中国人民大学
RENMIN UNIVERSITY OF CHINA

房产定价线性模型
Mid-term 期中汇报

财政金融学院
王晨曦
2024年10月31日

数据处理

- Assignment1：补充缺失值
- Assignment2：提取特征数据 (str -> int or float)

Step1：

merge 合并train, test && details

主要目的：

✓ 补充train, test中缺失的数据（Assignment1）
主要是 建筑结构
✓ 补全房屋特征（prepare for Assignment2）
主要补充 房屋总数、楼栋总数等

1.1 我们先来处理df_details，并将其与df_ori，df_pre合并

```
# print(df_details.head())
df_details = df_details.rename(columns={'名称': '小区名称', '建筑结构_x': '建筑结构'})
# 合并建筑结构数据，以便填充缺失值
# 假设 df_details 包含所有需要的列
columns_to_merge = ['建筑结构', '房屋总数', '楼栋总数', '绿 化 率', '容 积 率', '物 业 费', '供暖', '燃气费', '供热费', '停车位']
merged_df_ori = df_ori.merge(df_details[columns_to_merge + ['小区名称']], on='小区名称', how='left')
merged_df_ori.rename(columns={'建筑结构_x': '建筑结构',
                              '房屋总数_x': '房屋总数',
                              '楼栋总数_x': '楼栋总数',
                              '绿 化 率_x': '绿化率',
                              '容 积 率_x': '容积率',
                              '物 业 费_x': '物业费',
                              '供暖_x': '供暖',
                              '燃气费_x': '燃气费',
                              '供热费_x': '供热费',
                              '停车位_x': '停车位'
                              }, inplace=True)

# 合并建筑结构数据，以便填充缺失值
merged_df_pre = df_pre.merge(df_details[columns_to_merge + ['小区名称']], on='小区名称', how='left')
merged_df_pre.rename(columns={'建筑结构_x': '建筑结构',
                              '房屋总数_x': '房屋总数',
                              '楼栋总数_x': '楼栋总数',
                              '绿 化 率_x': '绿化率',
                              '容 积 率_x': '容积率',
                              '物 业 费_x': '物业费用',
                              '供暖_x': '供暖',
                              '燃气费_x': '燃气费',
                              '供热费_x': '供热费',
                              '停车位_x': '停车位'
                              }, inplace=True)
n_ori = merged_df_ori.shape[0]
print("原数据共有{}条数据".format(n_ori))
n_pre = merged_df_pre.shape[0]
print("预测集共有{}条数据".format(n_pre))
# merged_df_ori.to_csv('merged_df_ori.csv', encoding='utf-8-sig', index=False)
```

[4] ✓ 0.0s Python

原数据共有109307条数据
预测集共有19298条数据

详见：Code 1.1

数据处理

- Assignment1：补充缺失值
- Assignment2：提取特征数据 (str -> int or float)

Step2：

定义函数 clean_data 用于清洗数据

主要工作：

✓ 补充所有的缺失值（不同特征选择了不同的补充方式）Assignment1
✓ 提取特征值 Assignment2

```python
def clean_data(df_cleaned, df_ori, df_details = df_details):
    df_cleaned['城市'] = df_ori['城市']
    df_cleaned['区域'] = df_ori['区域']
    df_cleaned['板块'] = df_ori['板块']
    # 环线
    df_cleaned['小区名称'] = df_ori['小区名称']
    try:
        df_cleaned['价格'] = df_ori['价格']
    except KeyError:
        pass

    # 房屋户型
    df_ori = mode_complete(df_ori, '房屋户型')
    # 使用正则表达式提取房间数、厅数、厨数和卫数
    df_cleaned[['室数', '厅数', '厨数', '卫数']] = df_ori['房屋户型'].str.extract(r'(\d+)室(\d+)厅(\d+)厨(\d+)卫')

    # 所在楼层
    # 先截取前三个字符，然后使用 extract() 提取匹配的楼层字符串
    floor_levels = df_ori['所在楼层'].str[0:3].str.extract(r'(低楼层|中楼层|高楼层|地下室)')
    # 使用 replace() 方法将提取的楼层字符串映射到相应的数值
    df_ori['所在楼层代码'] = floor_levels.replace({
        '低楼层': 1,
        '中楼层': 2,
        '高楼层': 3,
        '地下室': 4
    })
    # 如果存在没有匹配到的楼层，可以设置一个默认值
    df_ori['所在楼层代码'] = df_ori['所在楼层代码'].astype(float)

    # 总层数，先截取"共"字后面的字符，然后提取数字
    df_cleaned['总层数'] = df_ori['所在楼层'].str[3:].astype(str).str.extract(r'共(\d+)层')
    # 若提取失败，则设置默认值
    df_cleaned['总层数'] = df_cleaned['总层数'].astype(float)

    # 赋予df_cleaned所处楼层的值，低 - 0.3， 中 - 0.5， 高 - 0.8，地下室 - (-1)
    # 如果为空，补充为0.5
    df_ori['所在楼层赋值'] = df_ori['所在楼层代码'].apply(lambda x: 0.3 if x == 1 else 0.5 if x == 2 else 0.8 if x == 3 else -1 if x == 4 else 0.5)
    df_cleaned['所在楼层'] = np.where(df_ori['所在楼层赋值'] != -1,
                                  df_ori['所在楼层赋值'] * df_cleaned['总层数'],
                                  -1)
```

部分截图
详见：Code 1.2

数据处理

Step2 further：

clean_data 中引用的函数

```python
# 众数赋值函数

def mode_complete(df_ori, col_name_complete, col_name_basis='小区名称', others=None):
    df_ori.loc[df_ori[col_name_complete].isnull(), col_name_complete] = df_ori.groupby(col_name_basis)[col_name_complete].transform(
    lambda x: x.mode()[0] if not x.mode().empty else others
    )
    df_ori = df_ori.dropna(subset=[col_name_complete])

    return df_ori
```

✓ 0.0s

Compared with **SimpleImputer** from scikit-learn

Advantages:

自己写的函数，相对简单清晰

可以随时做出调整

能够较为稳定的按另一特征组进行分类

数据处理

Step2 further：

clean_data 中引用的函数

```python
# 供热费
def hot_fee(text):
    # 先按"/㎡"分割字符串
    temp = text.split("元/㎡")[0]
    try:
        return float(temp)
    except ValueError:
        group = temp.split("-")
        return (float(group[0]) + float(group[1])) / 2
def calculate_heating_fee(row):
    # 首先查看是否为空
    if pd.isnull(row['供热费']):
        # 检查供暖是否为1
        if row['供暖'] == 1:
            # 尝试获取板块的供暖费众数
            try:
                mode_value = df_cleaned[df_cleaned['板块'] == row['板块']]['供暖'].mode()[0]
                return mode_value
            except (KeyError, IndexError):
                # 返回燃气费列的倍数，这里假设倍数为固定的1.2倍
                return row['燃气费'] * 1.2
        else:
            # 返回燃气费列的倍数，这里假设倍数为固定的1.2倍
            return row['燃气费'] * 1.2
    else:
        return row['供热费']
```

这里选取了供热费函数来讲解

处理方式为：

- 如果“供暖”一列包含集中供暖，则
  （1）大多数形况下会包含供热费
  （2）如果依然没有供热费，则众数进行填补

- 如果“供暖”一列不包含集中供暖，则补充为燃气费的1.2倍。

1.2是按照4个月冬季，每天没屏幕燃气0.01立方米假设

数据处理

Step2 further：
clean_data 中引用的函数

```python
# 周边配套
def surround_count(text):
    score = 0
    if pd.isna(text):
        return 0
    else:
        words = pseg.cut(text)
        descriptions = [word for word, flag in words if flag in ('n') and len(word) > 1]
        score += len(descriptions)*1

        return score

# 交通出行
def transport_count(text):
    score = 0
    if pd.isna(text):
        return 0
    else:
        words = pseg.cut(text)
        descriptions = [word for word, flag in words if flag in ('m')]
        score += len(descriptions)*1

        if score < 1:
            if '地铁' or '公交' or '火车' or '飞机' in text:
                score += 1
            elif '交通' or '出行' or '驾车' in text:
                score += 0.3

        return score
```

对于核心卖点、户型介绍、周边配套、交通出行长text文本

采用了jieba分词的处理方式，并使用pseg定义词性，

如周边配套选取了名词计数；交通出行选取了数词进行计数，并对特定文本进行了赋值处理

数据处理

Step2 further：
clean_data 中引用的函数

```python
# 周边配套
def surround_count(text):
    score = 0
    if pd.isna(text):
        return 0
    else:
        words = pseg.cut(text)
        descriptions = [word for word, flag in words if flag in ('n') and len(word) > 1]
        score += len(descriptions)*1

        return score

# 交通出行
def transport_count(text):
    score = 0
    if pd.isna(text):
        return 0
    else:
        words = pseg.cut(text)
        descriptions = [word for word, flag in words if flag in ('m')]
        score += len(descriptions)*1

        if score < 1:
            if '地铁' or '公交' or '火车' or '飞机' in text:
                score += 1
            elif '交通' or '出行' or '驾车' in text:
                score += 0.3

        return score
```

对于核心卖点、户型介绍、周边配套、交通出行长text文本

采用了jieba分词的处理方式，并使用pseg定义词性，

如周边配套选取了名词计数；交通出行选取了数词进行计数，并对特定文本进行了赋值处理

数据处理

Step2 further：
clean_data 中引用的函数

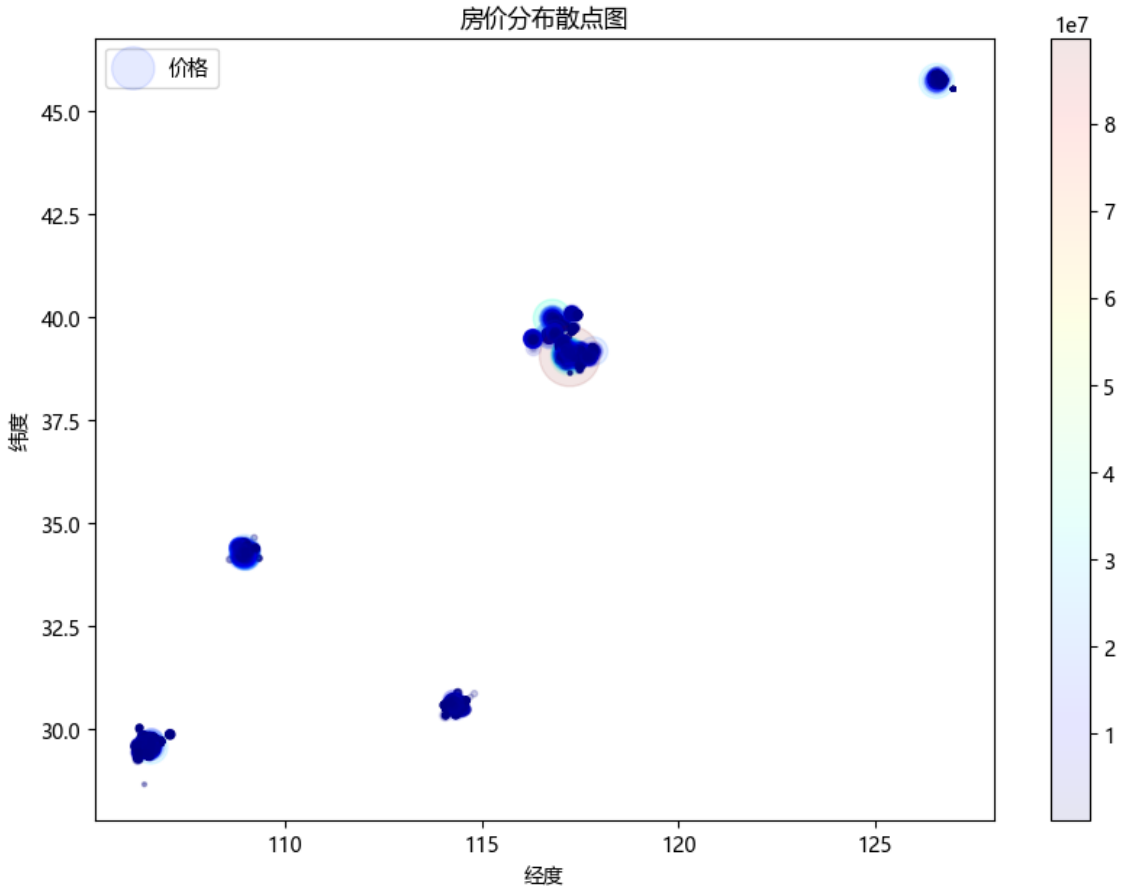
```
# 所在楼层
# 先截取前三个字符，然后使用 extract() 提取匹配的楼层字符串
floor_levels = df_ori['所在楼层'].str[0:3].str.extract(r'(低楼层|中楼层|高楼层|地下室)')
# 使用 replace() 方法将提取的楼层字符串映射到相应的数值
df_ori['所在楼层代码'] = floor_levels.replace({
    '低楼层': 1,
    '中楼层': 2,
    '高楼层': 3,
    '地下室': 4
})
# 如果存在没有匹配到的楼层，可以设置一个默认值
df_ori['所在楼层代码'] = df_ori['所在楼层代码'].astype(float)

# 总层数，先截取“共”字后面的字符，然后提取数字
df_cleaned['总层数'] = df_ori['所在楼层'].str[3:].astype(str).str.extract(r'共(\d+)层')
# 若提取失败，则设置默认值
df_cleaned['总层数'] = df_cleaned['总层数'].astype(float)

# 赋予df_cleaned所处楼层的值，低 - 0.3， 中 - 0.5， 高 - 0.8，地下室 - (-1)
# 如果为空，补充为0.5
df_ori['所在楼层赋值'] = df_ori['所在楼层代码'].apply(lambda x: 0.3 if x == 1 else 0.5 if x == 2 else 0.8 if x == 3 else -1 if x == 4 else 0.5)
df_cleaned['所在楼层'] = np.where(df_ori['所在楼层赋值'] != -1,
                                  df_ori['所在楼层赋值'] * df_cleaned['总层数'],
                                  -1)
```

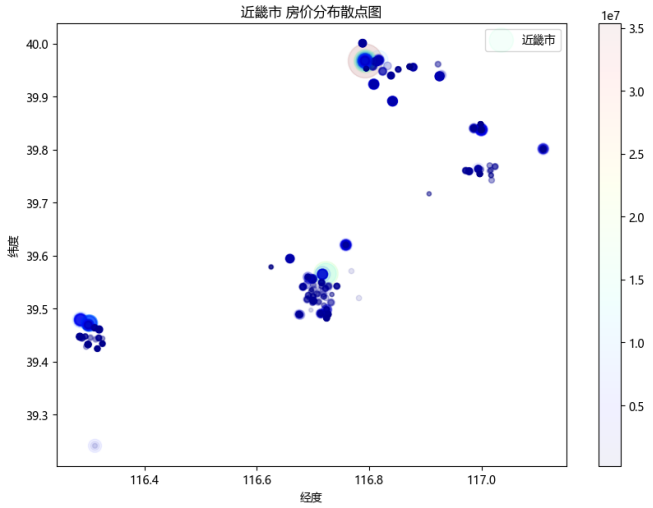
对于所在所在楼层，采取正则表达式的方式获取总层数，并对地 低 中 高分别赋值

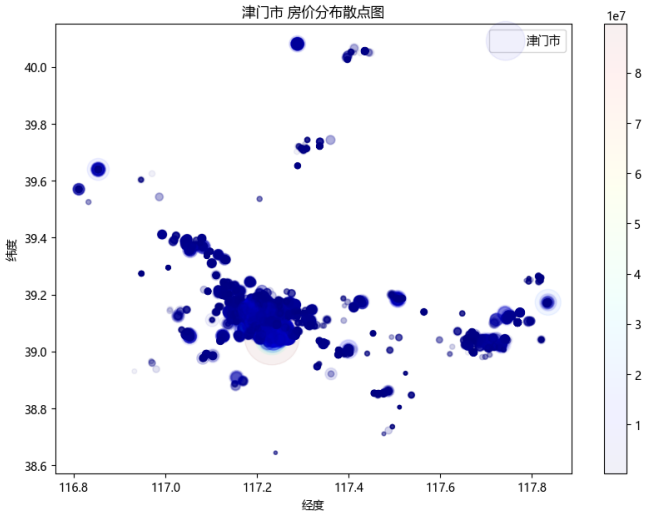
Step3：更好的确定经纬度如何使用

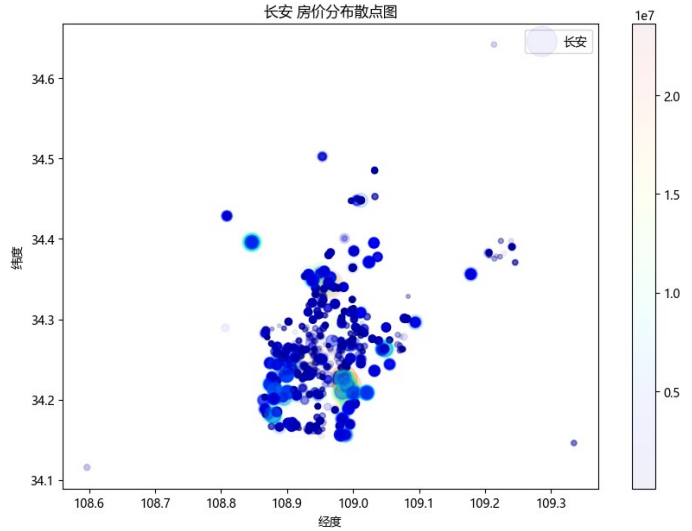


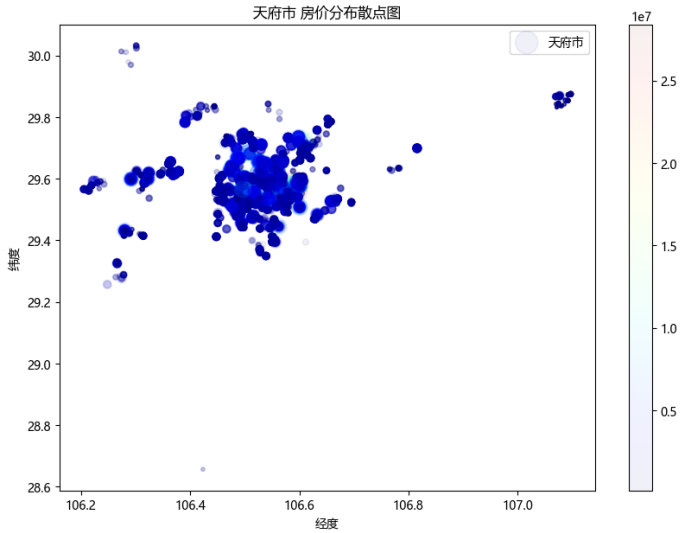
数据处理

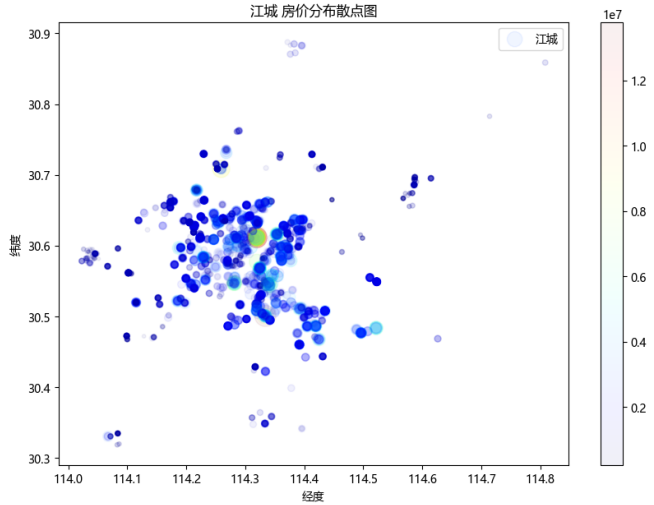
Step3：更好的确定经纬度如何使用





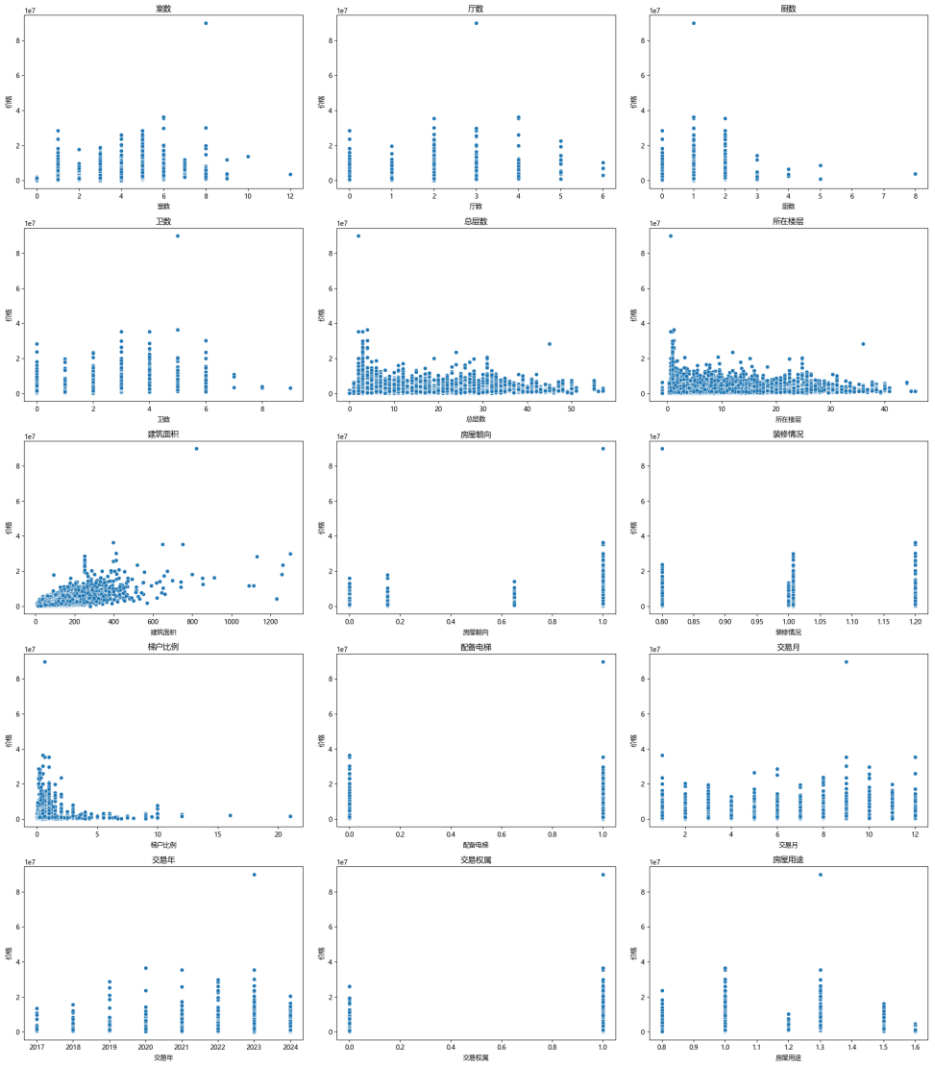


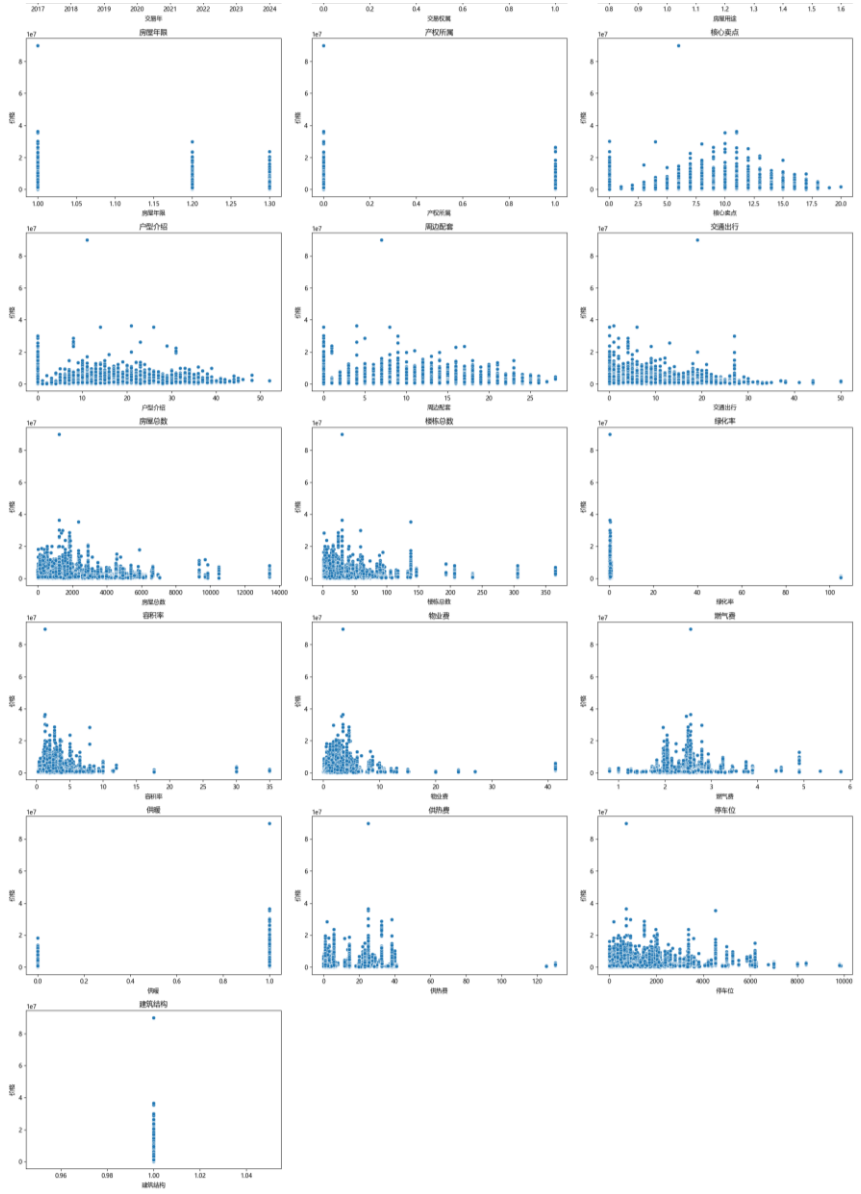




确定变量

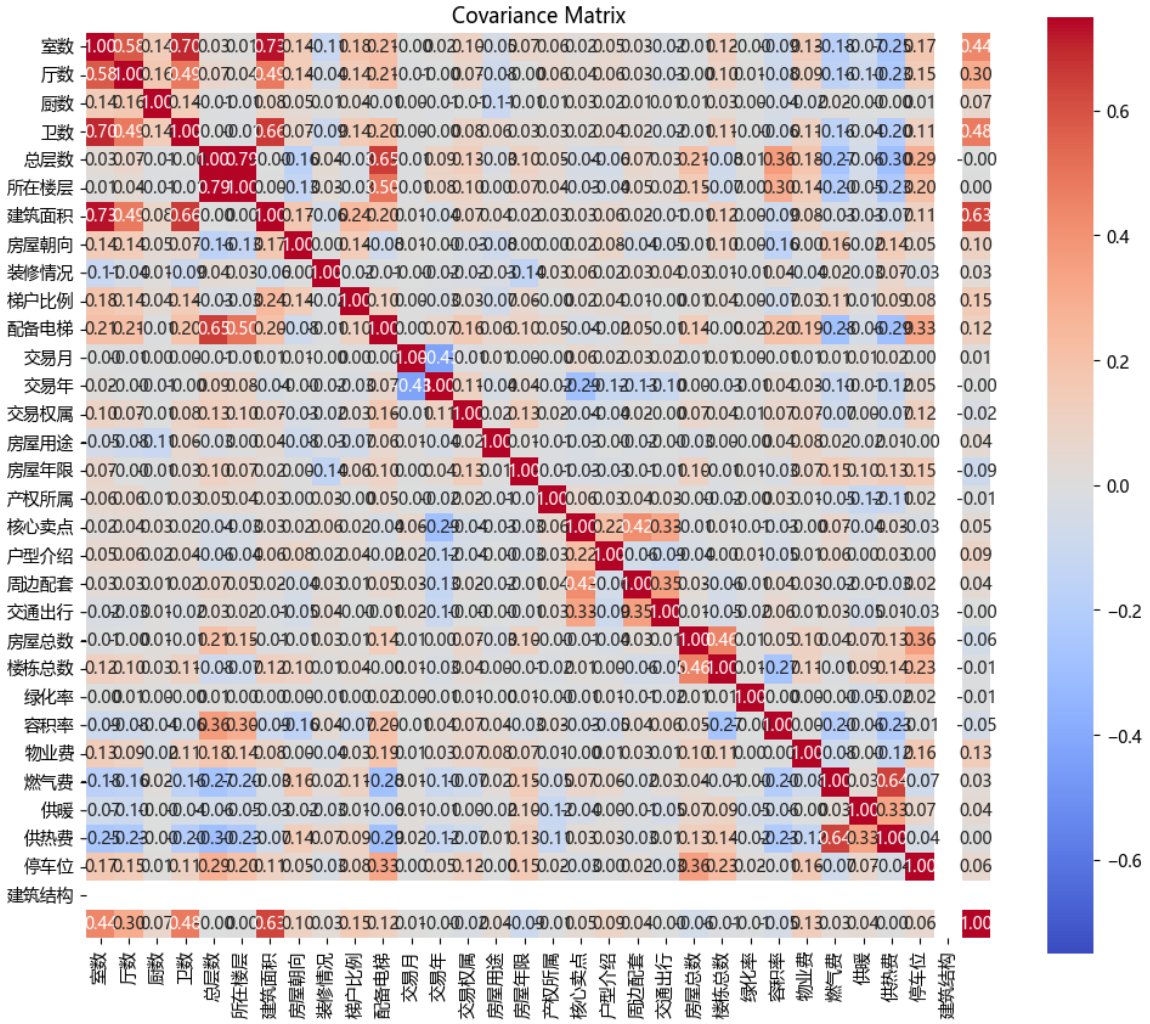
画出散点图，各变量x与y





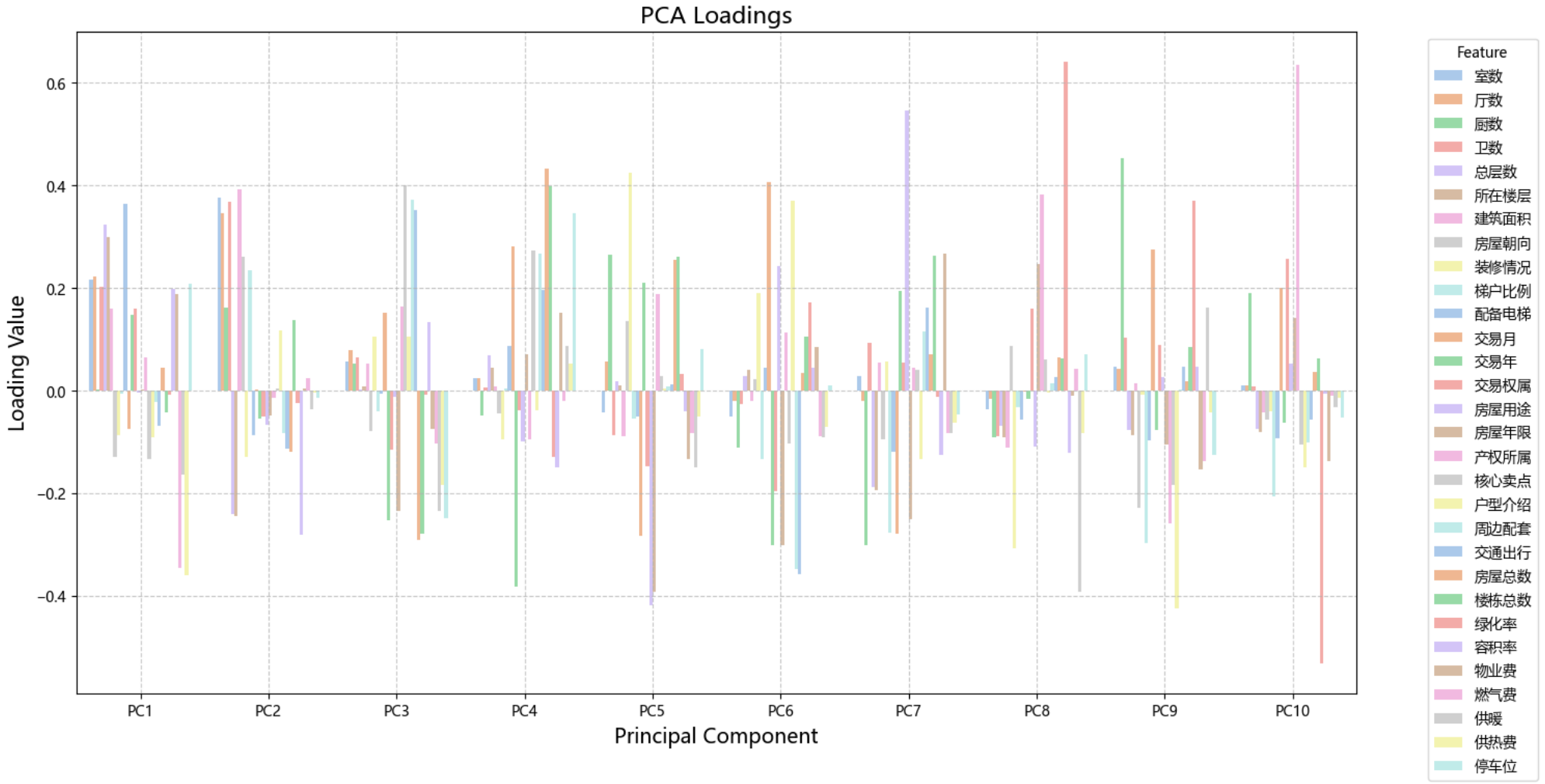
确定变量

X集的协方差矩阵



确定变量

主成分分析法测试



确定变量

设置出一些新的变量

```python
def reset_X(X):
    X['房屋数'] = X['室数'] + X['厅数'] + X['卫数'] + X['厨数']
    X['每屋面积'] = X['建筑面积'] / X['房屋数'].where(X['房屋数'] != 0)

    X.drop(['室数', '厅数', '卫数', '厨数'], axis=1, inplace=True)

    X['总层数&配备电梯'] = X['总层数'] * X['配备电梯']
    X['所在楼层&配备电梯'] = X['所在楼层'] * X['配备电梯']
    X['所在楼层/总层数'] = X['所在楼层'] / X['总层数'].where(X['总层数'] != 0)

    X.drop(['总层数', '配备电梯', '所在楼层'], axis=1, inplace=True)

    X['房屋总数/楼栋总数'] = X['房屋数'] / X['楼栋总数'].where(X['楼栋总数'] != 0)
    X.drop(['房屋总数'], axis=1, inplace=True)

    X['停车位/房屋总数'] = X['停车位'] / X['房屋数'].where(X['房屋数'] != 0)


    return X
```

0.0s Python

训练模型

这里训练模型相对比较简单

训练模型

```python
# 定义模型列表
models = {
    'OLS': LinearRegression(),
    'LASSO': Lasso(),
    'Ridge': Ridge(),
    'ElasticNet': ElasticNet()
}
```

训练结果表示

```
Model: LASSO
In sample predictions (MAE): 579701.89
Cross-validation predictions (MAE): 580061.22
Testing predictions (MAE): 582276.30
Sample in sample predictions: [1817097.28292316 1297527.56529653 1254627.87861(
  995702.49769294]
Sample test predictions: [ 811272.46641359 1591572.19028884 1459444.23152858 2(
  484072.50707066]
Coefficients: [ 1.89983894e+04 -5.38257110e+04  5.85335229e+05 -4.51332217e+04
  8.60109833e+03  7.81563360e+04 -3.05436586e+05  2.32302470e+05
 -1.32106772e+06 -7.71329492e+04  7.26640381e+03  7.60088406e+03
  3.16874886e+03 -1.66220056e+03 -4.80948845e+03 -2.52175825e+03
 -3.88939899e+02  6.49120924e+04  1.90530143e+05  2.74561760e+05
  4.53892187e+03  5.26152364e+01  3.04495391e+04 -3.18108933e+03
 -8.36537843e+03  1.93099738e+04 -5.35524904e+05 -4.83015901e+04
 -2.59801388e+02]
Intercept: -158070052.5540956
----------------------------------
```

```
Summary of results:
OLS | In sample: 579703.42, Cross-validation: 580062.75, Testing: 582278.13
LASSO | In sample: 579701.89, Cross-validation: 580061.22, Testing: 582276.30
Ridge | In sample: 579690.89, Cross-validation: 580047.89, Testing: 582264.30
ElasticNet | In sample: 591665.49, Cross-validation: 591914.66, Testing: 59318
```

训练模型 – 提升方向

– 关于经纬度信息的利用

– 超参数尝试，运行结果依然不太理想

```
param_grid = {
    'OLS': {'model__fit_intercept': [True, False]},  # 线性回归通常不需要调整太多参数
    'LASSO': {'model__alpha': [0.1, 1, 10, 100, 1000]},
    'Ridge': {'model__alpha': [0.1, 1, 10, 100, 1000]},
    'ElasticNet': {'model__alpha': [0.1, 1, 10, 100, 1000], 'model__l1_ratio': [0.1, 0.5, 0.7, 0.9, 1]}
}
```

中国人民大学
RENMIN UNIVERSITY OF CHINA

复兴栋梁 强国先锋

谢谢大家！

Thank the experts for listening and welcome the criticism!