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]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
from sklearn.impute import KNNImputer

sns.set(style="whitegrid")
df = pd.read_csv("data/train.csv")
```

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]:

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

```txt
各特征的缺失值数量：
 LotFrontage      259
Alley           1369
MasVnrType       872
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Electrical         1
FireplaceQu      690
```

```
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
GarageCond        81
PoolQC          1453
Fence           1179
MiscFeature     1406
dtype: int64

```

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]:

python

```python
# 假设我们只对数值型特征进行KNN填充
numeric_cols = df.select_dtypes(include=np.number).columns
imputer = KNNImputer(n_neighbors=5) # n_neighbors 可以调整

# 注意：KNNImputer 返回的是 numpy array，需要转换回 DataFrame
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

# 检查是否还有缺失值
print("填充后，数值型特征的缺失值数量：\n", df[numeric_cols].isnull().sum().sum())

print(df[0:0])
```

txt

```
填充后，数值型特征的缺失值数量：
 0
Empty DataFrame
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]
Index: []

[0 rows x 81 columns]

```

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]:

python

```python
# 以 price 属性为例
```

```python
Q1 = df['SalePrice'].quantile(0.25)
Q3 = df['SalePrice'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 找出异常值
outliers = df[(df['SalePrice'] < lower_bound) | (df['SalePrice'] > upper_bound)]
print(f"在 'SalePrice' 特征中检测到 {len(outliers)} 个异常值")
```

```txt
在 'SalePrice' 特征中检测到 61 个异常值

```

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]:

```python
# 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
df['SalePrice'] = np.where(df['SalePrice'] > upper_bound, upper_bound,
df['SalePrice'])
df['SalePrice'] = np.where(df['SalePrice'] < lower_bound, lower_bound,
df['SalePrice'])
print("异常值处理完成. ")
```

```txt
异常值处理完成.

```

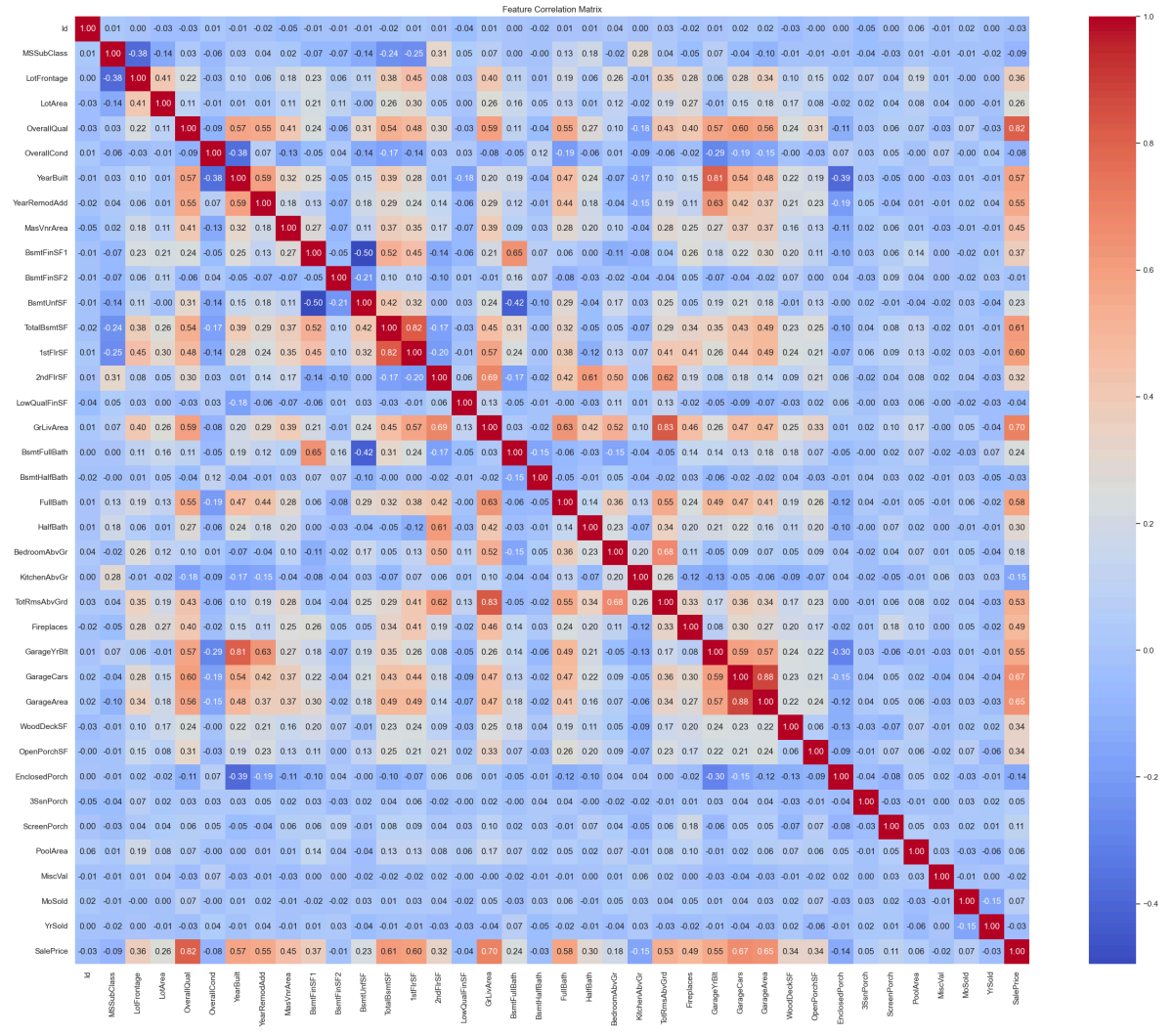
II.4 特征间的相关性分析

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]:

```python
# 计算数值型特征的相关性矩阵
correlation_matrix = df.select_dtypes(include=np.number).corr()

# 使用热力图进行可视化
plt.figure(figsize=(30, 24))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation Matrix')
plt.show()
```



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，从而为后续的特征选择和建模提供依据.

- 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]:

```python
# 初始化标准化器
scaler = StandardScaler()

# 对 price 列进行标准化
# fit_transform 需要一个二维数组，所以我们用 [[]] 来 reshape
df['price_scaled'] = scaler.fit_transform(df[['SalePrice']])

print("SalePrice 标准化后的前5行：\n", df[['SalePrice', 'price_scaled']].head())
```

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

```

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]:

```python
# 将 price 分为4个等频的箱（每个箱包含约25%的数据）
df['price_binned_quantile'] = pd.qcut(df['SalePrice'], q=4, labels=['Low', 'Medium', 'High', 'Very High'])

print("price 离散化后的分布：\n", df['price_binned_quantile'].value_counts())
```

```txt
price 离散化后的分布：
 price_binned_quantile
Medium       367
High         366
Low          365
Very High    362
Name: count, dtype: int64

```

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]:

```python
# 从相关性矩阵中提取与 'price' 相关的系数
price_correlation = correlation_matrix['SalePrice'].sort_values(ascending=False)

# '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
top_3_features = price_correlation[1:4] # 排除第一个（price本身）

print("与 'SalePrice' 相关性最高的三个特征是：\n", top_3_features)
```

```txt
与 'SalePrice' 相关性最高的三个特征是：
 OverallQual    0.816856
GrLivArea      0.699980
GarageCars     0.672293
Name: SalePrice, dtype: float64

```

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.