 0.0001429273782925814),
 ('brand_price_min', 2.0721302299502698e-05),
 ('brand_price_average', 5.308179717783439e-06),
 ('brand_amount', 2.8308531339942507e-06),
 ('brand_price_max', 6.764442596115763e-07),
 ('offerType', 1.6765966392995324e-10),
 ('seller', 9.308109838457312e-12),
 ('brand_price_sum', -1.3473184925468486e-10),
 ('name', -7.11403461065247e-08),
 ('brand_price_median', -1.7608143661053008e-06),
 ('brand_price_std', -2.7899058266986454e-06),
 ('used_time', -5.6142735899344175e-06),
 ('city', -0.0024992974087053223),
 ('v_14', -0.012754139659375262),
 ('kilometer', -0.013999175312751872),
 ('v_0', -0.04553774829634237),
 ('notRepairedDamage', -0.273686961116076),
 ('v_7', -0.7455902679730504),
 ('v_4', -0.9281349233755761),
 ('v_2', -1.2781892166433606),
 ('v_5', -1.5458846136756323),
 ('v_10', -1.8059217242413748),
 ('v_8', -42.611729973490604),
 ('v_6', -241.30992120503035)]
```

再次进行可视化，发现预测结果与真实值较为接近，且未出现异常状况

In [17]:

```
plt.scatter(train_X['v_9'][subsample_index], train_y[subsample_index], color='black')
plt.scatter(train_X['v_9'][subsample_index], np.exp(model.predict(train_X.loc[subsample_index])), co
plt.xlabel('v_9')
plt.ylabel('price')
plt.legend(['True Price','Predicted Price'],loc='upper right')
print('The predicted price seems normal after np.log transforming')
plt.show()
```

```
The predicted price seems normal after np.log transforming
```



5.4.2 - 2 五折交叉验证

在使用训练集对参数进行训练的时候，经常会发现人们通常会将一整个训练集分为三个部分（比如mnist手写训练集）。一般分为：训练集（train_set），评估集（valid_set），测试集（test_set）这三个部分。这其实是为了保证训练效果而特意设置的。其中测试集很好理解，其实就是完全不参与训练的数据，仅仅用来观测测试效果的数据。而训练集和评估集则牵涉到下面的知识了。

因为在实际的训练中，训练的结果对于训练集的拟合程度通常还是挺好的（初始条件敏感），但是对于训练集之外的数据的拟合程度通常就不那么令人满意了。因此我们通常并不会把所有的数据集都拿来训练，而是分出一部分来（这一部分不参加训练）对训练集生成的参数进行测试，相对客观的判断这些参数对训练集之外的数据的符合程度。这种思想就称为交叉验证（Cross Validation）

In [18]:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error,  make_scorer
```

In [19]:

```
def log_transfer(func):
    def wrapper(y, yhat):
        result = func(np.log(y), np.nan_to_num(np.log(yhat)))
        return result
    return wrapper
```

In [20]:

```
scores = cross_val_score(model, X=train_X, y=train_y, verbose=1, cv = 5, scoring=make_scorer(log_tr
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.1s finished
```

使用线性回归模型，对未处理标签的特征数据进行五折交叉验证（Error 1.36）

In [21]:

```
print('AVG:', np.mean(scores))
```

```
AVG: 1.3641908155886227
```

使用线性回归模型，对处理过标签的特征数据进行五折交叉验证（Error 0.19）

In [22]:

```
scores = cross_val_score(model, X=train_X, y=train_y_ln, verbose=1, cv = 5, scoring=make_scorer(mea
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.1s finished
```

In [23]:

```
print('AVG:', np.mean(scores))
```

```
AVG: 0.19382863663604424
```

In [24]:

```
scores = pd.DataFrame(scores.reshape(1,-1))
scores.columns = ['cv' + str(x) for x in range(1, 6)]
scores.index = ['MAE']
scores
```

Out[24]:

	cv1	cv2	cv3	cv4	cv5
MAE	0.191642	0.194986	0.192737	0.195329	0.19445

5.4.2.3 模拟真实业务情况

5.4.2 - 3 模拟真实业务情况

但在事实上，由于我们并不具有预知未来的能力，五折交叉验证在某些与时间相关的数据集上反而反映了不真实的情况。通过2018年的二手车价格预测2017年的二手车价格，这显然是不合理的，因此我们还可以采用时间顺序对数据集进行分隔。在本例中，我们选用靠前时间的4/5样本当作训练集，靠后时间的1/5当作验证集，最终结果与五折交叉验证差距不大

In [25]:

```
import datetime
```

In [26]:

```
sample_feature = sample_feature.reset_index(drop=True)
```

In [27]:

```
split_point = len(sample_feature) // 5 * 4
```

In [28]:

```
train = sample_feature.loc[:split_point].dropna()
val = sample_feature.loc[split_point:].dropna()

train_X = train[continuous_feature_names]
train_y_ln = np.log(train['price'] + 1)
val_X = val[continuous_feature_names]
val_y_ln = np.log(val['price'] + 1)
```

In [29]:

```
model = model.fit(train_X, train_y_ln)
```

In [30]:

```
mean_absolute_error(val_y_ln, model.predict(val_X))
```

Out[30]:

0.19443858353490887

5.4.2 - 4 绘制学习率曲线与验证曲线

In [32]:

```
from sklearn.model_selection import learning_curve, validation_curve
```

In [37]:

```
? learning_curve
```

In [38]:

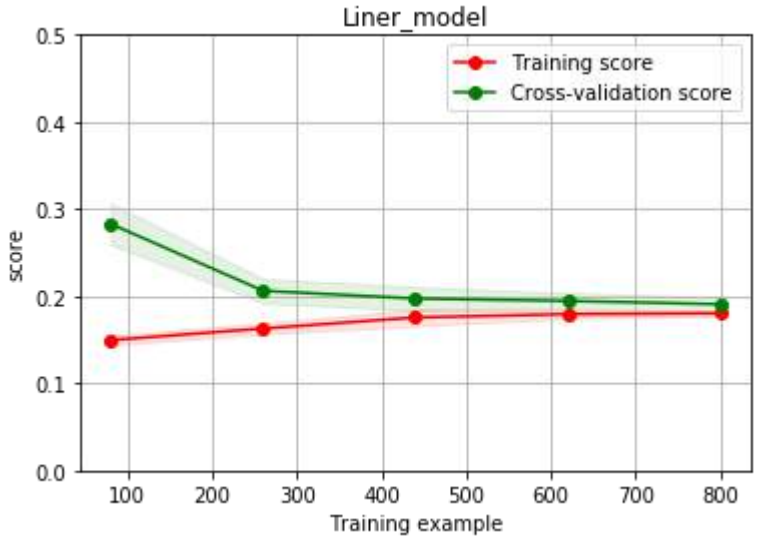
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,n_jobs=1, train_size=np.linspac
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel('Training example')
    plt.ylabel('score')
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()#区域
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color='r',
             label="Training score")
    plt.plot(train_sizes, test_scores_mean,'o-',color="g",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt
```

In [53]:

```
plot_learning_curve(LinearRegression(), 'Liner_model', train_X[:1000], train_y_ln[:1000], ylim=(0.0,
```

Out[53]:

```
<module 'matplotlib.pyplot' from 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\ma
tplotlib\\pyplot.py'>
```



5.4.3 多种模型对比

In [40]:

```
train = sample_feature[continuous_feature_names + ['price']].dropna()

train_X = train[continuous_feature_names]
train_y = train['price']
train_y_ln = np.log(train_y + 1)
```

5.4.3 - 1 线性模型 & 嵌入式特征选择

本章节默认，学习者已经了解关于过拟合、模型复杂度、正则化等概念。否则请寻找相关资料或参考如下连接：

- 用简单易懂的语言描述「过拟合 overfitting」？ https://www.zhihu.com/question/32246256/answer/55320482 (https://www.zhihu.com/question/32246256/answer/55320482)
- 模型复杂度与模型的泛化能力 http://yangyingming.com/article/434/ (http://yangyingming.com/article/434/)
- 正则化的直观理解 https://blog.csdn.net/jinping_shi/article/details/52433975 (https://blog.csdn.net/jinping_shi/article/details/52433975)

在过滤式和包裹式特征选择方法中，特征选择过程与学习器训练过程有明显的分别。而嵌入式特征选择在学习器训练过程中自动地进行特征选择。嵌入式选择最常用的是L1正则化与L2正则化。在对线性回归模型加入两种正则化方法后，他们分别变成了岭回归与Lasso回归。

In [41]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

In [42]:

```
models = [LinearRegression(),
          Ridge(),
          Lasso()]
```

In [43]:

```
result = dict()
for model in models:
    model_name = str(model).split('(')[0]
    scores = cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make_scorer
    result[model_name] = scores
    print(model_name + ' is finished')
```

```
LinearRegression is finished
Ridge is finished
Lasso is finished
```

对三种方法的效果对比

In [44]:

```
result = pd.DataFrame(result)
result.index = ['cv' + str(x) for x in range(1, 6)]
result
```

Out[44]:

	LinearRegression	Ridge	Lasso
cv1	0.191642	0.195665	0.382708
cv2	0.194986	0.198841	0.383916
cv3	0.192737	0.196629	0.380754
cv4	0.195329	0.199255	0.385683
cv5	0.194450	0.198173	0.383555

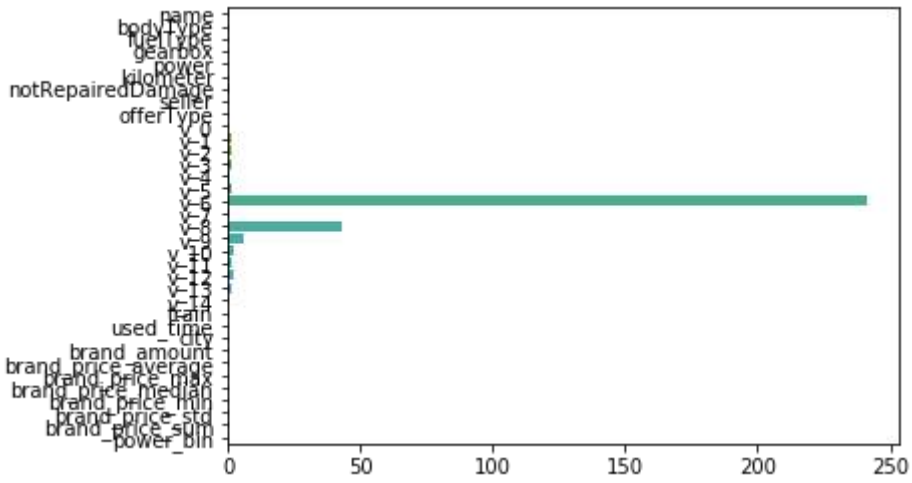
In [45]:

```
model = LinearRegression().fit(train_X, train_y_ln)
print('intercept:'+ str(model.intercept_))
sns.barplot(abs(model.coef_), continuous_feature_names)
```

```
intercept:23.515984499017883
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1feb933ca58>
```



L2正则化在拟合过程中通常都倾向于让权值尽可能小，最后构造一个所有参数都比较小的模型。因为一般认为参数值小的模型比较简单，能适应不同的数据集，也在一定程度上避免了过拟合现象。可以设想一下对于一个线性回归方程，若参数很大，那么只要数据偏移一点点，就会对结果造成很大的影响；但如果参数足够小，数据偏移得多一点也不会对结果造成什么影响，专业一点的说法是『抗扰动能力强』

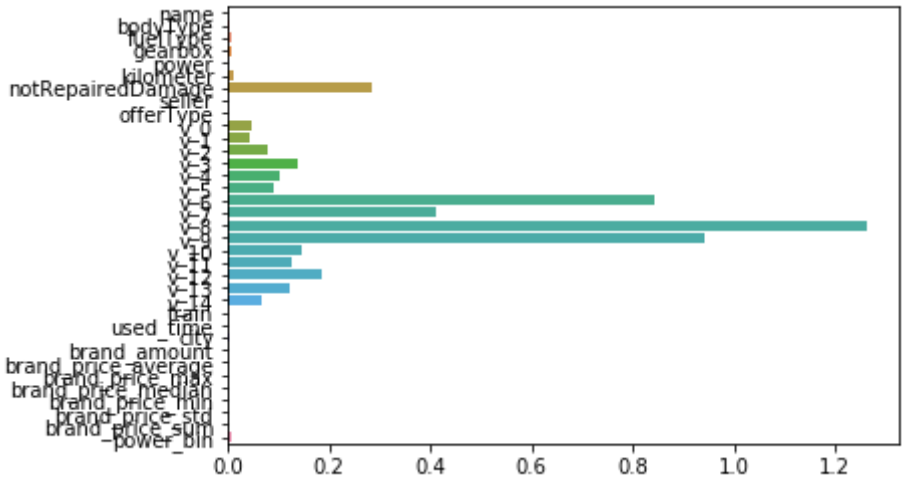
In [46]:

```
model = Ridge().fit(train_X, train_y_ln)
print('intercept:'+ str(model.intercept_))
sns.barplot(abs(model.coef_), continuous_feature_names)
```

```
intercept:5.901527844424091
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fea9056860>
```



L1正则化有助于生成一个稀疏权值矩阵，进而可以用于特征选择。如下图，我们发现power与userd_time特征非常重要。

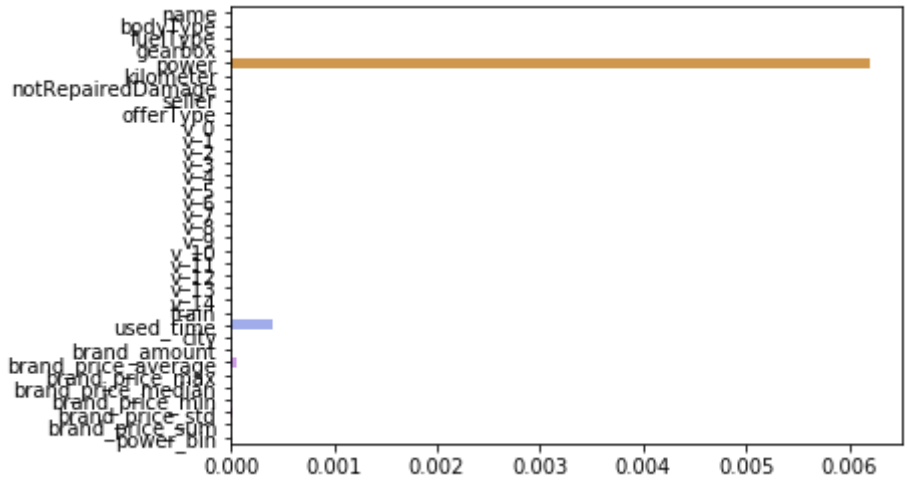
In [47]:

```
model = Lasso().fit(train_X, train_y_ln)
print('intercept:'+ str(model.intercept_))
sns.barplot(abs(model.coef_), continuous_feature_names)
```

```
intercept:8.674427764003347
```

Out[47]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fea90b69b0>
```



除此之外，决策树通过信息熵或GINI指数选择分裂节点时，优先选择的分裂特征也更加重要，这同样是一种特征选择的方法。XGBoost与LightGBM模型中的model_importance指标正是基于此计算的

5.4.3 - 2 非线性模型

除了线性模型以外，还有许多我们常用的非线性模型如下，在此篇幅有限不再一一讲解原理。我们选择了部分常用模型与线性模型进行效果比对。

In [48]:

```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from xgboost.sklearn import XGBRegressor
from lightgbm.sklearn import LGBMRegressor
```

In [49]:

```
models = [LinearRegression(),
          DecisionTreeRegressor(),
          RandomForestRegressor(),
          GradientBoostingRegressor(),
          MLPRegressor(solver='lbfgs', max_iter=100),
          XGBRegressor(n_estimators = 100, objective='reg:squarederror'),
          LGBMRegressor(n_estimators = 100)]
```

In [50]:

```
result = dict()
for model in models:
    model_name = str(model).split('(')[0]
    scores = cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make_scorer
    result[model_name] = scores
    print(model_name + ' is finished')
```

```
LinearRegression is finished
DecisionTreeRegressor is finished
RandomForestRegressor is finished
GradientBoostingRegressor is finished
MLPRegressor is finished
XGBRegressor is finished
LGBMRegressor is finished
```

In [51]:

```
result = pd.DataFrame(result)
result.index = ['cv' + str(x) for x in range(1, 6)]
result
```

Out[51]:

	LinearRegression	DecisionTreeRegressor	RandomForestRegressor	GradientBoostingRegres
cv1	0.191642	0.184566	0.136266	0.168
cv2	0.194986	0.187029	0.139693	0.171
cv3	0.192737	0.184839	0.136871	0.169
cv4	0.195329	0.182605	0.138689	0.172
cv5	0.194450	0.186626	0.137420	0.171

可以看到随机森林模型在每一个fold中均取得了更好的效果

5.4.4 模型调参

在此我们介绍了三种常用的调参方法如下：

- 贪心算法 https://www.jianshu.com/p/ab89df9759c8 (https://www.jianshu.com/p/ab89df9759c8)
- 网格调参 https://blog.csdn.net/weixin_43172660/article/details/83032029 (https://blog.csdn.net/weixin_43172660/article/details/83032029)
- 贝叶斯调参 https://blog.csdn.net/linxid/article/details/81189154 (https://blog.csdn.net/linxid/article/details/81189154)

In [52]:

```
## LGB的参数集合：

objective = ['regression', 'regression_l1', 'mape', 'huber', 'fair']

num_leaves = [3,5,10,15,20,40, 55]
max_depth = [3,5,10,15,20,40, 55]
bagging_fraction = []
feature_fraction = []
drop_rate = []
```

5.4.4 - 1 贪心调参

In [53]:

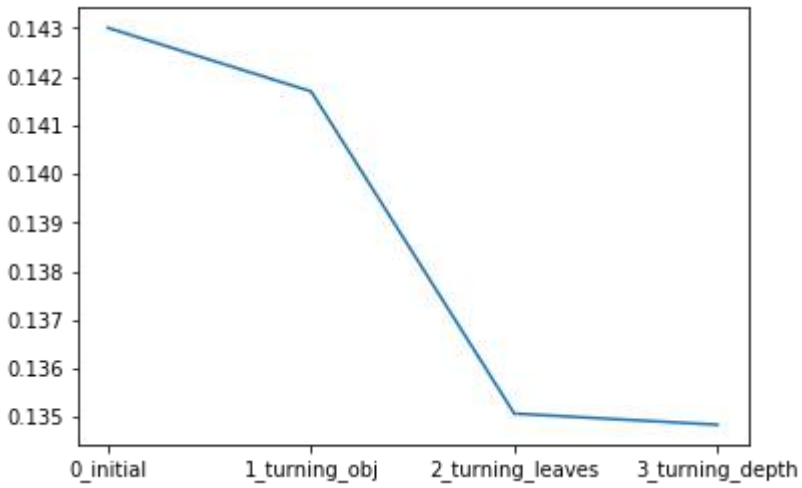
```
best_obj = dict()
for obj in objective:
    model = LGBMRegressor(objective=obj)
    score = np.mean(cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make
    best_obj[obj] = score

best_leaves = dict()
for leaves in num_leaves:
    model = LGBMRegressor(objective=min(best_obj.items(), key=lambda x:x[1])[0], num_leaves=leaves)
    score = np.mean(cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make
    best_leaves[leaves] = score

best_depth = dict()
for depth in max_depth:
    model = LGBMRegressor(objective=min(best_obj.items(), key=lambda x:x[1])[0],
                          num_leaves=min(best_leaves.items(), key=lambda x:x[1])[0],
                          max_depth=depth)
    score = np.mean(cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make
    best_depth[depth] = score
```

In [54]:

```
sns.lineplot(x=['0_initial','1_turning_obj','2_turning_leaves','3_turning_depth'], y=[0.143 ,min(bes
```

Out[54]:

<matplotlib.axes._subplots.AxesSubplot at 0x1fea93f6080>



5.4.4 - 2 Grid Search 调参

In [55]:

```
from sklearn.model_selection import GridSearchCV
```

In [56]:

```
parameters = {'objective': objective , 'num_leaves': num_leaves, 'max_depth': max_depth}
model = LGBMRegressor()
clf = GridSearchCV(model, parameters, cv=5)
clf = clf.fit(train_X, train_y)
```

In [57]:

```
clf.best_params_
```

Out[57]:

{'max_depth': 15, 'num_leaves': 55, 'objective': 'regression'}

In [58]:

```
model = LGBMRegressor(objective='regression',
                          num_leaves=55,
                          max_depth=15)
```

In [59]:

```
np.mean(cross_val_score(model, X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make_scorer(mean
```

Out[59]:

0.13626164479243302

5.4.4 - 3 贝叶斯调参

In [60]:

```
from bayes_opt import BayesianOptimization
```

In [61]:

```
def rf_cv(num_leaves, max_depth, subsample, min_child_samples):
    val = cross_val_score(
        LGBMRegressor(objective = 'regression_l1',
            num_leaves=int(num_leaves),
            max_depth=int(max_depth),
            subsample = subsample,
            min_child_samples = int(min_child_samples)
        ),
        X=train_X, y=train_y_ln, verbose=0, cv = 5, scoring=make_scorer(mean_absolute_error)
    ).mean()
    return 1 - val
```

In [62]:

```
rf_bo = BayesianOptimization(
    rf_cv,
    {
    'num_leaves': (2, 100),
    'max_depth': (2, 100),
    'subsample': (0.1, 1),
    'min_child_samples' : (2, 100)
    }
)
```

In [63]:

```
rf_bo.maximize()
```

```
|   iter    |  target   | max_depth | min_ch... | num_le... | subsample |
-------------------------------------------------------------------------
|  1        |  0.8649   |  89.57    |  47.3     |  55.13    |  0.1792   |
|  2        |  0.8477   |  99.86    |  60.91    |  15.35    |  0.4716   |
|  3        |  0.8698   |  81.74    |  83.32    |  92.59    |  0.9559   |
|  4        |  0.8627   |  90.2     |  8.754    |  43.34    |  0.7772   |
|  5        |  0.8115   |  10.07    |  86.15    |  4.109    |  0.3416   |
|  6        |  0.8701   |  99.15    |  9.158    |  99.47    |  0.494    |
|  7        |  0.806    |  2.166    |  2.416    |  97.7     |  0.224    |
|  8        |  0.8701   |  98.57    |  97.67    |  99.87    |  0.3703   |
|  9        |  0.8703   |  99.87    |  43.03    |  99.72    |  0.9749   |
|  10       |  0.869    |  10.31    |  99.63    |  99.34    |  0.2517   |
|  11       |  0.8703   |  52.27    |  99.56    |  98.97    |  0.9641   |
|  12       |  0.8669   |  99.89    |  8.846    |  66.49    |  0.1437   |
|  13       |  0.8702   |  68.13    |  75.28    |  98.71    |  0.153    |
|  14       |  0.8695   |  84.13    |  86.48    |  91.9     |  0.7949   |
|  15       |  0.8702   |  98.09    |  59.2     |  99.65    |  0.3275   |
|  16       |  0.87     |  68.97    |  98.62    |  98.93    |  0.2221   |
|  17       |  0.8702   |  99.85    |  63.74    |  99.63    |  0.4137   |
|  18       |  0.8703   |  45.87    |  99.05    |  99.89    |  0.3238   |
|  19       |  0.8702   |  79.65    |  46.91    |  98.61    |  0.8999   |
|  20       |  0.8702   |  99.25    |  36.73    |  99.05    |  0.1262   |
|  21       |  0.8702   |  85.51    |  85.34    |  99.77    |  0.8917   |
|  22       |  0.8696   |  99.99    |  38.51    |  89.13    |  0.9884   |
|  23       |  0.8701   |  63.29    |  97.93    |  99.94    |  0.9585   |
|  24       |  0.8702   |  93.04    |  71.42    |  99.94    |  0.9646   |
|  25       |  0.8701   |  99.73    |  16.21    |  99.38    |  0.9778   |
|  26       |  0.87     |  86.28    |  58.1     |  99.47    |  0.107    |
|  27       |  0.8703   |  47.28    |  99.83    |  99.65    |  0.4674   |
|  28       |  0.8703   |  68.29    |  99.51    |  99.4     |  0.2757   |
|  29       |  0.8701   |  76.49    |  73.41    |  99.86    |  0.9394   |
|  30       |  0.8695   |  37.27    |  99.87    |  89.87    |  0.7588   |
=========================================================================
```

In [64]:

```
1 - rf_bo.max['target']
```

Out[64]:

0.1296693644053145

总结

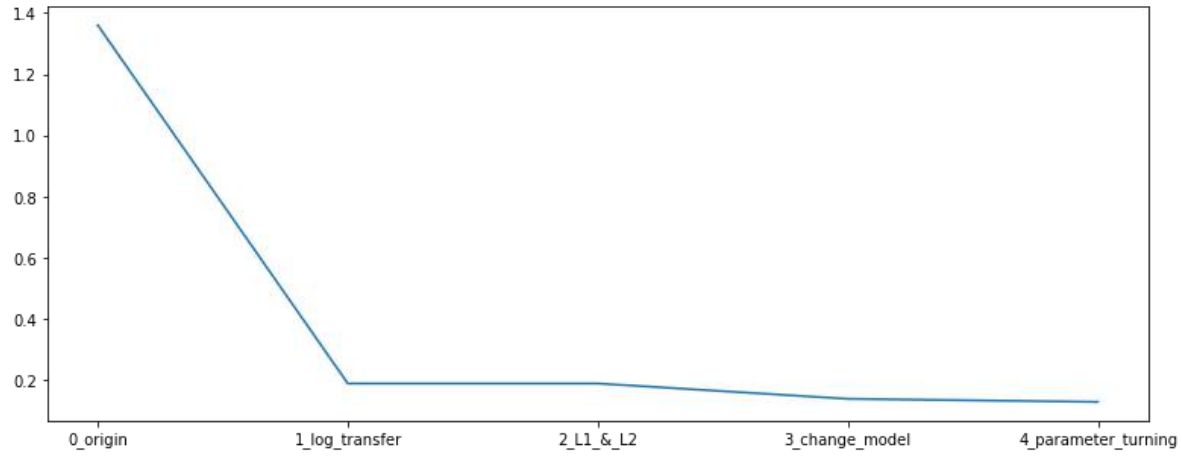
在本章中，我们完成了建模与调参的工作，并对我们的模型进行了验证。此外，我们还采用了一些基本方法来提高预测的精度，提升如下图所示。

In [70]:

```
plt.figure(figsize=(13,5))
sns.lineplot(x=['0_origin','1_log_transfer','2_L1_&_L2','3_change_model','4_parameter_turning'], y=
```

Out[70]:

<matplotlib.axes._subplots.AxesSubplot at 0x1feac73ceb8>



Task5 建模调参 END.

--- By: 小雨姑娘

数据挖掘爱好者，多次获比赛TOP名次。
作者的机器学习笔记：https://zhuanlan.zhihu.com/mlbasic

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：

Datawhale 零基础入门数据挖掘-Task5 模型融合

五、模型融合

Tip:此部分为零基础入门数据挖掘的 Task5 模型融合 部分，带你来了解各种模型结果的融合方式，在比赛的攻坚时刻冲刺Top，欢迎大家后续多多交流。

赛题：零基础入门数据挖掘 - 二手车交易价格预测

地址：https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX (https://tianchi.aliyun.com/competition/entrance/231784/introduction?spm=5176.12281957.1004.1.38b02448ausjSX)

5.1 模型融合目标

- 对于多种调参完成的模型进行模型融合。
- 完成对于多种模型的融合，提交融合结果并打卡。

5.2 内容介绍

模型融合是比赛后期一个重要的环节，大体来说有如下的类型方式。

1. 简单加权融合:
   - 回归（分类概率）：算术平均融合（Arithmetic mean），几何平均融合（Geometric mean）；
   - 分类：投票（Voting)
   - 综合：排序融合(Rank averaging)，log融合

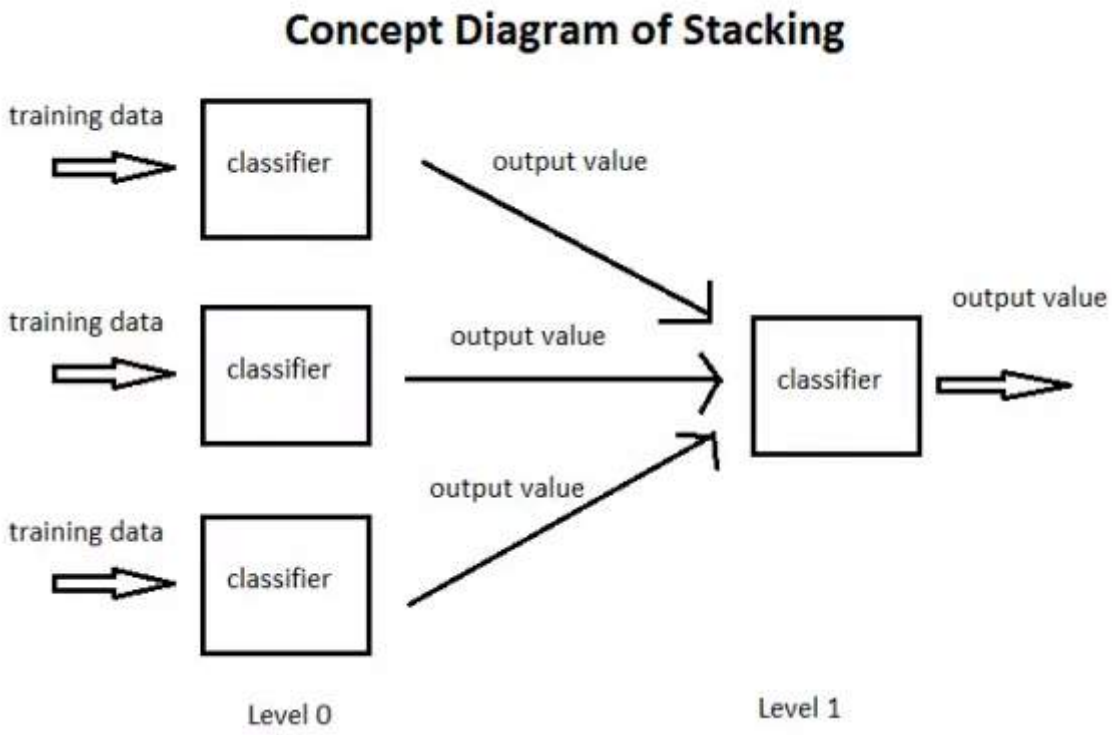
2. stacking/blending:
   - 构建多层模型，并利用预测结果再拟合预测。

4. boosting/bagging（在xgboost，Adaboost,GBDT中已经用到）：
   - 多树的提升方法

5.3 Stacking相关理论介绍

1) 什么是 stacking

简单来说 stacking 就是当用初始训练数据学习出若干个基学习器后，将这几个学习器的预测结果作为新的训练集，来学习一个新的学习器。



将个体学习器结合在一起的时候使用的方法叫做结合策略。对于分类问题，我们可以使用投票法来选择输出最多的类。对于回归问题，我们可以将分类器输出的结果求平均值。

上面说的投票法和平均法都是很有效的结合策略，还有一种结合策略是使用另外一个机器学习算法来将个体机器学习器的结果结合在一起，这个方法就是Stacking。

在stacking方法中，我们把个体学习器叫做初级学习器，用于结合的学习器叫做次级学习器或元学习器（meta-learner），次级学习器用于训练的数据叫做次级训练集。次级训练集是在训练集上用初级学习器得到的。

2) 如何进行 stacking

算法示意图如下：

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
初级学习算法 $\mathfrak{L}_1, \mathfrak{L}_2, \ldots, \mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.

过程：

```
for t = 1, 2, ..., T do
    h_t = L_t(D);
end for
D' = ∅;
for i = 1, 2, ..., m do
    for t = 1, 2, ..., T do
        z_it = h_t(x_i);
    end for
    D' = D' ∪ ((z_i1, z_i2, ..., z_iT), y_i);
end for
h' = L(D');
```

输出： $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$

图 8.9 Stacking 算法

> 引用自 西瓜书《机器学习》

- 过程1-3 是训练出来个体学习器，也就是初级学习器。
- 过程5-9是 使用训练出来的个体学习器来得预测的结果，这个预测的结果当做次级学习器的训练集。
- 过程11 是用初级学习器预测的结果训练出次级学习器，得到我们最后训练的模型。

3）Stacking的方法讲解

首先，我们先从一种“不那么正确”但是容易懂的Stacking方法讲起。

Stacking模型本质上是一种分层的结构，这里简单起见，只分析二级Stacking.假设我们有2个基模型 Model1_1、Model1_2 和 一个次级模型Model2

Step 1. 基模型 Model1_1，对训练集train训练，然后用于预测 train 和 test 的标签列，分别是P1，T1

Model1_1 模型训练:

$$\begin{pmatrix} \vdots \\ X_{train} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_1 Train}} \begin{pmatrix} \vdots \\ Y_{True} \\ \vdots \end{pmatrix}$$

训练后的模型 Model1_1 分别在 train 和 test 上预测，得到预测标签分别是P1，T1

$$\begin{pmatrix} \vdots \\ X_{train} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_1 Predict}} \begin{pmatrix} \vdots \\ P_1 \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \vdots \\ X_{test} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_1 Predict}} \begin{pmatrix} \vdots \\ T_1 \\ \vdots \end{pmatrix}$$

Step 2. 基模型 Model1_2 ，对训练集train训练，然后用于预测train和test的标签列，分别是P2，T2

Model1_2 模型训练:

$$\begin{pmatrix} \vdots \\ X_{train} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_2 Train}} \begin{pmatrix} \vdots \\ Y_{True} \\ \vdots \end{pmatrix}$$

训练后的模型 Model1_2 分别在 train 和 test 上预测，得到预测标签分别是P2，T2

$$\begin{pmatrix} \vdots \\ X_{train} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_2 Predict}} \begin{pmatrix} \vdots \\ P_2 \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \vdots \\ X_{test} \\ \vdots \end{pmatrix} \overbrace{\Longrightarrow}^{\text{Model1\_2 Predict}} \begin{pmatrix} \vdots \\ T_2 \\ \vdots \end{pmatrix}$$

Step 3. 分别把P1,P2以及T1,T2合并，得到一个新的训练集和测试集train2,test2.

$$\overbrace{\begin{pmatrix} \vdots & \vdots \\ P_1 & P_2 \\ \vdots & \vdots \end{pmatrix}}^{\text{Train\_2}} and \overbrace{\begin{pmatrix} \vdots & \vdots \\ T_1 & T_2 \\ \vdots & \vdots \end{pmatrix}}^{\text{Test\_2}}$$

再用 次级模型 Model2 以真实训练集标签为标签训练,以train2为特征进行训练，预测test2,得到最终的测试集预测的标签列 Y_{Pre}。

$$\overbrace{\begin{pmatrix} \vdots & \vdots \\ P_1 & P_2 \\ \vdots & \vdots \end{pmatrix}}^{\text{Train\_2}} \overset{\text{Model2 Train}}{\Longrightarrow} \begin{pmatrix} \vdots \\ Y_{True} \\ \vdots \end{pmatrix}$$

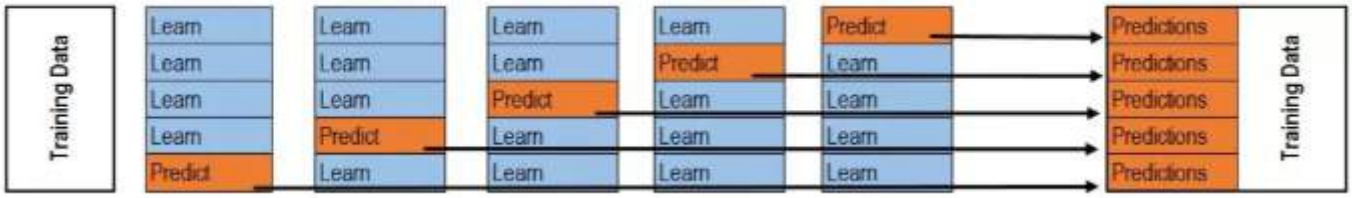
$$\overbrace{\begin{pmatrix} \vdots & \vdots \\ T_1 & T_2 \\ \vdots & \vdots \end{pmatrix}}^{\text{Test\_2}} \overset{\text{Model1\_2 Predict}}{\Longrightarrow} \begin{pmatrix} \vdots \\ Y_{Pre} \\ \vdots \end{pmatrix}$$

这就是我们两层堆叠的一种基本的原始思路想法。在不同模型预测的结果基础上再加一层模型，进行再训练，从而得到模型最终的预测。

Stacking本质上就是这么直接的思路，但是直接这样有时对于如果训练集和测试集分布不那么一致的情况下是有一点问题的，其问题在于用初始模型训练的标签再利用真实标签进行再训练，毫无疑问会导致一定的模型过拟合训练集，这样或许模型在测试集上的泛化能力或者说效果会有一定的下降，因此现在的问题变成了如何降低再训练的过拟合性，这里我们一般有两种方法。

- 1. 次级模型尽量选择简单的线性模型
- 2. 利用K折交叉验证

K-折交叉验证： 训练：



预测：

5.4 代码示例

5.4.1 回归\分类概率-融合：

1）简单加权平均，结果直接融合

In [1]:

```
## 生成一些简单的样本数据，test_prei 代表第i个模型的预测值
test_pre1 = [1.2, 3.2, 2.1, 6.2]
test_pre2 = [0.9, 3.1, 2.0, 5.9]
test_pre3 = [1.1, 2.9, 2.2, 6.0]

# y_test_true 代表第模型的真实值
y_test_true = [1, 3, 2, 6]
```

In [2]:

```
import numpy as np
import pandas as pd

## 定义结果的加权平均函数
def Weighted_method(test_pre1,test_pre2,test_pre3,w=[1/3,1/3,1/3]):
    Weighted_result = w[0]*pd.Series(test_pre1)+w[1]*pd.Series(test_pre2)+w[2]*pd.Series(test_pre3)
    return Weighted_result
```

In [3]:

```
from sklearn import metrics
# 各模型的预测结果计算MAE
print('Pred1 MAE:',metrics.mean_absolute_error(y_test_true, test_pre1))
print('Pred2 MAE:',metrics.mean_absolute_error(y_test_true, test_pre2))
print('Pred3 MAE:',metrics.mean_absolute_error(y_test_true, test_pre3))
```

```
Pred1 MAE: 0.175
Pred2 MAE: 0.075
Pred3 MAE: 0.1
```

In [4]:

```
## 根据加权计算MAE
w = [0.3,0.4,0.3] # 定义比重权值
Weighted_pre = Weighted_method(test_pre1,test_pre2,test_pre3,w)
print('Weighted_pre MAE:',metrics.mean_absolute_error(y_test_true, Weighted_pre))
```

```
Weighted_pre MAE: 0.0575
```

可以发现加权结果相对于之前的结果是有提升的，这种我们称其为简单的加权平均。

还有一些特殊的形式，比如mean平均，median平均

In [5]:

```
## 定义结果的加权平均函数
def Mean_method(test_pre1,test_pre2,test_pre3):
    Mean_result = pd.concat([pd.Series(test_pre1),pd.Series(test_pre2),pd.Series(test_pre3)],axis=1)
    return Mean_result
```

In [6]:

```
Mean_pre = Mean_method(test_pre1,test_pre2,test_pre3)
print('Mean_pre MAE:',metrics.mean_absolute_error(y_test_true, Mean_pre))
```

```
Mean_pre MAE: 0.0666666666667
```

In [7]:

```
## 定义结果的加权平均函数
def Median_method(test_pre1,test_pre2,test_pre3):
    Median_result = pd.concat([pd.Series(test_pre1),pd.Series(test_pre2),pd.Series(test_pre3)],axis=
    return Median_result
```

In [8]:

```
Median_pre = Median_method(test_pre1,test_pre2,test_pre3)
print('Median_pre MAE:',metrics.mean_absolute_error(y_test_true, Median_pre))
```

```
Median_pre MAE: 0.075
```

2) Stacking融合(回归):

In [9]:

```
from sklearn import linear_model

def Stacking_method(train_reg1,train_reg2,train_reg3,y_train_true,test_pre1,test_pre2,test_pre3,mod
    model_L2.fit(pd.concat([pd.Series(train_reg1),pd.Series(train_reg2),pd.Series(train_reg3)],axis=
    Stacking_result = model_L2.predict(pd.concat([pd.Series(test_pre1),pd.Series(test_pre2),pd.Serie
    return Stacking_result
```

In [10]:

```
## 生成一些简单的样本数据，test_prei 代表第i个模型的预测值
train_reg1 = [3.2, 8.2, 9.1, 5.2]
train_reg2 = [2.9, 8.1, 9.0, 4.9]
train_reg3 = [3.1, 7.9, 9.2, 5.0]
# y_test_true 代表第模型的真实值
y_train_true = [3, 8, 9, 5]

test_pre1 = [1.2, 3.2, 2.1, 6.2]
test_pre2 = [0.9, 3.1, 2.0, 5.9]
test_pre3 = [1.1, 2.9, 2.2, 6.0]

# y_test_true 代表第模型的真实值
y_test_true = [1, 3, 2, 6]
```

In [11]:

```
model_L2= linear_model.LinearRegression()
Stacking_pre = Stacking_method(train_reg1,train_reg2,train_reg3,y_train_true,
                               test_pre1,test_pre2,test_pre3,model_L2)
print('Stacking_pre MAE:',metrics.mean_absolute_error(y_test_true, Stacking_pre))
```

```
Stacking_pre MAE: 0.0421348314607
```

可以发现模型结果相对于之前有进一步的提升，这是我们需要注意的一点是，对于第二层Stacking的模型不宜选取的过于复杂，这样会导致模型在训练集上过拟合，从而使得在测试集上并不能达到很好的效果。

5.4.2 分类模型融合：

对于分类，同样的可以使用融合方法，比如简单投票，Stacking...

In [12]:

```
from sklearn.datasets import make_blobs
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.metrics import accuracy_score,roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
```

1）Voting投票机制：

Voting即投票机制，分为软投票和硬投票两种，其原理采用少数服从多数的思想。

In [16]:

```
'''
硬投票：对多个模型直接进行投票，不区分模型结果的相对重要度，最终投票数最多的类为最终被预测的类。
'''
iris = datasets.load_iris()

x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

clf1 = XGBClassifier(learning_rate=0.1, n_estimators=150, max_depth=3, min_child_weight=2, subsampl
                     colsample_bytree=0.6, objective='binary:logistic')
clf2 = RandomForestClassifier(n_estimators=50, max_depth=1, min_samples_split=4,
                              min_samples_leaf=63,oob_score=True)
clf3 = SVC(C=0.1)

# 硬投票
eclf = VotingClassifier(estimators=[('xgb', clf1), ('rf', clf2), ('svc', clf3)], voting='hard')
for clf, label in zip([clf1, clf2, clf3, eclf], ['XGBBoosting', 'Random Forest', 'SVM', 'Ensemble']
    scores = cross_val_score(clf, x, y, cv=5, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.97 (+/- 0.02) [XGBBoosting]
Accuracy: 0.33 (+/- 0.00) [Random Forest]
Accuracy: 0.95 (+/- 0.03) [SVM]
Accuracy: 0.94 (+/- 0.04) [Ensemble]
```

In [17]:

```
'''
软投票：和硬投票原理相同，增加了设置权重的功能，可以为不同模型设置不同权重，进而区别模型不同的重要度
'''
x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

clf1 = XGBClassifier(learning_rate=0.1, n_estimators=150, max_depth=3, min_child_weight=2, subsampl
                     colsample_bytree=0.8, objective='binary:logistic')
clf2 = RandomForestClassifier(n_estimators=50, max_depth=1, min_samples_split=4,
                              min_samples_leaf=63,oob_score=True)
clf3 = SVC(C=0.1, probability=True)

# 软投票
eclf = VotingClassifier(estimators=[('xgb', clf1), ('rf', clf2), ('svc', clf3)], voting='soft', wei
clf1.fit(x_train, y_train)

for clf, label in zip([clf1, clf2, clf3, eclf], ['XGBBoosting', 'Random Forest', 'SVM', 'Ensemble']
    scores = cross_val_score(clf, x, y, cv=5, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.96 (+/- 0.02) [XGBBoosting]
Accuracy: 0.33 (+/- 0.00) [Random Forest]
Accuracy: 0.95 (+/- 0.03) [SVM]
Accuracy: 0.96 (+/- 0.02) [Ensemble]
```

2）分类的Stacking\Blending融合：

stacking是一种分层模型集成框架。

以两层为例，第一层由多个基学习器组成，其输入为原始训练集，第二层的模型则是以第一层基学习器的输出作为训练集进行再训练，从而得到完整的stacking模型, stacking两层模型都使用了全部的训练数据。

In [18]:

```
'''
5-Fold Stacking
'''
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier,GradientBoostingClassifier
import pandas as pd
#创建训练的数据集
data_0 = iris.data
data = data_0[:100,:]

target_0 = iris.target
target = target_0[:100]

#模型融合中使用到的各个单模型
clfs = [LogisticRegression(solver='lbfgs'),
        RandomForestClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='entropy'),
        GradientBoostingClassifier(learning_rate=0.05, subsample=0.5, max_depth=6, n_estimators=5)]

#切分一部分数据作为测试集
X, X_predict, y, y_predict = train_test_split(data, target, test_size=0.3, random_state=2020)

dataset_blend_train = np.zeros((X.shape[0], len(clfs)))
dataset_blend_test = np.zeros((X_predict.shape[0], len(clfs)))

#5折stacking
n_splits = 5
skf = StratifiedKFold(n_splits)
skf = skf.split(X, y)

for j, clf in enumerate(clfs):
    #依次训练各个单模型
    dataset_blend_test_j = np.zeros((X_predict.shape[0], 5))
    for i, (train, test) in enumerate(skf):
        #5-Fold交叉训练，使用第i个部分作为预测，剩余的部分来训练模型，获得其预测的输出作为第i部分的
        X_train, y_train, X_test, y_test = X[train], y[train], X[test], y[test]
        clf.fit(X_train, y_train)
        y_submission = clf.predict_proba(X_test)[:, 1]
        dataset_blend_train[test, j] = y_submission
        dataset_blend_test_j[:, i] = clf.predict_proba(X_predict)[:, 1]
    #对于测试集，直接用这k个模型的预测值均值作为新的特征。
    dataset_blend_test[:, j] = dataset_blend_test_j.mean(1)
    print("val auc Score: %f" % roc_auc_score(y_predict, dataset_blend_test[:, j]))

clf = LogisticRegression(solver='lbfgs')
clf.fit(dataset_blend_train, y)
y_submission = clf.predict_proba(dataset_blend_test)[:, 1]

print("Val auc Score of Stacking: %f" % (roc_auc_score(y_predict, y_submission)))
```

```
val auc Score: 1.000000
val auc Score: 0.500000
val auc Score: 0.500000
val auc Score: 0.500000
val auc Score: 0.500000
Val auc Score of Stacking: 1.000000
```

Blending，其实和Stacking是一种类似的多层模型融合的形式

> 其主要思路是把原始的训练集先分成两部分，比如70%的数据作为新的训练集，剩下30%的数据作为测试集。

> 在第一层，我们在这70%的数据上训练多个模型，然后去预测那30%数据的label，同时也预测test集的label。

> 在第二层，我们就直接用这30%数据在第一层预测的结果做为新特征继续训练，然后用test集第一层预测的label做特征，用第二层训练的模型做进一步预测

其优点在于：

- 1.比stacking简单（因为不用进行k次的交叉验证来获得stacker feature）
- 2.避开了一个信息泄露问题：generlizers和stacker使用了不一样的数据集

缺点在于：

- 1.使用了很少的数据（第二阶段的blender只使用training set10%的量）
- 2.blender可能会过拟合
- 3.stacking使用多次的交叉验证会比较稳健 '''

In [19]:

```
'''
Blending
'''

#创建训练的数据集
#创建训练的数据集
data_0 = iris.data
data = data_0[:100,:]

target_0 = iris.target
target = target_0[:100]

#模型融合中使用到的各个单模型
clfs = [LogisticRegression(solver='lbfgs'),
        RandomForestClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        RandomForestClassifier(n_estimators=5, n_jobs=-1, criterion='entropy'),
        ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        #ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='entropy'),
        GradientBoostingClassifier(learning_rate=0.05, subsample=0.5, max_depth=6, n_estimators=5)]

#切分一部分数据作为测试集
X, X_predict, y, y_predict = train_test_split(data, target, test_size=0.3, random_state=2020)

#切分训练数据集为d1,d2两部分
X_d1, X_d2, y_d1, y_d2 = train_test_split(X, y, test_size=0.5, random_state=2020)
dataset_d1 = np.zeros((X_d2.shape[0], len(clfs)))
dataset_d2 = np.zeros((X_predict.shape[0], len(clfs)))

for j, clf in enumerate(clfs):
    #依次训练各个单模型
    clf.fit(X_d1, y_d1)
    y_submission = clf.predict_proba(X_d2)[:, 1]
    dataset_d1[:, j] = y_submission
    #对于测试集，直接用这k个模型的预测值作为新的特征。
    dataset_d2[:, j] = clf.predict_proba(X_predict)[:, 1]
    print("val auc Score: %f" % roc_auc_score(y_predict, dataset_d2[:, j]))

#融合使用的模型
clf = GradientBoostingClassifier(learning_rate=0.02, subsample=0.5, max_depth=6, n_estimators=30)
clf.fit(dataset_d1, y_d2)
y_submission = clf.predict_proba(dataset_d2)[:, 1]
print("Val auc Score of Blending: %f" % (roc_auc_score(y_predict, y_submission)))
```

```
val auc Score: 1.000000
val auc Score: 1.000000
val auc Score: 1.000000
val auc Score: 1.000000
val auc Score: 1.000000
Val auc Score of Blending: 1.000000
```

参考博客：https://blog.csdn.net/Noob_daniel/article/details/76087829 (https://blog.csdn.net/Noob_daniel/article/details/76087829)

3）分类的Stacking融合(利用mlxtend)：

In [ ]:

```
!pip install mlxtend

import warnings
warnings.filterwarnings('ignore')
import itertools
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier

from sklearn.model_selection import cross_val_score
from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions

# 以python自带的鸢尾花数据集为例
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target

clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)

label = ['KNN', 'Random Forest', 'Naive Bayes', 'Stacking Classifier']
clf_list = [clf1, clf2, clf3, sclf]

fig = plt.figure(figsize=(10,8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

clf_cv_mean = []
clf_cv_std = []
for clf, label, grd in zip(clf_list, label, grid):

    scores = cross_val_score(clf, X, y, cv=3, scoring='accuracy')
    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
    clf_cv_mean.append(scores.mean())
    clf_cv_std.append(scores.std())

    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf)
    plt.title(label)

plt.show()
```

可以发现 基模型 用 'KNN', 'Random Forest', 'Naive Bayes' 然后再这基础上 次级模型加一个 'LogisticRegression'，模型测试效果有着很好的提升。

5.4.3 一些其它方法：

将特征放进模型中预测，并将预测结果变换并作为新的特征加入原有特征中再经过模型预测结果 （Stacking变化）

（可以反复预测多次将结果加入最后的特征中）

In [22]:

```
def Ensemble_add_feature(train,test,target,clfs):

    # n_flods = 5
    # skf = list(StratifiedKFold(y, n_folds=n_flods))

    train_ = np.zeros((train.shape[0],len(clfs*2)))
    test_ = np.zeros((test.shape[0],len(clfs*2)))

    for j,clf in enumerate(clfs):
        '''依次训练各个单模型'''
        # print(j, clf)
        '''使用第1个部分作为预测，第2部分来训练模型，获得其预测的输出作为第2部分的新特征。'''
        # X_train, y_train, X_test, y_test = X[train], y[train], X[test], y[test]

        clf.fit(train,target)
        y_train = clf.predict(train)
        y_test = clf.predict(test)

        ## 新特征生成
        train_[:,j*2] = y_train**2
        test_[:,j*2] = y_test**2
        train_[:, j+1] = np.exp(y_train)
        test_[:, j+1] = np.exp(y_test)
        # print("val auc Score: %f" % r2_score(y_predict, dataset_d2[:, j]))
        print('Method ',j)

    train_ = pd.DataFrame(train_)
    test_ = pd.DataFrame(test_)
    return train_,test_
```

In [23]:

```
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()

data_0 = iris.data
data = data_0[:100,:]

target_0 = iris.target
target = target_0[:100]

x_train,x_test,y_train,y_test=train_test_split(data,target,test_size=0.3)
x_train = pd.DataFrame(x_train) ; x_test = pd.DataFrame(x_test)

#模型融合中使用到的各个单模型
clfs = [LogisticRegression(),
        RandomForestClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='gini'),
        ExtraTreesClassifier(n_estimators=5, n_jobs=-1, criterion='entropy'),
        GradientBoostingClassifier(learning_rate=0.05, subsample=0.5, max_depth=6, n_estimators=5)]

New_train,New_test = Ensemble_add_feature(x_train,x_test,y_train,clfs)

clf = LogisticRegression()
# clf = GradientBoostingClassifier(learning_rate=0.02, subsample=0.5, max_depth=6, n_estimators=30)
clf.fit(New_train, y_train)
y_emb = clf.predict_proba(New_test)[:, 1]

print("Val auc Score of stacking: %f" % (roc_auc_score(y_test, y_emb)))
```

```
Method  0
Method  1
Method  2
Method  3
Method  4
Val auc Score of stacking: 1.000000
```

5.4.4 本赛题示例

In [25]:

```
import pandas as pd
import numpy as np
import warnings
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')
%matplotlib inline

import itertools
import matplotlib.gridspec as gridspec
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
# from mlxtend.classifier import StackingClassifier
from sklearn.model_selection import cross_val_score, train_test_split
# from mlxtend.plotting import plot_learning_curves
# from mlxtend.plotting import plot_decision_regions

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split

from sklearn import linear_model
from sklearn import preprocessing
from sklearn.svm import SVR
from sklearn.decomposition import PCA,FastICA,FactorAnalysis,SparsePCA

import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [30]:

```
## 数据读取
Train_data = pd.read_csv('datalab/231784/used_car_train_20200313.csv', sep=' ')
TestA_data = pd.read_csv('datalab/231784/used_car_testA_20200313.csv', sep=' ')

print(Train_data.shape)
print(TestA_data.shape)
```

```
(150000, 31)
(50000, 30)
```

In [31]:

```
Train_data.head()
```

Out[31]:

	SaleID	name	regDate	model	brand	bodyType	fuelType	gearbox	power	kilometer	...
0	0	736	20040402	30.0	6	1.0	0.0	0.0	60	12.5	...
1	1	2262	20030301	40.0	1	2.0	0.0	0.0	0	15.0	...
2	2	14874	20040403	115.0	15	1.0	0.0	0.0	163	12.5	...
3	3	71865	19960908	109.0	10	0.0	0.0	1.0	193	15.0	...
4	4	111080	20120103	110.0	5	1.0	0.0	0.0	68	5.0	...

5 rows × 31 columns

In [32]:

```
numerical_cols = Train_data.select_dtypes(exclude = 'object').columns
print(numerical_cols)
```

```
Index(['SaleID', 'name', 'regDate', 'model', 'brand', 'bodyType', 'fuelType',
       'gearbox', 'power', 'kilometer', 'regionCode', 'seller', 'offerType',
       'creatDate', 'price', 'v_0', 'v_1', 'v_2', 'v_3', 'v_4', 'v_5', 'v_6',
       'v_7', 'v_8', 'v_9', 'v_10', 'v_11', 'v_12', 'v_13', 'v_14'],
      dtype='object')
```

In [33]:

```
feature_cols = [col for col in numerical_cols if col not in ['SaleID','name','regDate','price']]
```

In [34]:

```
X_data = Train_data[feature_cols]
Y_data = Train_data['price']

X_test  = TestA_data[feature_cols]

print('X train shape:',X_data.shape)
print('X test shape:',X_test.shape)
```

```
X train shape: (150000, 26)
X test shape: (50000, 26)
```

In [35]:

```
def Sta_inf(data):
    print('_min',np.min(data))
    print('_max:',np.max(data))
    print('_mean',np.mean(data))
    print('_ptp',np.ptp(data))
    print('_std',np.std(data))
    print('_var',np.var(data))
```

In [36]:

```
print('Sta of label:')
Sta_inf(Y_data)
```

```
Sta of label:
_min 11
_max: 99999
_mean 5923.32733333
_ptp 99988
_std 7501.97346988
_var 56279605.9427
```

In [37]:

```
X_data = X_data.fillna(-1)
X_test = X_test.fillna(-1)
```

In [43]:

```
def build_model_lr(x_train,y_train):
    reg_model = linear_model.LinearRegression()
    reg_model.fit(x_train,y_train)
    return reg_model

def build_model_ridge(x_train,y_train):
    reg_model = linear_model.Ridge(alpha=0.8)#alphas=range(1,100,5)
    reg_model.fit(x_train,y_train)
    return reg_model

def build_model_lasso(x_train,y_train):
    reg_model = linear_model.LassoCV()
    reg_model.fit(x_train,y_train)
    return reg_model

def build_model_gbdt(x_train,y_train):
    estimator =GradientBoostingRegressor(loss='ls',subsample= 0.85,max_depth= 5,n_estimators = 100)
    param_grid = {
            'learning_rate': [0.05,0.08,0.1,0.2],
            }
    gbdt = GridSearchCV(estimator, param_grid,cv=3)
    gbdt.fit(x_train,y_train)
    print(gbdt.best_params_)
    # print(gbdt.best_estimator_ )
    return gbdt

def build_model_xgb(x_train,y_train):
    model = xgb.XGBRegressor(n_estimators=120, learning_rate=0.08, gamma=0, subsample=0.8,\
        colsample_bytree=0.9, max_depth=5) #, objective ='reg:squarederror'
    model.fit(x_train, y_train)
    return model

def build_model_lgb(x_train,y_train):
    estimator = lgb.LGBMRegressor(num_leaves=63,n_estimators = 100)
    param_grid = {
        'learning_rate': [0.01, 0.05, 0.1],
    }
    gbm = GridSearchCV(estimator, param_grid)
    gbm.fit(x_train, y_train)
    return gbm
```

2）XGBoost的五折交叉回归验证实现

In [41]:

```
## xgb
xgr = xgb.XGBRegressor(n_estimators=120, learning_rate=0.1, subsample=0.8,\
        colsample_bytree=0.9, max_depth=7) #,objective ='reg:squarederror'

scores_train = []
scores = []

## 5折交叉验证方式
sk=StratifiedKFold(n_splits=5,shuffle=True,random_state=0)
for train_ind,val_ind in sk.split(X_data,Y_data):

    train_x=X_data.iloc[train_ind].values
    train_y=Y_data.iloc[train_ind]
    val_x=X_data.iloc[val_ind].values
    val_y=Y_data.iloc[val_ind]

    xgr.fit(train_x,train_y)
    pred_train_xgb=xgr.predict(train_x)
    pred_xgb=xgr.predict(val_x)

    score_train = mean_absolute_error(train_y,pred_train_xgb)
    scores_train.append(score_train)
    score = mean_absolute_error(val_y,pred_xgb)
    scores.append(score)

print('Train mae:',np.mean(score_train))
print('Val mae',np.mean(scores))
```

```
Train mae: 558.212360169
Val mae 693.120168439
```

3）划分数据集，并用多种方法训练和预测

In [42]:

```
## Split data with val
x_train,x_val,y_train,y_val = train_test_split(X_data,Y_data,test_size=0.3)

## Train and Predict
print('Predict LR...')
model_lr = build_model_lr(x_train,y_train)
val_lr = model_lr.predict(x_val)
subA_lr = model_lr.predict(X_test)

print('Predict Ridge...')
model_ridge = build_model_ridge(x_train,y_train)
val_ridge = model_ridge.predict(x_val)
subA_ridge = model_ridge.predict(X_test)

print('Predict Lasso...')
model_lasso = build_model_lasso(x_train,y_train)
val_lasso = model_lasso.predict(x_val)
subA_lasso = model_lasso.predict(X_test)

print('Predict GBDT...')
model_gbdt = build_model_gbdt(x_train,y_train)
val_gbdt = model_gbdt.predict(x_val)
subA_gbdt = model_gbdt.predict(X_test)
```

```
Predict LR...
Predict Ridge...
Predict Lasso...
Predict GBDT...
{'learning_rate': 0.1, 'n_estimators': 80}
```

一般比赛中效果最为显著的两种方法

In [44]:

```
print('predict XGB...')
model_xgb = build_model_xgb(x_train,y_train)
val_xgb = model_xgb.predict(x_val)
subA_xgb = model_xgb.predict(X_test)

print('predict lgb...')
model_lgb = build_model_lgb(x_train,y_train)
val_lgb = model_lgb.predict(x_val)
subA_lgb = model_lgb.predict(X_test)
```

```
predict XGB...
predict lgb...
```

In [45]:

```
print('Sta inf of lgb:')
Sta_inf(subA_lgb)
```

```
Sta inf of lgb:
_min -126.864734992
_max: 90152.4775557
_mean 5917.96632163
_ptp 90279.3422907
_std 7358.88582391
_var 54153200.5693
```

1）加权融合

In [46]:

```
def Weighted_method(test_pre1,test_pre2,test_pre3,w=[1/3,1/3,1/3]):
    Weighted_result = w[0]*pd.Series(test_pre1)+w[1]*pd.Series(test_pre2)+w[2]*pd.Series(test_pre3)
    return Weighted_result

## Init the Weight
w = [0.3,0.4,0.3]

## 测试验证集准确度
val_pre = Weighted_method(val_lgb,val_xgb,val_gbdt,w)
MAE_Weighted = mean_absolute_error(y_val,val_pre)
print('MAE of Weighted of val:',MAE_Weighted)

## 预测数据部分
subA = Weighted_method(subA_lgb,subA_xgb,subA_gbdt,w)
print('Sta inf:')
Sta_inf(subA)
## 生成提交文件
sub = pd.DataFrame()
sub['SaleID'] = X_test.index
sub['price'] = subA
sub.to_csv('./sub_Weighted.csv',index=False)
```

```
MAE of Weighted of val: 730.877443666
Sta inf:
_min -2816.93914153
_max: 88576.7842223
_mean 5920.38233546
_ptp 91393.7233639
_std 7325.20946801
_var 53658693.7502
```

In [47]:

```
## 与简单的LR（线性回归）进行对比
val_lr_pred = model_lr.predict(x_val)
MAE_lr = mean_absolute_error(y_val,val_lr_pred)
print('MAE of lr:',MAE_lr)
```

```
MAE of lr: 2597.45638384
```

2）Starking融合

2) Starking融合

In [48]:

```
## Starking

## 第一层
train_lgb_pred = model_lgb.predict(x_train)
train_xgb_pred = model_xgb.predict(x_train)
train_gbdt_pred = model_gbdt.predict(x_train)

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_val = pd.DataFrame()
Strak_X_val['Method_1'] = val_lgb
Strak_X_val['Method_2'] = val_xgb
Strak_X_val['Method_3'] = val_gbdt

Strak_X_test = pd.DataFrame()
Strak_X_test['Method_1'] = subA_lgb
Strak_X_test['Method_2'] = subA_xgb
Strak_X_test['Method_3'] = subA_gbdt
```

In [49]:

```
Strak_X_test.head()
```

Out[49]:

	Method_1	Method_2	Method_3
0	39682.037093	41029.078125	40552.596813
1	239.498371	266.032654	393.909761
2	6915.162439	7345.680664	7623.552178
3	11861.783785	11721.493164	11463.293245
4	583.773267	513.307983	520.665295

In [50]:

```
## level2-method
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('MAE of Stacking-LR:',mean_absolute_error(y_train,train_pre_Stacking))

## 验证集
val_pre_Stacking = model_lr_Stacking.predict(Strak_X_val)
print('MAE of Stacking-LR:',mean_absolute_error(y_val,val_pre_Stacking))

## 预测集
print('Predict Stacking-LR...')
subA_Stacking = model_lr_Stacking.predict(Strak_X_test)
```

```
MAE of Stacking-LR: 628.399441036
MAE of Stacking-LR: 707.673951794
Predict Stacking-LR...
```

In [55]:

```
subA_Stacking[subA_Stacking<10]=10  ## 去除过小的预测值

sub = pd.DataFrame()
sub['SaleID'] = X_test.index
sub['price'] = subA_Stacking
sub.to_csv('./sub_Stacking.csv',index=False)
```

In [56]:

```
print('Sta inf:')
Sta_inf(subA_Stacking)
```

```
Sta inf:
_min 10.0
_max: 90849.3729816
_mean 5917.39429976
_ptp 90839.3729816
_std 7396.09766172
_var 54702260.6217
```

3.4 经验总结

比赛的融合这个问题，个人的看法来说其实涉及多个层面，也是提分和提升模型鲁棒性的一种重要方法：

- 1）**结果层面的融合**，这种是最常见的融合方法，其可行的融合方法也有很多，比如根据结果的得分进行加权融合，还可以做Log，exp处理等。在做结果融合的时候，有一个很重要的条件是模型结果的得分要比较近似，然后结果的差异要比较大，这样的结果融合往往有比较好的效果提升。
- 2）**特征层面的融合**，这个层面其实感觉不叫融合，准确说可以叫分割，很多时候如果我们用同种模型训练，可以把特征进行切分给不同的模型，然后在后面进行模型或者结果融合有时也能产生比较好的效果。
- 3）**模型层面的融合**，模型层面的融合可能就涉及模型的堆叠和设计，比如加Staking层，部分模型的结果作为特征输入等，这些就需要多实验和思考了，基于模型层面的融合最好不同模型类型要有一定的差异，用同种模型不同的参数的收益一般是比较小的。

Task 5-模型融合 END.

--- By: ML67

Email: maolinw67@163.com
PS: 华中科技大学研究生，长期混迹Tianchi等，希望和大家多多交流。
github: https://github.com/mlw67 （近期会做一些书籍推导和代码的整理）

关于Datawhale：

> Datawhale是一个专注于数据科学与AI领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以“for the learner，和学习者一起成长”为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。

本次数据挖掘路径学习，专题知识将在天池分享，详情可关注Datawhale：