

数据科学导论 -HW 2 报告

孙育泉 10234900421

2025.09.26

I 实验要求

在这次的实验中，我们需要对给定的房价的数据集进行如下操作：

1. 检测缺失值，并进行填充
2. 检测异常值
3. 对特征间的相关性进行分析
4. 标准化 price 属性
5. 根据 price 属性进行离散化
6. 找出和 price 属性相关性较高的属性并给出理由

II 具体实现

II.1 准备工作

首先导入一些我们需要使用的库，并读入数据

In [1]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
6 from sklearn.impute import KNNImputer
7
8 sns.set(style="whitegrid")
9 df = pd.read_csv("data/train.csv")
```

python

II.2 缺失值的检测与处理

首先，我们需要检测数据中的缺失值。

In [2]:

```
1 # 查看每列的缺失值数量
2 missing_values = df.isnull().sum()
3 print("各特征的缺失值数量：\n", missing_values[missing_values > 0])
```

python

```
1 各特征的缺失值数量：
2 | LotFrontage      259
3 Alley            1369
4 MasVnrType        872
5 MasVnrArea         8
6 BsmtQual          37
7 BsmtCond          37
8 BsmtExposure      38
9 BsmtFinType1       37
10 BsmtFinType2       38
11 Electrical         1
12 FireplaceQu       690
```

txt

```

13 GarageType      81
14 GarageYrBlt     81
15 GarageFinish    81
16 GarageQual      81
17 GarageCond      81
18 PoolQC          1453
19 Fence           1179
20 MiscFeature     1406
21 dtype: int64
22

```

接着，对于缺失值的处理，我们使用 KNNImputer 进行填充. 这样做的好处是能够利用数据中的其他信息来估计缺失值，从而提高数据的完整性和准确性。

In [3]:

```

1 # 假设我们只对数值型特征进行KNN填充
2 numeric_cols = df.select_dtypes(include=np.number).columns
3 imputer = KNNImputer(n_neighbors=5) # n_neighbors 可以调整
4
5 # 注意：KNNImputer 返回的是 numpy array，需要转换回 DataFrame
6 df[numeric_cols] = imputer.fit_transform(df[numeric_cols])
7
8 # 检查是否还有缺失值
9 print("填充后，数值型特征的缺失值数量：\n", df[numeric_cols].isnull().sum().sum())
10
11 print(df[0:0])

```

python

txt

```

1 填充后，数值型特征的缺失值数量：
2 0
3 Empty DataFrame
4 Columns: [Id, MSSubClass, MSZoning, LotFrontage, LotArea, Street, Alley, LotShape,
LandContour, Utilities, LotConfig, LandSlope, Neighborhood, Condition1, Condition2,
BldgType, HouseStyle, OverallQual, OverallCond, YearBuilt, YearRemodAdd, RoofStyle,
RoofMatl, Exterior1st, Exterior2nd, MasVnrType, MasVnrArea, ExterQual, ExterCond,
Foundation, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinSF1, BsmtFinType2,
BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, Heating, HeatingQC, CentralAir, Electrical,
1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, BsmtHalfBath, FullBath,
HalfBath, BedroomAbvGr, KitchenAbvGr, KitchenQual, TotRmsAbvGrd, Functional,
Fireplaces, FireplaceQu, GarageType, GarageYrBlt, GarageFinish, GarageCars,
GarageArea, GarageQual, GarageCond, PavedDrive, WoodDeckSF, OpenPorchSF,
EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, PoolQC, Fence, MiscFeature, MiscVal,
MoSold, YrSold, SaleType, SaleCondition, SalePrice]
5 Index: []
6
7 [0 rows x 81 columns]
8

```

II.3 异常值检测与处理

接下来，我们以 price 属性为例，使用 IQR 方法检测异常值. IQR 方法通过计算四分位数来识别异常值，能够有效地捕捉到数据中的极端值。

In [4]:

```

1 # 以 price 属性为例

```

python

```
2 Q1 = df['SalePrice'].quantile(0.25)
3 Q3 = df['SalePrice'].quantile(0.75)
4 IQR = Q3 - Q1
5 lower_bound = Q1 - 1.5 * IQR
6 upper_bound = Q3 + 1.5 * IQR
7
8 # 找出异常值
9 outliers = df[(df['SalePrice'] < lower_bound) | (df['SalePrice'] > upper_bound)]
10 print(f"在 'SalePrice' 特征中检测到 {len(outliers)} 个异常值")
```

txt

```
1 在 'SalePrice' 特征中检测到 61 个异常值
2
```

而对于这些异常值，我们选择将超出上下界限的值替换为边界值，这样可以避免异常值对后续分析产生过大的影响。

In [5]:

```
1 # 将 price 中超过上界的值替换为上界，低于下界的值替换为下界
2 df['SalePrice'] = np.where(df['SalePrice'] > upper_bound, upper_bound,
3 df['SalePrice'])
4 df['SalePrice'] = np.where(df['SalePrice'] < lower_bound, lower_bound,
5 df['SalePrice'])
6 print("异常值处理完成。")
```

python

txt

```
1 异常值处理完成.
2
```

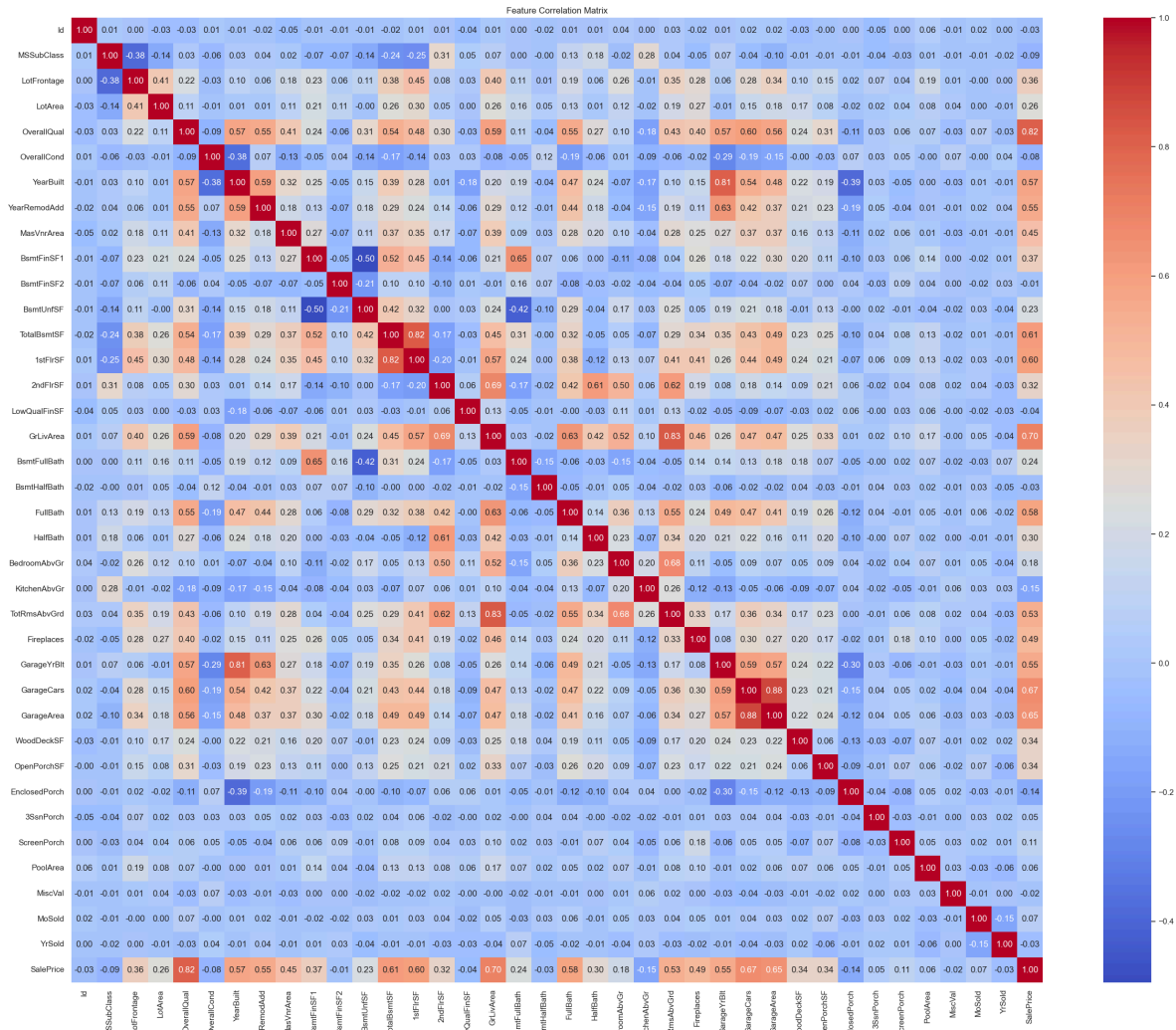
II.4 特征间的相关性分析

我们使用相关性矩阵和热力图来分析特征之间的线性关系。

In [6]:

```
1 # 计算数值型特征的相关性矩阵
2 correlation_matrix = df.select_dtypes(include=np.number).corr()
3
4 # 使用热力图进行可视化
5 plt.figure(figsize=(30, 24))
6 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
7 plt.title('Feature Correlation Matrix')
8 plt.show()
```

python



通过热力图，我们可以直观地看到各个特征之间的相关性强弱。

- 热力图中的颜色越接近 1（深红）或 -1（深蓝），表示两个特征的线性相关性越强。
- 接近 0 的值表示线性相关性很弱。
- 这有助于识别冗余特征（例如两个特征相关性 > 0.9 ）

II.5 对 price 属性进行标准化

标准化就是将数据按比例缩放，使其分布均值为 0，方差为 1。在这里，我们使用 StandardScaler 来对 price 属性进行标准化。

In [7]:

```
1 # 初始化标准化器
2 scaler = StandardScaler()
3
4 # 对 price 列进行标准化
5 # fit_transform 需要一个二维数组，所以我们用 [[]] 来 reshape
6 df['price_scaled'] = scaler.fit_transform(df[['SalePrice']])
7
8 print("SalePrice 标准化后的前5行: \n", df[['SalePrice', 'price_scaled']].head())
```

python

txt

```
1 SalePrice 标准化后的前5行 :
2 | SalePrice price_scaled
3 0 208500.0 0.463935
4 1 181500.0 0.062047
5 2 223500.0 0.687207
6 3 140000.0 -0.555671
7 4 250000.0 1.081653
8
```

II.6 对 price 属性进行离散化

离散化或分箱是将连续变量转换为分类变量的过程. 在这里, 我们使用等频分箱的方法将 price 属性分为 4 个类别.

In [8]:

```
1 # 将 price 分为4个等频的箱 (每个箱包含约25%的数据)
2 df['price_binned_quantile'] = pd.qcut(df['SalePrice'], q=4, labels=['Low', 'Medium',
3                               'High', 'Very High'])
4 print("price 离散化后的分布: \n", df['price_binned_quantile'].value_counts())
```

python

txt

```
1 price 离散化后的分布 :
2 | price_binned_quantile
3 Medium      367
4 High        366
5 Low         365
6 Very High   362
7 Name: count, dtype: int64
8
```

II.7 找出和 price 属性相关性较高的特征

对于相关性最高的特征, 我们可以直接从之前计算的相关性矩阵中获取这个信息.

In [9]:

```
1 # 从相关性矩阵中提取与 'price' 相关的系数
2 price_correlation = correlation_matrix['SalePrice'].sort_values(ascending=False)
3
4 # 'price' 与自身的相关性是1, 所以我们排除掉它, 取接下来的三个
5 top_3_features = price_correlation[1:4] # 排除第一个 (price本身)
6
7 print("与 'SalePrice' 相关性最高的三个特征是: \n", top_3_features)
```

python

txt

```
1 与 'SalePrice' 相关性最高的三个特征是 :
2 | OverallQual    0.816856
3 GrLivArea       0.699980
4 GarageCars      0.672293
5 Name: SalePrice, dtype: float64
6
```

对于这 3 个特征, 我们可以给出以下理由:

- **OverallQual** (整体质量): 这个特征直接反映了房屋的建筑质量和材料使用情况, 通常质量越高的房屋价格也越高, 因此与价格有较强的正相关关系.
- **GrLivArea** (地上居住面积): 这个特征表示房屋的实际居住面积, 面积越大的房屋通常价格也越高, 因此与价格有较强的正相关关系.
- **GarageCars** (车库容量): 这个特征表示车库可以容纳的车辆数量, 车库容量较大的房屋通常价格也较高, 因此与价格有较强的正相关关系.

Remark

完整代码见 `./hw2.ipynb` 文件.