# 二手车交易价格预测

第三组成员：

刘刚健、吴腾达、林坤、何洪波、胡晓峰、朱伟章

# 目录

- 项目背景
- 数据的预处理与分析
- 特征工程
- 模型训练
- 结果分析

# 背景与意义


- 我国每年乘用车二手车市场交易规模已经高达300万辆以上
- 《4月二手车在线拍卖大数据报告》显示，在选择卖车渠道时，58. 8%的车主优先选择线上二手车电商平台来卖车
- 在二手车市场上存在以下问题：
  - 二手评估车的盲目性
  - 二手车信息的滞后性
  - 交易双方信息的不对等性
- 二手车交易中如何对车辆进行合理的估值已经成为消费者和经销商最为关注的问题
- 通过数据分析，了解影响二手车成交价格的影响因子，建立模型来预测二手车价格，为买卖双方提供交易价格参考依据，更好的为二手车交易中的买家与卖家群体服务。

# 数据获取

| Field | Description |
| --- | --- |
| SaleID | 交易ID，唯一编码 |
| name | 汽车交易名称，已脱敏 |
| regDate | 汽车注册日期，例如20160101，2016年01月01日 |
| model | 车型编码，已脱敏 |
| brand | 汽车品牌，已脱敏 |
| bodyType | 车身类型：豪华轿车：0，微型车：1，厢型车：2，大巴车：3，敞篷车：4，双门汽车：5，商务车：6，搅拌车：7 |
| fuelType | 燃油类型：汽油：0，柴油：1，液化石油气：2，天然气：3，混合动力：4，其他：5，电动：6 |
| gearbox | 变速箱：手动：0，自动：1 |
| power | 发动机功率：范围 [ 0, 600 ] |
| kilometer | 汽车已行驶公里，单位万km |
| notRepairedDamage | 汽车有尚未修复的损坏：是：0，否：1 |
| regionCode | 地区编码，已脱敏 |
| seller | 销售方：个体：0，非个体：1 |
| offerType | 报价类型：提供：0，请求：1 |
| creatDate | 汽车上线时间，即开始售卖时间 |
| price | 二手车交易价格（预测目标） |
| v系列特征 | 匿名特征，包含v0-14在内15个匿名特征 |

数据来自天池的开源数据集，其中15万条作为训练集，还有5万条作为测试集

包含31列变量信息：

二手车交易价格（预测目标），交易ID，汽车交易名称，汽车注册日期，车型编码，汽车品牌，车型类型，燃油类型，变速箱，发动机功率，汽车已行驶公里，汽车有尚未修复的损坏，地区编码，销售方，报价类型，汽车上线时间，还有15列为匿名变量。

# 数据预处理

**2. 判断缺失与异常**

```
In [50]: Train_data.isnull().sum()
```

```
Out[50]: 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
```

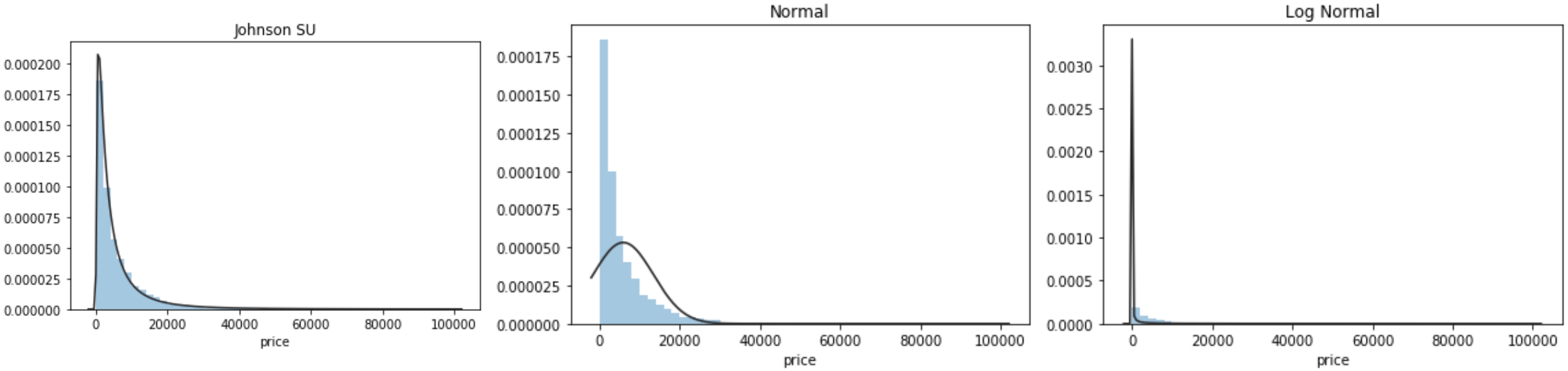
```
In [79]: TestA_data.isnull().sum()
```

```
Out[79]: SaleID                0
         name                  0
         regDate               0
         gearbox            1968
         power                 0
         kilometer             0
         notRepairedDamage     0
         regionCode            0
         seller                0
         offerType             0
         creatDate             0
         v_0                   0
```

缺失值处理

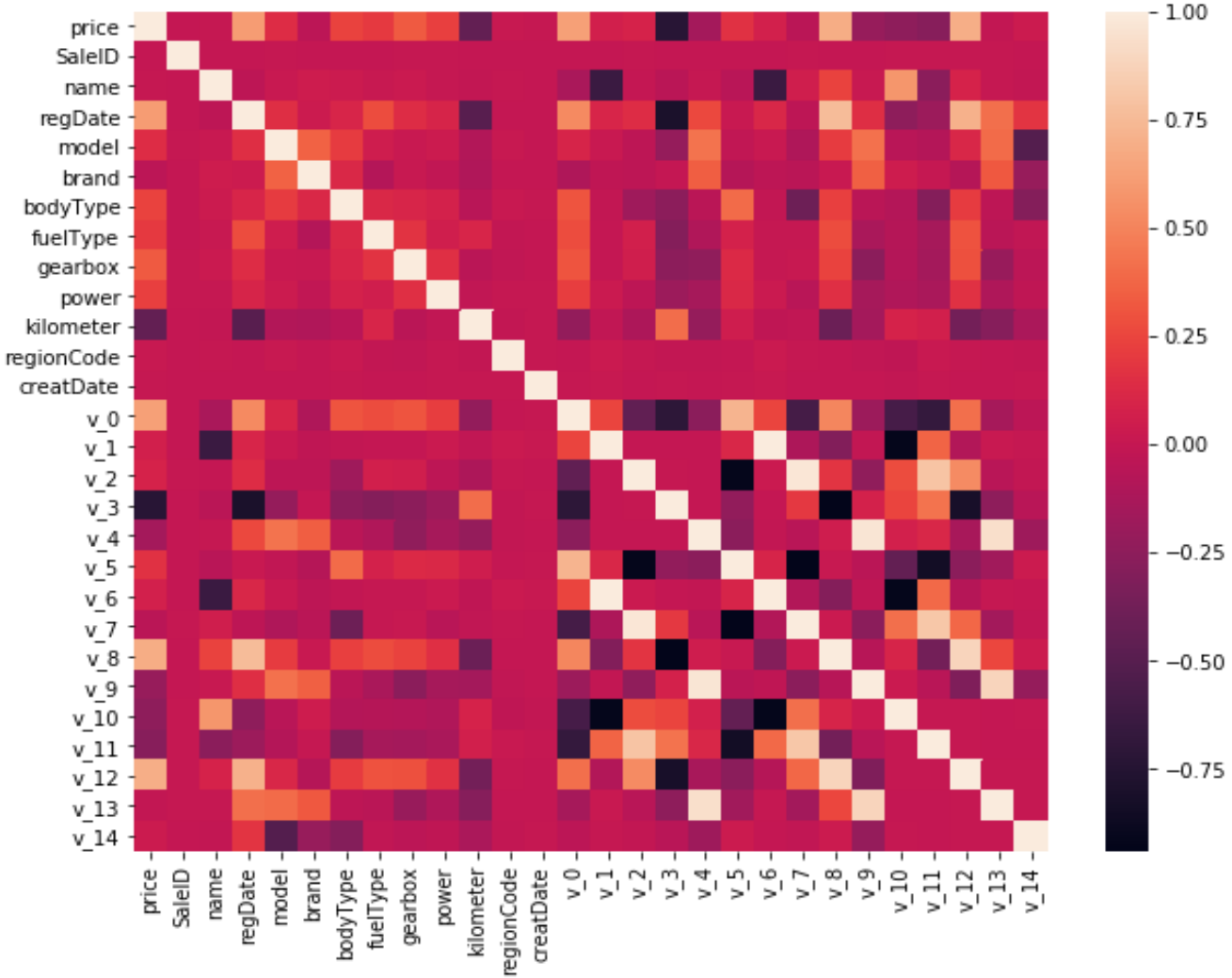
- bodyType 缺失4506条
- fuelType 缺失8680条
- gearbox 5981条
- model缺失1条
- notRepairedDamage项的数据，发现有24324值为‘-’，也属于缺失值。
- 将缺失的数值特征填充为NaN
- 缺失的分类特征填充为‘MISSING’

# 数据分析
## 价格分布情况




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

# 数据分析
## 相关性分析

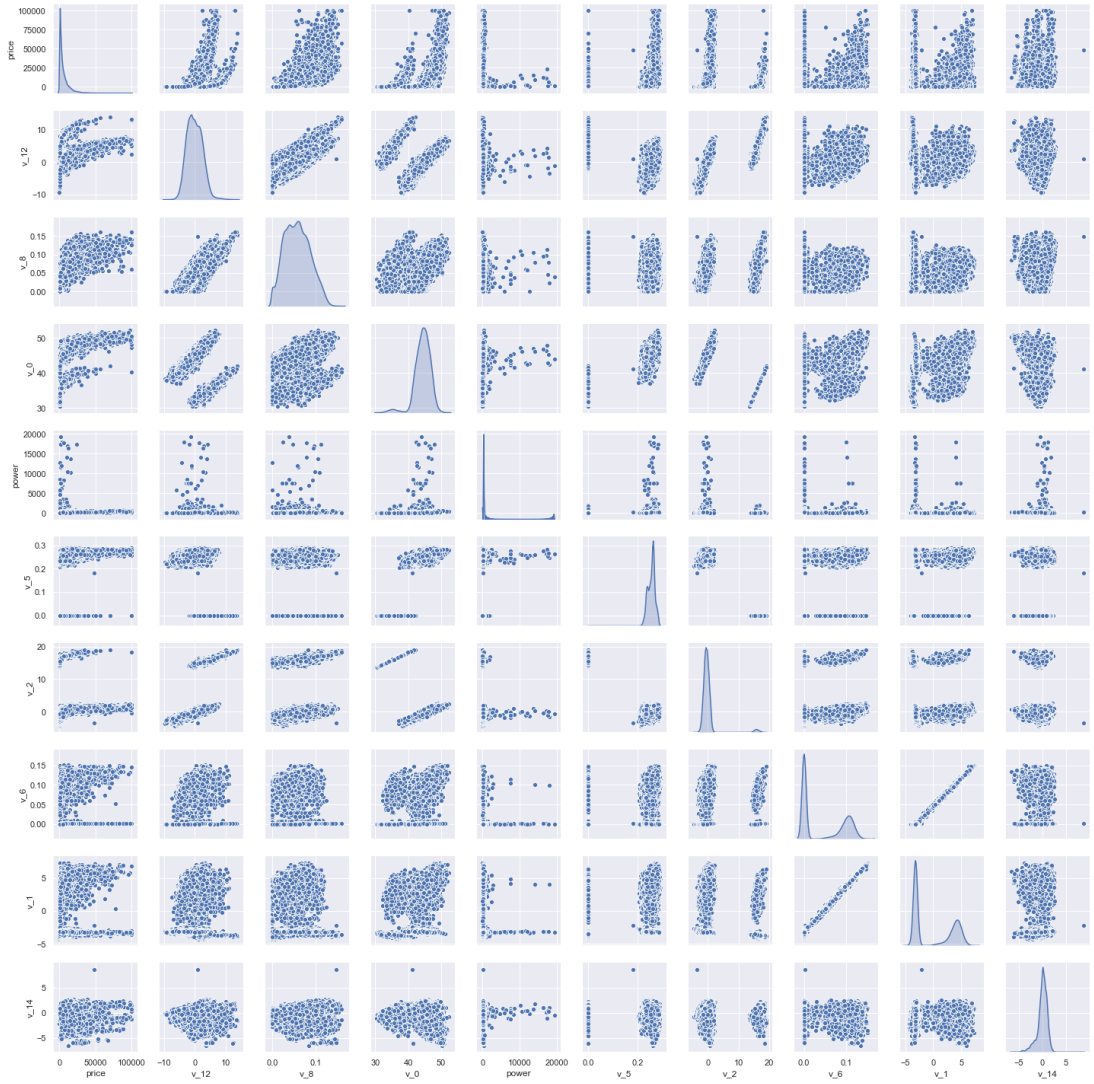



从图中可以看出：

- 二手车价格与汽车**已行驶公里数**呈明显**负**相关关系
- 与**汽车注册日期**呈现明显**正**相关关系

# 数据分析

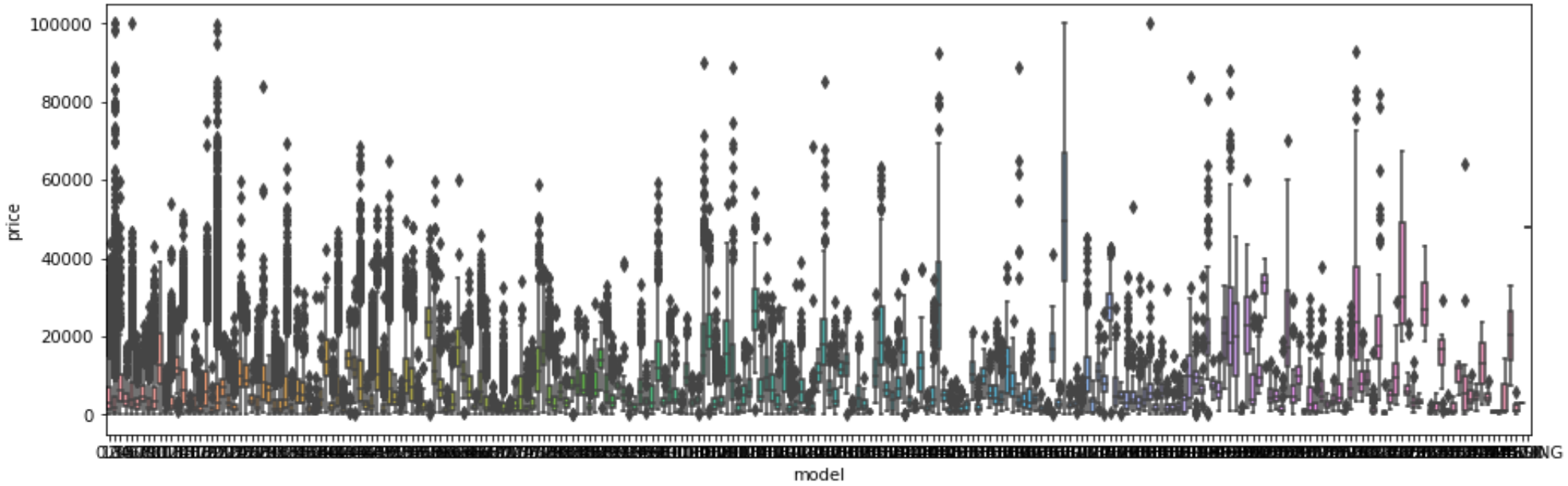
## 数字特征相互之间的关系可视化




- model特征有个248不同的值
- brand特征有个40不同的值
- bodyType特征有个8不同的值
- fuelType特征有个7不同的值
- gearbox特征有个2不同的值
- name和regionCode较为稀疏，暂不考虑

# 特征工程

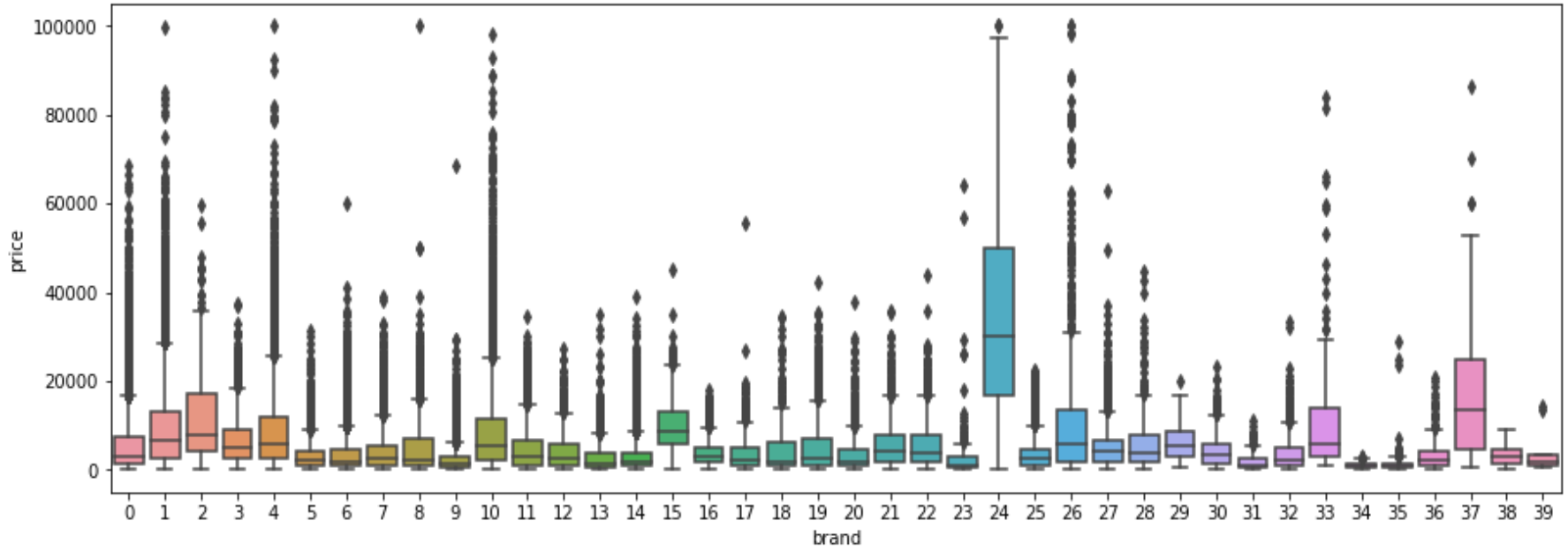
## 分析特征对价格的影响




车型编码与价格的箱线图

# 特征工程

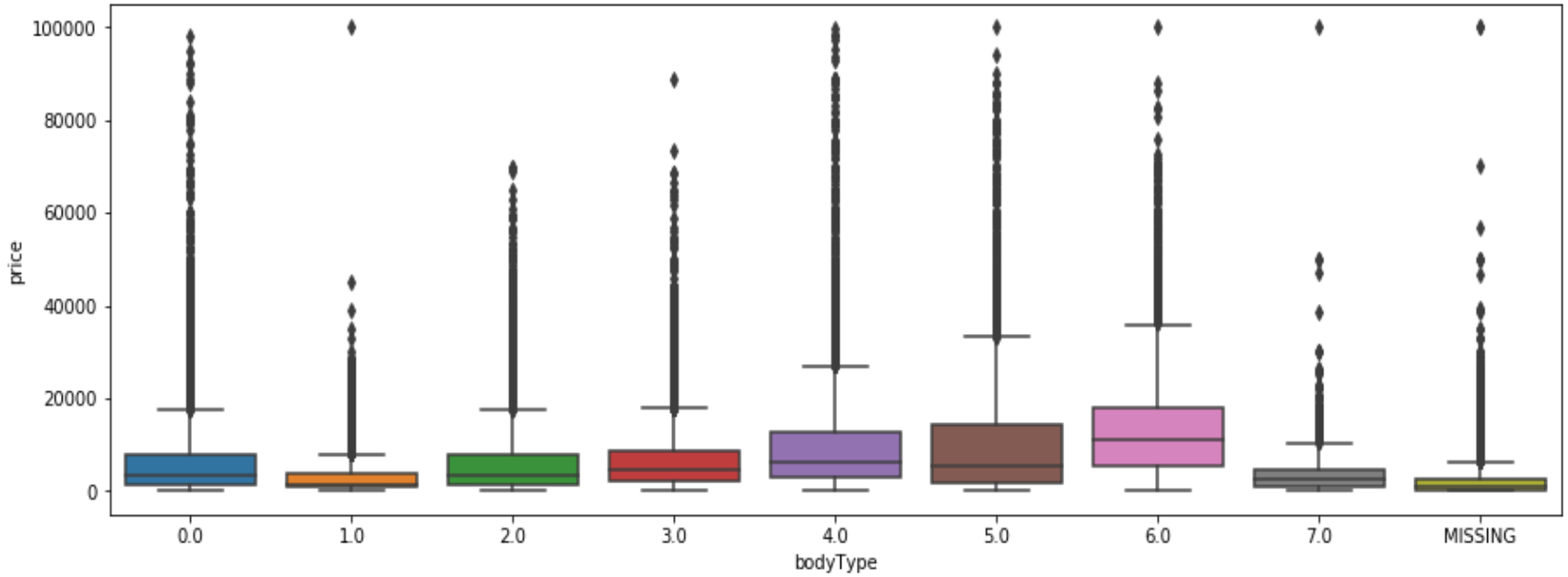
## 分析特征对价格的影响




汽车品牌与价格的箱线图

# 特征工程

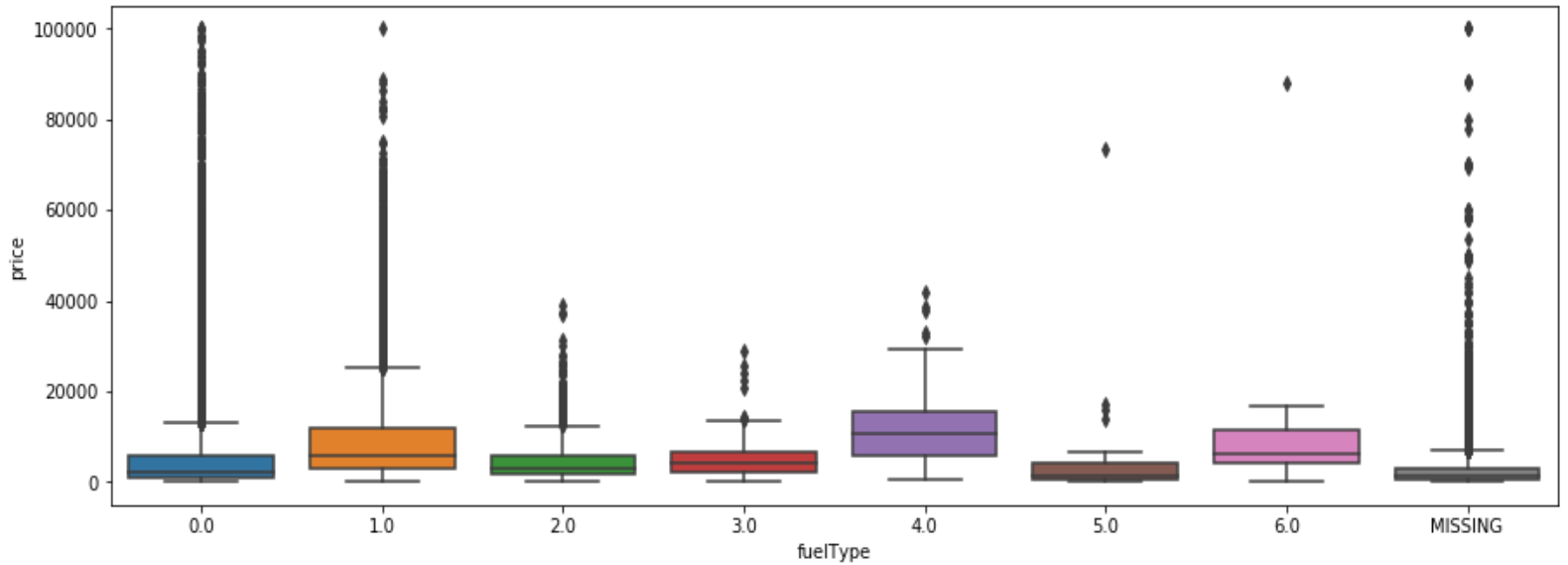
## 分析特征对价格的影响




车身类型与价格的箱线图

# 特征工程

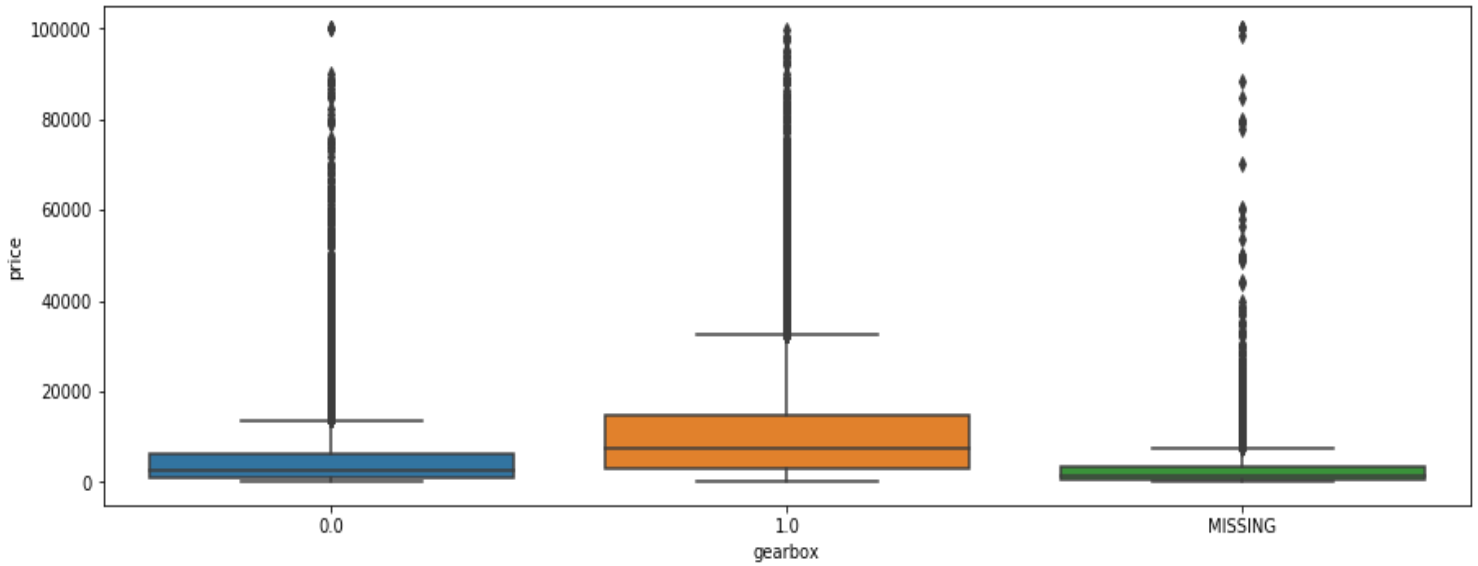
## 分析特征对价格的影响




燃油类型与价格的箱线图

# 特征工程

## 分析特征对价格的影响




变速箱与价格的箱线图

# 特征工程

## 特征编码

```
Train_data = pd.get_dummies(Train_data)
TestA_data = pd.get_dummies(TestA_data)
```

特征中存在一些离散的分类特征信息，包括以下这些特征：

brand 品牌编码
bodyType 车身类型
fuelType 燃油类型
没有选择使用model：分类数过多

对于这些分类特征信息，我们采用独热编码，生成了以下这些新特征：

```
['brand_0', 'brand_1', 'brand_10', 'brand_11', 'brand_12', 'brand_13', 'brand_14',
'brand_15', 'brand_16', 'brand_17', 'brand_18', 'brand_19', 'brand_2', 'brand_20',
'brand_21', 'brand_22', 'brand_23', 'brand_24', 'brand_25', 'brand_26', 'brand_27',
'brand_28', 'brand_29', 'brand_3', 'brand_30', 'brand_31', 'brand_32', 'brand_33',
'brand_34', 'brand_35', 'brand_36', 'brand_37', 'brand_38', 'brand_39', 'brand_4',
'brand_5', 'brand_6', 'brand_7', 'brand_8', 'brand_9', 'bodyType_0.0', 'bodyType_1.0',
'bodyType_2.0', 'bodyType_3.0', 'bodyType_4.0', 'bodyType_5.0', 'bodyType_6.0',
'bodyType_7.0', 'bodyType_nan', 'fuelType_0.0', 'fuelType_1.0', 'fuelType_2.0',
'fuelType_3.0', 'fuelType_4.0', 'fuelType_5.0', 'fuelType_6.0', 'fuelType_nan']
```

# 特征工程

## 特征组合

```
# 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'
Train_data['used_time'] = (pd.to_datetime(Train_data['creatDate'], format='%Y%m%d', errors='coerce') -
                           pd.to_datetime(Train_data['regDate'], format='%Y%m%d', errors='coerce')).dt.days

TestA_data['used_time'] = (pd.to_datetime(Train_data['creatDate'], format='%Y%m%d', errors='coerce') -
                           pd.to_datetime(Train_data['regDate'], format='%Y%m%d', errors='coerce')).dt.days
```

观察到汽车注册日期(regDate)和价格关连比较大，并且原始数据中还包含交易日期(creatDate)
构造这两个日期之间的天数作为新特征used_time

used_time= creatDate-regDate

改进方向：对特征编码进行降维

# 特征工程

**2. 特征编码**

In [21]:
```
Train_data = pd.get_dummies(Train_data)
TestA_data = pd.get_dummies(TestA_data)
```

In [22]:
```
Train_data.head()
```

Out[22]:

| | SaleID | name | regDate | power | kilometer | notRepairedDamage | regionCode | seller | offerType | creatDate | ... | fuelType_1.0 | fuelType_2.0 | fuelType_3.0 | fuelT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 736 | 20040402 | 60 | 12.5 | 0.0 | 1046 | 0 | 0 | 20160404 | ... | 0 | 0 | 0 | |
| **1** | 1 | 2262 | 20030301 | 0 | 15.0 | 1.0 | 4366 | 0 | 0 | 20160309 | ... | 0 | 0 | 0 | |
| **2** | 2 | 14874 | 20040403 | 163 | 12.5 | 0.0 | 2806 | 0 | 0 | 20160402 | ... | 0 | 0 | 0 | |
| **3** | 3 | 71865 | 19960908 | 193 | 15.0 | 0.0 | 434 | 0 | 0 | 20160312 | ... | 0 | 0 | 0 | |
| **4** | 4 | 111080 | 20120103 | 68 | 5.0 | 0.0 | 6977 | 0 | 0 | 20160313 | ... | 0 | 0 | 0 | |

5 rows × 336 columns

In [23]:
```
TestA_data.head()
```

Out[23]:

| | SaleID | name | regDate | gearbox | power | kilometer | notRepairedDamage | regionCode | seller | offerType | ... | bodyType_7.0 | bodyType_nan | fuelType_0.0 | fu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 200000 | 133777 | 20000501 | 0.0 | 101 | 15.0 | 0.0 | 5019 | 0 | 0 | ... | 0 | 0 | 1 | |
| **1** | 200001 | 61206 | 19950211 | 0.0 | 73 | 6.0 | 0.0 | 1505 | 0 | 0 | ... | 0 | 0 | 1 | |
| **2** | 200002 | 67829 | 20090606 | 0.0 | 120 | 5.0 | 1.0 | 1776 | 0 | 0 | ... | 0 | 0 | 1 | |
| **3** | 200003 | 8892 | 20020601 | 0.0 | 58 | 15.0 | 0.0 | 26 | 0 | 0 | ... | 0 | 0 | 1 | |
| **4** | 200004 | 76998 | 20030301 | 0.0 | 116 | 15.0 | 0.0 | 738 | 0 | 0 | ... | 0 | 0 | 0 | |

5 rows × 329 columns

# 模型训练

- lightgbm：由于现在的比赛数据越来越大，想要获得一个比较高的预测精度，同时又要减少内存占用以及提升训练速度，lightgbm是一个非常不错的选择，其可达到与xgboost相似的预测效果。

- xgboost：在lightgbm出来之前，是打比赛的不二之选，现在由于需要做模型融合以提高预测精度，所以也需要使用到xgboost。

# 模型训练

## XGBoost

XGBoost模型特点：

- 不需要操心特征工程
- 不需要进行特征变换（归一化、标准化、取log等）
- 不需要进行特征选择
- 能够处理缺失值

训练方式：

- 模型切分：随机抽取数据集中70%的样本作为训练集，剩下的30%作为验证集
- 参数调节：
  - 利用xgb进行五折交叉验证调参
  - 对参数learning_rate采用网格搜索，选取最佳的模型参数
- 模型融合：

```
# 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
# 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，由此我们进行对应的后修正
val_Weighted[val_Weighted<0]=10
print('MAE of val with Weighted ensemble:',mean_absolute_error(y_val,val_Weighted))
```

# 模型训练

## XGBoost

**1. 利用xgb进行五折交叉验证查看模型的参数效果**

```
## 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))
```

# 模型训练

## XGBoost

**2. 定义xgb和lgb模型函数**

```
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
```

# 模型训练

**3. 切分数据集（Train,Val）进行模型训练，评价和预测**

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

```
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: 652.636634980716
Predict lgb...
Sta of Predict lgb:
_min -639.6063215735171
_max: 92372.27497141116
_mean 5906.438922690525
_ptp 93011.88129298468
_std 7342.960815249283
_var 53919073.53428641
```

```
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...
[21:44:58] WARNING: src/objective/regression_obj.cu:152: re
MAE of val with xgb: 685.8261277876218
Predict xgb...
[21:49:06] WARNING: src/objective/regression_obj.cu:152: re
Sta of Predict xgb:
_min -92.93175
_max: 91331.2
_mean 5905.853
_ptp 91424.13
_std 7325.025
_var 53655990.0
```

# 模型融合

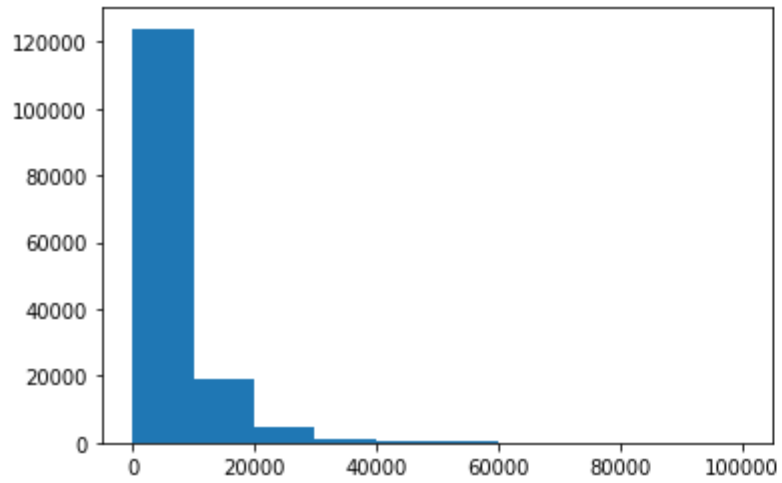
**4. 两模型加权融合**

```
## 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，由此我们进行对应的后修正
print('MAE of val with Weighted ensemble:',mean_absolute_error(y_val,val_Weighted))
```

```
MAE of val with Weighted ensemble: 617.4812625330834
```

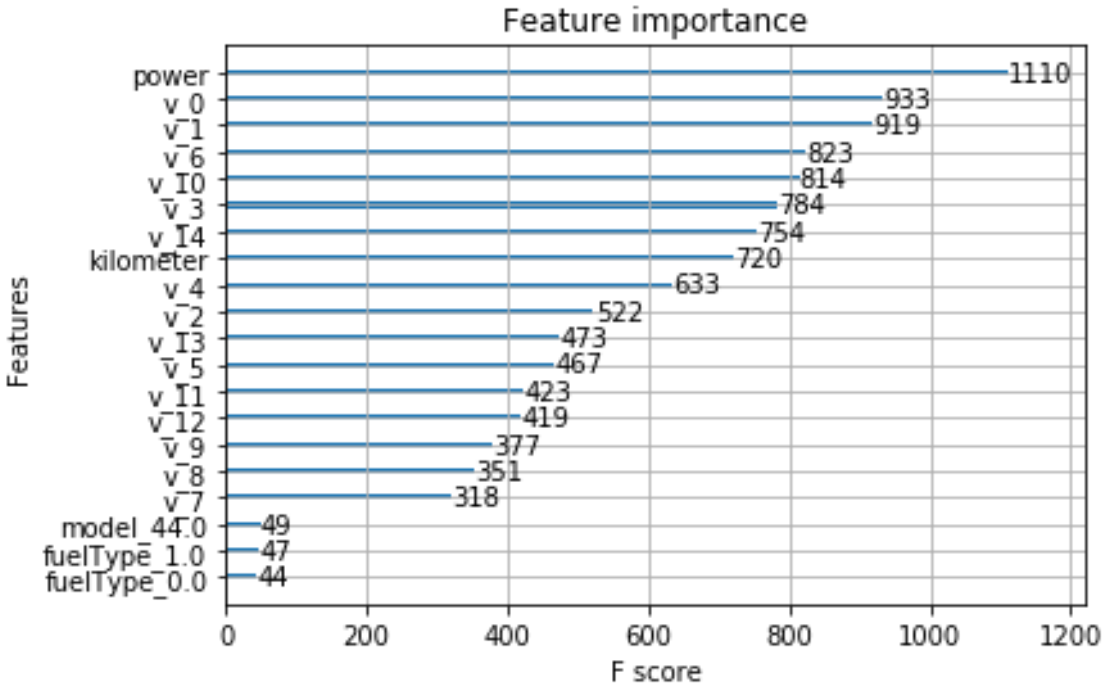
```
sub_Weighted = (1-MAE_lgb/(MAE_xgb+MAE_lgb))*subA_lgb+(1-MAE_xgb/(MAE_xgb+MAE_lgb))*subA_xgb

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

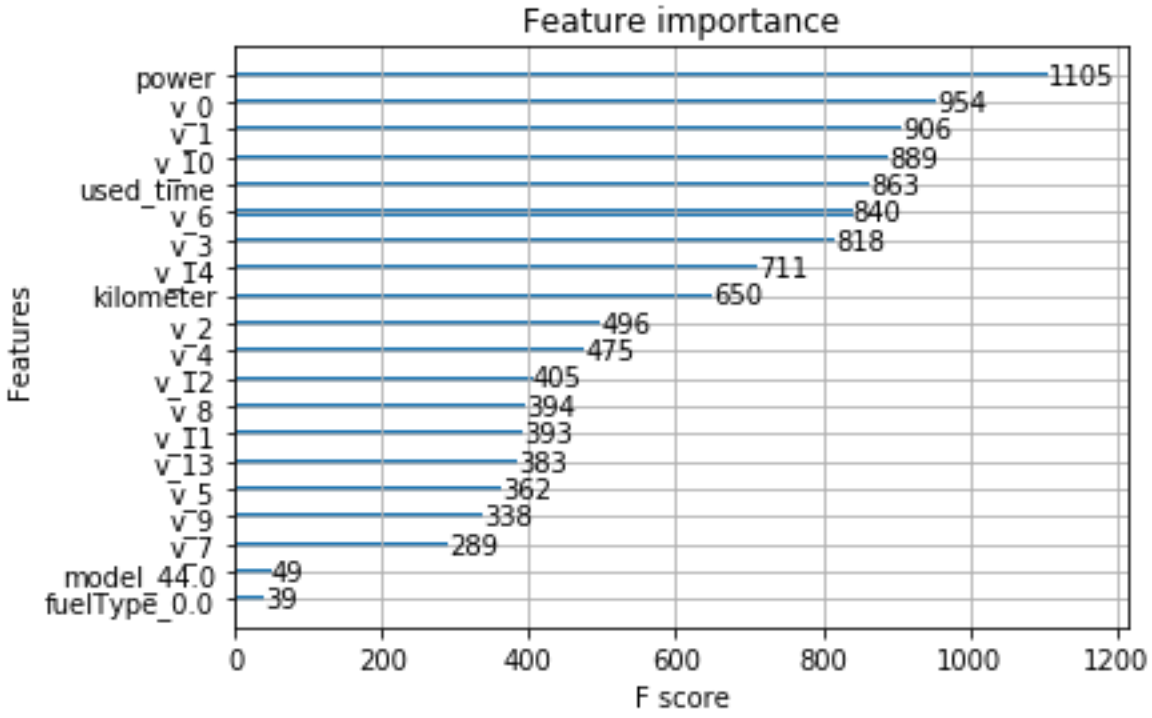


# 模型结果

```
from xgboost import plot_importance
plot_importance(model_xgb,max_num_features=20)
plt.show()
plt.close()
```




不考虑使用时长used_time




考虑使用时长used_time

# 模型结果

- 没有进行特征工程的模型
  - 不考虑分类特征
  - 验证集MAE为681
  - 天池测试集MAE为672
- 没有加入used_time的模型
  - 验证集MAE为617
  - 天池测试集MAE为647
- 加入了used_time
  - 验证集MAE为504
  - 天池测试集MAE为1412
- 特征工程对模型准确度的作用很重要
- 猜测可能是天池的训练集和测试集之间存在一些差别

# 谢谢