2023-2024 学年第一学期计算机 23C1-C2 班人工智能模块

《机器学习》期末大作业

班级：计算机 23C2

学号：2023031471

姓名：林爱娣

2024 年 11 月

基于随机森林算法的二手车价格预测

一、利用机器学习解决问题

1.1 基于随机森林模型的个人信用风险评估研究

1.2 基于 LiDAR 数据和随机森林算法的森林滑坡检测

1.3 基于改进随机森林优化算法在医疗数据中的应用研究

1.4 基于随机森林算法的推荐系统的设计与实现

1.5 基于矩阵分解与随机森林的多准则推荐算法

参考文献

[1]何静.基于随机森林模型的个人信用评估研究[D].上海工程技术大学,2020.DOI:10.27715/d.cnki.gshgj.2020.000609.

[5] 林栢全，肖菁. 基于矩阵分解与随机森林的多准则推荐算法[D]. 华南师范大学，2019. DOI:10.6054/j.jscnun.2019036.

二、利用随机森林算法进行二手车价格预测

2.1 背景介绍

2.2 随机森林算法背景介绍

2.3 选择随机森林算法原因

2.3 二手车价格预测项目详情

2.3.1 确定问题

2.3.2 收集数据

2.3.3 数据预处理

2.3.4 数据可视化

2.3.5 模型训练

2.3.6 算法优化

三、项目详细实现过程

3.1 Python 库导入及数据读取

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import GridSearchCV
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
```

```
train_df = pd.read_csv("data/train.csv")
test_df = pd.read_csv("data/test.csv")
```

3.2 数据预处理

3.2.1 数据预览

查看训练集和训练集信息

```
train_df.head()
```

	id	brand	model	model_year	milage	fuel_type	engine	transmission	accident	clean_title	price
0	0	MINI	Cooper S Base	2007	213000	Gasoline	172.0HP 1.6L 4 Cylinder Engine Gasoline Fuel	A/T	None reported	Yes	4200
1	1	Lincoln	LS V8	2002	143250	Gasoline	252.0HP 3.9L 8 Cylinder Engine Gasoline Fuel	A/T	At least 1 accident or damage reported	Yes	4999
2	2	Chevrolet	Silverado 2500 LT	2002	136731	E85 Flex Fuel	320.0HP 5.3L 8 Cylinder Engine Flex Fuel Capab...	A/T	None reported	Yes	13900
3	3	Genesis	G90 5.0 Ultimate	2017	19500	Gasoline	420.0HP 5.0L 8 Cylinder Engine Gasoline Fuel	Transmission w/Dual Shift Mode	None reported	Yes	45000
4	4	Mercedes-Benz	Metris Base	2021	7388	Gasoline	208.0HP 2.0L 4 Cylinder Engine Gasoline Fuel	7-Speed A/T	None reported	Yes	97500

```
# 打印训练集相关信息
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188533 entries, 0 to 188532
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            188533 non-null  int64 
 1   brand         188533 non-null  object
 2   model         188533 non-null  object
 3   model_year    188533 non-null  int64 
 4   milage        188533 non-null  int64 
 5   fuel_type     183450 non-null  object
 6   engine        188533 non-null  object
 7   transmission  188533 non-null  object
 8   accident      186081 non-null  object
 9   clean_title   167114 non-null  object
 10  price         188533 non-null  int64 
dtypes: int64(4), object(7)
memory usage: 15.8+ MB
```

```
# 打印测试集相关信息
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125690 entries, 0 to 125689
Data columns (total 10 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            125690 non-null  int64 
 1   brand         125690 non-null  object
 2   model         125690 non-null  object
 3   model_year    125690 non-null  int64 
 4   milage        125690 non-null  int64 
 5   fuel_type     122307 non-null  object
 6   engine        125690 non-null  object
 7   transmission  125690 non-null  object
 8   accident      124058 non-null  object
 9   clean_title   111451 non-null  object
dtypes: int64(3), object(7)
memory usage: 9.6+ MB
```

说明：训练集共有 188533 条数据，用于训练模型；测试集共有 125690 数据用于预测价格。

3.2.2 重复值

重复值

```
# 查看重复值
print(f'训练集重复值：{train_df.duplicated().sum()}')
print(f'测试集重复值：{test_df.duplicated().sum()}')
```

```
训练集重复值：0
测试集重复值：0
```

3.2.3 缺失值

缺失值

```
# 查看缺失值
print('训练集缺失值:')
print(train_df.isnull().sum())
print('-' * 50)
print('测试集缺失值:')
print(test_df.isnull().sum())
```

测试集缺失值:

id	0
brand	0
model	0
model_year	0
milage	0
fuel_type	3383
engine	0
transmission	0
accident	1632
clean_title	14239

dtype: int64

训练集缺失值:

id	0
brand	0
model	0
model_year	0
milage	0
fuel_type	5083
engine	0
transmission	0
accident	2452
clean_title	21419
price	0

dtype: int64

fuel_type燃料类型缺失值

```
# 处理字段fuel_type燃料类型缺失值：按品牌和型号填充，若无法确定则使用品牌的众数
# 首先将 '-' 标记为缺失值
train_df['fuel_type'] = train_df['fuel_type'].replace('-', np.nan)
test_df['fuel_type'] = test_df['fuel_type'].replace('-', np.nan)
# 处理燃料类型缺失值：按品牌和型号填充，若无法确定则使用品牌的众数
fuel_type_mode_by_brand_model = train_df.groupby(['brand', 'model'])['fuel_type'].apply(
    lambda x: x.mode().iloc[0] if not x.mode().empty else None)
fuel_type_mode_by_brand = train_df.groupby('brand')['fuel_type'].apply(
    lambda x: x.mode().iloc[0] if not x.mode().empty else None)


def fill_fuel_type(row):
    if pd.isnull(row['fuel_type']):
        # 先按品牌和型号填充
        brand_model_fill = fuel_type_mode_by_brand_model.get((row['brand'], row['model']), None)
        # 若品牌和型号均缺失，按品牌填充
        return brand_model_fill if brand_model_fill is not None else
        fuel_type_mode_by_brand.get(row['brand'],'Gasoline')
    return row['fuel_type']


# 应用填充方法到训练集和测试集
train_df['fuel_type'] = train_df.apply(fill_fuel_type, axis=1)
test_df['fuel_type'] = test_df.apply(fill_fuel_type, axis=1)
```

说明：处理字段 fuel_type 燃料类型缺失值：按品牌和型号填充，若无法确定则使用品牌的众数。

accident处理事故历史缺失值

```
# 处理事故历史缺失值：删除有缺失值的行
train_df.dropna(subset=['accident'], inplace=True)
test_df.dropna(subset=['accident'], inplace=True)
```

clean_title是否拥有健全良好的所有权证明缺失值

```
# 删除训练集和测试集中的 clean_title 列
train_df.drop(columns=['clean_title'], inplace=True)
test_df.drop(columns=['clean_title'], inplace=True)
```

说明：处理事故历史缺失值删除有缺失值的行，删除训练集和测试集中的 clean_title 列。

处理字段engine排量信息

```
# 定义函数，用于提取符合 "X.XL" 格式的排量信息
def extract_displacement(engine_info):
    # 匹配标准格式的排量
    match = re.search(r'(\d+\.\d+)L', str(engine_info))
    if match:
        # 提取数值部分并转化为浮点数
        return float(match.group(1))
        # 若没有匹配到排量信息，则返回 None
    return None


# 应用函数提取排量信息，生成新的 'displacement' 列
train_df['displacement'] = train_df['engine'].apply(extract_displacement)
test_df['displacement'] = test_df['engine'].apply(extract_displacement)
```

```
# 统计训练集和测试集中 'displacement' 列的缺失值数量
print(f'训练集displacement缺失值:{train_df['displacement'].isnull().sum()}')
print(f'测试集displacement缺失值:{test_df['displacement'].isnull().sum()}')
```

```
训练集displacement缺失值:14078
测试集displacement缺失值:9334
```

```
# 删除displacement缺失值
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)
```

说明：规划 engine 字段的数据格式，并删除缺失值。

3.2.4 划分数据等级

```
# 定义函数，根据排量划分等级
def categorize_displacement(displacement):
    # 对电动车直接返回 Electric
    if displacement == "Electric":
        return "Electric"
    # 判断数值的区间，并返回相应的类别
    elif displacement <= 1.0:
        return "Small"
    elif 1.0 < displacement <= 1.6:
        return "Medium"
    elif 1.6 < displacement <= 2.5:
        return "Large"
    elif 2.5 < displacement <= 4.0:
        return "Extra-large"
    else:
        return "Ultra-large"


# 应用划分函数到训练集和测试集
train_df['displacement'] = train_df['displacement'].apply(categorize_displacement)
test_df['displacement'] = test_df['displacement'].apply(categorize_displacement)
```

```
# 删除原来训练集和测试集中的 engine 列
train_df.drop(columns=['engine'], inplace=True)
test_df.drop(columns=['engine'], inplace=True)
```

说明：将字段 displacement 划分成 Electric、Small、Medium、Large、Extra-large、Ultra-large。

处理字段transmission变速器类型

```
# 定义函数，将变速器类型分为 'Automatic'、'Manual' 和 'Other'
def categorize_transmission(transmission):
    transmission = str(transmission).lower()
    if any(keyword in transmission for keyword in ["automatic", "a/t", "cvt", "speed"]):
        return "Automatic"
    elif any(keyword in transmission for keyword in ["manual", "m/t"]):
        return "Manual"
    else:
        return "Other"


# 应用函数到训练集和测试集
train_df['transmission'] = train_df['transmission'].apply(categorize_transmission)
test_df['transmission'] = test_df['transmission'].apply(categorize_transmission)
```

说明：将字段 transmission 划分成 Automatic、Manual、Other，并赋值到原来字段。

3.2.5 数据预处理后数据

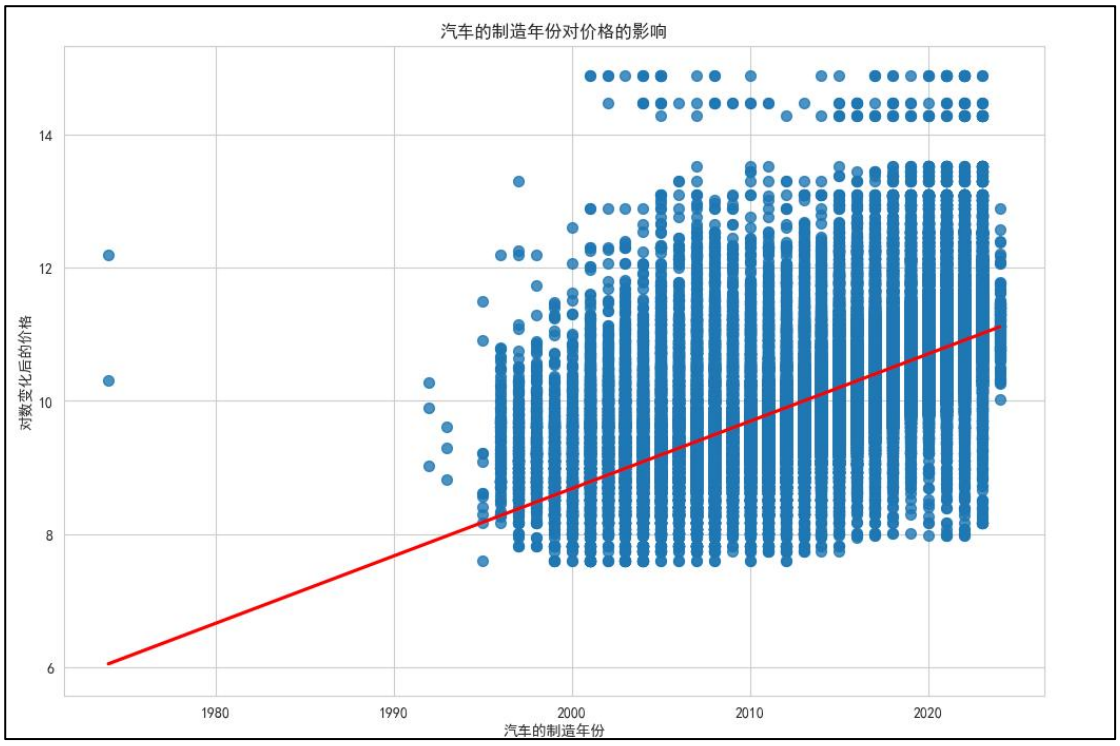
```
train_df.head()
```

	id	brand	model	model_year	milage	fuel_type	transmission	accident	price	displacement
0	0	MINI	Cooper S Base	2007	213000	Gasoline	Automatic	None reported	4200	Medium
1	1	Lincoln	LS V8	2002	143250	Gasoline	Automatic	At least 1 accident or damage reported	4999	Extra-large
2	2	Chevrolet	Silverado 2500 LT	2002	136731	E85 Flex Fuel	Automatic	None reported	13900	Ultra-large
3	3	Genesis	G90 5.0 Ultimate	2017	19500	Gasoline	Other	None reported	45000	Ultra-large
4	4	Mercedes-Benz	Metris Base	2021	7388	Gasoline	Automatic	None reported	97500	Large

3.3 数据可视化

3.3.1 汽车的制造年份与价格的关系图

```
# 绘制汽车的制造年份与价格的关系图
plt.figure(figsize=(12, 8))
sns.regplot(x='model_year', y='log_price', data=train_data,
            scatter_kws={'s': 50}, line_kws={'color': 'red'})
plt.title("汽车的制造年份对价格的影响")
plt.xlabel("汽车的制造年份")
plt.ylabel("对数变化后的价格")
plt.show()
```

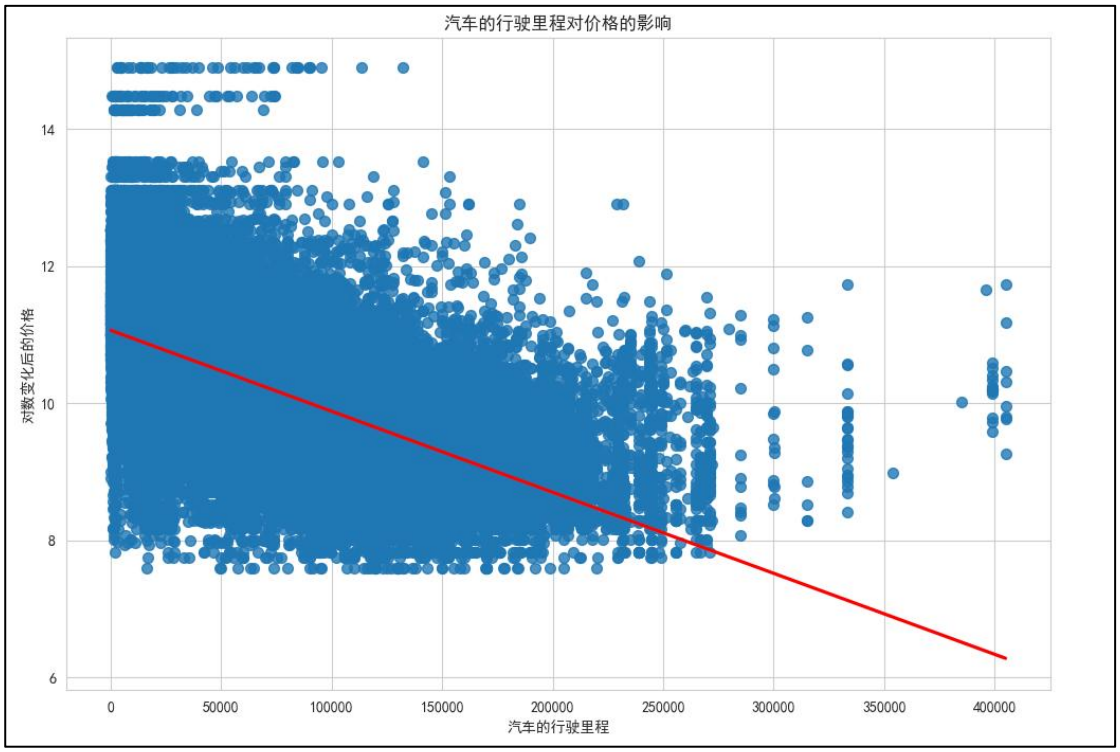


说明：红线显示了一个明显的上升趋势，表明随着制造年份越新，汽车的平均价格呈现稳定增长。对于每个年份，价格都呈现出较大的垂直分布范围，表明同一

年份的车型存在显著的价格差异，这种价格差异在 2000 年以后变得更加明显，分布范围更广。

3.3.2 汽车的行驶里程与价格的关系图

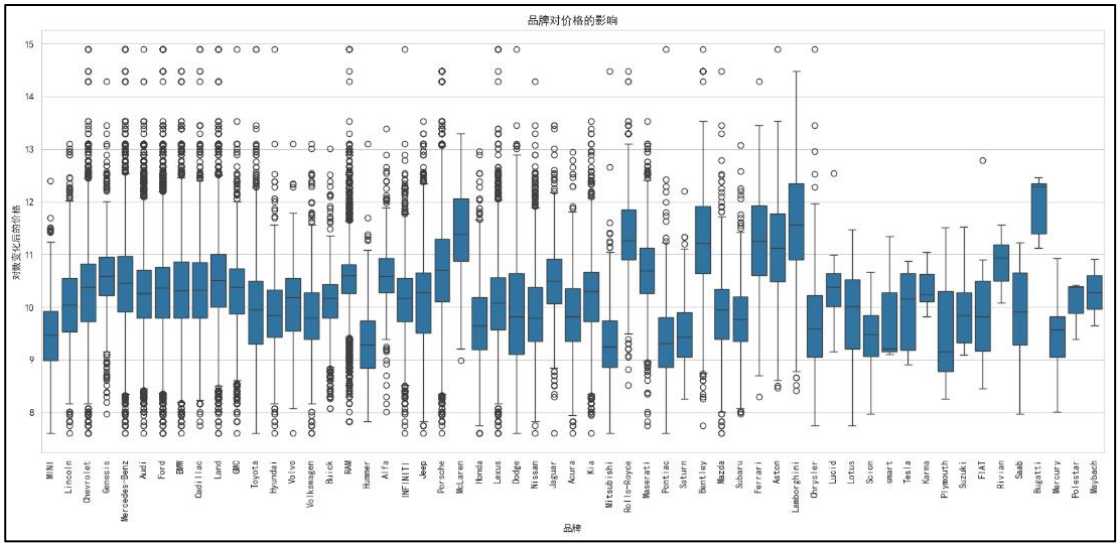
```
# 绘制汽车的行驶里程与价格的关系图
plt.figure(figsize=(12, 8))
sns.regplot(x='milage', y='log_price', data=train_data,
            scatter_kws={'s': 50}, line_kws={'color': 'red'})
plt.title("汽车的行驶里程对价格的影响")
plt.xlabel("汽车的行驶里程")
plt.ylabel("对数变化后的价格")
plt.show()
```



说明：红线清晰地显示了一个下降趋势，表明随着行驶里程的增加，汽车价格整体呈下降趋势。

3.3.3 品牌对价格影响的关系图

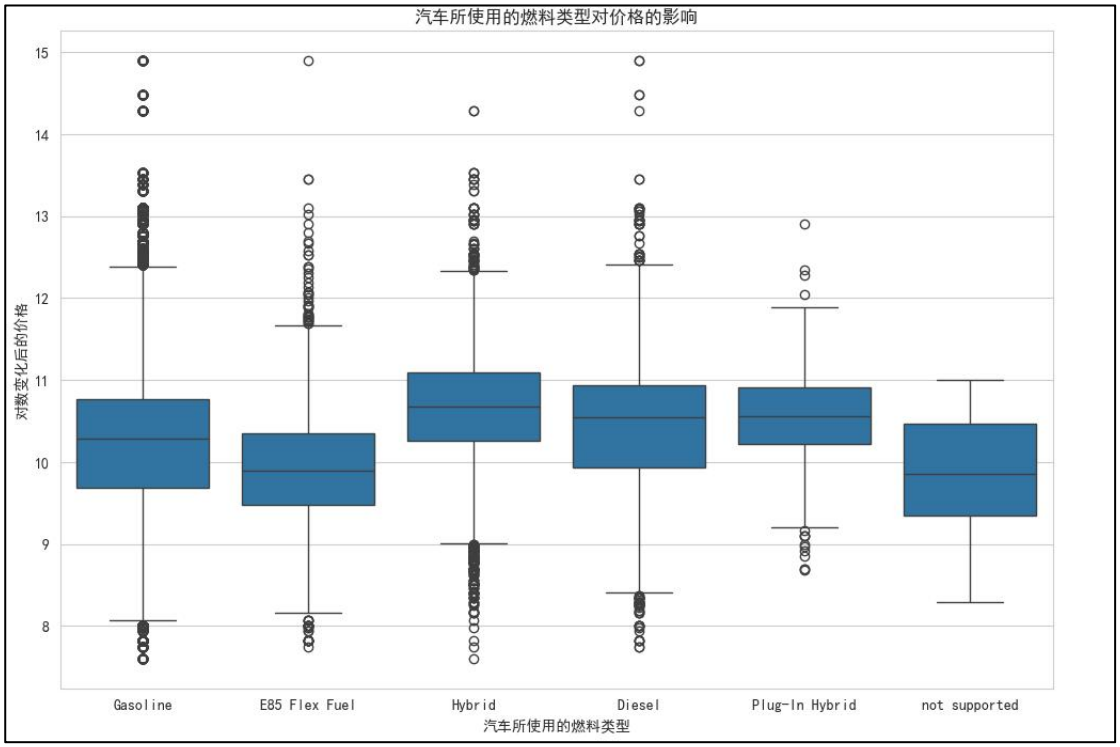
```
# 品牌对价格的影响
plt.figure(figsize=(20, 8))
sns.boxplot(data=train_data, x='brand', y='log_price')
plt.title("品牌对价格的影响")
plt.xticks(rotation=90)
plt.xlabel("品牌")
plt.ylabel("对数变化后的价格")
plt.show()
```



说明：Bugatti（布加迪）的价格中位数最高，并且价格分布比较窄，总体价格较高。Lamborghini（兰博基尼）的价格中位数第二高，价格分布比较广，也是豪华品牌代表。Hummer（悍马）、Mitsubishi（三菱）、Pontiac（庞蒂亚克）、Plymouth（普利茅斯）的价格中位数相对较低。可以看出，不同的品牌价格不同，因此品牌可能是影响汽车价格的因素之一。

3.3.4 汽车所使用的燃料类型和价格的关系图

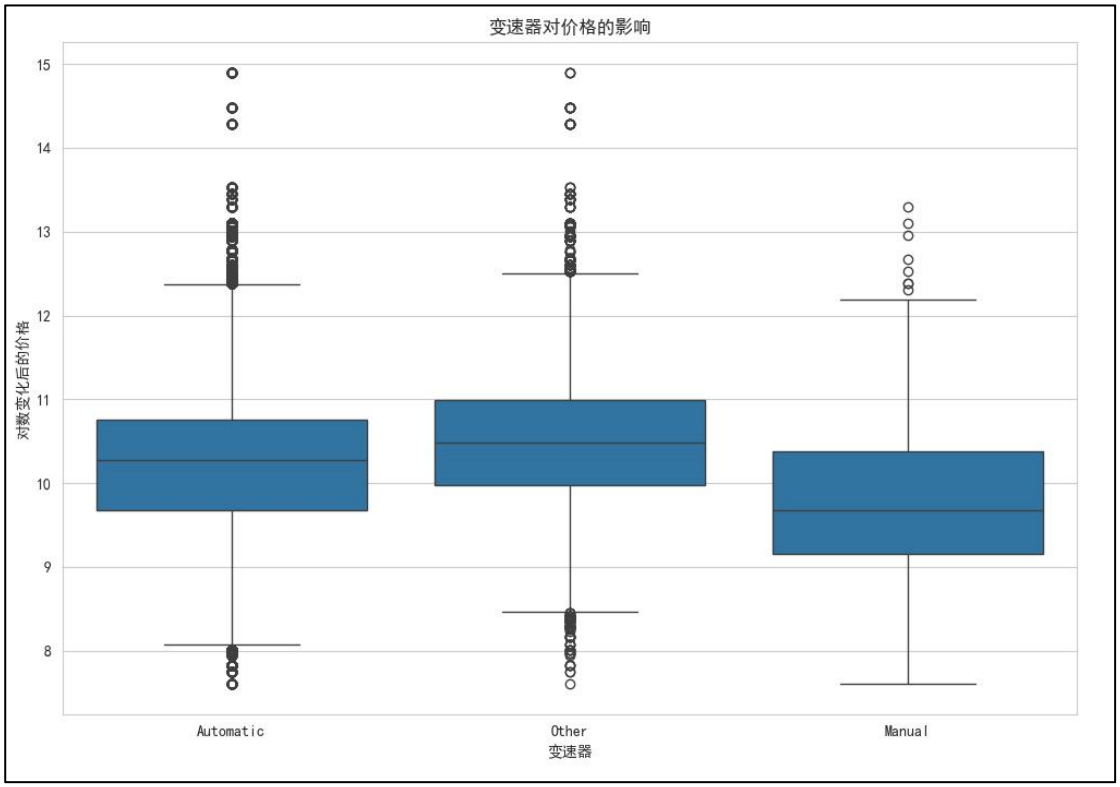
```
# 绘制汽车所使用的燃料类型和价格的关系图
plt.figure(figsize=(12,8))
sns.boxplot(data=train_data,x='fuel_type', y='log_price')
plt.title('汽车所使用的燃料类型对价格的影响')
plt.xlabel('汽车所使用的燃料类型')
plt.ylabel('对数变化后的价格')
plt.show()
```



说明：Hybrid（混合动力）和 Plug-In Hybrid（插电式混合动力）的二手车中位数价格高于其他类型。not supported（不支持）类别价格分布最窄，中位数价格最低，可能代表一些特殊或老旧车型。

3.3.5 变速器和价格的关系图

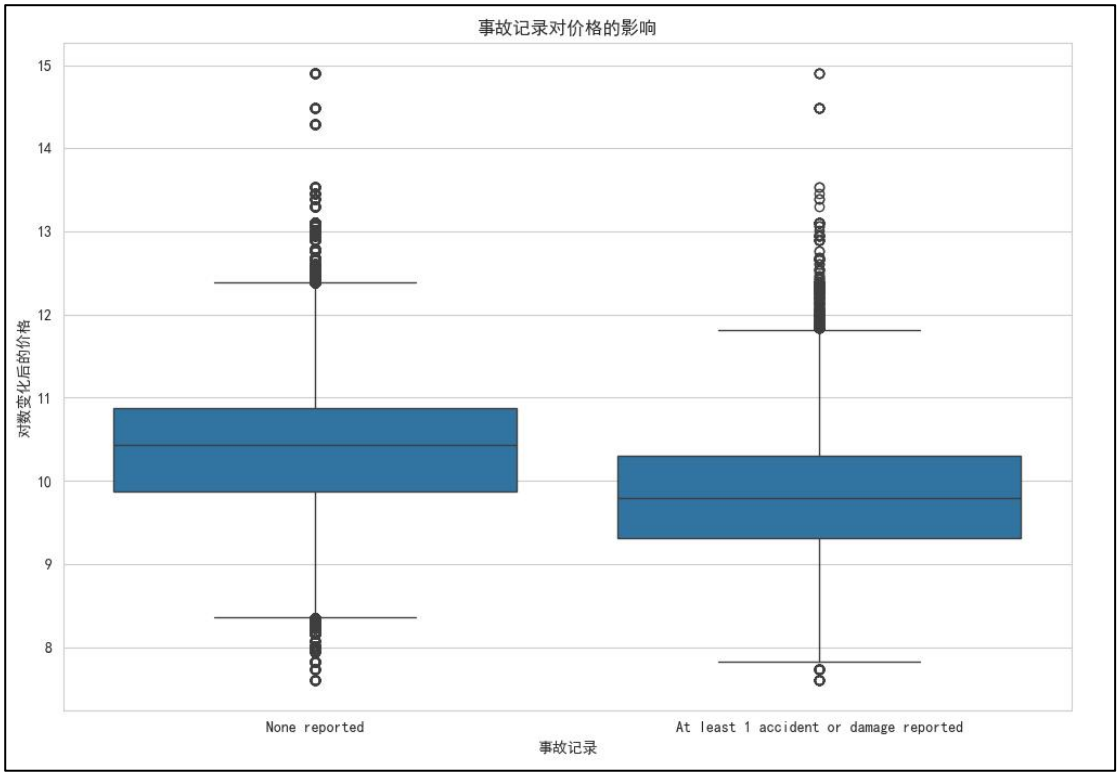
```
# 绘制变速器和价格的关系图
plt.figure(figsize=(12, 8))
sns.boxplot(data=train_data, x='transmission', y='log_price')
plt.title("变速器对价格的影响")
plt.xlabel("变速器")
plt.ylabel("对数变化后的价格")
plt.show()
```



说明：其他类型变速器的中位数价格最高，其次是自动档，手动档最低。

3.3.6 事故记录和价格的关系图

```
# 绘制事故记录和价格的关系图
plt.figure(figsize=(12, 8))
sns.boxplot(data=train_data, x='accident', y='log_price')
plt.title("事故记录对价格的影响")
plt.xlabel("事故记录")
plt.ylabel("对数变化后的价格")
plt.show()
```



说明：没发生过事故的二手车价格更高，发生过事故或者损坏的二手车价格就比较低。

3.4 模型训练

3.4.1 数据预处理

数据预处理

```
model_train_data = train_df.copy()
model_test_data = test_df.copy()
```

```
# 增加新特征 车龄current_year
current_year = datetime.now().year
model_train_data['car_age'] = current_year - model_train_data['model_year']
model_test_data['car_age'] = current_year - model_test_data['model_year']
# 删除原来的生产列model_year
model_train_data.drop('model_year', inplace=True, axis=1)
model_test_data.drop('model_year', inplace=True, axis=1)
# 删除 dataset列
model_train_data.drop('dataset', inplace=True, axis=1)
model_test_data.drop('dataset', inplace=True, axis=1)
model_train_data
```

	id	brand	model	milage	fuel_type	transmission	accident	price	displacement	log_price	car_age
0	0	MINI	Cooper S Base	213000	Gasoline	Automatic	None reported	4200	Medium	8.343078	17
1	1	Lincoln	LS V8	143250	Gasoline	Automatic	At least 1 accident or damage reported	4999	Extra-large	8.517193	22
2	2	Chevrolet	Silverado 2500 LT	136731	E85 Flex Fuel	Automatic	None reported	13900	Ultra-large	9.539716	22
3	3	Genesis	G90 5.0 Ultimate	19500	Gasoline	Other	None reported	45000	Ultra-large	10.714440	7
4	4	Mercedes-Benz	Metris Base	7388	Gasoline	Automatic	None reported	97500	Large	11.487618	3
...	...	...	...	...	...	...	...	...	...	...	...
188528	188528	Cadillac	Escalade ESV Platinum	49000	Gasoline	Other	None reported	27500	Ultra-large	10.221978	7
188529	188529	Mercedes-Benz	AMG C 43 AMG C 43 4MATIC	28600	Gasoline	Automatic	At least 1 accident or damage reported	30000	Extra-large	10.308986	6
188530	188530	Mercedes-Benz	AMG GLC 63 Base 4MATIC	13650	Gasoline	Automatic	None reported	86900	Extra-large	11.372525	3
188531	188531	Audi	S5 3.0T Prestige	13895	Gasoline	Automatic	None reported	84900	Extra-large	11.349241	2
188532	188532	Porsche	Macan Base	59500	Gasoline	Other	None reported	28995	Large	10.274913	8

172003 rows × 11 columns

说明：把汽车的制造年份，转为车龄：当前年份-汽车的制造年份，这样可以使模型更直接地学习车辆新旧程度对价格的影响。

```
# 定义需要编码的类别特征
categorical_features = model_train_data.select_dtypes(include=['object']).columns.tolist()

# 创建一个字典来保存每个特征的 LabelEncoder
label_encoders = {}
# 对 new_train_data 进行编码并保存编码器
for col in categorical_features:
    le = LabelEncoder()
    col1=col+'_LE'
    model_train_data[col1] = le.fit_transform(model_train_data[col].astype(str))
    label_encoders[col] = le  # 保存编码器，供测试数据使用

# 使用相同的编码器对 new_test_data 进行编码
for col in categorical_features:
    if col in model_test_data.columns:  # 确保测试数据包含该列
        col1=col+'_LE'
        le = label_encoders[col]
        # 处理测试数据中出现的新类别（即训练数据中没有见过的类别）
        model_test_data[col1] = model_test_data[col].map(lambda s: le.transform([s])[0] if s in le.classes_ else -1)
```

model_train_data

	id	brand	model	milage	fuel_type	transmission	accident	price	displacement	log_price	car_age	brand_LE	model_LE	fuel_type_LE
0	0	MINI	Cooper S Base	213000	Gasoline	Automatic	None reported	4200	Medium	8.343078	17	31	492	2
1	1	Lincoln	LS V8	143250	Gasoline	Automatic	At least 1 accident or damage reported	4999	Extra-large	8.517193	22	28	922	2
2	2	Chevrolet	Silverado 2500 LT	136731	E85 Flex Fuel	Automatic	None reported	13900	Ultra-large	9.539716	22	9	1543	1
3	3	Genesis	G90 5.0 Ultimate	19500	Gasoline	Other	None reported	45000	Ultra-large	10.714440	7	16	753	2
4	4	Mercedes-Benz	Metris Base	7388	Gasoline	Automatic	None reported	97500	Large	11.487618	3	36	1068	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
188528	188528	Cadillac	Escalade ESV Platinum	49000	Gasoline	Other	None reported	27500	Ultra-large	10.221978	7	8	599	2
188529	188529	Mercedes-Benz	AMG C 43 AMG C 43 4MATIC	28600	Gasoline	Automatic	At least 1 accident or damage reported	30000	Extra-large	10.308986	6	36	205	2

说明：针对分类特征，使用标签编码。

```python
from sklearn.preprocessing import StandardScaler
# 特征标准化
# 数值型特征（车龄和对数化的里程数）
numerical_features = ['car_age', 'milage']
# 对数值型特征进行标准化
scaler = StandardScaler()
model_train_data[numerical_features] = scaler.fit_transform(model_train_data[numerical_features])
model_train_data[numerical_features] = scaler.transform(model_train_data[numerical_features])
```

代码 Markdown

```python
x = model_train_data[['car_age', 'brand_LE', 'model_LE',
                      'fuel_type_LE', 'transmission_LE', 'accident_LE', 'displacement_LE']]
y = model_train_data['price']
# 分割数据
x_train, x_test, y_train, y_test,  = train_test_split(x, y, test_size=0.2, random_state=15)
```

✓ [55] 31毫秒

说明：数值型特征（车龄和里程数）并拆分数据集。

3.4.2 岭回归模型

```python
from sklearn.linear_model import RidgeCV     RidgeCV
from sklearn.linear_model import Ridge
```

✓ [97] < 10 毫秒

```python
alpha_range = np.logspace(-3, 3, 10)
ridge_cv = RidgeCV(alphas=alpha_range, scoring='r2', cv=5)  # 5折
# 对原始价格进行训练
ridge_cv.fit(x_train, y_train)
# 输出最佳alpha值
best_alpha = ridge_cv.alpha_
print("最佳 alpha:", best_alpha)
```

✓ [98] 964毫秒

```
最佳 alpha: 0.46415888336127775
```

```python
# 使用最佳 alpha 创建 Ridge 模型
ridge_model_original = Ridge(alpha=best_alpha, random_state=15)
ridge_model_original.fit(x_train, y_train)
```

✓ [99] 20毫秒

```
Ridge(alpha=np.float64(0.46415888336127775), random_state=15)
```

```
y_pred_original = ridge_model_original.predict(x_test)
mse_original = mean_squared_error(y_test_original, y_pred_original)
mae_original = mean_absolute_error(y_test_original, y_pred_original)
rmse_original = np.sqrt(mse_original)
r2_original = r2_score(y_test_original, y_pred_original)

print("直接预测原始价格的性能:")
print(f"MSE: {mse_original:.2f}")
print(f"MAE: {mae_original:.2f}")
print(f"RMSE: {rmse_original:.2f}")
print(f"R2 Score: {r2_original:.4f}")
✓ [41] 12毫秒

直接预测原始价格的性能:
MSE: 4691705706.57
MAE: 24240.54
RMSE: 68496.03
R2 Score: 0.0752
```

说明：模型准确度得分 0.0752。

3.4.2 随机森林模型

```
# 网格搜索法
# 定义参数网格
param_grid = {
    'n_estimators': [5, 10, 50, 100, 200],  # 决策树的数量
    'max_depth': [None, 10, 15, 20],  # 树的最大深度
    'min_samples_split': [2, 5, 10]  # 节点划分最少样本数
}

# 初始化随机森林回归器
rf_model = RandomForestRegressor(random_state=15, n_jobs=-1)
# 初始化网格搜索
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
# 执行网格搜索
grid_search.fit(x_train, y_train)
# 输出最佳参数
print("最佳参数:", grid_search.best_params_)
✓ [94] 4分钟 24秒

Fitting 3 folds for each of 60 candidates, totalling 180 fits
最佳参数: {'max_depth': 15, 'min_samples_split': 10, 'n_estimators': 200}

rf_model = RandomForestRegressor(random_state=15, n_jobs=-1,
                                 max_depth=15, min_samples_split=10, n_estimators=200)
rf_model.fit(x_train, y_train)
```

```
y_pred = rf_model.predict(x_test)

# 计算对数域中的性能指标
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("\n对数域中的性能:")
print(f"MSE : {mse:.5f}")
print(f"MAE : {mae:.5f}")
print(f"RMSE : {rmse:.5f}")
print(f"R2 Score : {r2:.5f}")
✓ [96] 176毫秒
```

```
对数域中的性能:
对数域中的性能:
MSE (log): 0.29056
MAE (log): 0.38142
RMSE (log): 0.53904
R2 Score (log): 0.59243
```

说明：模型准确度得分 0.59。

3.4.3 预测测试集数据价格

```
x_test_new = model_test_data[
    ['car_age', 'brand_LE', 'model_LE', 'fuel_type_LE', 'transmission_LE', 'accident_LE', 'displacement_LE']]
# 使用模型预测对数价格
y_pred_new = rf_model.predict(x_test_new)
# 将预测的对数价格转换回原始价格
y_pred_new = np.exp(y_pred_new)
y_pred_new = np.round(y_pred_new).astype(int)
# 将预测结果添加到原始数据框中
test_df['rf_predicted_price'] = y_pred_new
test_df.head()
✓ [101] 231毫秒
```

5 行 × 10 列

	id	brand	model	model_year	milage	fuel_type	transmission
0	188533	Land	Rover LR2 Base	2015	98000	Gasoline	Automatic
1	188534	Land	Rover Defender SE	2020	9142	Hybrid	Automatic
2	188535	Ford	Expedition Limited	2022	28121	Gasoline	Automatic
4	188537	Audi	A6 2.0T Premium Plus	2018	59000	Gasoline	Automatic
5	188538	Chevrolet	Express 2500 Work Van	2013	99524	Gasoline	Automatic

说明：将预测的数据添加到新的一列中。

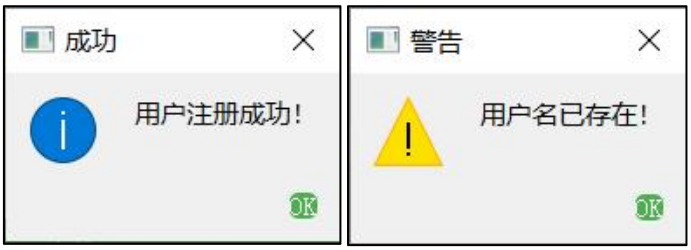
3.5 UI 界面展示

3.5.1 主界面


```
class MainWindow(QWidget):  1个用法
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):  1个用法
        self.setWindowTitle('二手车价格预测平台')
        self.setGeometry(400, 200, 250, 200)
        self.setFixedSize(1000, 650)
        self.setStyleSheet(
            "QWidget {  border: 1px solid #888888; border-radius: 10px; }")
        # 加载图片
        pixmap = QPixmap('image/main_window.png')
        # 缩小图片到指定尺寸，同时保持宽高比
        scaled_pixmap = pixmap.scaled(600, 600, Qt.KeepAspectRatio, Qt.SmoothTransformation)
        # 创建QLabel来显示缩小后的图片
        label = QLabel(self)
        label.setPixmap(scaled_pixmap)
        label.resize(600, 600)  # 设置QLabel的大小，以适应缩小后的图片
        label.move(350, 25)  # 设置QLabel的位置
        self.show()
        layout = QVBoxLayout()
        self.register_button = QPushButton('注册账号')
        self.register_button.setFixedSize(300, 50)
        self.register_button.clicked.connect(self.open_register)
        layout.addWidget(self.register_button)
        self.register_button.move(500, 20)
        self.login_button = QPushButton('登录账号')
        self.login_button.setFixedSize(300, 50)
        self.login_button.clicked.connect(self.open_login)
        layout.addWidget(self.login_button)
        self.setLayout(layout)
```

```
def open_register(self):  1个用法
    self.register_window = RegisterWindow()
    self.register_window.show()

def open_login(self):  1个用法
    self.login_window = LoginWindow()
    self.login_window.show()
```

3.5.2 注册界面



```
class RegisterWindow(QWidget):  1个用法
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):  1个用法
        self.setWindowTitle('注册界面')
        self.setFixedSize(500, 400)  # 只需要设置一次固定大小
        self.setGeometry(620, 350, 250, 200)

        # 设置窗口的样式表，背景为纯白色，边框为圆角
        self.setStyleSheet("QWidget {border-radius: 10px; }"
                           "QLineEdit { border: 1px solid #cccccc; padding: 2px; }"  # 为输入框添加样式
                           "QPushButton { background-color: #4CAF50; color: white; border-radius: 5px; }"
                           "QPushButton:hover { background-color: #45a049; }")  # 为按钮添加悬停样式
        layout = QVBoxLayout()
        # 创建用户名的标签和输入框，并使用水平布局管理器
        username_layout = QHBoxLayout()
        self.label_username = QLabel('用户名:')
        username_layout.addWidget(self.label_username)
        self.entry_username = QLineEdit()
        username_layout.addWidget(self.entry_username)

        # 创建密码的标签和输入框，并使用水平布局管理器
        password_layout = QHBoxLayout()
        self.label_password = QLabel('密码:')
        password_layout.addWidget(self.label_password)
        self.entry_password = QLineEdit()
        self.entry_password.setEchoMode(QLineEdit.Password)
        password_layout.addWidget(self.entry_password)
```

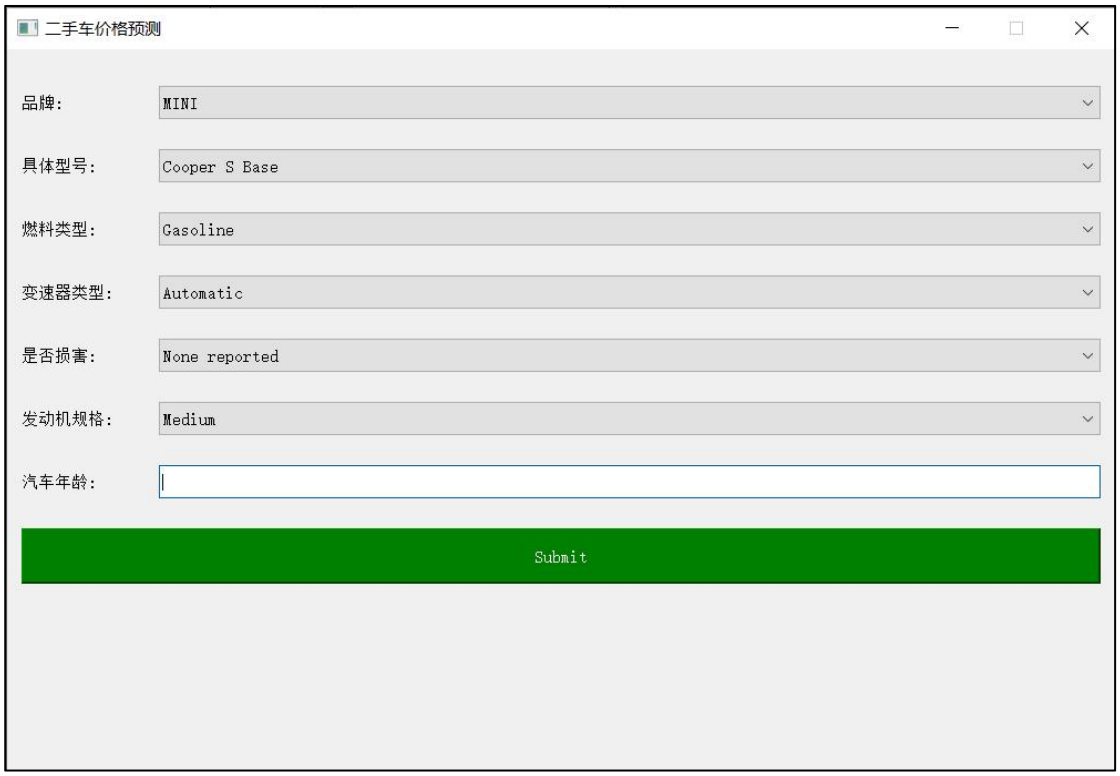
```
        # 将水平布局添加到垂直布局中
        layout.addLayout(username_layout)
        layout.addLayout(password_layout)

        # 注册按钮
        self.register_button = QPushButton('注册')
        self.register_button.clicked.connect(self.register)
        self.register_button.setFixedSize(473, 40)
        layout.addWidget(self.register_button)

        self.setLayout(layout)

    def register(self):  1个用法
        username = self.entry_username.text()
        password = self.entry_password.text()
        # 设置窗口的样式表，背景为纯白色，边框为圆角
        self.setStyleSheet("QWidget {border-radius: 10px; }"
                           "QLineEdit { border: 1px solid #cccccc; padding: 2px; }"  # 为输入框添加样式
                           "QPushButton { background-color: #4CAF50; color: white; border-radius: 5px; }"
                           "QPushButton:hover { background-color: #45a049; }")  # 为按钮添加悬停样式

        if username in users:
            QMessageBox.warning(self, '警告', '用户名已存在!')
        else:
            users[username] = password
            self.close()
            QMessageBox.information(self, '成功', '用户注册成功!')
            # 注册成功后，可以打开登录窗口
            LoginWindow()
```

3.5.3 登录界面


```python
class LoginWindow(QWidget):  2 用法
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):  1 个用法
        self.setWindowTitle('登录界面')
        self.setFixedSize(500, 400)  # 只需要设置一次固定大小
        self.setGeometry(620, 350, 250, 200)

        # 设置窗口的样式表，背景为纯白色，边框为圆角
        self.setStyleSheet("QWidget {border-radius: 10px; }"
                           "QLineEdit { border: 1px solid #cccccc; padding: 2px; }"  # 为输入框添加样式
                           "QPushButton { background-color: #4CAF50; color: white; border-radius: 5px; }"
                           "QPushButton:hover { background-color: #45a049; }")  # 为按钮添加悬停样式

        layout = QVBoxLayout()

        # 创建用户名的标签和输入框，并使用水平布局管理器
        username_layout = QHBoxLayout()
        self.label_username = QLabel('用户名:')
        username_layout.addWidget(self.label_username)
        self.entry_username = QLineEdit()
        username_layout.addWidget(self.entry_username)

        # 创建密码的标签和输入框，并使用水平布局管理器
        password_layout = QHBoxLayout()
        self.label_password = QLabel('密码:')
        password_layout.addWidget(self.label_password)
        self.entry_password = QLineEdit()
        self.entry_password.setEchoMode(QLineEdit.Password)
        password_layout.addWidget(self.entry_password)
```

```
        # 将水平布局添加到垂直布局中
        layout.addLayout(username_layout)
        layout.addLayout(password_layout)

        # 登录按钮
        self.login_button = QPushButton('登录')
        self.login_button.clicked.connect(self.login)
        self.login_button.setFixedSize(473, 40)
        layout.addWidget(self.login_button)

        self.setLayout(layout)

    def login(self):  1个用法
        username = self.entry_username.text()
        password = self.entry_password.text()
        # 设置窗口的样式表，背景为纯白色，边框为圆角
        self.setStyleSheet("QWidget {border-radius: 10px; }"
                           "QLineEdit { border: 1px solid #cccccc; padding: 2px; }"  # 为输入框添加样式
                           "QPushButton { background-color: #4CAF50; color: white; border-radius: 5px; }"
                           "QPushButton:hover { background-color: #45a049; }")  # 为按钮添加悬停样式

        if username in users and users[username] == password:
            QMessageBox.information(self, '成功', '用户登录成功！')
            self.close()
            self.forecast_window = ForecastWindow()
            self.forecast_window.show()
        else:
            QMessageBox.warning(self, '错误', '用户名或密码错误！')
```

3.5.4 预测界面




```python
def create_RandomForestRegressor(brand, model, fuel_type, transmission, accident, displacement, car_age):  1个用法
    model_train_data = pd.read_csv('data/model_train_data.csv')
    x = model_train_data[['brand_LE', 'model_LE', 'fuel_type_LE', 'transmission_LE', 'accident_LE', 'displacement_LE', 'car_age']]
    y = model_train_data['price']
    # 分割数据
    x_train, x_test, y_train, y_test = train_test_split( *arrays: x, y, test_size=0.2, random_state=15)
    # 分割数据
    rf_model = RandomForestRegressor(random_state=15, n_jobs=-1,
                                     max_depth=15, min_samples_split=10, n_estimators=200)
    rf_model.fit(x_train, y_train)
    brand = brand
    model = model
    fuel_type = fuel_type
    transmission = transmission
    accident = accident
    displacement = displacement
    car_age = car_age
    brand_LE = model_train_data[model_train_data['brand'] == brand].brand_LE.values[0]
    model_LE = model_train_data[model_train_data['model'] == model].model_LE.values[0]
    fuel_type_LE = model_train_data[model_train_data['fuel_type'] == fuel_type].fuel_type_LE.values[0]
    transmission_LE = model_train_data[model_train_data['transmission'] == transmission].transmission_LE.values[0]
    accident_LE = model_train_data[model_train_data['accident'] == accident].accident_LE.values[0]
    displacement_LE = model_train_data[model_train_data['displacement'] == displacement].displacement_LE.values[0]
    pre_df = {'brand_LE': [brand_LE],'model_LE': [model_LE],
              'fuel_type_LE': [fuel_type_LE],
              'transmission_LE': [transmission_LE],
              'accident_LE': [accident_LE],
              'displacement_LE': [displacement_LE],
              'car_age': [car_age],
              }
    pre_df = pd.DataFrame(pre_df)
    y_pred_log = rf_model.predict(pre_df)
    return int(y_pred_log[0])
```

```python
class ForecastWindow(QWidget):  1个用法
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):  1个用法
        self.setWindowTitle('二手车价格预测')
        self.setGeometry(400, 200, 250, 200)
        self.setFixedSize(1000, 650)

        layout = QVBoxLayout()
        # brand_layout
        brand_layout = QHBoxLayout()
        self.label_brand = QLabel('品牌:')
        self.brand_LE = QComboBox()
        self.brand_LE.addItems([...])
        self.brand_LE.setFixedSize(850, 30)
        brand_layout.addWidget(self.label_brand)
        brand_layout.addWidget(self.brand_LE)
        # 将品牌选择区域的布局添加到主布局
        layout.addLayout(brand_layout)
        # model_layout
        model_layout = QHBoxLayout()
        self.label_model = QLabel('具体型号:')
        self.model_LE = QComboBox()
        self.model_LE.addItems([...])
        self.model_LE.setFixedSize(850, 30)
        model_layout.addWidget(self.label_model)
        model_layout.addWidget(self.model_LE)
        # 将具体类型选择区域的布局添加到主布局
        layout.addLayout(model_layout)
```

```python
        # fuel_type_layout
        fuel_type_layout = QHBoxLayout()
        self.label_fuel_type = QLabel('燃料类型:')
        self.fuel_type_LE = QComboBox()
        self.fuel_type_LE.addItems(['Gasoline', 'E85 Flex F
                                    'Diesel', 'Plug-In Hybr
        self.fuel_type_LE.setFixedSize(850, 30)
        fuel_type_layout.addWidget(self.label_fuel_type)
        fuel_type_layout.addWidget(self.fuel_type_LE)
        layout.addLayout(fuel_type_layout)

        # transmission_LE_layout
        transmission_LE_layout = QHBoxLayout()
        self.label_transmission = QLabel('变速器类型:')
        self.transmission_LE = QComboBox()
        self.transmission_LE.addItems(['Automatic', 'Other'
        self.transmission_LE.setFixedSize(850, 30)
        transmission_LE_layout.addWidget(self.label_transmi
        transmission_LE_layout.addWidget(self.transmission_
        layout.addLayout(transmission_LE_layout)

        # accident_LE_layout
        accident_LE_layout = QHBoxLayout()
        self.label_accident = QLabel('是否损害:')
        self.accident_LE = QComboBox()
        self.accident_LE.addItems(['None reported',
                                   'At least 1 accident or
        self.accident_LE.setFixedSize(850, 30)
        accident_LE_layout.addWidget(self.label_accident)
        accident_LE_layout.addWidget(self.accident_LE)
        layout.addLayout(accident_LE_layout)
```