# 房地产价格预测模型验证报告

## 作者及分工

- **郭立为**: 负责模型性能评估，包括超参数选择问题、特征选择问题和后处理问题等方面的分析
- **梁艺高**: 负责分析缺失值处理、异常值处理和分箱编码方面的问题，提供了关于公摊比填充、面积异常处理和建筑面积区间划分的建议
- **徐子禾**: 负责代码规范性评估，包括命名规范、注释规范、代码模块化、依赖管理和运行环境说明等方面的问题分析
- **曹馨元**: 负责数据泄露问题的评估，包括代码运行检查、数据处理评估等

## 1. 简介

房地产价格预测一直是机器学习在金融领域应用的重要场景，其准确性对投资决策、风险管理和市场分析有重要作用。本报告针对团队1开发的房地产价格预测模型进行全面验证，评估其科学性和可靠性。

根据美联储SR 11-7指引，模型验证是"一系列旨在验证模型按预期执行、符合其设计目标和业务用途的过程和活动"。有效的验证有助于确保模型的健全性，并识别潜在的局限性和假设，评估其可能的影响。本次验证工作遵循了标准化的验证流程，包括：1）验证模型输入和输出的质量；2）检查模型输入的稳定性和代表性；3）验证模型实现的技术正确性；4）分析模型性能；5）与基准模型进行比较分析；6）评估模型的稳定性和校准过程的稳健性。

本报告将首先评估代码可运行性，然后深入分析数据处理方法的合理性，重点关注样本划分的科学性和潜在的数据泄露问题，最后提出具体的改进建议。

# 2. 代码运行检查

## 2.1 代码是否能顺利运行

这里运用了绝对路径，并不是标准做法，相对路径会更加方便对方运行代码：

```
# 加载训练集数据
train_data_path = r"C:\Users\21517\Desktop\期中考试
\input\quant4533\train_data.csv"
train_df = pd.read_csv(train_data_path)
```

## 2.2 代码规范性问题

### 2.2.1 命名规范

代码中的变量主要使用的是中文命名，但有一些变量使用了中英文混合的命名方式，建议统一命名风格，避免中英文混用。

如建筑面积data、套内面积data，与总体风格不符合，建议修改为"建筑面积数值"和"套内面积数值"，从而提高团队协作与维护的一致性。

### 2.2.2 代码注释

代码中已有一些基础的注释，如"去除建筑面积数据中的单位"等。但对于一些关键操作，缺乏足够的解释，例如为何选择某种模型、模型参数的意义等。

```
#使用KNeighborsClassifier填充缺失的"环线"变量
known_ringline = train_df[train_df['环线'].notna()]
unknown_ringline = train_df[train_df['环线'].isna()]
train_df['lon'] = pd.to_numeric(train_df['lon'], errors='coerce')
train_df['lat'] = pd.to_numeric(train_df['lat'], errors='coerce')
X_known = known_ringline[['lon', 'lat']]
y_known = known_ringline['环线']
X_unknown = unknown_ringline[['lon', 'lat']]
knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(X_known, y_known)
predicted_ringlines = knn_clf.predict(X_unknown)
train_df.loc[unknown_ringline.index, '环线'] = predicted_ringlines
```

这部分代码采用了 KNN 填充缺失值的方式，使用了经纬度信息来预测缺失的"环线"变量。这个操作可以通过增加注释来解释为何使用 KNN，以及选择 n_neighbors=3 的理由，可补充如下注释：

```
# 使用经纬度作为特征，利用KNN对"环线"字段进行缺失值填补
# 选择K=3是基于小样本实验下精度与速度的折中
knn_clf = KNeighborsClassifier(n_neighbors=3)
```

### 2.2.3 代码模块化

代码中随着预测方法变化而重复出现数据处理、特征工程、缺失值处理等操作，可维护性较低，如果后续需要修改某个处理步骤，需要修改多处重复的代码，较为麻烦且容易产生错误，建议将这些操作封装为函数，这样可以使得代码更加模块化，可读性更强，减少冗余代码。这样每次处理 train_df 或 test_df 时，可以调用同一个函数来完成相同的操作。

```
# 结合建筑面积计算总房价
test_df_original = pd.read_csv(test_data_path)
test_df_original['建筑面积data'] = test_df_original['建筑面
积'].str.extract('(\d+\.?\d*)').astype(float)
total_price_pred = y_pred * test_df_original['建筑面积data']

#重新读取原始数据
raw_test_df = pd.read_csv(test_data_path)

# 创建包含编号、预测总房价和板块编号的表格
result_df = pd.DataFrame({
    'id': range(0, len(total_price_pred)),
    'price': total_price_pred,
    '单位房价': y_pred,
    '板块': raw_test_df['板块'],
    'area': test_df_original['建筑面积data'] })

# 去除房价低于 200000 的数据
filtered_df = result_df[result_df['price'] >= 200000]

# 按板块分组计算每个板块的单位房价均值
block_means = filtered_df.groupby('板块')['单位房价'].mean()

# 遍历每个板块，将低于 200000 的值用该板块的单位房价乘以房屋面积所得的值替换
for block in result_df['板块'].unique():
    block_mask = result_df['板块'] == block
```

```
    low_price_mask = result_df['price'] < 200000
    combined_mask = block_mask & low_price_mask
    result_df.loc[combined_mask, 'price'] =
result_df.loc[block_mask, '单位房价'] * result_df.loc[block_mask,
'area']

# 删除除了 id 和 price 以外的列
result_df = result_df[['id', 'price']]
```

这部分代码在OLS，RIDGE和LASSO方法中重复出现，建议：

1. 首先封装一个函数来结合建筑面积和预测的单位房价计算总房价
2. 接着封装一个函数来创建包含 id、price、单位房价、板块和建筑面积的结果表格
3. 封装房价低于 200000 的替换逻辑
4. 最后封装一个主函数，集成上述所有步骤

封装函数后，只需要调用主函数即可实现功能，减少了重复代码，并且提升了可维护性。

### 2.2.4 依赖管理

代码中使用了多个第三方库，比如 pandas、scikit-learn等，但未提供环境依赖说明，导致其他开发人员在运行时可能出现库缺失或版本不兼容的问题。对于团队协作项目，建议使用 requirements.txt 或 Pipfile 来明确所需的库版本，方便其他开发人员正确使用。

### 2.2.5 运行环境说明

文档或 notebook 中未说明运行所依赖的基础环境。建议在文档中增加如下说明：

1. Python 版本（如 Python 3.9+）
2. 建议 IDE（如 Jupyter Lab / VSCode）
3. 是否需要 GPU、并行库等计算资源
4. 推荐执行方式：notebook运行 or .py脚本一键执行

### 2.2.6 主流程封装

Notebook 结构适合演示与交互，但在部署与复现实验时，建议增加主函数封装，提供一键运行能力。建议在notebook末尾增加main()调用或封装为 .py 脚本并提供命令行入口。

## 3. 数据处理评估

### 3.1 特征提取和转换

这里team1对建筑面积和套内面积进行了正则表达式的数值提取：

```
train_df['建筑面积data'] = train_df['建筑面积'].str.extract('(\d+\.?\d*)').astype(float)
train_df['套内面积data'] = train_df['套内面积'].str.extract('(\d+\.?\d*)').astype(float)
```

这是合理的方法，但没有对提取失败的情况进行处理。如果加上对NaN值的处理，将会更加稳健得应对原始数据中不符合模式的值。

同时，代码创建了"公摊比"这个衍生特征。通过查询相关资料，发现，这里并不应该命名为"公摊比"而应该是"套内占比"，如果想要构建公摊比，应该是`(建筑面积-套内面积)/建筑面积`

### 3.2 缺失值处理评估

这里的缺失值处理有两种：均值和KNN填充

```
# 用公摊比的均值填充缺失值
mean_share_ratio = train_df['公摊比'].mean()
train_df['公摊比'].fillna(mean_share_ratio, inplace=True)

# 使用KNN填充缺失的"环线"变量
known_ringline = train_df[train_df['环线'].notna()]
unknown_ringline = train_df[train_df['环线'].isna()]
```

两种都是合理的方法，KNN会比简单的均值/众数填充更好。

但"套内面积"缺失值比例高达70.11%，直接使用均值填充欠妥，且不同房产性质，如商品房、公房、安置房等房产的得房率（公摊比）存在差异，使用统一的均值填充也存在不妥。

且在后面代码中，去除了面积和房价的异常值，但填充公摊比是在这之前进行的。如果填充之后再进行数据过滤，可能会导致填充的值没有考虑到之后的数据筛选。如果某些样本因为面积异常被过滤掉，那么填充时使用的均值可能包含了这些异常样本的信息。因此，可能需要调整填充步骤的顺序，先处理异常值再填充。

## 3.3 异常值处理评估

异常值是利用IQR去除的，这是一种合理的方式去除异常值：

```
# 去除异常面积
train_df = train_df[(train_df['建筑面积data'] > 20) & (train_df['建筑面积data'] < 1000)]

# 计算IQR去除房价异常
q1 = train_df['每平方米房价'].quantile(0.25)
q3 = train_df['每平方米房价'].quantile(0.75)
iqr = q3 - q1
train_df = train_df[(train_df['每平方米房价'] >= (q1 - 1.5 * iqr)) & (train_df['每平方米房价'] <= (q3 + 1.5 * iqr))]
```

个人不推荐该剔除outlier的方法，因为某些在保留区间外的"面积-房价"对其实是满足数据总体的分布规律的，只是其数据较大或较小，而保留在区间内的"面积-房价"对有可能是不满足数据总体分布规律的，例如，可能存在面积位于保留下界，而房价位于保留上界的极端情况，这样的数据显然是不合理的，但会被保留在训练集中。

## 3.4 独热编码

首先代码进行了对公摊比每0.1进行区间划分，并对建筑面积进行区间划分，从而进行编码。这样的处理可以捕捉非线性关系，但会增加很多特征维度，可能会导致过拟合。

```
# 对公摊比每 0.1 进行划分
bins = np.arange(0, train_df['公摊比'].max() + 0.1, 0.1)
labels = [f'{bins[i]:.1f}-{bins[i + 1]:.1f}' for i in
range(len(bins) - 1)]
train_df['公摊比区间'] = pd.cut(train_df['公摊比'], bins=bins,
labels=labels, right=False)

# 对建筑面积进行区间划分
bins = np.arange(0, train_df['建筑面积data'].max() + 20, 20)
labels = [f'{int(bins[i])}-{int(bins[i+1])}' for i in
range(len(bins) - 1)]
train_df['建筑面积区间'] = pd.cut(train_df['建筑面积data'], bins=bins,
labels=labels, right=False)
```

在测试集处理部分，代码使用了train_df的建筑面积最大值来生成bins，这可能存在问题，因为测试集的建筑面积可能超过训练集的范围。如果测试集的某个房屋建筑面积超过训练集的最大值，分箱时会被归入最后一个区间或者可能被排除。

```
# 对测试集建筑面积进行处理
bins = np.arange(0, train_df['建筑面积data'].max() + 20, 20)
labels = [f'{int(bins[i])}-{int(bins[i+1])}' for i in
range(len(bins) - 1)]
test_df['建筑面积data'] = test_df['建筑面积'].str.extract('(\d+\.?
\d*)').astype(float)
test_df['建筑面积区间'] = pd.cut(test_df['建筑面积data'], bins=bins,
labels=labels, right=False)
test_area_range_dummies = pd.get_dummies(test_df['建筑面积区间'],
prefix='建筑面积区间')
test_df = pd.concat([test_df, test_area_range_dummies], axis=1)
```

此外，注意到小区名称作为特征可能会导致维度爆炸：

```
categorical_cols = ['城市', '区域', '板块','环线', '小区名称', '装修情
况', '配备电梯', '交易权属', '房屋用途', '房屋年限',
'产权所属', '交易时间', '年份', '所在楼层', '别墅类型', '梯户比例', '房屋优
势']
```

# 4. 数据泄露问题

## 4.1 数据预处理顺序问题

这里team1选择在整个数据集`train_df`上计算IQR，然后应用于训练集来过滤异常值。这样会将测试集的信息泄露到训练过程中。正确的做法是仅在训练集上计算IQR，然后分别应用于训练集和测试集：

```
# 原始代码
q1 = train_df['每平方米房价'].quantile(0.25)
q3 = train_df['每平方米房价'].quantile(0.75)
iqr = q3 - q1
train_df = train_df[(train_df['每平方米房价'] >= (q1 - 1.5 * iqr)) &
(train_df['每平方米房价'] <= (q3 + 1.5 * iqr))]

X_train_train, X_train_test, y_train_train, y_train_test =
train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
# 正确的做法
X_train_train, X_train_test, y_train_train, y_train_test =
train_test_split(train_df[features], train_df['每平方米房价'],
test_size=0.2, random_state=42)

q1 = y_train_train.quantile(0.25)
q3 = y_train_train.quantile(0.75)
iqr = q3 - q1
# 再分别应用于训练集和测试集
train_indices = y_train_train[(y_train_train >= (q1 - 1.5 * iqr)) &
(y_train_train <= (q3 + 1.5 * iqr))].index
X_train_train_filtered = X_train_train.loc[train_indices]
y_train_train_filtered = y_train_train.loc[train_indices]


test_indices = y_train_test[(y_train_test >= (q1 - 1.5 * iqr)) &
(y_train_test <= (q3 + 1.5 * iqr))].index
X_train_test_filtered = X_train_test.loc[test_indices]
y_train_test_filtered = y_train_test.loc[test_indices]
```

## 4.2 特征工程存在不一致

```
# 训练集数据归一化
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)

# 测试集数据处理
test_df = scaler.transform(test_df)
```

这里，利用在整个数据集`train_df`上拟合的`scaler`来转换验证集。正确的做法是只在训练子集上`fit_transform`，然后在验证集上使用`transform`。

## 4.3 缺失值填充中存在信息泄露

这里计算了测试集自身的均值来填充测试集的缺失值。但正确的做法是使用训练集的均值来填充测试集的缺失值：

```
# 训练集
mean_share_ratio = train_df['公摊比'].mean()
train_df['公摊比'].fillna(mean_share_ratio, inplace=True)

# 测试集
test_mean_share_ratio = test_df['公摊比'].mean()
test_df['公摊比'].fillna(test_mean_share_ratio, inplace=True)
```

## 4.4 交叉验证中的问题

```
for train_index, test_index in kf.split(X_train):
    X_train_train, X_train_test = X_train[train_index],
X_train[test_index]
    # ...
    model = LinearRegression()
    model.fit(X_train_train, y_train_train)
```

这里，在交叉验证之前代码已经完成了数据的预处理，这导致验证折中包含了训练折的信息。应该在每个交叉验证折内分别执行数据预处理。

也许正确一些的版本：

```
# 进行六折交叉验证，在每个折内分别执行数据预处理
for train_index, test_index in kf.split(train_df):
    # 1. 分割每折的训练集和验证集
    cv_train_df = train_df.iloc[train_index].copy()
    cv_val_df = train_df.iloc[test_index].copy()

    # 2.1 处理缺失值 - 例如：用训练折均值填充公摊比
    mean_share_ratio = cv_train_df['公摊比'].mean()
    cv_train_df['公摊比'].fillna(mean_share_ratio, inplace=True)
    cv_val_df['公摊比'].fillna(mean_share_ratio, inplace=True)  # 使
用训练折均值
```

### 4.5 KNN填充的问题

```
# 训练集KNN填充
known_ringline = train_df[train_df['环线'].notna()]
unknown_ringline = train_df[train_df['环线'].isna()]
# ...

# 测试集KNN填充
known_ringline = train_df[train_df['环线'].notna()]
unknown_ringline = test_df[test_df['环线'].isna()]
```

这里选取KNN进行填充，但是在填充测试集缺失值时利用了完整数据集的已知数据来训练KNN模型，是一个数据泄露点，因为测试集信息本应该是未知的。

## 5. 模型性能评估

### 5.1 模型选择合理性分析

#### 5.1.1 超参数选择的问题

在Ridge回归模型中，对于超参数的选择方式存在改进空间：

```
#构建Ridge回归模型
alphas = [0.01, 0.1, 1, 10, 100]
model_2 = RidgeCV(alphas=alphas, cv=5).fit(X_train, y_train)
print(f"最佳alpha: {model_2.alpha_}")
```

这里对于超参数的选择，仅采用了0.01、0.1、1、10、100共5个值，而没有尝试更多的选择。建议：如果选择的最佳参数为0.01或100，则可以进一步尝试更大（更小）的值；如果选择的最佳参数是中间的某个值，也可以多加调整进行优化，而非就采用这五个值。

### 5.1.2 特征选择问题

模型选择了大量分类特征进行独热编码：

```
# 选择特征变量
categorical_cols = ['城市', '区域', '板块','环线', '小区名称', '装修情
况', '配备电梯', '交易权属', '房屋用途', '房屋年限',
                    '产权所属', '交易时间', '年份', '所在楼层', '别墅类
型', '梯户比例', '房屋优势']
train_features = ['城市', '区域', '板块', '环线','年份', '小区名称', '装
修情况', '配备电梯', '交易权属', '房屋用途',
                  '房屋年限', '产权所属', '交易时间', '所在楼层', '别墅类
型', '梯户比例', '房屋优势', '多手房'] +
list(area_range_dummies.columns) +
list(share_ratio_dummies.columns)

X_train = train_df[train_features]
y_train = train_df['每平方米房价']

# 对需要独热编码的特征进行处理
X_train = pd.get_dummies(X_train, columns=categorical_cols)
```

其中将"房屋优势"这样完全主观填写的变量也生成虚拟变量进行处理，由于这个变量每套房屋填写的都不一样，所以几乎没有任何意义，只是将每套房又增加一个虚拟变量，对后续的预测起不到任何帮助。此外，由于许多变量（例如"城市"、"区域"、"板块"）存在很强的共线性（自相关）问题，全部加入预测中可能会带来模型偏差（特别是针对Ridge和Lasso），建议慎重考虑。

### 5.1.3 随机种子不一致

模型训练过程中使用的随机种子与要求不符：

```
#要先把train_df打乱一下顺序，然后再划分训练集和测试集
train_df = train_df.sample(frac=1,
random_state=42).reset_index(drop=True)
X_train = train_df[train_features]
X_train = pd.get_dummies(X_train, columns=categorical_cols)
y_train = train_df['每平方米房价']

# 划分数据集，80% 用于训练，20% 用于验证
X_train_train, X_train_test, y_train_train, y_train_test =
train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# 创建六折交叉验证对象
kf = KFold(n_splits=6, shuffle=True, random_state=42)
```

这里老师要求的random_state=111，而代码采用的是42，导致不符合要求。

## 5.2 后处理问题

在预测结果的后处理过程中，发现了如下问题：

```
# 去除房价低于 200000 的数据
filtered_df = result_df[result_df['price'] >= 200000]

# 按板块分组计算每个板块的单位房价均值
block_means = filtered_df.groupby('板块')['单位房价'].mean()

# 遍历每个板块，将低于 200000 的值用该板块的单位房价乘以房屋面积所得的值替换
for block in result_df['板块'].unique():
    block_mask = result_df['板块'] == block
    low_price_mask = result_df['price'] < 200000
    combined_mask = block_mask & low_price_mask
    result_df.loc[combined_mask, 'price'] =
result_df.loc[block_mask, '单位房价'] * result_df.loc[block_mask,
'area']
```

这里将房价预测值低于200000的值用板块单位房价均值乘以面积来替换，有一定道理；但是考虑到：

1. 预测的城市和区域众多，20万的总价对于特定区域而言不是不可能的；

2. 如一定要进行缩尾处理，那么20万这个截断值的选取是否过于武断，而缺乏一定的证据支持；
3. 使用"该板块预测的均价"来填充是否不妥，可采用另一套数据中该板块真实的均值，或许更准确；
4. 后续计算MAE和RMSE时，似乎没有将此处的调整纳入其中，可能会导致模型选择和评估的效果受影响。

因此这里的处理还值得商榷。

## 6. 改进建议

1. 在数据集划分后进行数据预处理，避免测试集信息泄露到训练阶段。且所有的基于统计的预处理操作（例如均值、方差和分箱界限）都应该仅在训练集上计算，然后应用到测试集。
2. 统一变量命名规范，避免中英文混用，提高代码可读性。
3. 增加关键操作的详细注释，特别是模型选择和参数设置的理由。
4. 将重复的数据处理步骤封装为函数，提高代码模块化程度和可维护性。
5. 提供完整的环境依赖说明，包括Python版本和必要的库版本。
6. 考虑更合理的缺失值处理方法，尤其是对于缺失比例高达70%的"套内面积"字段。
7. 重新考虑异常值处理策略，可能需要考虑多变量关系而不仅仅是边界值。
8. 在分箱编码时，考虑测试集可能超出训练集范围的情况。
9. 重新设计"公摊比"特征，正确命名为"套内占比"，或根据实际含义调整计算公式。
10. 在交叉验证过程中，确保每个折内分别进行完整的数据预处理。
11. 为代码增加主函数封装，提供一键运行能力。

## 7. 总结

本报告对团队1开发的房地产价格预测模型进行了全面验证，从代码运行检查、数据处理评估到数据泄露问题分析，并提出了具体的改进建议。主要发现包括数据预处理顺序不当导致的数据泄露、缺失值处理方法有待优化、代码模块化程度不足等问题。