**北京房价信息机器学习**

背景：

近几年房价持续高涨，买房成了大部分中国人一生的目标。本次实验通过基于对北京房价信息的数据分析进行机器学习，以获得机器对于房价的可能回归与认知。

# 数据可视化部分

首先对数据进行可视化操作，来进行数据筛选和处理，以达到更好的学习效果。

## 初步统计分类

住房按照分类可有多种划分，首先对数据进行预处理，按照多种类型进行分类。

本次按照地名，建造年份，装修类型，房价，面积，建造年份分类

### 代码实现：

df\_dm\_mean=df["District"].value\_counts()#.sort\_values(ascending=False)#按照地名来分类

df\_2=df['Construction time'].sort\_values()#df['Construction time'].value\_counts()#按照建造年份分类

df\_zx\_mean=df['renovation condition'].value\_counts()#按照装修类型分类

df\_jg\_mean=df['Total price']#房价

df\_dx\_mean=df['Square']#面积

df\_nf\_mean=df['Construction time'].value\_counts()#按照建造年份分类

## 装修情况一览

住房装修是住房的一大重要属性，不同的装修程度直接影响住房的价格，对此属性进行统计分析，可得到住房价格的一个基本认知。

### 代码实现：

df\_zx\_mean.plot(kind='bar',figsize=(18,6),color=['darkkhaki','tan','beige','steelblue'],alpha=0.55)

plt.xticks(size=18,rotation=0)

am1=dict(df\_zx\_mean).values()

text1=list(am1)

for i in range(len(text1)):

plt.text(i,text1[i],text1[i],size=18)

plt.title('装修情况')

plt.xlabel('装修类型',size=18,color='g')

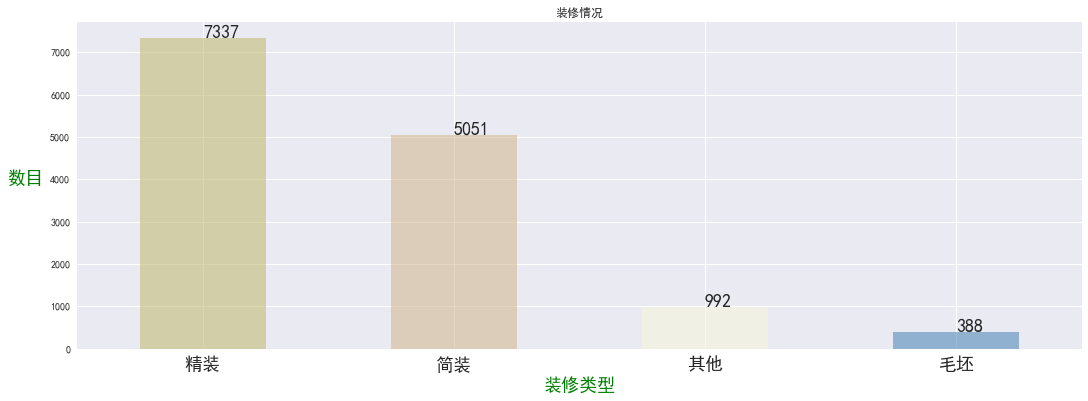
plt.ylabel('数目',size=18,color='g',rotation=0,labelpad=25)

plt.show()

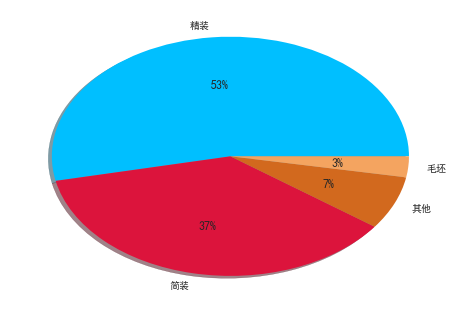
#画出饼图，展现出各种装修所占的比例

labels\_zx=['精装','简装','其他','毛坯']

plt.pie(df\_zx\_mean, autopct='%0.0f%%',labels=labels\_zx,colors=['deepskyblue','crimson','chocolate','sandybrown'],shadow=True)

plt.show()

柱形图



饼图

## 所有房子的价格信息

散点图具有直观观察数据分布特点，以散点图观测房价的基本分布情况。同时，插入一条平均价格的水平线，用以观测房价在均价附近的分布。

### 代码实现：

#显示房价的基本信息

cankao\_1=[df\_jg\_mean.mean()]\*len(df\_jg\_mean)#生成一条平均价格的参考线

fig=plt.figure(figsize=(20,6))

ax1=fig.add\_subplot(111)

ax1.set\_title('初步价格情况')

cankao\_1

cankao\_2=[i for i in range(13769)]

plt.scatter(cankao\_2,df\_jg\_mean,label='房产单价',s=5)

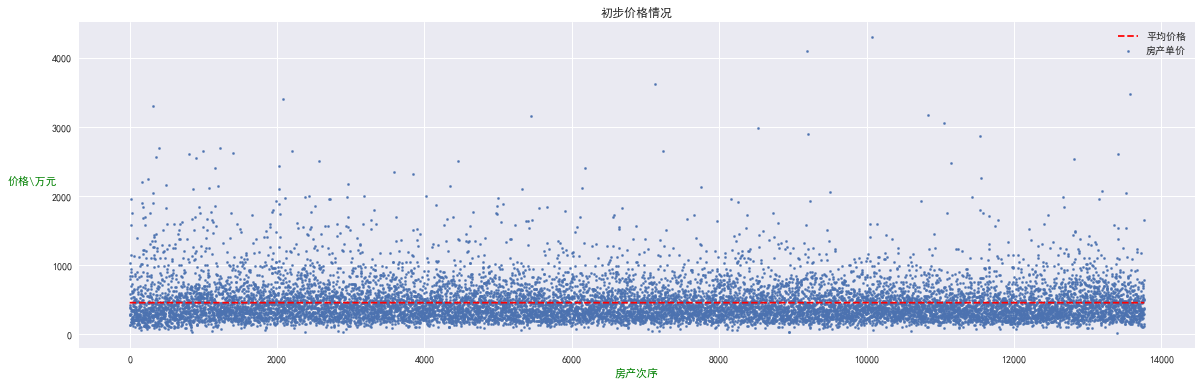
plt.plot(cankao\_1,'r--',label='平均价格')

plt.legend(loc='best')

plt.ylabel('价格\万元',rotation=0,labelpad=20,color='g')

plt.xlabel('房产次序',color='g')

plt.show()



散点图无法直观观察在某价位层级的分布情况，所以在此基础之上加入分布直方图，观测房价的档位分布。

### 代码实现：

#按200万进行分档

chuli\_fj=[df\_jg\_mean[df\_jg\_mean<200].count(),

df\_jg\_mean[df\_jg\_mean<400][df\_jg\_mean>200].count(),

df\_jg\_mean[600>df\_jg\_mean][df\_jg\_mean>400].count(),

df\_jg\_mean[800>df\_jg\_mean][df\_jg\_mean>600].count(),

df\_jg\_mean[1000>df\_jg\_mean][df\_jg\_mean>800].count(),

df\_jg\_mean[1200>df\_jg\_mean][df\_jg\_mean>1000].count()]#把房价按照200分档

fig=plt.figure(figsize=(20,6))

ax=fig.add\_subplot(111)

ax.bar([1,2,3,4,5,6],chuli\_fj,color=['gray','crimson','chocolate','sandybrown','salmon','tomato'],alpha=0.55)

ax.set\_xticklabels(['0','200','400','600','800','1000','1200'])

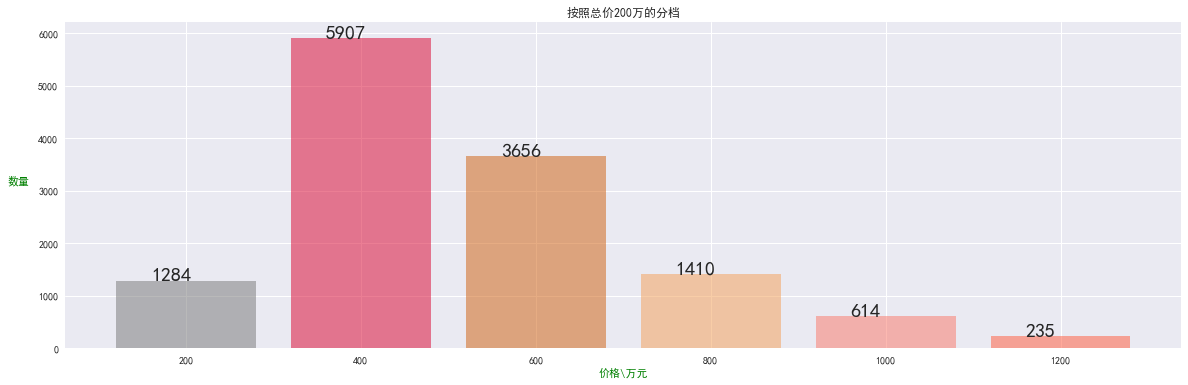
ax.set\_ylabel('数量',rotation=0,labelpad=20,color='g')

ax.set\_xlabel('价格\万元',color='g')

for i in range(6):

plt.text(i+0.8,chuli\_fj[i],chuli\_fj[i],size=20)

ax.set\_title('按照总价200万的分档')

plt.show()

## 各年份房价统计

对不同年份建造的房屋进行价格分析，观察房价在不同年份的分布情况

### 代码实现：

df\_nf=df\_nf\_mean.sort\_index()

cankao\_3=[df\_nf.mean()]\*74#平均线

cankao\_4=[100]\*74#起算线，低于起算线的值数据量不足，不纳入学习和测试集

df\_nf.plot(kind='bar',figsize=(18,6),color='sandybrown',alpha=0.55,label='年度值')

am3=dict(df\_nf).values()

text3=list(am3)

for i in range(len(text3)):

plt.text(i-0.25,text3[i],text3[i],color='r',size=8)

plt.title('随年度变化的情况')

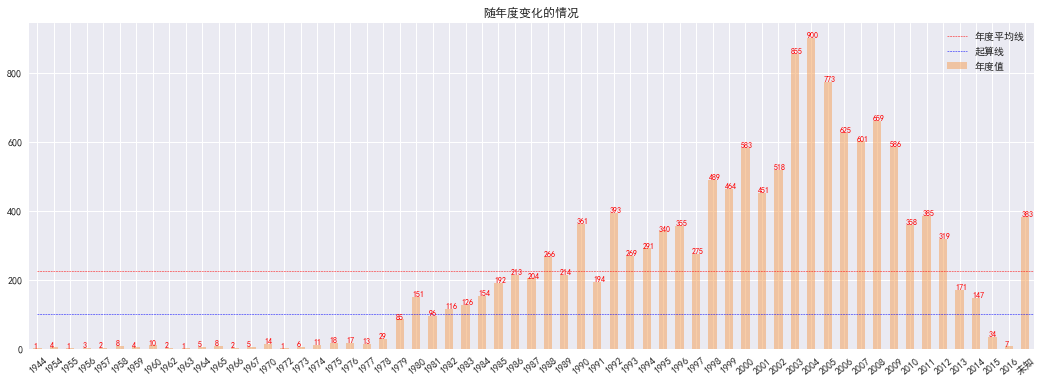
plt.xticks(rotation=40)

plt.plot(cankao\_3,'r--',label='年度平均线',linewidth=0.5)

plt.plot(cankao\_4,'b--',label='起算线',linewidth=0.5)

plt.legend(loc='best')

plt.show()



## 各区住房价格统计

地区也是影响住房价格的一个直接属性，通过对此进行分析，观测房价分布情况

### 代码实现：

df\_dm\_mean.plot(kind='bar',figsize=(18,6),color=

['gray','crimson','chocolate','sandybrown','salmon','tomato','deepskyblue'],alpha=0.55)

plt.xticks(size=18,rotation=0)

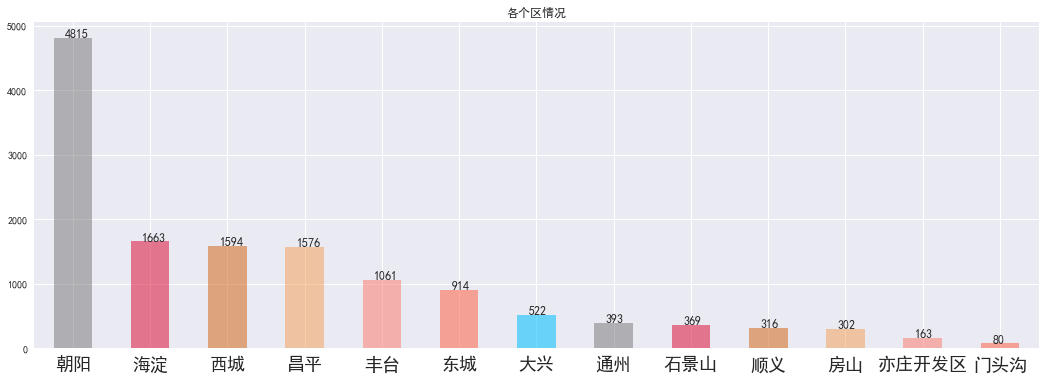
am2=dict(df\_dm\_mean).values()

text2=list(am2)

for i in range(len(text2)):

plt.text(i-0.1,text2[i],text2[i])

plt.title('各个区情况')

plt.show()

## 房价随建造时间的变化

随着时间的变化，房价也会产生一定的波动，观测时间轴上房价的统计信息。

### 代码实现：

df\_x=df[(df['Construction time']<='2014')&(df['Construction time']>'1980')]

jgx=df\_x[['Construction time','Total price']]

item=np.arange(1981,2015)

pj\_ja\_nf=[] #求出每年的平均房价

dj\_ja\_nf=[] #每年的最低价

gj\_ja\_nf=[] #每年的最高价

for i in range(1981,2015):

pj\_ja\_nf.append(jgx[jgx['Construction time']==str(i)]['Total price'].mean())

dj\_ja\_nf.append(jgx[jgx['Construction time']==str(i)]['Total price'].min())

gj\_ja\_nf.append(jgx[jgx['Construction time']==str(i)]['Total price'].max())

fig=plt.figure(figsize=(20,10))

ax1=fig.add\_subplot(111,facecolor='linen')

ax1.scatter(x=jgx['Construction time'].values,y=jgx['Total price'].values,s=1,color='g',label='房间单价')

ax1.plot(item,pj\_ja\_nf,'o--',alpha=0.55,label='每年的均价')

ax1.plot(item,dj\_ja\_nf,'o--',alpha=0.55,label='每年最低价',color='brown')

ax1.plot(item,gj\_ja\_nf,'o--',alpha=0.55,label='每年最高价',color='salmon')

plt.xticks(size=15)

plt.yticks(size=15)

plt.title('房价与时间的一些信息',color='red',size=24)

plt.xlabel('建造年份',color='tomato',size=15)

plt.ylabel('总价 \万元',color='tomato',size=15)

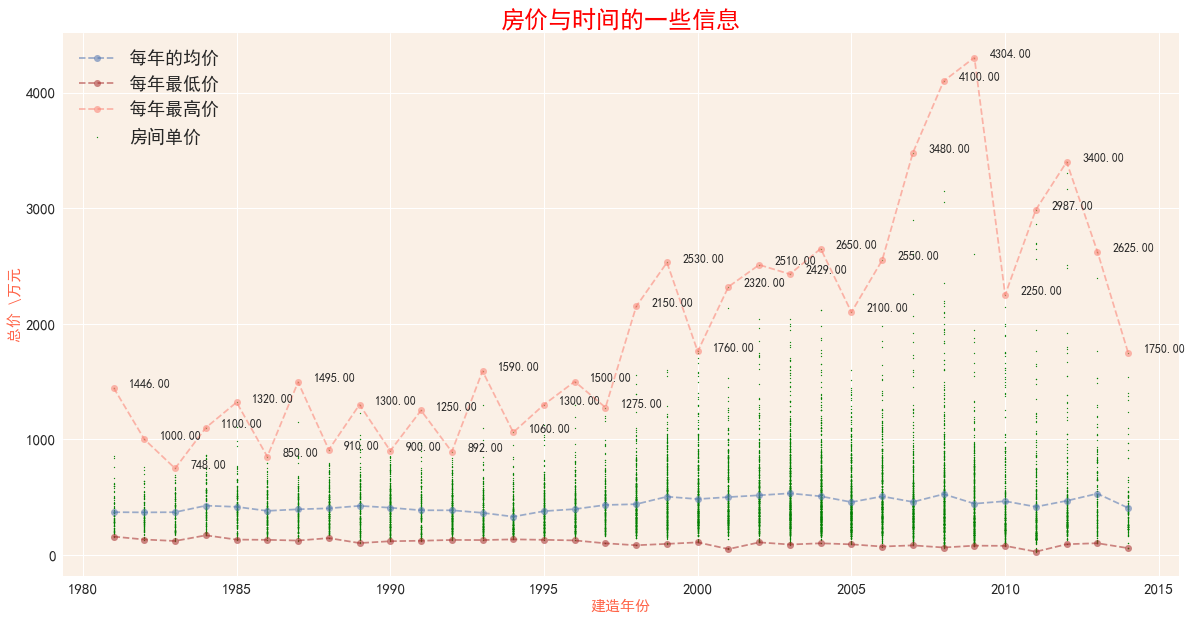
for i in range(len(item)):

plt.text(item[i]+0.5,gj\_ja\_nf[i],'%.2f'%gj\_ja\_nf[i])

plt.legend(loc=2,fontsize=18)

plt.savefig('房价与时间的关系.jpg')

plt.show()



可以发现房价随时间的变化没有过于剧烈的变化，证明总价和时间的关系不能作为回归分析的数据

## 发现平米价格和建造时间的关系

房价的总价格与建筑面积有关，为了更好地观测房屋的价格，观测平米价格而不是总价，更能体现价格和建造时间的关系。

### 代码实现：

pj\_dj=df\_x[['Construction time','Total price','Square']]

pj\_dj['price']=pj\_dj['Total price']/pj\_dj['Square']

pj\_dj

pj\_dj\_nf=[] #求出每年的平米均价

for i in range(1981,2015):

pj\_dj\_nf.append(pj\_dj[pj\_dj['Construction time']==str(i)]['price'].mean())

pj\_dj\_nf

fig=plt.figure(figsize=(14,6))

ax1=fig.add\_subplot(111,facecolor='linen')

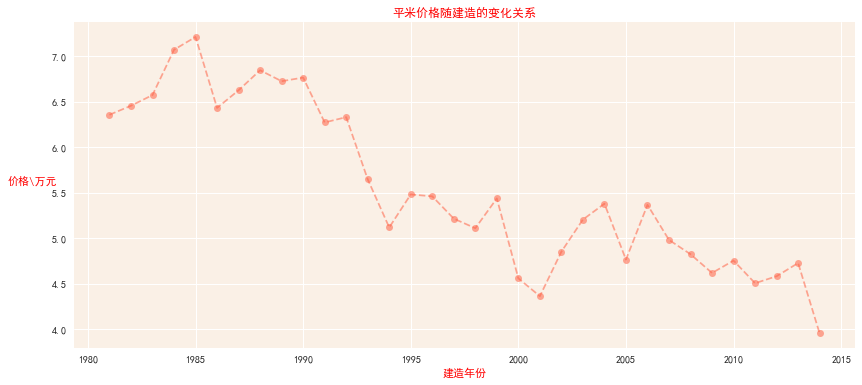
ax1.plot(item,pj\_dj\_nf,'o--',color='tomato',alpha=0.55,label='平米单价')

ax1.set\_title('平米价格随建造的变化关系',color='r',size=12)

ax1.set\_xlabel('建造年份',color='r')

ax1.set\_ylabel('价格\万元',rotation=0,labelpad=20,color='r')

plt.show()



## 各区的平米单价

同样的，观测各区域的平米价格，来更好地体现价格水平

### 代码实现：

qwfj=df[['District','Price']]#区位房价

#各州的平均价格

cy=qwfj[qwfj['District']=='朝阳']['Price'].mean()

cp=qwfj[qwfj['District']=='昌平']['Price'].mean()

dc=qwfj[qwfj['District']=='东城']['Price'].mean()

sy=qwfj[qwfj['District']=='顺义']['Price'].mean()

xc=qwfj[qwfj['District']=='西城']['Price'].mean()

ft=qwfj[qwfj['District']=='丰台']['Price'].mean()

hd=qwfj[qwfj['District']=='海淀']['Price'].mean()

dx=qwfj[qwfj['District']=='大兴']['Price'].mean()

fs=qwfj[qwfj['District']=='房山']['Price'].mean()

yz=qwfj[qwfj['District']=='亦庄开发区']['Price'].mean()

sjs=qwfj[qwfj['District']=='石景山']['Price'].mean()

mtg=qwfj[qwfj['District']=='门头沟']['Price'].mean()

tz=qwfj[qwfj['District']=='通州']['Price'].mean()

jg\_zhou=[cy,cp,dc,sy,xc,ft,hd,dx,fs,yz,sjs,mtg,tz]

dn=qwfj['District'].unique()[:13]

item2=np.arange(0,13)

fig=plt.figure(figsize=(14,6))

ax1=fig.add\_subplot(111,axisbg='linen')

ax1.bar(np.arange(len(jg\_zhou)),jg\_zhou,color=['gray','crimson','chocolate','sandybrown','salmon','tomato','cyan'],alpha=0.55)

ax1.set\_xticks(item2)

ax1.set\_xticklabels(dn,color='tomato')

ax1.set\_xlabel('地区',color='tomato')

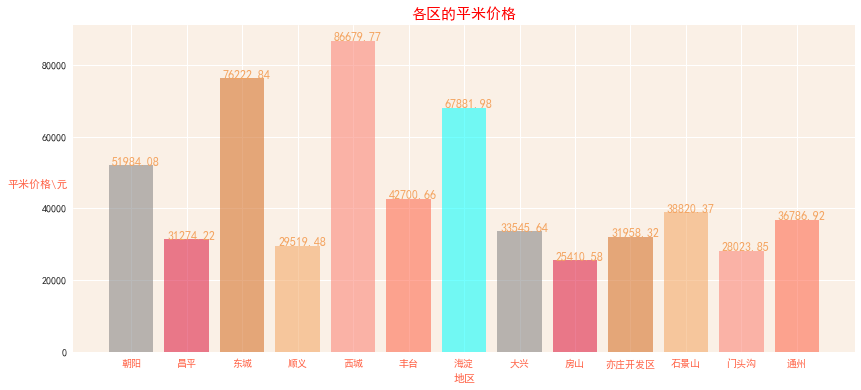
ax1.set\_ylabel('平米价格\元',color='tomato',rotation=0)

ax1.set\_title('各区的平米价格',color='r',size=15)

for i in range(13):

plt.text(i-0.35,jg\_zhou[i],'%.2f'%jg\_zhou[i],color='sandybrown')

plt.show()



## 交易和时间的关系

观察交易价格和时间的关系，以便于体现价格和时间的关系达到更好的回归效果

### 代码实现：

jm=df[['Price','TradeTime','Total price']].sort\_values(by=['TradeTime'])

jm['TradeTime']=pd.to\_datetime(jm['TradeTime'])

jo=jm[['TradeTime','Price']]

jo.set\_index('TradeTime',inplace=True)

jo2=jo.groupby('TradeTime').mean()

dt=pd.Series(jysj)

jysj=pd.to\_datetime(dt)

jam=df[['TradeTime','Price']]

jam.drop([13768])

c=jam.groupby('TradeTime').count()

c.values

dd=[]

for i in c.values:

dd.append(int(i))

fig=plt.figure(figsize=(18,4))

ax1=fig.add\_subplot(111,facecolor='linen')

ax1.bar(c.index,dd,color='skyblue')

ax1.set\_ylabel('成交量/次',rotation=0,color='skyblue',size=15,labelpad=15)

ax2=ax1.twinx()

ax2.plot(jo2,color='tomato',alpha=0.55,label='当日平均价格')

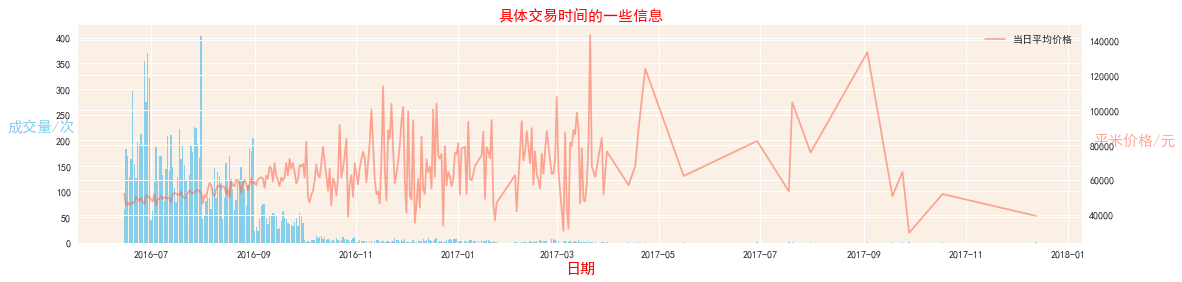
ax2.set\_ylabel('平米价格/元',rotation=0,color='tomato',alpha=0.55,size=15,labelpad=15)

ax1.set\_xlabel('日期',color='r',size=15)

plt.legend(loc='best')

ax1.set\_title('具体交易时间的一些信息',size=15,color='r')

plt.show()



# 机器学习部分

目标实现：通过对数据集进行学习训练，达到输入一定的房屋属性即可得到预期的房屋价格

## 数据清洗

数据中存在文字标签，机器无法识别，将其转化为字符或数字形式，可以识别，以便于学习

### 代码实现：

ren\_cond\_list=data['renovation condition'].unique()#所有装修状况的列表

ren\_cond\_dict={}

for item in ren\_cond\_list:#将装修状况与其平均价格相对应

ren\_cond\_dict[item]=data[data['renovation condition']==item]['Price'].mean()

bud\_stru\_list=data['building structure'].unique()#所有结构的列表

bud\_stru\_dict={}

for item in bud\_stru\_list:#将结构与其平均价格相对应

bud\_stru\_dict[item]=data[data['building structure']==item]['Price'].mean()

bud\_typ\_list=data['Building Type'].unique()#所有类型的列表

bud\_typ\_dict={}

for item in bud\_typ\_list:#将类型与其平均价格相对应

bud\_typ\_dict[item]=data[data['Building Type']==item]['Price'].mean()

'''dist\_list=data['District'].unique()#所有地区的列表

dist\_dict={}

for item in dist\_list:#将地区与其平均价格相对应

dist\_dict[item]=data[data['District']==item]['Price'].mean()

'''

floor\_list=data['楼'].unique()#所有楼层的列表

floor\_dict={}

for item in floor\_list:#将楼层与其平均价格相对应

floor\_dict[item]=data[data['楼']==item]['Price'].mean()

#将文字信息转化为对应的数值信息

yearunknown=[]

for row in data.iterrows():

data.loc[row[0],'楼']=floor\_dict[data.loc[row[0],'楼']]

data.loc[row[0],'Building Type']=bud\_typ\_dict[data.loc[row[0],'Building Type']]

data.loc[row[0],'renovation condition']=ren\_cond\_dict[data.loc[row[0],'renovation condition']]

data.loc[row[0],'building structure']=bud\_stru\_dict[data.loc[row[0],'building structure']]

data.loc[row[0],'TradeTime']=pd.to\_datetime(data.loc[row[0],'TradeTime'])

#data.loc[row[0],'District']=dist\_dict[data.loc[row[0],'District']]

try:

int(data.loc[row[0],'Construction time'])

except:

yearunknown.append(row[0])

data.drop(yearunknown,inplace=True)

#dist\_list=data['District'].unique()

dist1=data.loc[data['District']=='朝阳']

dist1



结果预览

## 数据预处理

通过对数据进行预处理，以便更接近预期效果。

### 代码实现

data=pd.read\_csv('lianjia.csv',encoding='gbk')

data=data.iloc[0:1000,2:27]

data.dropna(axis=0,how='any',inplace=True)

data=data.loc[data['District']=='朝阳']

yearunknown=[]

for row in data.iterrows():

data.loc[row[0],'TradeTime']=pd.to\_datetime(data.loc[row[0],'TradeTime'])

#data.loc[row[0],'District']=dist\_dict[data.loc[row[0],'District']]

try:

int(data.loc[row[0],'Construction time'])

except:

yearunknown.append(row[0])

data.drop(yearunknown,inplace=True)

## 训练与测试

选取一部分数据作为训练集和测试集，开始进行机器学习

### 代码实现

yearunknown=[]

for row in data.iterrows():

data.loc[row[0],'TradeTime']=pd.to\_datetime(data.loc[row[0],'TradeTime'])

#data.loc[row[0],'District']=dist\_dict[data.loc[row[0],'District']]

try:

int(data.loc[row[0],'Construction time'])

except:

yearunknown.append(row[0])

data.drop(yearunknown,inplace=True)

## 用不同方法进行回归

使用LinearRegression，Lasso，Ridge三种方式进行回归。

### 代码实现

#多变量回归分析

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import Lasso

from sklearn.linear\_model import Ridge

model=LinearRegression()

model.fit(x\_train,y\_train)

a=model.intercept\_

b=model.coef\_

try:

lasmodel=Lasso(alpha=00.01)

lasmodel.fit(x\_train,y\_train)

lsa=lasmodel.intercept\_

lsb=lasmodel.coef\_

except:

print('err2')

try:

ridgemodel=Ridge()

ridgemodel.fit(x\_train,y\_train)

rida=ridgemodel.intercept\_

ridb=ridgemodel.coef\_

except:

print('err3')

#斜率与截距

print('osl:',a,b)

print('lasso:',lsa,lsb)

print('ridge:',rida,ridb)

y\_pred=model.predict(x\_test)

y\_predlas = lasmodel.predict(x\_test)

y\_predrid=ridgemodel.predict(x\_test)

from sklearn import metrics

# 用scikit-learn计算MSE

print ("olsMSE:",metrics.mean\_squared\_error(y\_test, y\_pred))

print ("lasMSE:",metrics.mean\_squared\_error(y\_test, y\_predlas))

print ("ridgeMSE:",metrics.mean\_squared\_error(y\_test, y\_predrid))

# 用scikit-learn计算RMSE

print ("olsRMSE:",np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

print ("lasRMSE:",np.sqrt(metrics.mean\_squared\_error(y\_test, y\_predlas)))

print ("ridgeRMSE:",np.sqrt(metrics.mean\_squared\_error(y\_test, y\_predrid)))

print('olsscore:',model.score(x\_test,y\_pred))

print('lasscore:',model.score(x\_test,y\_pred))

print('ridgescore:',model.score(x\_test,y\_pred))

ori=[]

osl=[]

las=[]

i=0

for row in y\_test.iteritems():

ori.append(row[1])

osl.append(y\_pred[i])

las.append(y\_predlas[i])

print('ori:',row[1],'osl:',y\_pred[i],'las:',y\_predlas[i])

i+=1

predlst=list(zip(ori,osl,las))

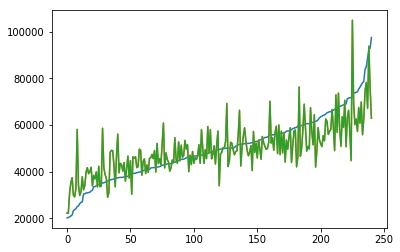
preddt=pd.DataFrame(predlst)

preddt=preddt.sort\_values(by=0)

preddt=preddt.reset\_index(drop=True)

preddt

plt.plot(preddt)



回归与学习效果展示