链家网二手房房价预测模型

SJTU-IS303-数据挖掘 课程设计

商进秩 519030910174

贾睿辰 519030910164

项目简介

链家网

北京链家成立于2001年，是中国领先的房地产服务企业，业务覆盖二手房、新房、租房等全方位房 产交易和居住服务，目前北京链家有1000余家门店。20年来，北京链家致力于提供安全有品质的服 务，在行业率先承诺“不吃差价”、首倡“真房源”、推出“交易不成 佣金全退”“电话营销 扰一赔百” 等18项安心服务承诺，持续推动行业进步，不断提供更有品质的服务。

项目背景

本项目爬取了2022年6月17日链家网上海二手房（https://sh.lianjia.com/ershoufang） 数据作为本项目数据集，尝试对上海市二手房价格进行建模，并通过建立的模型预测上海市二手房价格。

项目目录

```
base
│  main.py                  // 主程序入口
│  requirements.txt
│
├─data
│      count_fitment.png    // 测试图形
│      ershoufang.csv       // 爬取的原始数据
│      model_count.png      // 测试图形
│      rawdata.csv          // 原始数据备份
│      res.csv              // 预处理得到的结果
│      spider.py            // 爬虫文件
│      test.csv             // 测试文件
│
├─pretreatment
│  │  address.py            // 处理属性address
│  │  direction.py          // 处理属性direction
│  │  fitment.py            // 处理属性fitment
│  │  floor.py              // 处理属性floor
│  │  model.py              // 处理属性model
│  │  overview.py           // 浏览csv相关信息
│  │  pretreat.py           // 预处理函数入口
│  │  transform.py          // 字符串数字化
│  │
│  └─__pycache__
│
├─train
```

```
│   │   train.py              // 训练函数入口
│   │
│   └──__pycache__
│
├─venv
│
└─report
│   │   train.py              // 训练函数入口
│   │
│   └──static                   // 报告静态文件
```

互联网数据爬取

爬虫编写

通过在网页源码中搜索关键字的方式，首先确定链家网上海二手房网站为静态网站。

```
▼ <div class="info clear">
  ▼ <div class="title">
      <a class href="https://sh.lianjia.com/ershoufang/107105237186.html" target="_blank"
      data-log_index="1" data-el="ershoufang" data-housecode="107105237186" data-is_focus
      data-sl>客厅朝南，户型好，带两个固定车位，精装修</a>
      <!-- 拆分标签 只留一个优先级最高的标签-->
      <span class="goodhouse_tag tagBlock">必看好房</span>
      ::after
    </div>
  ▶ <div class="flood">…</div>
  ▶ <div class="address">…</div>
  ▶ <div class="followInfo">…</div>
  ▶ <div class="tag">…</div>
  ▶ <div class="priceInfo">…</div>
    ::after
</div>
```

再浏览不同页面进行观察，总结出链家网上海二手房网站的URL具有如下规律：

```
第一页：https://sh.lianjia.com/ershoufang/pg1/
第二页：https://sh.lianjia.com/ershoufang/pg2/
第三页：https://sh.lianjia.com/ershoufang/pg3/
第n页：https://sh.lianjia.com/ershoufang/pgn/
闵行区第n页：https://sh.lianjia.com/ershoufang/minhang/pgn/
浦东区第n页：https://sh.lianjia.com/ershoufang/pudong/pgn/
```

依照以上规律，我们可以通过爬虫实现对链家网上海二手房网站每一页数据的爬取。

爬取方式

由于链家网上海二手房网站每一页数据有限，每一页呈现30条数据，最多可访问100页。那么，如果仅对 首页进行爬取，得到的数据量在3000条左右，远远不足以训练出较为准确的模型。经过分析，我们发现 如果按照上海市的行政区进行筛选，每个行政区子页面会呈现出不同的房源，可以获得客观的数据量。我们最终 便采取此种方式对数据进行爬取，成功获得了约20,000条数据。

数据集

在我们为通过爬虫爬取得到的原始数据集增加属性后，得到示例如下：

```
name,model,area,direction,fitment,floor,address,total_list,price_list
汤臣豪园(二期) ,4室3厅,268.73平米,南,精装,下叠(共5层),浦东,2280,"84,844元/平"
汇锦城二期 ,4室2厅,140.59平米,南 北,精装,15层,浦东,493,"35,067元/平"
大华锦绣华城(十一街区) ,3室2厅,108.83平米,南 北,精装,低楼层(共20层),浦东,993,"91,244
元/平"
玉兰香苑二期A块 ,3室2厅,116.29平米,南,精装,高楼层(共6层),浦东,680,"58,475元/平"
昱丽家园 ,1室1厅,57.07平米,南,简装,低楼层(共14层),浦东,230,"40,302元/平"
```

可以看到数据集中包含如下属性：

- name：二手房所在小区的名称
- model：二手房的户型
- area：二手房的面积
- direction：二手房的朝向
- fitment：二手房的装修情况
- floor：二手房的楼层
- address：二手房所在的区
- total_list：二手房的总价（万元）
- price_list：二手房每平米的单价

下面我们介绍如何对数据集进行预处理。

数据预处理

数据总览

调用overview函数对爬取的原始数据集进行观察：

```
# 查看每个属性的所有可能取值
def overview(path):
esf = pd.read_csv(path, encoding='utf-8')

print(esf['model'].unique())
print(esf['direction'].unique())
print(esf['fitment'].unique())
print(esf['floor'].unique())
print(esf['address'].unique())
print(esf[esf.duplicated()])
print(esf.info())
```

得到数据集的总览信息如下：

```
RangeIndex: 19740 entries, 0 to 19739
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  ------      --------------  -----
0   name        19740 non-null  object
1   model       19740 non-null  object
2   area        19740 non-null  object
3   direction   19740 non-null  object
4   fitment     19740 non-null  object
5   floor       19740 non-null  object
6   address     19740 non-null  object
7   total_list  19740 non-null  object
8   price_list  19739 non-null  object
dtypes: object(9)
```

可以发现，在整个数据集中，有一条数据的price_list属性为空，因此设计clean函数 对为空的数据进行清洗。

```
# 清除空值
def clean(file):
    file.dropna(axis=0, how='any', inplace=True)
    return file
```

name

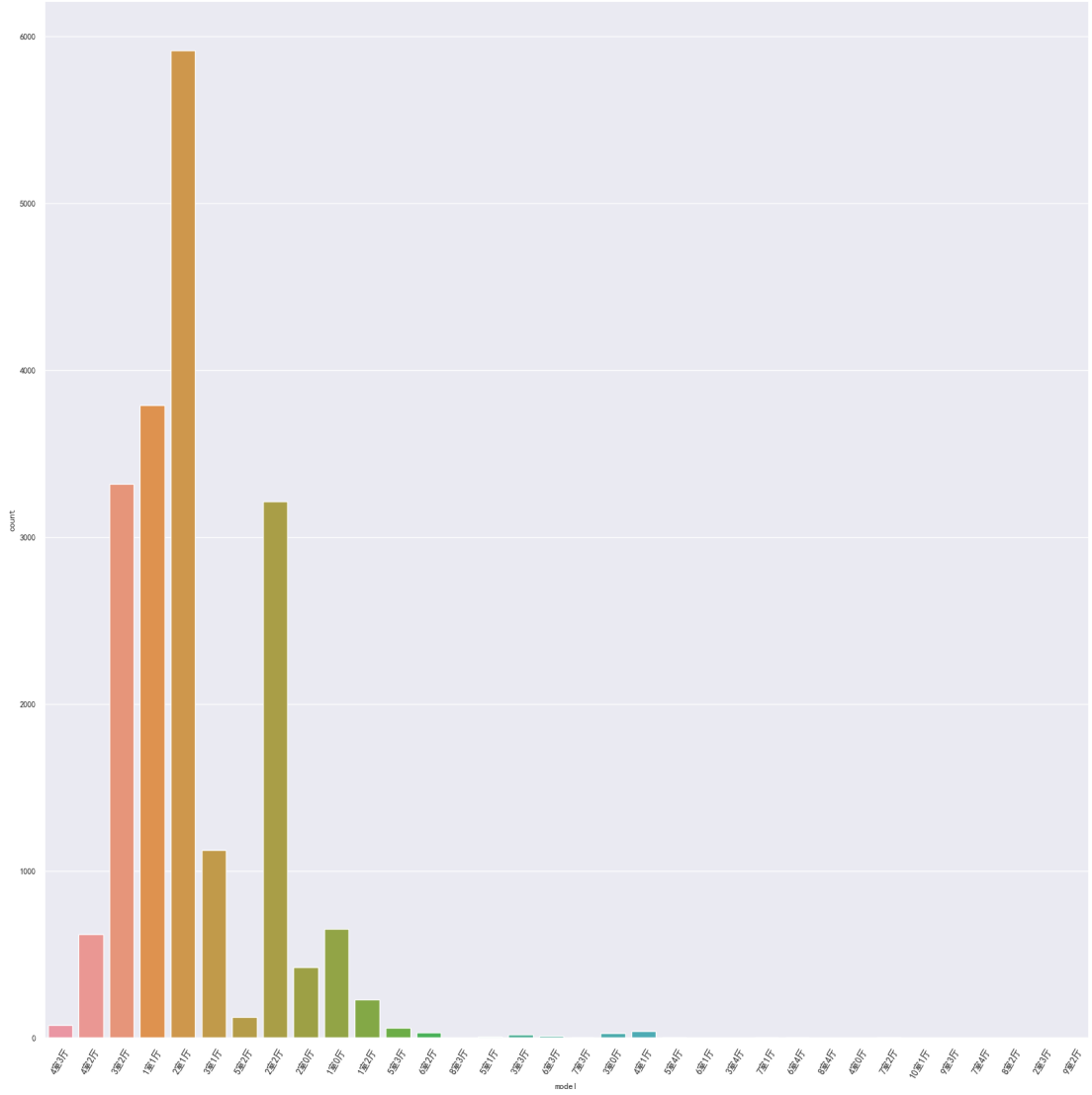
经过我们分析，name属性仅仅作为类似于id的标识作用出现，对于模型建立 并没有实际的帮助，并且无法进行处理，因此我们在实际建模时选择移除name属性。

model

model属性在数据集中一共有如下出现形式：

```
['4室3厅' '4室2厅' '3室2厅' '1室1厅' '2室1厅' '3室1厅' '5室2厅' '2室2厅' '2室0厅' '1
室0厅'
 '1室2厅' '5室3厅' '6室2厅' '8室3厅' '5室1厅' '3室3厅' '6室3厅' '7室3厅' '3室0厅' '4
室1厅'
 '5室4厅' '6室1厅' '3室4厅' '7室1厅' '6室4厅' '8室4厅' '4室0厅' '7室2厅' '10室1厅' '9
室3厅'
 '7室4厅' '8室2厅' '2室3厅' '9室2厅']
```

由于元素取值种类过多，我们决定对model属性进行可视化观察。通过调用 visualize_model函数，我们得到了如下图像：



通过图像我们发现，大多数数据集中在前12个属性取值中，因此，在对model属性 的预处理操作中，我们对其进行了筛选，将model取值不为前12个取值的数据剔除。

```
    for index, row in file.iterrows():
        if ((row['model'] == '4室3厅' or row['model'] == '4室2厅' or row['model']
== '3室2厅' or
                row['model'] == '1室1厅' or row['model'] == '2室1厅' or
row['model'] == '3室1厅' or
                row['model'] == '5室2厅' or row['model'] == '2室2厅' or
row['model'] == '2室0厅' or
                row['model'] == '1室0厅' or row['model'] == '1室2厅' or
row['model'] == '5室3厅')
                and (float(row['total_list']) < 1500)):
            df_res = df_res.append(row, ignore_index=True)
```

同时，为了对数据进行建模，我们决定对model属性进行拆分处理，将其拆分为独立的两个属性：

- room：室的数量
- hall：厅的数量

```
    # 将 model 属性拆分为 室 (room) 和厅 (hall)
    df1 = pd.concat([file, file['model'].str.split('室', expand=True)], axis=1)
    del df1['model']
    df1.rename(columns={0: 'room', 1: 'hall'}, inplace=True)
    file = df1
    word_to_num(file, 'hall')
```

通过筛选、拆分并去除中文文本，我们完成了对model属性的预处理。

area、price_list

area属性和price_list属性在数据集中的呈现形式示例如下：

```
268.73平米
"84,844元/平"
```

可以看出，其呈现方式为数字、汉字、字符混合的方式。因此，对这两个属性预处理的主要工作是将其数据化， 去除其中多余的汉字和字符，将其变为数值型数据。我们通过word_to_num函数完成了这项工作。

```
# 将数字汉字符号混合的原始数据转换为数值
def word_to_num(file, attribute):
    simple_punctuation = '[\" /,)]'
    data = file.loc[:, attribute]
    res_data = []

    for i in data:
        tmp = re.sub('[\u4e00-\u9fa5]', '', i)
        res = re.sub(simple_punctuation, '', tmp)
        res_data.append(res)

    df_res = pd.DataFrame(res_data)
    file[attribute] = df_res

    return file
```

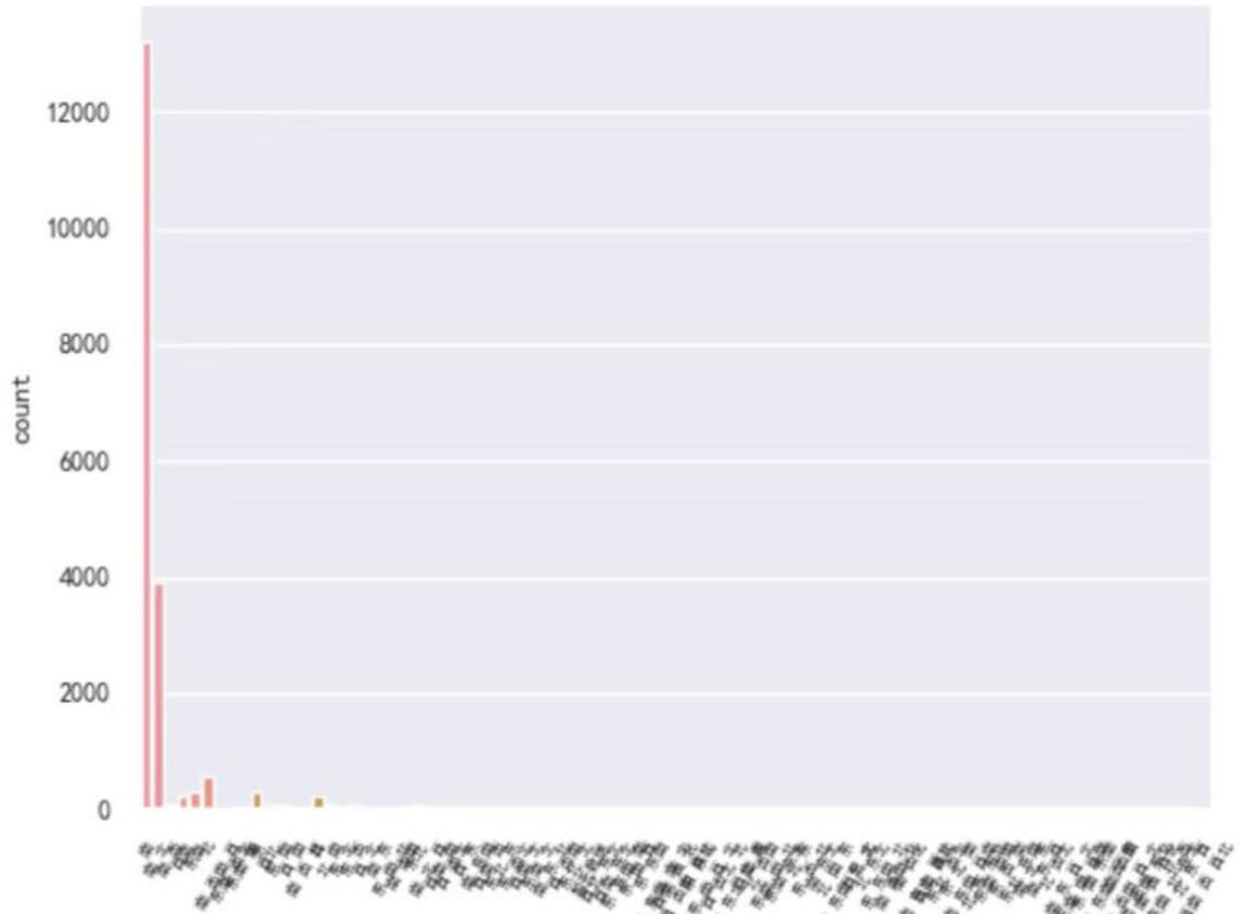
direction

direction属性在数据集中一共有如下出现形式：

```
['南' '南 北' '南 西' '西南' '东南' '北' '南 西南 西' '东南 西北' '东南 南' '东' '西
北' '东 南'
```

```
 '西 南' '南 西 北' '西' '北 南' '东 北' '东 西' '西 北' '南 东' '东 南 北' '南 西南'
'东北'
 '南 北 西' '西南 南' '西 西北' '西 东' '南 东南' '西南 西' '东北 东' '东 西北' '西南
北' '东南 北'
 '南 东 北' '西 西南' '东北 北' '暂无数据' '西北 东北' '西南 东北' '东 东南 南' '东 南
西' '东 东南'
 '东 东南 南 北' '西南 西 北' '西 西南 西北' '东 东北' '东 南 西 北' '西 西北 北' '西南
东北 南 东'
 '西北 西' '北 西南' '东南 南 北' '东南 东北' '北 东' '东 西 北' '南 东北' '北 南 东'
'南 东南 东 北'
 '西北 北' '北 东北' '东 东南 北' '南 西南 北' '南 西北' '东南 东北 西北' '西南 西北'
'东北 西北' '东 北 南'
 '南 东南 西南' '北 东 东北' '东南 西南' '东 西 南' '东北 东南' '南 北 东' '南 东 西'
'北 西北'
 '南 东 西 北' '东南 南 西南' '东 南 东南' '东南 东 南 西南' '南 西 东' '南 西南 西 北'
'东南 西南 东北'
 '南 西 西北' '东 东南 北 东北' '北 西' '北 东 西' '西南 西 西北']
```

显然，direction属性的复杂程度也令我们需要对其进行筛选。我们通过visualize_direction 函数实现了对direction属性的可视化，结果如下：



通过图像我们可以看出，绝大部分数据的direction值属性为南，其他大部分值对应的情况均可以忽略不计。 考虑到购房时对阳光的实际需求，我们决定对direction属性做如下划分：

- direction属性值中包含汉字"南"：置为1
- direction属性值中不包含汉字"南"：置为0

采用函数pre_direction实现了对direction属性的预处理。

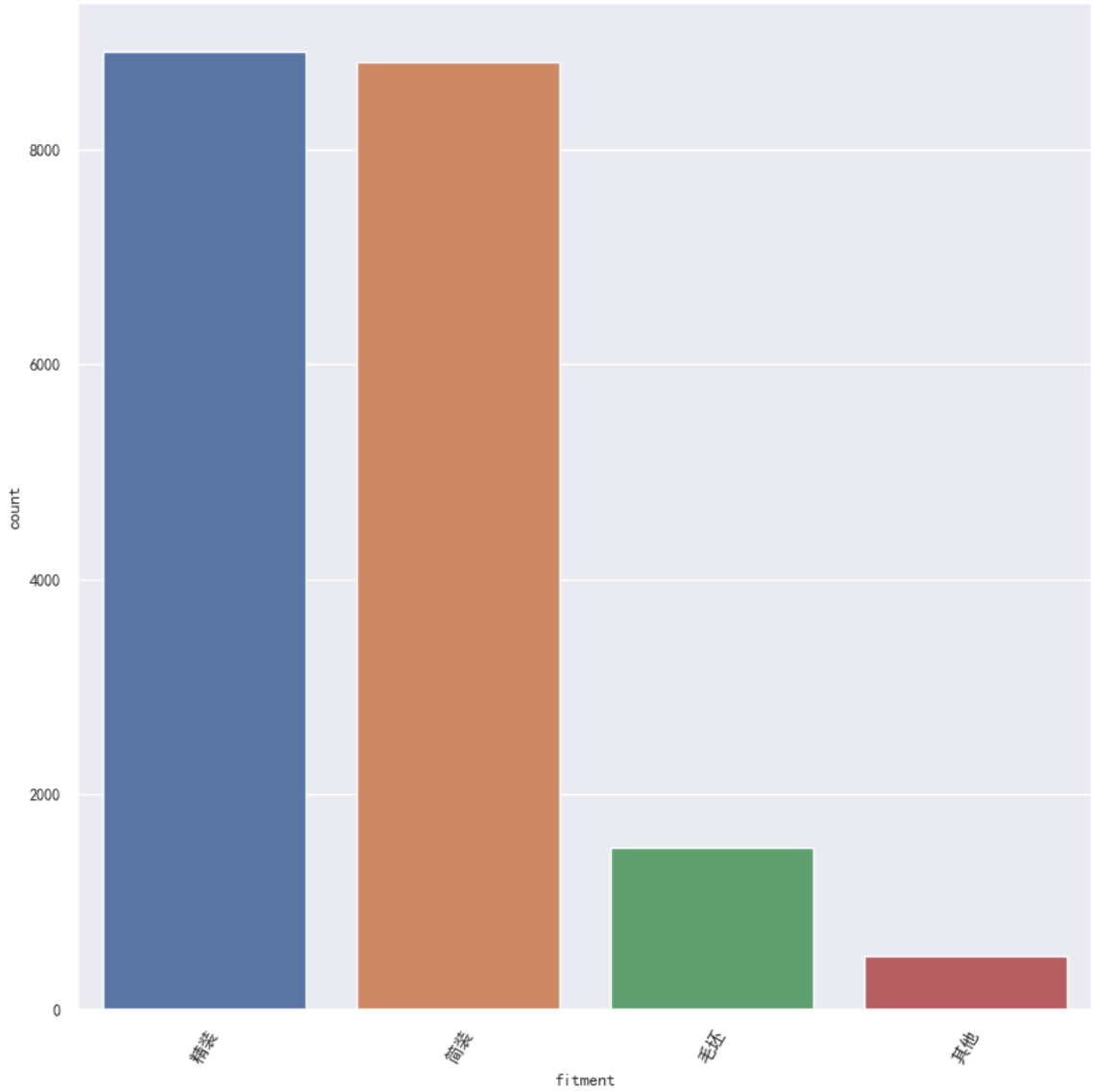
```
# 预处理 direction 属性，凡是有“南“朝向存在均置1，反之置0
def pre_direction(file):
    data = file.loc[:, 'direction']
    res_data = []

    for i in data:
        cnt = 0
        for tmp in i:
            if tmp == '\u5357':  # 识别是否有”南“ (\u5357)
                cnt = cnt + 1
        if cnt >= 1:
            res_data.append(1)
        else:
            res_data.append(0)

    df_res = pd.DataFrame(res_data)
    file['direction'] = df_res

    return file
```

fitment

fitment属性在数据集中一共有以下出现形式：

```
['精装' '简装' '毛坯' '其他']
```

通过可视化函数visualize_fitment观察其分布为：



可以看到不同类型的属性值均有足够多的数据存在，因此不必对数据进行筛选操作。考虑到fitment属性的特征， 不同取值间没有大小关系只有分类的意义，我们选择采用独热码对其进行编码，方便分类器对属性数据进行处理。

```
def pre_fitment(file):
    dummies = pd.get_dummies(file['fitment'])
    df1 = pd.concat([file, dummies], axis=1)
    del df1['fitment']
    df1.rename(columns={'精装': 'jingzhuang', '简装': 'jianzhuang', '毛坯':
'maopi', '其他': 'qita'}, inplace=True)
    file = df1

    return file
```

编码后，原fitment属性被移除，数据集中增加如下四个新属性：

- jingzhuang：代表精装房
- jianzhuang：代表简装房
- maopi：代表毛坯房
- qita：代表其他

floor

floor属性在数据集中一共有以下出现形式：(仅选取部分)

```
['下叠(共5层)' '15层' '低楼层(共20层)' '高楼层(共6层)' '低楼层(共14层)' '高楼层(共4
层)' '高楼层(共12层)'
 '中楼层(共14层)' '中楼层(共4层)' '中楼层(共6层)' '低楼层(共7层)' '低楼层(共6层)' '低楼
层(共5层)'
 '中楼层(共18层)' '低楼层(共18层)' '高楼层(共18层)' '低楼层(共11层)' '低楼层(共12层)'
'高楼层(共7层)'
 '低楼层(共19层)' '14层' '低楼层(共17层)' '低楼层(共13层)' '低楼层(共8层)' '高楼层(共
11层)'
 '低楼层(共21层)' '低楼层(共25层)' '中楼层(共5层)' '低楼层(共31层)' '高楼层(共5层)'
'中楼层(共12层)'
 '中楼层(共24层)' '低楼层(共16层)' '中楼层(共17层)' '高楼层(共24层)' '高楼层(共14层)'
'24层' '17层'
 '中楼层(共8层)' '高楼层(共16层)' '低楼层(共28层)' '20层' '高楼层(共13层)' '中楼层(共
11层)'
 '低楼层(共27层)' '高楼层(共8层)' '中楼层(共7层)' '中楼层(共15层)' '低楼层(共24层)'
'16层'
 '中楼层(共13层)' '中楼层(共19层)' '低楼层(共15层)' '中楼层(共20层)' '6层' '低楼层(共
26层)' '18层'
 '中楼层(共9层)' '低楼层(共41层)' '49层' '低楼层(共3层)' '低楼层(共4层)' '中楼层(共30
层)' '41层'
 ……
```

由于篇幅原因，下面还有超过百分之50的数据被省略掉了。我们可以发现，floor数据的出现是有规律的，一般以以下格式出现：

```
X楼层(共X层)
```

因此我们通过pre_floor函数对数据做拆分处理，将floor拆分为height和max_height两个属性：

- height：存储前半部分"X楼层"数据
- max_height：存储后半部分"共X层数据"

拆分完毕后，对height做独热码处理，对max_height进行word_to_num处理，完成对floor的预处理

```
# 将floor属性预处理，分成 height 和 max_height 两个属性，对 height做独热码
def pre_floor(file):
    file = file[file['floor'].str.contains('楼层')]
```

```
    file = file.reset_index()
    del file['index']
    # file['tmp'] = range(len(file))

    # file1 = pd.read_csv('floor.csv', encoding='utf-8')
    # print(file1['floor'].unique())

    df1 = pd.concat([file, file['floor'].str.split('(', expand=True)], axis=1)
    del df1['floor']
    df1.rename(columns={0: 'height', 1: 'max_height'}, inplace=True)
    file = df1
    file = word_to_num(file, 'max_height')

    dummies = pd.get_dummies(file['height'])
    df2 = pd.concat([file, dummies], axis=1)
    del df2['height']
    df2.rename(columns={'低楼层': 'low', '中楼层': 'medium', '高楼层': 'high'},
inplace=True)
    file = df2
    # del file['tmp']
    file.to_csv("new.csv", encoding='utf-8', index=False)

    return file
```

address

address属性的取值情况比较简单，是我们在爬取数据时主动指定的：

```
'浦东': 'pudong', '嘉定': 'jiading', '金山': 'jinshan', '虹口': 'hongkou', '静安':
'jingan',
'黄浦': 'huangpu', '闵行': 'minhang', '宝山': 'baoshan', '徐汇': 'xuhui', '普陀':
'putuo',
'杨浦': 'yangpu', '长宁': 'changning', '松江': 'songjiang', '青浦': 'qingpu', '奉
贤': 'fengxian','崇明':'chongming'
```

对address属性的处理也较为简单，直接对其进行独热码编码：

```
def pre_address(file):
    dummies = pd.get_dummies(file['address'])
    df1 = pd.concat([file, dummies], axis=1)
    del df1['address']
    df1.rename(columns={'浦东': 'pudong', '嘉定': 'jiading', '金山': 'jinshan', '虹
口': 'hongkou', '静安': 'jingan',
                        '黄浦': 'huangpu', '闵行': 'minhang', '宝山': 'baoshan',
'徐汇': 'xuhui', '普陀': 'putuo',
                        '杨浦': 'yangpu', '长宁': 'changning', '松江': 'songjiang',
'青浦': 'qingpu', '奉贤': 'fengxian',
                        '崇明':'chongming'}, inplace=True)
    file = df1
```

```
    return file
```

total_list

观察发现，原始数据中total_list属性的值均为数据值，并不需要预处理即可直接使用。

数据建模、调优和测试

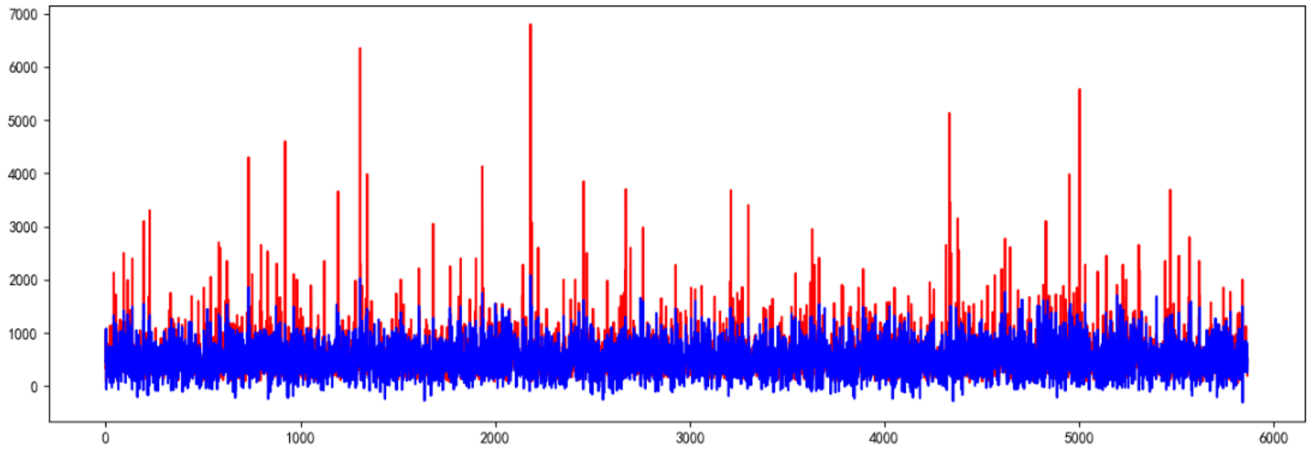
数据建模

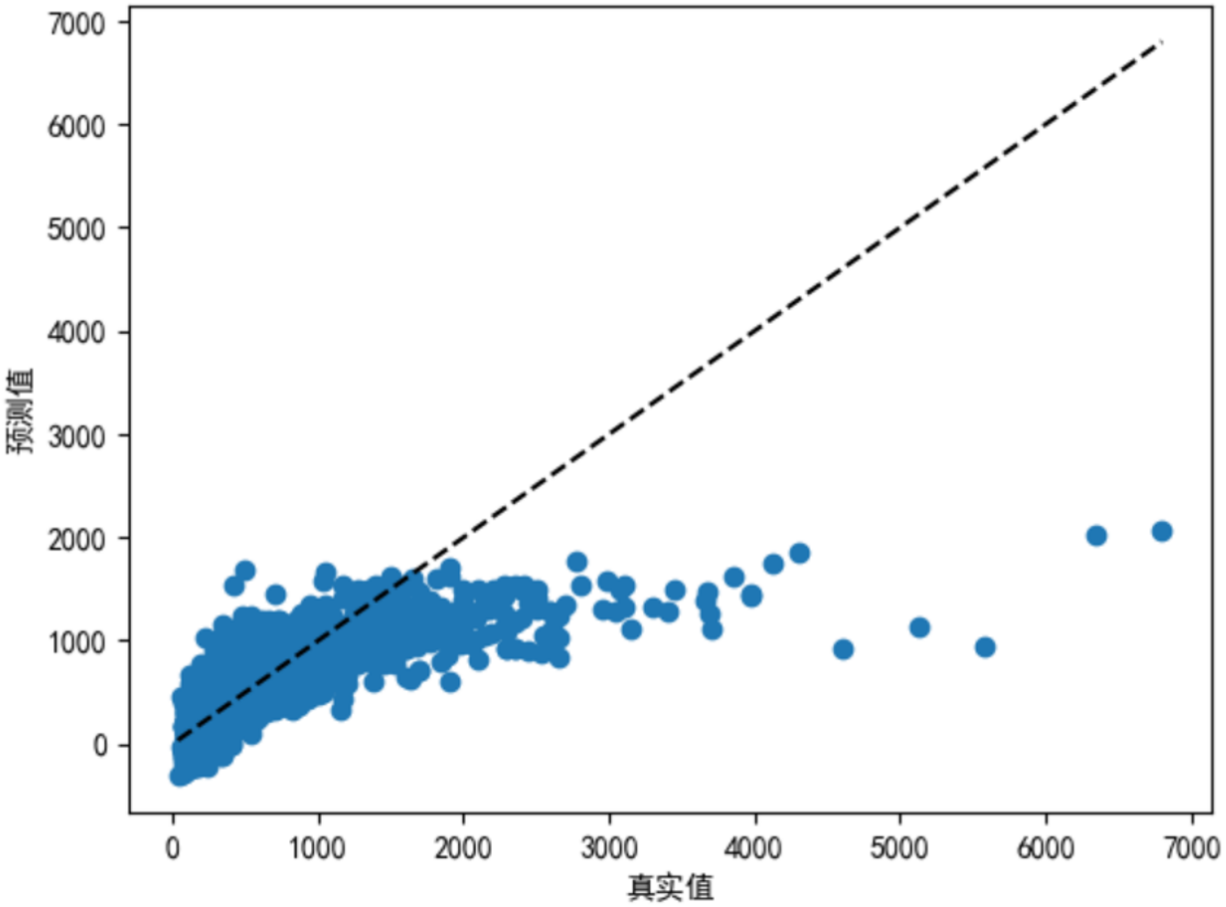
在数据建模上，我们选择采用sklearn库中的LinearRegression线性回归模型进行建模。具体到代码中，封装成为函数train：

```
def train(path):
    file = pd.read_csv(path, encoding='utf-8')
    file.drop(file.tail(1).index, inplace=True)
    del file['name']
    del file['floor']

    # 平滑处理y
    y = np.log1p(file['total_list'])
    del file['total_list']
    # file = preprocessing.scale(file)
    x = file

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=1)
    # print(x_train.shape)
    # print(x_test.shape)

    lr = LinearRegression()
    lr.fit(x_train, y_train)  # 把训练集的自变量，因变量添加到函数fit()中，进行训练
    # print(lr.coef_)  # lr.coef_得到的是每个自变量的权重系数
    # print(lr.intercept_)  # lr.intercept_得到的是截距

    y_pred = lr.predict(x_test)

    MSE = metrics.mean_squared_error(y_test, y_pred)
    RMSE = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    Rsquare = metrics.r2_score(y_test, y_pred)

    print("Train success! \nSome results :")
    print('MSE:', MSE)
    print('RMSE:', RMSE)
    print('Rsquare:', Rsquare)


    return file
```

模型调优

初次对模型进行训练，我们得到的结果可视化如下：



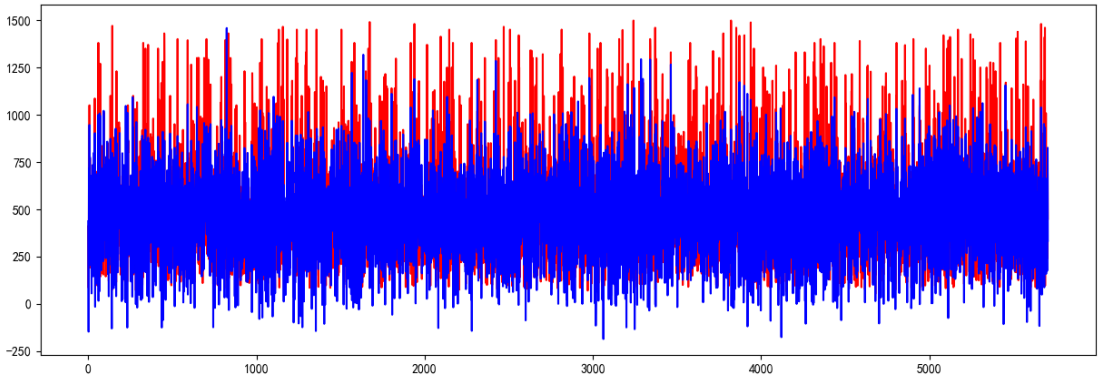


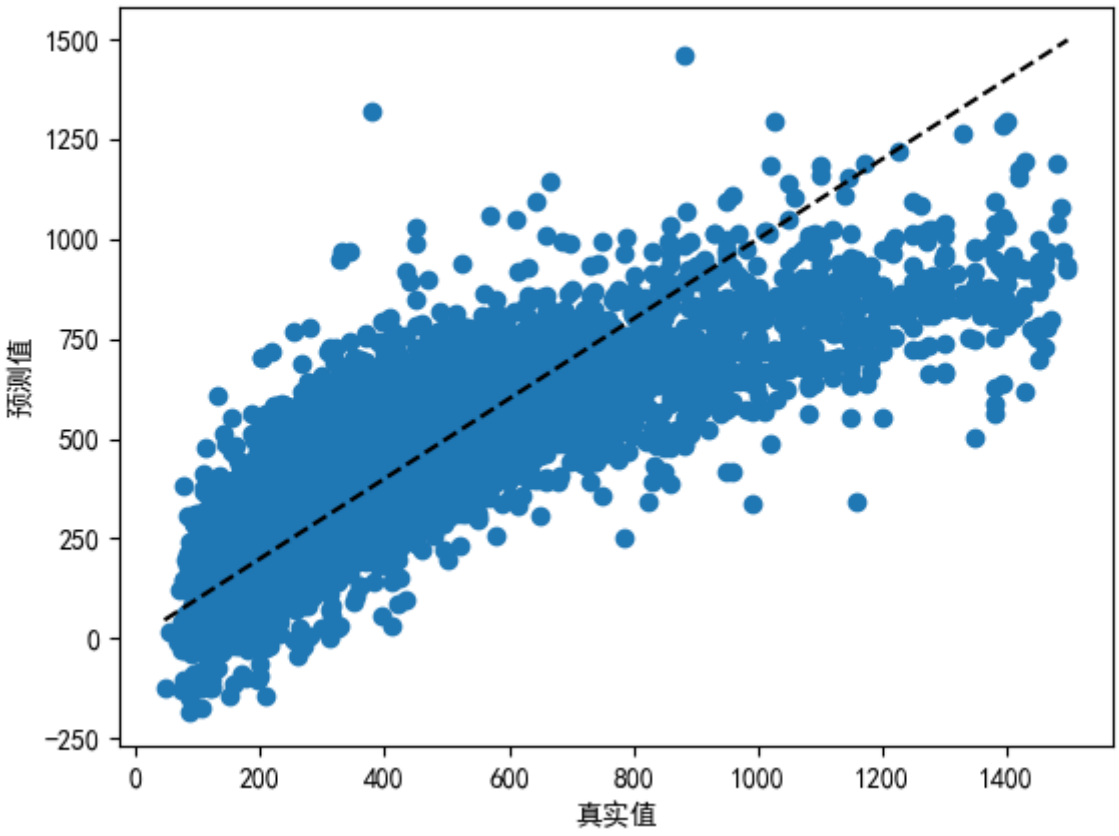
通过可视化图像可以看出，有大量偏离回归的噪点，主要集中在真实值>2000的范围内。这些噪点影响了模型的准确性。 在发现模型存在噪点后，我们决定对total_list进行筛选，筛去总价过高的二手房。最后，经过统计分析，我们选择 筛去了所有total_list>=1500的二手房，大概占总体数据量的1%。具体删去的代码集成在model属性的处理实现。

```
    for index, row in file.iterrows():
        if ((row['model'] == '4室3厅' or row['model'] == '4室2厅' or row['model']
== '3室2厅' or
                row['model'] == '1室1厅' or row['model'] == '2室1厅' or
```

```
    row['model'] == '3室1厅' or
                    row['model'] == '5室2厅' or row['model'] == '2室2厅' or
    row['model'] == '2室0厅' or
                    row['model'] == '1室0厅' or row['model'] == '1室2厅' or
    row['model'] == '5室3厅')
                    and (float(row['total_list']) < 1500)):
                df_res = df_res.append(row, ignore_index=True)
```

删去后得到的结果可视化如下：

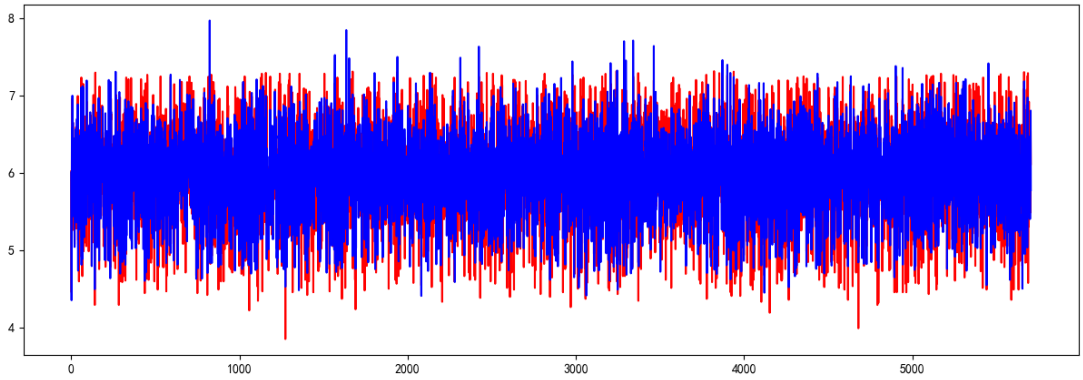


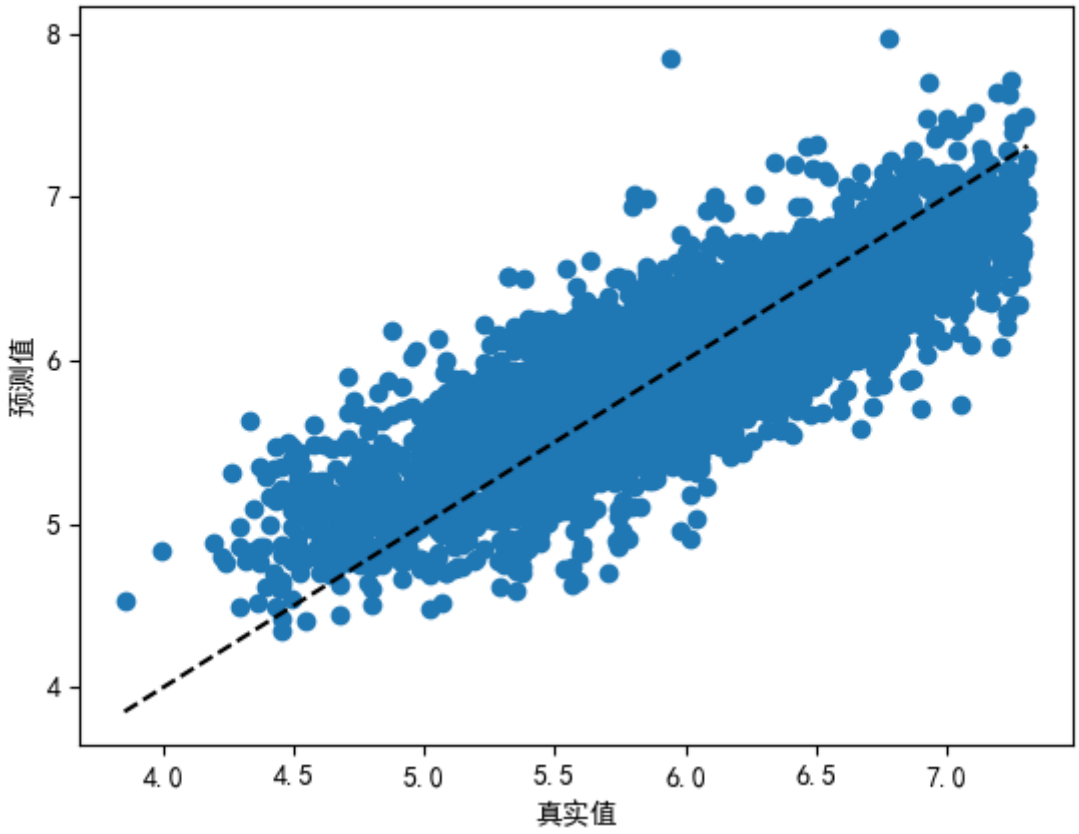


此时，得出的结果拟合程度仍然不够好。经过对数据进行分析，我们选择对total_list值进行平滑处理，尽可能减少 噪声对于模型的影响：

```
    # 平滑处理y
    y = np.log1p(file['total_list'])
```

平滑处理后，得到的结果可视化如下：





可以看到，此时模型的拟合程度相较调优之初有了很大提升。

模型测试

运行main.py对模型进行测试，得到输出结果如下：

```
MSE: 0.10027270238502037
RMSE: 0.3166586527872251
Rsquare: 0.7155737941932878
```

总结

在学习数据挖掘这门课程之前,我们小组的大部分成员对数据挖掘领域的相关知识都是懵懂的，也对数据挖掘技术的 广泛应用没有深入的了解。在学习了这门课程后，我们惊叹于数据挖掘技术的强大与丰富，在老师的讲述下第一次 体会到了数据挖掘的魅力，这也激发了我们进一步探索数据挖掘领域的兴趣。

来到课程设计部分，在学习了老师提供的课程讲述和学习材料后，我们选择了对链家网二手房数据进行分析挖掘， 以期对上海市二手房价进行预测。在设计实现过程中，我们通力合作、技术互补，按照既定的分工有条 不紊的完成了分配的工作。由于疫情客观条件的限制，我们无法进行线下的合作与开发，这无疑为我们的课程设计带 来了极大的挑战。面对空间的隔阂，我们选择在线上利用视频会议软件定期召开开发会议，交流前一阶段完成的工作 和碰到的困难，一起制定下一阶段的开发计划。经过我们的不懈努力，小组的课程设计在既定时间内圆满完成，小组 的全部成员都为之欢欣鼓舞。

当然，在开发过程中，遇到的最大难关还是技术问题。我们在阅读了大量学习资料的基础上集思广益、开拓思路，完 成了数据预处理中最关键的几个函数的逻辑设计与具体实现。在设计实现过程中，我们也遇到过一系列问题。这一系 列难关虽然短暂的困扰了我们，但最终都在我们的通力合作之下被克服了。在攻克难关的过程中，我们也学到了很多 宝贵的知识，这大概就是课程设计的魅力所在吧。

最后,感谢回老师和数据挖掘这门课程提供给我们宝贵的学习机会，让我们得以深入了解数据挖掘背后的技术与应 用，让我们感受到数据挖掘技术的迷人魅力，谢谢!