入门 python 数据分析最好的实战项目（一）

作者：xiaoyu　　知乎：python 数据分析师

数据集下载：

链接: https://pan.baidu.com/s/178DXNbeTIrAbHzvuejEShg 提取码: g5h5

数据初探

首先导入要使用的科学计算包 `numpy`,`pandas`,可视化 `matplotlib`,`seaborn`,以及机器学习包 `sklearn`。

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib as mpl



import matplotlib.pyplot as plt

from IPython.display import display

plt.style.use("fivethirtyeight")

sns.set_style({'font.sans-serif':['simhei','Arial']})

%matplotlib inline
```

```
# 检查 Python 版本
from sys import version_info
if version_info.major != 3:
    raise Exception('请使用 Python 3 来完成此项目')
```

然后导入数据，并进行初步的观察，这些观察包括了解数据特征的**缺失值**，**异常值**，以及大概的**描述性统计**。

```
# 导入链家二手房数据
lianjia_df = pd.read_csv('lianjia.csv')
display(lianjia_df.head(n=2))
```

	Direction	District	Elevator	Floor	Garden	Id	Layout	Price	Region	Renovation	Size	Year
0	东西	灯市口	NaN	6	锡拉胡同21号院	101102647043	3室1厅	780.0	东城	精装	75.0	1988
1	南北	东单	无电梯	6	东华门大街	101102650978	2室1厅	705.0	东城	精装	60.0	1988

初步观察到一共有 `11` 个特征变量，`Price` 在这里是我们的目标变量，然后我们继续深入观察一下。

```
# 检查缺失值情况
lianjia_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23677 entries, 0 to 23676
Data columns (total 12 columns):
Direction     23677 non-null object
District      23677 non-null object
Elevator      15440 non-null object
Floor         23677 non-null int64
Garden        23677 non-null object
Id            23677 non-null int64
Layout        23677 non-null object
Price         23677 non-null float64
Region        23677 non-null object
Renovation    23677 non-null object
Size          23677 non-null float64
Year          23677 non-null int64
dtypes: float64(2), int64(3), object(7)
memory usage: 2.2+ MB
```

发现了数据集一共有 `23677` 条数据，其中 `Elevator` 特征有明显的缺失值。

```
lianjia_df.describe()
```

	Floor	Id	Price	Size	Year
count	23677.000000	2.367700e+04	23677.000000	23677.000000	23677.000000
mean	12.765088	1.011024e+11	610.668319	99.149301	2001.326519
std	7.643932	5.652477e+05	411.452107	50.988838	9.001996
min	1.000000	1.010886e+11	60.000000	2.000000	1950.000000
25%	6.000000	1.011022e+11	365.000000	66.000000	1997.000000
50%	11.000000	1.011025e+11	499.000000	88.000000	2003.000000
75%	18.000000	1.011027e+11	717.000000	118.000000	2007.000000
max	57.000000	1.011028e+11	6000.000000	1019.000000	2017.000000

上面结果给出了特征值是数值的一些统计值，包括**平均数**，**标准差**，**中位数**，**最小值**，**最大值**，**25%分位数**，**75%分位数**。这些统计结果简单直接，对于初始了解一个特征好坏非常有用，比如我们观察到 `Size` 特征的最大值为 1019 平米，最小值为 2 平米，那么我们就要思考这个在实际中是不是存在的，如果不存在没有意义，那么这个数据就是一个异常值，会严重影响模型的性能。

当然，这只是初步观察，后续我们会用数据可视化来清晰的展示，并证实我们的猜测。

```
# 添加新特征房屋均价

df = lianjia_df.copy()

df['PerPrice'] = lianjia_df['Price']/lianjia_df['Size']
```

```
# 重新摆放列位置

columns = ['Region', 'District', 'Garden', 'Layout', 'Floor', 'Year',
 'Size', 'Elevator', 'Direction', 'Renovation', 'PerPrice', 'Price']

df = pd.DataFrame(df, columns = columns)
```

```
# 重新审视数据集

display(df.head(n=2))
```

我们发现 `Id` 特征其实没有什么实际意义，所以将其移除。由于房屋单价分析起来比较方便，简单的使用总价/面积就可得到，所以增加一个新的特征 `PerPrice`（只用于分析，不是预测特征）。另外，特征的顺序也被调整了一下，看起来比较舒服。

	Region	District	Garden	Layout	Floor	Year	Size	Elevator	Direction	Renovation	PerPrice	Price
0	东城	灯市口	锡拉胡同21号院	3室1厅	6	1988	75.0	NaN	东西	精装	10.40	780.0
1	东城	东单	东华门大街	2室1厅	6	1988	60.0	无电梯	南北	精装	11.75	705.0

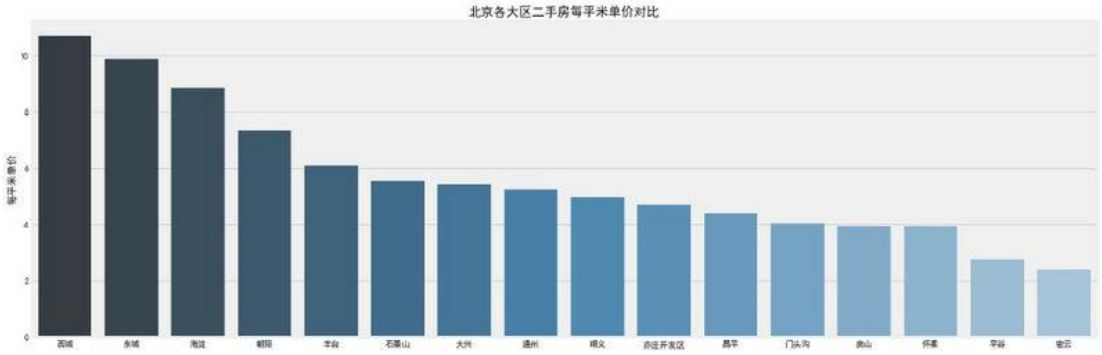
数据可视化分析

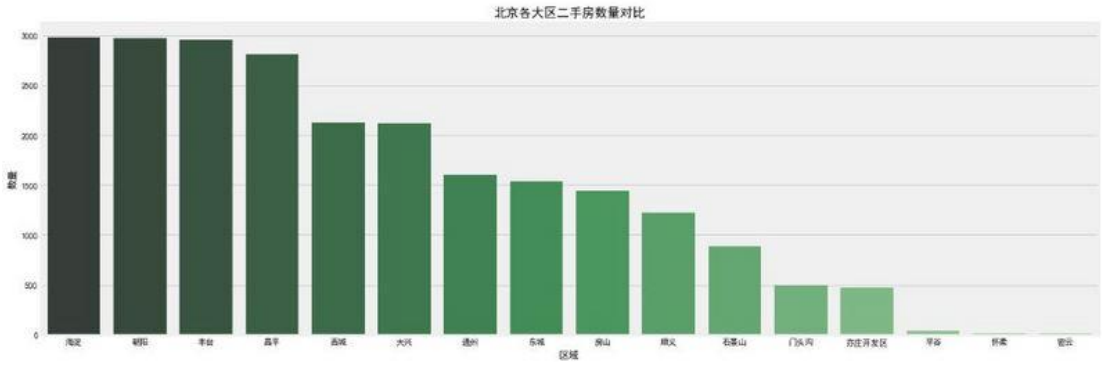
Region 特征分析

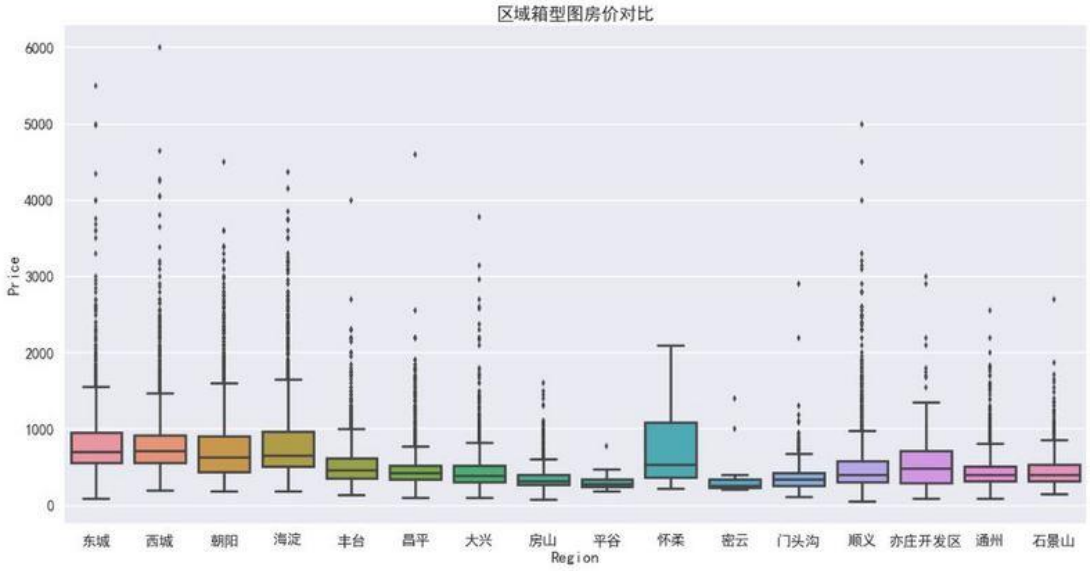
对于区域特征，我们可以分析不同区域房价和数量的对比。

```
# 对二手房区域分组对比二手房数量和每平米房价

df_house_count = df.groupby('Region')['Price'].count().sort_values(a
scending=False).to_frame().reset_index()

df_house_mean = df.groupby('Region')['PerPrice'].mean().sort_values
(ascending=False).to_frame().reset_index()



f, [ax1,ax2,ax3] = plt.subplots(3,1,figsize=(20,15))

sns.barplot(x='Region', y='PerPrice', palette="Blues_d", data=df_hou
se_mean, ax=ax1)

ax1.set_title('北京各大区二手房每平米单价对比',fontsize=15)

ax1.set_xlabel('区域')

ax1.set_ylabel('每平米单价')



sns.barplot(x='Region', y='Price', palette="Greens_d", data=df_house
_count, ax=ax2)

ax2.set_title('北京各大区二手房数量对比',fontsize=15)

ax2.set_xlabel('区域')

ax2.set_ylabel('数量')



sns.boxplot(x='Region', y='Price', data=df, ax=ax3)
```

```
ax3.set_title('北京各大区二手房房屋总价',fontsize=15)

ax3.set_xlabel('区域')

ax3.set_ylabel('房屋总价')


plt.show()
```







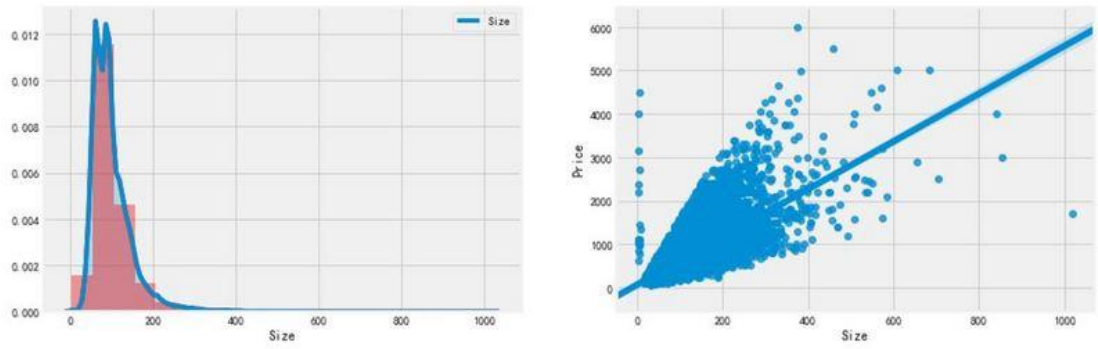
使用了 `pandas` 的网络透视功能 `groupby` 分组排序。区域特征可视化直接采用 `seaborn` 完成，颜色使用调色板 `palette` 参数，颜色渐变，越浅说明越少，反之越多。

可以观察到：

- **二手房均价**：西城区的房价最贵均价大约 11 万/平，因为西城在二环以里，且是热门学区房的聚集地。其次是东城大约 10 万/平，然后是海淀大约 8.5 万/平，其它均低于 8 万/平。
- **二手房房数量**：从数量统计上来看，目前二手房市场上比较火热的区域。海淀区和朝阳区二手房数量最多，差不多都接近 3000 套，毕竟大区，需求量也大。然后是丰台区，近几年正在改造建设，有赶超之势。
- **二手房总价**：通过箱型图看到，各大区域房屋总价中位数都都在 1000 万以下，且房屋总价离散值较高，西城最高达到了 6000 万，说明房屋价格特征不是理想的正太分布。

Size 特征分析

```
f, [ax1,ax2] = plt.subplots(1, 2, figsize=(15, 5))

# 建房时间的分布情况

sns.distplot(df['Size'], bins=20, ax=ax1, color='r')

sns.kdeplot(df['Size'], shade=True, ax=ax1)

# 建房时间和出售价格的关系

sns.regplot(x='Size', y='Price', data=df, ax=ax2)

plt.show()
```



- **Size 分布**:

通过 `distplot` 和 `kdeplot` 绘制柱状图观察 Size 特征的分布情况，属于长尾类型的分布，这说明了有很多面积很大且超出正常范围的二手房。

- **Size 与 Price 的关系**:

通过 `regplot` 绘制了 Size 和 Price 之间的散点图，发现 Size 特征基本与 Price 呈现线性关系，符合基本常识，面积越大，价格越高。但是有两组明显的异常点：

1. 面积不到 10 平米，但是价格超出 10000 万；2. 一个点面积超过了 1000 平米，价格很低，需要查看是什么情况。

```
df.loc[df['Size']< 10]
```

	Region	District	Garden	Layout	Floor	Year	Size	Elevator	Direction	Renovation	PerPrice	Price
1168	房山	长阳	世茂维拉	叠拼别墅	5	2015	5.0	无电梯	240.97平米	NaN	216.000000	1080.0
1458	房山	长阳	世茂维拉	叠拼别墅	5	2015	5.0	无电梯	242.78平米	NaN	220.000000	1100.0
1797	房山	长阳	世茂维拉	叠拼别墅	5	2015	5.0	无电梯	242.96平米	NaN	196.000000	980.0
2268	顺义	顺义其它	龙湖好望山	叠拼别墅	4	2014	4.0	无电梯	295.88平米	NaN	250.000000	1000.0
2274	顺义	顺义其它	骏豪国际	叠拼别墅	4	2014	5.0	无电梯	295.01平米	NaN	290.000000	1450.0
2276	顺义	顺义其它	龙湖好望山	叠拼别墅	3	2014	4.0	无电梯	292.31平米	NaN	215.000000	860.0
2432	顺义	顺义其它	龙湖好望山	叠拼别墅	5	2013	6.0	无电梯	294.42平米	NaN	163.333333	980.0
4078	大兴	西红门	鸿坤林语墅	叠拼别墅	3	2015	4.0	无电梯	427.5平米	NaN	787.500000	3150.0
4079	大兴	西红门	鸿坤林语墅	叠拼别墅	4	2015	4.0	无电梯	361.8平米	NaN	595.000000	2380.0
4761	大兴	西红门	鸿坤林语墅	叠拼别墅	3	2015	5.0	无电梯	386.83平米	NaN	540.000000	2700.0
7533	昌平	回龙观	龙城花园N区	叠拼别墅	4	1997	2.0	无电梯	107.93平米	NaN	310.000000	620.0
8765	通州	通州其它	旭辉御锦	叠拼别墅	6	2014	5.0	无电梯	195.32平米	NaN	156.000000	780.0
9020	通州	通州其它	旭辉御锦	叠拼别墅	6	2014	4.0	无电梯	259.87平米	NaN	280.000000	1120.0
9080	通州	通州其它	旭辉御锦	叠拼别墅	6	2014	4.0	无电梯	259.76平米	NaN	262.500000	1050.0
9203	通州	通州其它	旭辉御锦	叠拼别墅	6	2014	4.0	无电梯	260.07平米	NaN	262.500000	1050.0
9254	通州	通州其它	旭辉御锦	叠拼别墅	6	2014	4.0	无电梯	264.6平米	NaN	275.000000	1100.0
11531	丰台	丽泽	西宸原著	叠拼别墅	6	2016	4.0	无电梯	335.51平米	NaN	1000.000000	4000.0
14298	海淀	西山	中间建筑一区	叠拼别墅	3	2007	8.0	无电梯	266.61平米	NaN	168.750000	1350.0
15334	海淀	西山	西山美墅馆F区	叠拼别墅	4	2004	4.0	无电梯	203.73平米	NaN	550.000000	2200.0
17311	朝阳	大望路	首府官邸	叠拼别墅	5	2007	5.0	无电梯	523.4平米	NaN	900.000000	4500.0

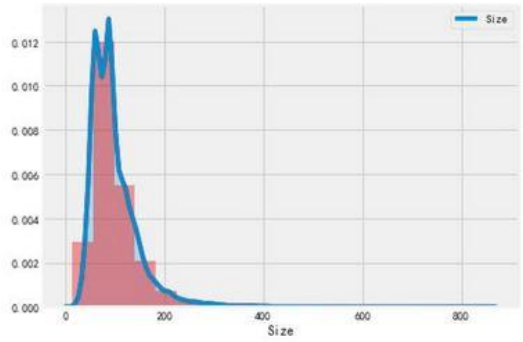
经过查看发现这组数据是别墅，出现异常的原因是由于别墅结构比较特殊（无朝向无电梯），字段定义与二手商品房不太一样导致爬虫爬取数据错位。也因别墅类型二手房不在我们的考虑范围之内，故将其移除再次观察 Size 分布和 Price 关系。

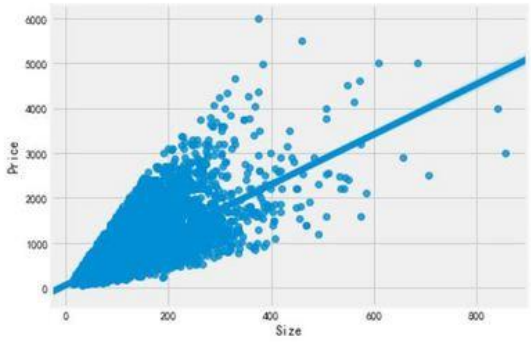
```
df.loc[df['Size']>1000]
```

	Region	District	Garden	Layout	Floor	Year	Size	Elevator	Direction	Renovation	PerPrice	Price
8754	通州	通州其它	新华联科技大厦	1房间0卫	8	2009	1019.0	有电梯	南	简装	1.668302	1700.0

经观察这个异常点不是普通的民用二手房，很可能是商用房，所以才有 1 房间 0 厅确有如此大超过 1000 平米的面积，这里选择移除。

```
df = df[(df['Layout']!='叠拼别墅')&(df['Size']<1000)]
```

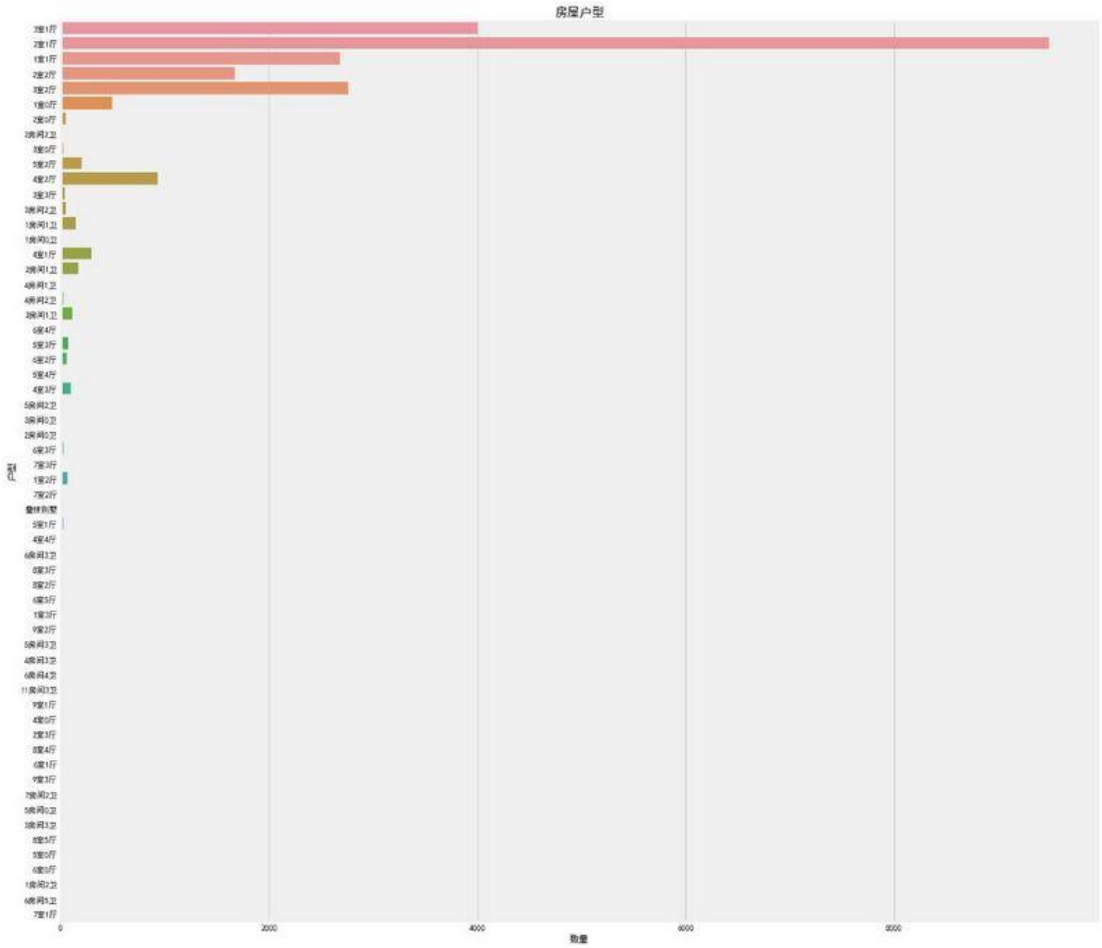




重新进行可视化发现就没有明显的异常点了。

Layout 特征分析

```
f, ax1= plt.subplots(figsize=(20,20))

sns.countplot(y='Layout', data=df, ax=ax1)

ax1.set_title('房屋户型',fontsize=15)

ax1.set_xlabel('数量')

ax1.set_ylabel('户型')

plt.show()
```



这个特征真是不看不知道，各种厅室组合搭配，竟然还有 9 室 3 厅，4 室 0 厅等奇怪的结构。其中，2 室一厅占绝大部分，其次是 3 室一厅，2 室 2 厅，3 室两厅。但是仔细观察特征分类下有很多不规则的命名，比如 2 室一厅与 2 房间 1 卫，还有别墅，没有统一的叫法。这样的特征肯定是不能作为机器学习模型的数据输入的，需要使用特征工程进行相应的处理。

Renovation 特征分析

```
df['Renovation'].value_counts()
```

精装 11345

简装 8497

其他 3239

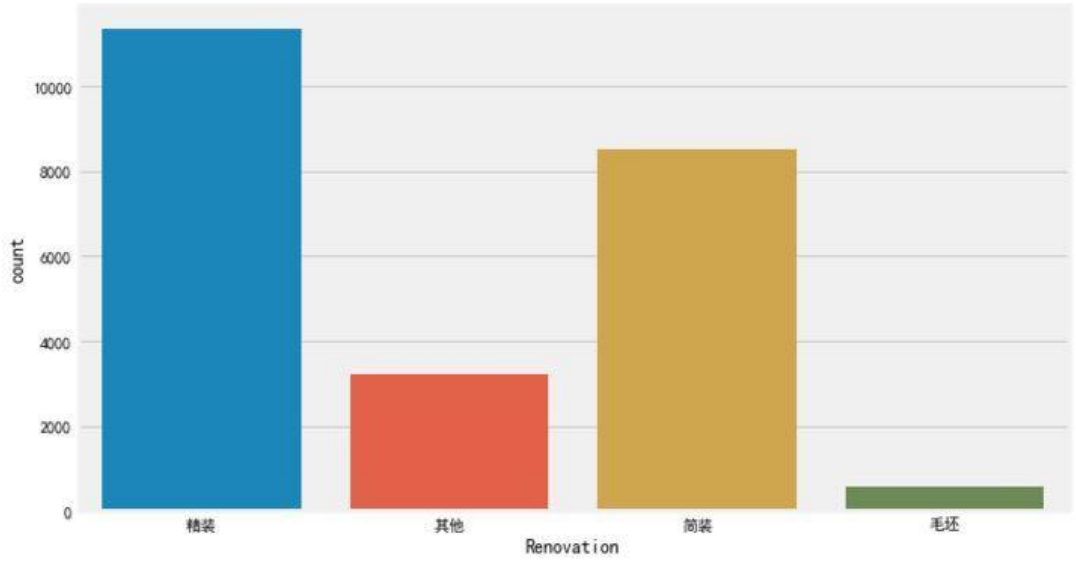
毛坯 576

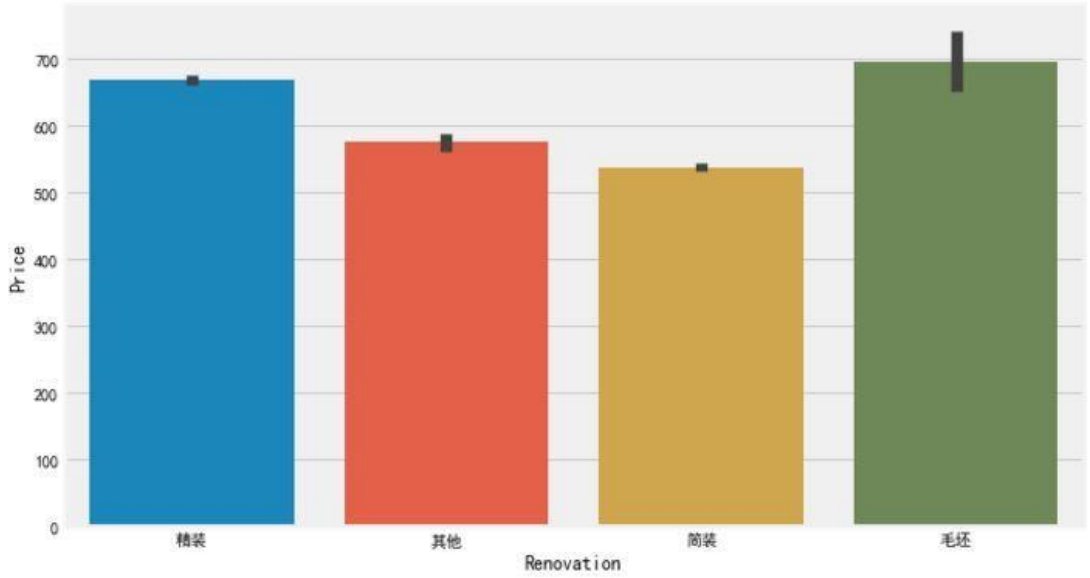
南北 20

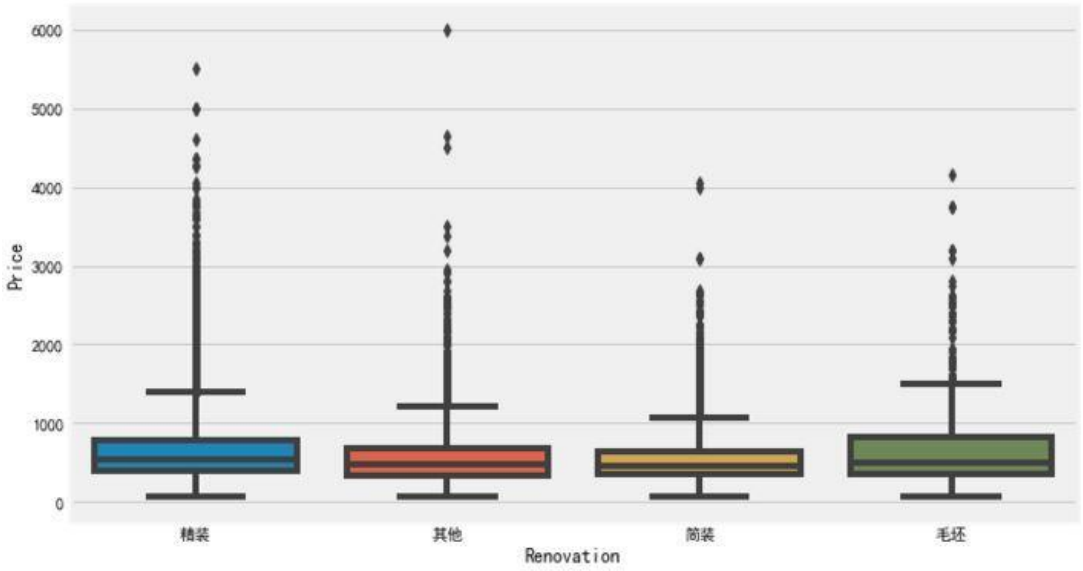
Name: Renovation, dtype: int64

发现 Renovation 装修特征中竟然有南北，它属于朝向的类型，可能是因为爬虫过程中一些信息位置为空，导致“Direction”朝向特征出现在这里，所以需要清除或替换掉。

```
# 去掉错误数据“南北”，因为爬虫过程中一些信息位置为空，导致“Direction”的特征出现在这里，需要清除或替换

df['Renovation'] = df.loc[(df['Renovation'] != '南北'), 'Renovation']



# 画幅设置

f, [ax1,ax2,ax3] = plt.subplots(1, 3, figsize=(20, 5))

sns.countplot(df['Renovation'], ax=ax1)

sns.barplot(x='Renovation', y='Price', data=df, ax=ax2)

sns.boxplot(x='Renovation', y='Price', data=df, ax=ax3)

plt.show()
```


count
10000
8000
6000
4000
2000
0
精装
其他
简装
毛坯
Renovation


Price
700
600
500
400
300
200
100
0
精装
其他
简装
毛坯
Renovation



观察到，精装修的二手房数量最多，简装其次，也是我们平日常见的。而对于价格来说，毛坯类型却是最高，其次是精装修。

Elevator 特征分析

初探数据的时候，我们发现 **Elevator** 特征是有大量缺失值的，这对于我们是十分不利的，首先我们先看看有多少缺失值：

```
misn = len(df.loc[(df['Elevator'].isnull()), 'Elevator'])
print('Elevator 缺失值数量为：'+ str(misn))
```

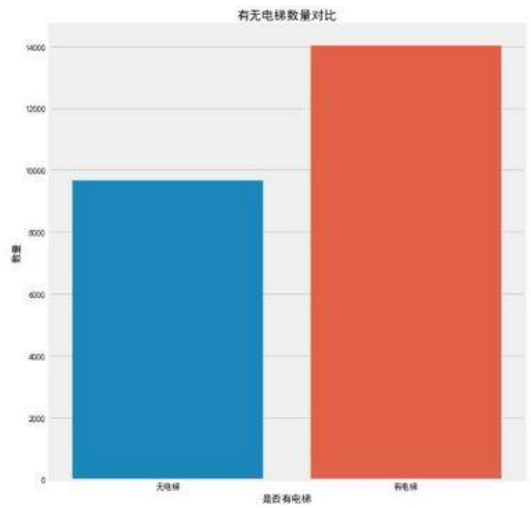
Elevator 缺失值数量为：`8237`

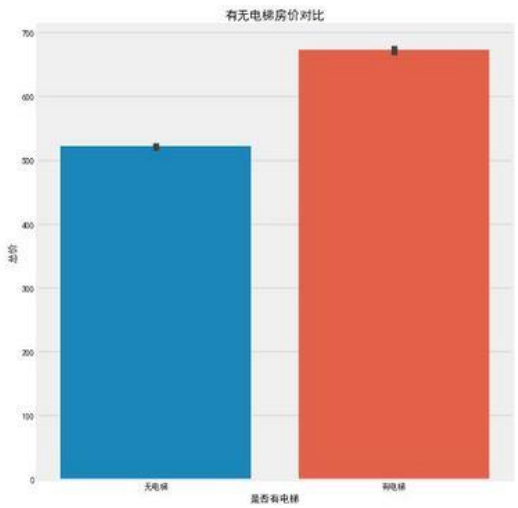
这么多的缺失值怎么办呢？这个需要根据实际情况考虑，常用的方法有**平均值/中位数填补法，直接移除，或者根据其他特征建模预测等。**

这里我们考虑填补法，但是有无电梯不是数值，不存在平均值和中位数，怎么填补呢？这里给大家提供一种思路：**就是根据楼层 Floor 来判断有无电梯，一般的楼**

层大于 6 的都有电梯，而小于等于 6 层的一般都没有电梯。有了这个标准，那么剩下的就简单了。

```
# 由于存在个别类型错误，如简装和精装，特征值错位，故需要移除

df['Elevator'] = df.loc[(df['Elevator'] == '有电梯')|(df['Elevator'] == '无电梯'), 'Elevator']



# 填补 Elevator 缺失值

df.loc[(df['Floor']>6)&(df['Elevator'].isnull()), 'Elevator'] = '有电梯'

df.loc[(df['Floor']<=6)&(df['Elevator'].isnull()), 'Elevator'] = '无电梯'



f, [ax1,ax2] = plt.subplots(1, 2, figsize=(20, 10))

sns.countplot(df['Elevator'], ax=ax1)

ax1.set_title('有无电梯数量对比',fontsize=15)

ax1.set_xlabel('是否有电梯')

ax1.set_ylabel('数量')

sns.barplot(x='Elevator', y='Price', data=df, ax=ax2)

ax2.set_title('有无电梯房价对比',fontsize=15)

ax2.set_xlabel('是否有电梯')

ax2.set_ylabel('总价')

plt.show()
```

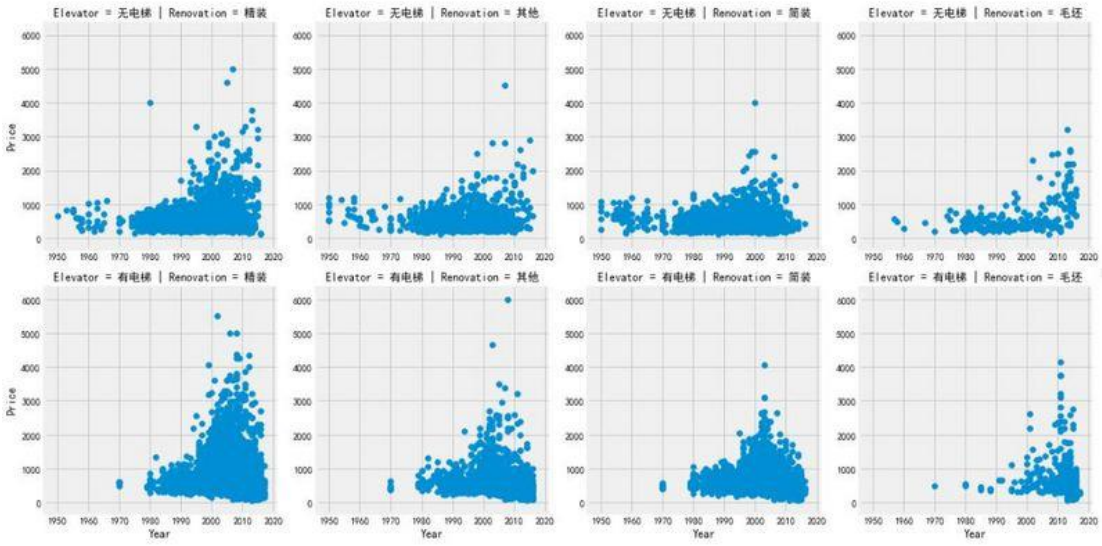




结果观察到，有电梯的二手房数量居多一些，毕竟高层土地利用率比较高，适合北京庞大的人群需要，而高层就需要电梯。相应的，有电梯二手房房价较高，因为电梯前期装修费和后期维护费包含内了（但这个价格比较只是一个平均的概念，比如无电梯的 6 层豪华小区当然价格更高了）。

Year 特征分析

```
grid = sns.FacetGrid(df, row='Elevator', col='Renovation', palette='seismic',size=4)

grid.map(plt.scatter, 'Year', 'Price')

grid.add_legend()
```



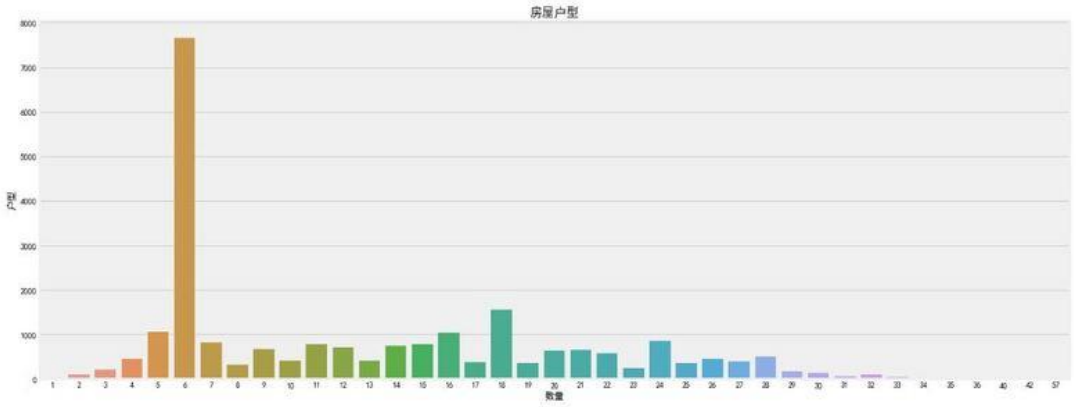
在 Renovation 和 Elevator 的分类条件下，使用 `FaceGrid` 分析 Year 特征，观察结果如下：

- 整个二手房房价趋势是随着时间增长而增长的；
- 2000 年以后建造的二手房房价相较于 2000 年以前有很明显的价格上涨；
- 1980 年之前几乎不存在有电梯二手房数据，说明 1980 年之前还没有大面积安装电梯；
- 1980 年之前无电梯二手房中，简装二手房占绝大多数，精装反而很少；

Floor 特征分析

```
f, ax1= plt.subplots(figsize=(20,5))

sns.countplot(x='Floor', data=df, ax=ax1)

ax1.set_title('房屋户型',fontsize=15)

ax1.set_xlabel('数量')

ax1.set_ylabel('户型')
```

```
plt.show()
```



可以看到，6 层二手房数量最多，但是单独的楼层特征没有什么意义，因为每个小区住房的总楼层数都不一样，我们需要知道楼层的相对意义。另外，楼层与文化也有很重要联系，比如中国文化七上八下，七层可能受欢迎，房价也贵，而一般也不会有 4 层或 18 层。当然，正常情况下中间楼层是比较受欢迎的，价格也高，底层和顶层受欢迎度较低，价格也相对较低。所以楼层是一个非常复杂的特征，对房价影响也比较大。

总结

本次分享旨在让大家了解如何用 Python 做一个简单的数据分析，对于刚刚接触数据分析的朋友无疑是一个很好的练习。不过，这个分析还存在很多问题需要解决，比如：

- 解决爬虫获取的数据源准确度问题；
- 需要爬取或者寻找更多好的售房特征；
- 需要做更多地特征工程工作，比如数据清洗，特征选择和筛选；

- 使用统计模型建立回归模型进行价格预测；

更多内容会慢慢介绍和分享，敬请期待。

入门 python 数据分析最好的实战项目（二）

本篇将继续上一篇数据分析之后进行数据挖掘建模预测，这两部分构成了一个简单的完整项目。结合两篇文章通过数据分析和挖掘的方法可以达到二手房屋价格预测的效果。

下面从特征工程开始讲述。

特征工程

特征工程包括的内容很多，有特征清洗，预处理，监控等，而预处理根据单一特征或多特征又分很多种方法，如归一化，降维，特征选择，特征筛选等等。这么多的方法，为的是什么呢？**其目的是让这些特征更友好的作为模型的输入**，处理数据的好坏会严重的影响模型性能，而好的特征工程有的时候甚至比建模调参更重要。

下面是继上一次分析之后对数据进行的特征工程，博主将一个一个帮大家解读。

```
"""
特征工程
"""
```

```
# 移除结构类型异常值和房屋大小异常值

df = df[(df['Layout']!='叠拼别墅')&(df['Size']<1000)]


# 去掉错误数据“南北”，因为爬虫过程中一些信息位置为空，导致“Direction”的特征出现在这里，需要清除或替换

df['Renovation'] = df.loc[(df['Renovation'] != '南北'), 'Renovation']


# 由于存在个别类型错误，如简装和精装，特征值错位，故需要移除

df['Elevator'] = df.loc[(df['Elevator'] == '有电梯')|(df['Elevator']
== '无电梯'), 'Elevator']


# 填补 Elevator 缺失值

df.loc[(df['Floor']>6)&(df['Elevator'].isnull()), 'Elevator'] = '有电
梯'

df.loc[(df['Floor']<=6)&(df['Elevator'].isnull()), 'Elevator'] = '无
电梯'


# 只考虑“室”和“厅”，将其它少数“房间”和“卫”移除

df = df.loc[df['Layout'].str.extract('^\d(.*?)\d.*?') == '室']


# 提取“室”和“厅”创建新特征

df['Layout_room_num'] = df['Layout'].str.extract('(^\d).*', expand=F
alse).astype('int64')
```

```
df['Layout_hall_num'] = df['Layout'].str.extract('^\d.*?(\d).*', exp
and=False).astype('int64')


# 按中位数对“Year”特征进行分箱

df['Year'] = pd.qcut(df['Year'],8).astype('object')


# 对“Direction”特征

d_list_one = ['东','西','南','北']

d_list_two = ['东西','东南','东北','西南','西北','南北']

d_list_three = ['东西南','东西北','东南北','西南北']

d_list_four = ['东西南北']

df['Direction'] = df['Direction'].apply(direct_func)

df = df.loc[(df['Direction']!='no')&(df['Direction']!='nan')]


# 根据已有特征创建新特征

df['Layout_total_num'] = df['Layout_room_num'] + df['Layout_hall_num
']

df['Size_room_ratio'] = df['Size']/df['Layout_total_num']


# 删除无用特征

df = df.drop(['Layout','PerPrice','Garden'],axis=1)
```

```
# 对于 object 特征进行 onehot 编码

df,df_cat = one_hot_encoder(df)
```

由于一些清洗处理在上一篇文章已经提到，博主从 17 行代码开始。

Layout

先来看看没经处理的 Layout 特征值是什么样的。

```
df['Layout'].value_counts()
```

```
In [148]: df['Layout'].value_counts()

Out[148]: 2室1厅       9485
          3室1厅       3999
          3室2厅       2765
          1室1厅       2681
          2室2厅       1671
          4室2厅        930
          1室0厅        499
          4室1厅        295
          5室2厅        200
          2房间1卫       170
          1房间1卫       146
          3房间1卫       116
          4室3厅         96
          5室3厅         75
          1室2厅         67
          6室2厅         59
          3房间2卫        53
          2室0厅         50
          3室3厅         43
          4房间2卫        31
          3室0厅         29
          6室3厅         29
          5室1厅         27
          叠拼别墅          20
          2房间2卫        18
          4房间1卫        15
          1房间0卫        15
          5房间2卫        10
          7室3厅          7
          4房间3卫         7
          7室2厅          6
          5房间3卫         6
          2室3厅          5
          6室4厅          4
          5室4厅          4
          8室3厅          4
          4室4厅          4
          8室2厅          3
          6房间4卫         3
          3房间0卫         3
          4室0厅          3
          3房间3卫         2
          2房间0卫         2
          1房间2卫         2
          6房间3卫         2
          6室0厅          2
          6房间5卫         1
          11房间3卫        1
          9室1厅          1
          9室2厅          1
          8室4厅          1
          5房间0卫         1
          6室5厅          1
          6室1厅          1
          9室3厅          1
          7室1厅          1
          8室5厅          1
          7房间2卫         1
          1室3厅          1
          5室0厅          1
          Name: Layout, dtype: int64
```

大家也都看到了，特征值并不是像想象中的那么理想。有两种格式的数据，一种是"xx 室 xx 厅"，另一种是"xx 房间 xx 卫"，但是绝大多数都是 xx 室 xx 厅的数据。而

对于像"11 房间 3 卫"或者"5 房间 0 卫"这些的 Layout 明显不是民住的二手房（不在我们的考虑范围之内），因此最后决定将所有"xx 房间 xx 卫"格式的数据都移除掉，只保留"xx 室 xx 厅"的数据。

Layout 特征的处理如下：

第 2 行的意思是只保留"xx 室 xx 厅"数据，但是保留这种格式的数据也是不能作为模型的输入的，我们不如干脆将"室"和"厅"都提取出来，单独作为两个新特征（如第 5 和 6 行），这样效果可能更好。

具体的用法就是使用 `str.extract()` 方法，里面写的是正则表达式。

```
# 只考虑“室”和“厅”，将其它少数“房间”和“卫”移除

df = df.loc[df['Layout'].str.extract('^\d(.*?)\d.*?') == '室']


# 提取“室”和“厅”创建新特征

df['Layout_room_num'] = df['Layout'].str.extract('(^\d).*', expand=False).astype('int64')

df['Layout_hall_num'] = df['Layout'].str.extract('^\d.*?(\d).*', expand=False).astype('int64')
```

Year

我们还有一个 `Year` 特征，为建房的年限时间。年限种类很多，分布在 1950 和 2018 之间，如果每个不同的 Year 值都作为特征值，我们并不能找出 Year 对

Price 有什么影响，因为年限划分的太细了。因此，我们只有将连续数值型特征 Year **离散化，做分箱处理。**

如何分箱还要看实际业务需求，博主为了方便并没有手动分箱，而使用了 pandas 的 qcut 采用中位数进行分割，分割数为 8 等份。

```
# 按中位数对“Year”特征进行分箱

df['Year'] = pd.qcut(df['Year'],8).astype('object')
```

这是将 Year 进行分箱的结果：

```
df['Year'].value_counts()
```

```
(2000.0, 2003.0]      3677
(2004.0, 2007.0]      3362
(1990.0, 1997.0]      3097
(1949.999, 1990.0]    3007
(1997.0, 2000.0]      2801
(2010.0, 2017.0]      2682
(2007.0, 2010.0]      2565
(2003.0, 2004.0]      1755
Name: Year, dtype: int64
```

Direction

这个特征没处理之前更乱，原以为是爬虫的问题，但是亲自到链家看过，朝向确实是这样的。

```
df['Direction'].value_counts()
```

```
南北          11367
南             2726
东西           1388
东南           1311
西南           1094
东              843
西              802
西北            733
东北            645
北              484
东南北           465
南西北           370
南西             158
东西北           139
东南西           133
西南北           124
东南西北           90
西南东北           23
南东北             19
东南西南           15
东南南             13
西东北             10
东东南             10
西南西北           10
东西南             9
南西南             9
东南东北            7
东南南北            5
西南西             5
东西南北            3
南西东北            3
西北东北            2
南西北北            2
东北东北            2
西西南             2
东东北             2
西北北             2
南西西北            2
南西南西            2
南东              2
南西南北            2
西西北             2
东西北北            1
东西北东北           1
东南南西北           1
东西东北            1
东南西南北           1
东南西北北           1
北西              1
西南西北北           1
北南              1
东南西北东北          1
东南西南东北          1
东东南南            1
东南北西            1
南北西             1
南北东             1
南北东北            1
西南西北东北          1
Name: Direction, dtype: int64
```

如上所见，像"西南西北北"或者"东东南南"这样的朝向是不符合常识的（反正我是理解不了）。因此，我们需要将这些凌乱的数据进行处理，具体实现方式是博主自

己写了一个函数 `direct_func`，主要思想就是将各种重复但顺序不一样的特征值合并，比如`"西南北"`和`"南西北"`，并将不合理的一些值移除，如`"西南西北北"`等。

然后通过 `apply()` 方法将 Direction 数据格式转换，代码如下：

```
# 对“Direction”特征

d_list_one = ['东','西','南','北']

d_list_two = ['东西','东南','东北','西南','西北','南北']

d_list_three = ['东西南','东西北','东南北','西南北']

d_list_four = ['东西南北']

df['Direction'] = df['Direction'].apply(direct_func)

df = df.loc[(df['Direction']!='no')&(df['Direction']!='nan')]
```

处理完结果如下，所有的内容相同而顺序不同的朝向都合并了，异常朝向也被移除了。

```
df['Direction'].value_counts()
```

```
南北      11368
南         2726
东西       1388
东南       1313
西南       1252
东          843
西          802
西北         734
东北         645
西南北        495
东南北        485
北          484
东西北        149
东西南        142
东西南北       120
Name: Direction, dtype: int64
```

创建新特征

有时候仅靠已有的一些特征是不够的，需要根据对业务的理解，定义一些的新特征，然后尝试这些新特征对模型的影响，在实战中会经常使用这种方法。

这里尝试将"室"与"厅"的数量相加作为一个总数量特征，然后将房屋大小 Size 与总数量的比值作为一个新特征，可理解为 `"每个房间的平均面积大小"`。当然，新特征不是固定的，可根据自己的理解来灵活的定义。

```
# 根据已有特征创建新特征

df['Layout_total_num'] = df['Layout_room_num'] + df['Layout_hall_num
']

df['Size_room_ratio'] = df['Size']/df['Layout_total_num']


# 删除无用特征

df = df.drop(['Layout','PerPrice','Garden'],axis=1)
```

最后删除旧的特征 Layout，PerPrice，Garden。

One-hot coding

这部分是 `One-hot 独热编码`，因为像 Region，Year（离散分箱后），Direction，Renovation，Elevator 等特征都是定类的非数值型类型，而作为模型的输入我们需要将这些非数值量化。

在没有一定顺序（定序类型）的情况下，使用独热编码处理定类数据是非常常用的做法，在 pandas 中非常简单，就是使用 `get_dummies()` 方法，而对于像 Size 这

样的定比数据则不使用独热，博主这里用了一个自己封装的函数实现了定类数据的自动量化处理。

> 对于**定类，定序，定距，定比**这四个非常重要的数据类型相信加入知识星球的伙伴都非常熟悉了，想要了解的同学可以扫描最后二维码查看。

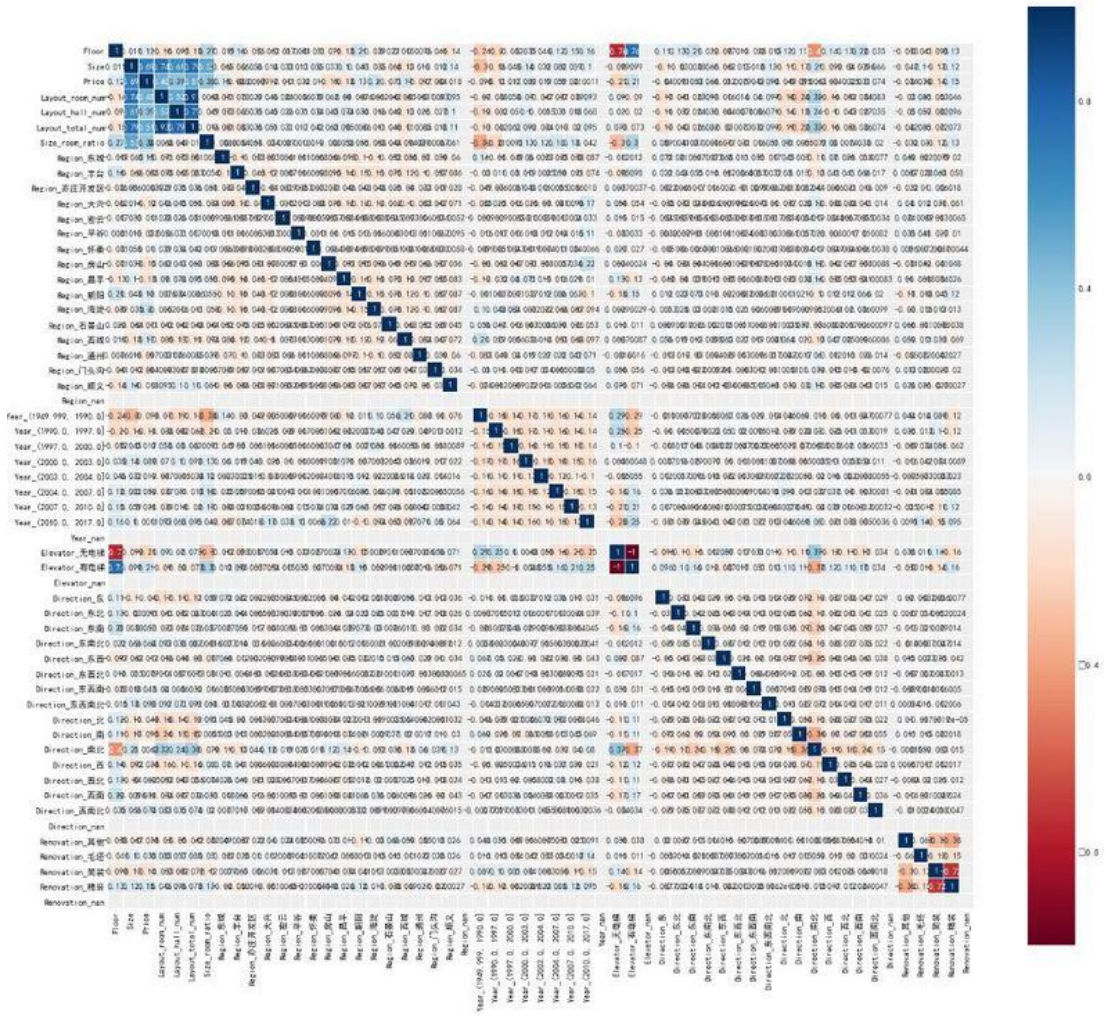
```
# 对于 object 特征进行 onehot 编码

df,df_cat = one_hot_encoder(df)
```

以上的特征工程就完成了。

特征相关性

下面使用 `seaborn` 的 `heatmap` 方法对特征相关性进行可视化。

```
# data_corr

colormap = plt.cm.RdBu

plt.figure(figsize=(20,20))

# plt.title('Pearson Correlation of Features', y=1.05, size=15)

sns.heatmap(df.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=col
ormap, linecolor='white', annot=True)
```



颜色偏红或者偏蓝都说明相关系数较大，即两个特征对于目标变量的影响程度相似，即存在严重的重复信息，会造成过拟合现象。因此，通过特征相关性分析，我们可以找出哪些特征有严重的重叠信息，然后择优选择。

数据建模预测

为了方便理解，博主在建模上做了一些精简，模型策略方法如下：

- 使用 `Cart 决策树`的回归模型对二手房房价进行分析预测

- 使用`交叉验证`方法充分利用数据集进行训练，避免数据划分不均匀的影响。
- 使用 `GridSearchCV` 方法优化模型参数
- 使用 `R2 评分`方法对模型预测评分

上面的建模方法比较简单，旨在让大家了解建模分析的过程。随着逐渐的深入了解，博主会介绍更多实战内容。

数据划分

```
# 转换训练测试集格式为数组

features = np.array(features)

prices = np.array(prices)



# 导入 sklearn 进行训练测试集划分

from sklearn.model_selection import train_test_split

features_train, features_test, prices_train, prices_test = train_test
_split(features, prices, test_size=0.2, random_state=0)
```

将以上数据划分为训练集和测试集，训练集用于建立模型，测试集用于测试模型预测准确率。使用 `sklearn 的 model_selection` 实现以上划分功能。

建立模型

```
from sklearn.model_selection import KFold

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import make_scorer
```

```
from sklearn.model_selection import GridSearchCV


# 利用 GridSearchCV 计算最优解
def fit_model(X, y):
    """ 基于输入数据 [X,y]，利于网格搜索找到最优的决策树模型"""


    cross_validator = KFold(10, shuffle=True)

    regressor = DecisionTreeRegressor()


    params = {'max_depth':[1,2,3,4,5,6,7,8,9,10]}

    scoring_fnc = make_scorer(performance_metric)

    grid = GridSearchCV(estimator = regressor, param_grid = params, s
coring = scoring_fnc, cv = cross_validator)


    # 基于输入数据 [X,y]，进行网格搜索

    grid = grid.fit(X, y)

#     print pd.DataFrame(grid.cv_results_)

    return grid.best_estimator_


# 计算 R2 分数
def performance_metric(y_true, y_predict):
```

```
    """计算并返回预测值相比于预测值的分数"""

    from sklearn.metrics import r2_score

    score = r2_score(y_true, y_predict)


    return score
```

使用了 `KFold` 方法减缓过拟合，`GridSearchCV` 方法进行最优参数自动搜查，最后使用 `R2` 评分来给模型打分。

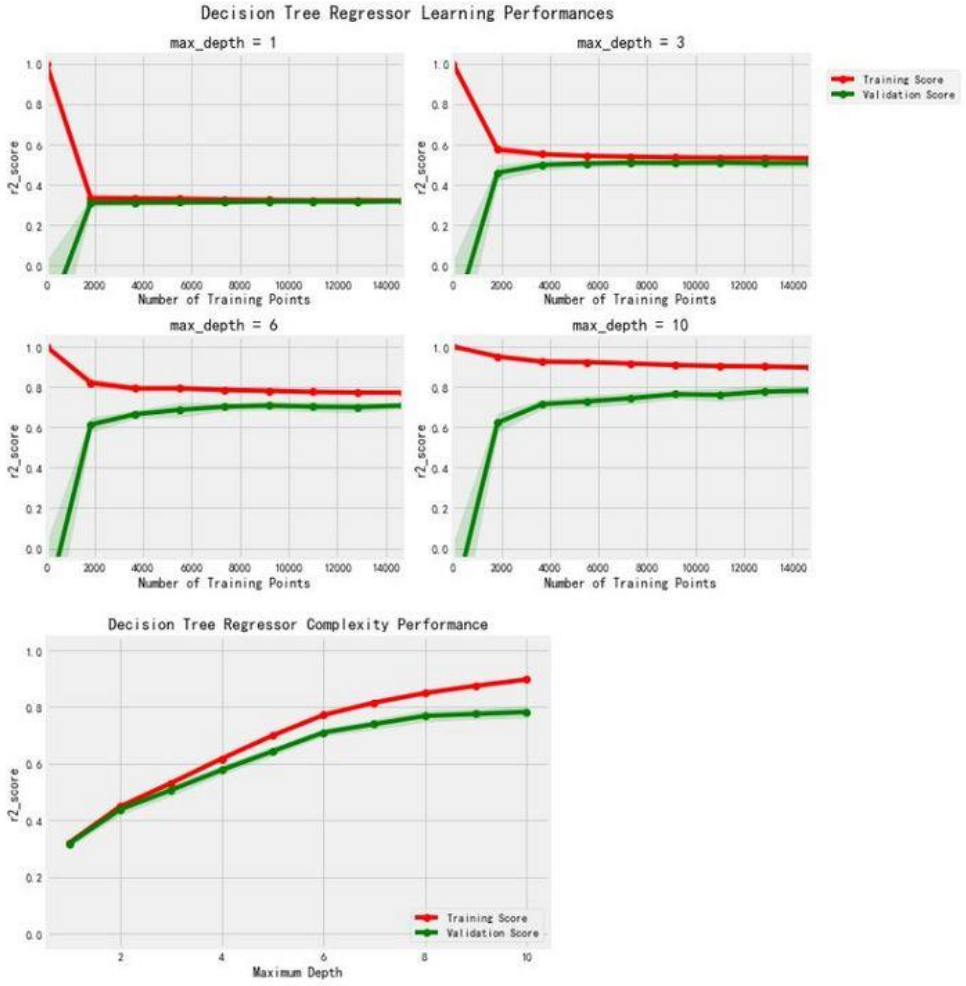
调参优化模型

```
import visuals as vs


# 分析模型

vs.ModelLearning(features_train, prices_train)

vs.ModelComplexity(features_train, prices_train)


optimal_reg1 = fit_model(features_train, prices_train)


# 输出最优模型的 'max_depth' 参数

print("最理想模型的参数 'max_depth' 是 {} 。".format(optimal_reg1.get_
params()['max_depth']))


predicted_value = optimal_reg1.predict(features_test)
```

```
r2 = performance_metric(prices_test, predicted_value)
```

```
print("最优模型在测试数据上 R^2 分数 {:,.2f}。".format(r2))
```

由于决策树容易过拟合的问题，我们这里采取观察学习曲线的方法查看决策树深度，并判断模型是否出现了过拟合现象。以下是观察到的学习曲线图形：



通过观察，最理想模型的参数`"max_depth"`是 10，此种情况下达到了偏差与方差的最优平衡，最后模型在测试数据上的 R2 分数，也即二手房房价预测的准确率为：`0.81`。

总结

以上一个完整的从数据分析到挖掘的项目就结束了，对于项目而言比较简单，目的是让大家了解整个分析的过程。可提升改进的地方非常多，可以有更好更健壮的方案代替，一些改进思考如下：

- 获取更多有价值的特征信息，比如学区，附近地铁，购物中心等
- 完善特征工程，如进行有效的特征选择
- 使用更优秀的模型算法建模或者使用模型融合