基于梯度提升模型和多变量综合评价法分析波士顿房价数据

摘　要

本文研究波士顿房价的预测和宜居性评估问题，首先探究了 13 个特征值对房价的影响。在综合比较了十余种不同的线性回归及非线性回归模型后，基于改进的梯度提升算法对决策树模型进行优化，建立了波士顿房价预测模型，其次分别采用两种不同的多变量评估模型，基于对数据采集地区的人均综合素质，环境优劣，房屋价格高低与交通等因素的综合考量采取多方案决策（AHP 模型）并进行统计，得到宜居性指标对波士顿的房子进行分类，最终基于对以上问题探究对出不同人群给出买房建议。

针对问题 1，首先借助于 Python 中的 pandas 库对大规模、多种类数据处理的优势进行数据的操作和处理，分别形成了 14 个变量数据两两之间的定性关系，并通过 matplotlib 库将数据进行可视化，分别形成了因变量与自变量，自变量与自变量之间的关系图，通过直线进行了线性拟合，

针对问题二建立房价预测模型的问题，基于模型套用比较选择了 GBRr 模型（后文简称 GBR 模型）。

模型关键参数为：

learning rate	max_depth	max_features	min_samples_leaf	n_estimator
0.1	6	13	3	100

针对问题 3 的宜居性评估问题，利用多变量综合评价法构造相对偏差矩阵，采用加权求和法客观赋权，确定了不同指标的权重。

最后，针对问题 4 给出买房建议的问题，不同人群在购房的时候所关注的问题重点不同，本文重点关注购房人群的特点，选择相关的房屋参数，给出了建议。

针对 GBR 模型的求解，本文使用 Grid Search 来选择性能表现最好的超参数，大大提高了效率，并用相对偏差矩阵求解出宜居性指标权重，进一步求解出宜居性的等级。

GBR 模型能较完美地预测真实房价的走势，鲁棒性较强，但在趋势拐点处往往出现较高的预测偏差。在数据分布稠密的数据上，泛化能力和表征能力都很好。后续可搭建神经网络模型与本次的训练模型进行评估比较。

相对偏差矩阵 R 的元素，消除了量纲，使指标具有可比性。局限性则在于将若干个指标数值综合成一个数值，损失了原有指标带来的大量信息，结果较抽象，难释其经济意义；由于主观性很强，评价的结果不具有惟一性。

关键词

房价预测 改进梯度提升决策树模型 可视化 主成分分析 多变量综合评价

一、 问题重述

1.1. 问题背景

近年来,我国房地产市场持续发展,住房价格也随之上涨,房地产价格已成为衡量我国房地产市场健康稳定的重要指标。人们对房地产价格的预测需求不断增长,要求也在不断提高。住房价格问题已成为我国人民关注的焦点,有关住房价格问题的许多研究已经在我国和国际的期刊上发表。

房价是体现经济运转好坏的重要指标，房地产开发商与购房者都密切关注着房价波动，构建有效的房价预测模型对金融市场、民情民生有着重要意义。房价预测模型可以有效地解决当今房地产市场所存在的一些问题,帮助人们理性的投资,切实地解决人们买房难的问题,创造更加和谐美满的社会。

1.2． 问题重述

已知条件：

1. 波士顿当前房价的中位数
2. 其余 13 类住房环境表格数据

解决问题：

1. 基于表格数据的分析及数据间关系的探究得出 13 个其他变量对房价的影响。
2. 13 个外界影响因素中，有的对房价影响很大，有的可能甚至们没有直接的相关关系，基于第（1）问比较选择波士顿房价较佳的预测模型，搭建模型，并对该模型的预测结果进行分析。
3. 基于第（1）（2）问不同外界因素影响房价的重要程度以及相关资料及常识给出房屋宜居性的评判标准，并以此标准为房屋划分等级
4. 不同人群的购房需求有所不同，比如收入高的人可能更喜欢住宜居性好的地方，但是收入低的人则可能优先考虑经济因素，比如考虑一些离公路较近，收税较少的房屋，即便这些地域人均犯罪率高一些，要综合考虑这些因素，结合以上问题的分析结果，给出对不同人群的购房建议。

二、 问题分析

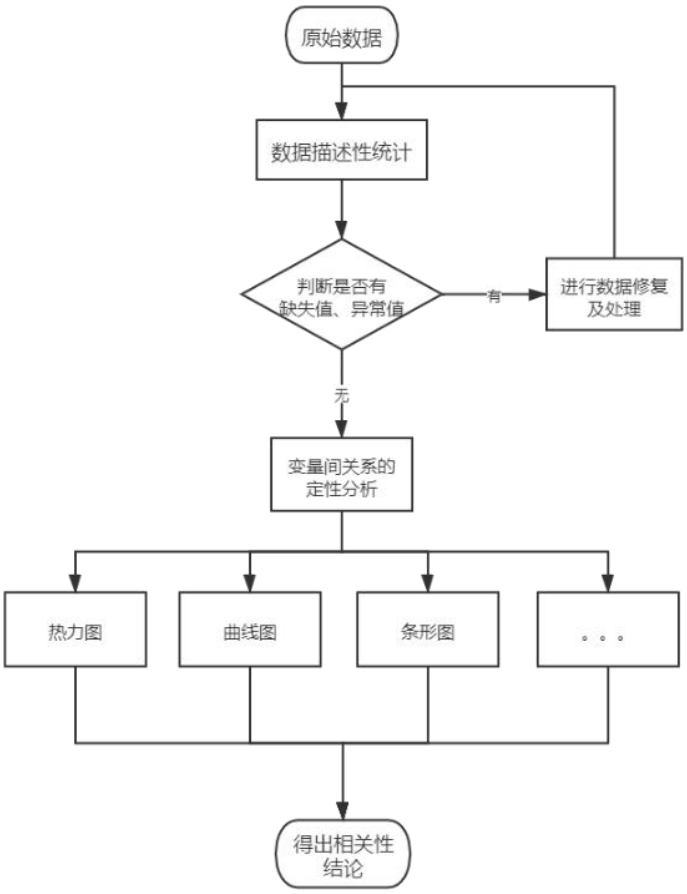
本题要求对波士顿未来房价进行预测，并对不同参数指标的房屋进行宜居性划分，制定针对不同人群的购房策略。解题思路主要分为 3 步：第一步，表格所给数据进行分析，将其对房价的重要程度进行定性评估；第二步，根据不同模型的训练情况搭建模型，对房价进行预测；第三步，对外界影响对房价的重要程度进行量化评估，给出宜居性标准并制定不同人群购房策略。第一问的问题可以用第一步的思路解决，第二问的解决则需要用到第一问的结论及第二步的思路，以此类推，所以，整个问题的解决过程与解题思路是一脉相承，层层递进的。

2.1. 问题一分析

题目给出的数据收集于 1978 年，506 个条目代表来自波士顿各个郊区的 14 个特征的汇总信息。问题需要根据已有表格数据分析变量之间的关系，因此需要有以下几个步骤：

1. 对数据进行描述性统计，获得缺失值、异常值，以及得到房价的总体特征。
2. 如果有缺失值或异常值，需要对这些数据进行处理，并对影响房价的因素进行划分，建立自变量之间以及自变量与因变量之间的定性拟合曲线。
3. 换用不同的统计方式对影响因素进行评估，得到更为全面，多维度的评估指标，最终，我们明确，房价可能与 LSTAT(底层人口的百分比),RM(每个住宅的平均房间数),PTRATIO(各镇的师生比率),TAX(每 $ 10,000 的全值财产税率),NOX(一氧化氮的浓度)有一定但不算太强的相关性。

问题一流程如图 1 所示：

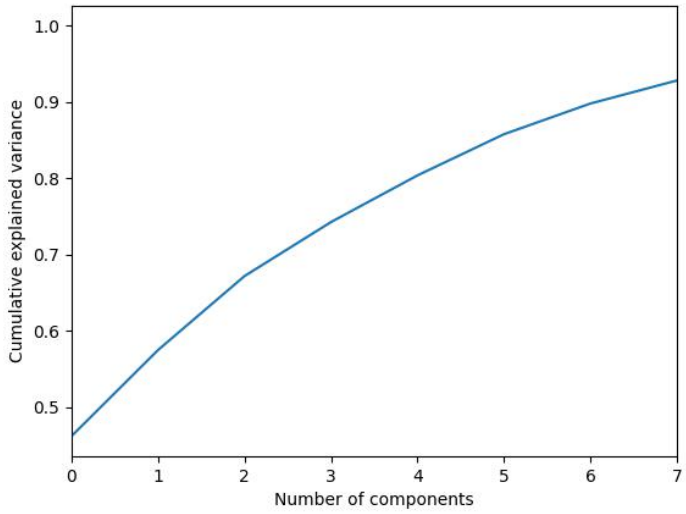


图表 1 问题 1 流程图

2.2. 问题二分析

问题二是建立合适的预测模型并进行模型评估，但是模型的选择肯定需要一些因素的考量，因此需要有以下几个步骤：

1. 由第一问的结论，考虑到不少特征与房价并无强相关性，若带入所有数据可能会对模型训练引入噪音，所以需要对原始数据进行主成分分析（PCA），对特征进行提取。
2. 考虑到主成分各特征维度的含义具有模糊性，不如原始样本特征的解释性强且方差小的成分降维丢弃可能对后续数据处理有影响，所以需要考察成分方差，看降维保留了多少差异性，经过可视化，如下图所示，可以看出，最终 PCA 参数为 5 时便可以达到要求。



图表 2 差异性可视化

3. 以 5 为参数，进行 PCA 特征提取，用处理后的数据对模型进行评估，生成对比表格（见文件包）
4. 重点依据以下指标对模型进行评估：

   learning_rate: 学习率，模型是 0

   n_estimators: 弱学习器的数目，默认值 100

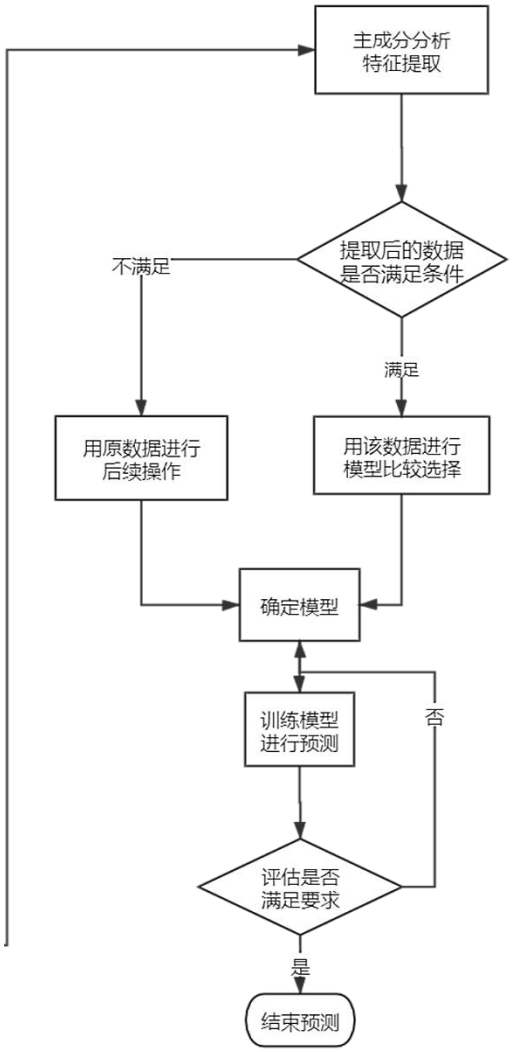
   max_depth: 每一个学习器的最大深度，默认为 3

   min_samples_split: 可以划分为内部节点的最小样本数，默认为 2

   min_samples_leaf: 叶节点所需的最小样本数，默认为 1

   由表格数据分析可得，GBRr 模型是十余种模型种综合指标最好的，确定模型。
5. 用该选定模型对数据进行预测，并且对模型预测结果进行评估，最终得到模型 MAE，MSE，ESV，R2 指标，对模型进行评测。

问题二流程如图 3 所示：



图表 3 问题 2 流程

2.3. 问题三分析

问题三需要通过分析给出宜居性判断，并且需要以宜居性为指标对波士顿房子进行分类。宜居性是一个比较抽象的概念，我们有两种思路来解决这个问题。

第一种：我们依据前两问的分析，觉得宜居性和房价呈现某种正相关性，则，本题中依照宜居性对房子进行分类就转换为依照房价对房子进行分类，也就是依据之前得到的不同因素对于房价的影响大小，化为权重，最终得出宜居性评分和等级。

第二种：我们并不假定宜居性与房价间有关系，我们直接将 14 种因素分为四类：

① 人均犯罪率&学生/教师比例，这两个变量归为人文类；

② 每个镇的非零售业务英亩的比例&查尔斯河虚拟变量&一氧化氮的浓度&到五个波士顿就业中心的加权距离&径向公路通达性的指标，这五个变

量归为环境类；

③ 每$ 10,000 的全值财产税率&户主拥有住房价值的中位数，这两个归为经济类；

④ 其余四个因素归为其他类；

⑤ 于是，我们得到四大类参考。

两种模型分别需要如下的步骤：

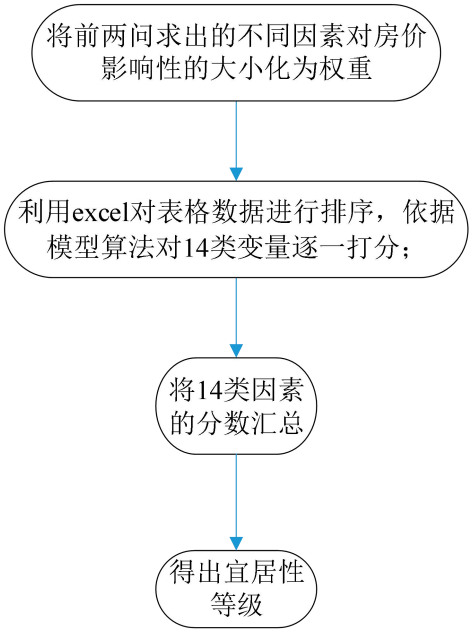
模型一：

1．将前两问求出的不同因素对房价影响性的大小化为权重；

2．利用 excel 对表格数据进行排序，依据模型算法对 14 类变量逐一打分；

3．将 14 类因素的分数汇总；

4．得出宜居性等级。



图表 4 模型 1 的求解流程

模型二：

1． 进行一致性检验，检验各元素重要度之间的协调性，避免出现 A 比 B 重要，B 比 C 重要，而 C 又比 A 重要，这样的矛盾情况出现，若指标 CR 满足<0.10 的范围，就可以利用 AHP 模型进行层次分析，如不满足，则处理到满足为止。

2． 规定加权方式，生成判断矩阵的简单方法，本次加权依照第二问得到的重要程度数据对不同小类进行打分，依照下图所示的条形图排序以及对应的具体数值，规定打分区间为 1~150，LSTAT 影响最大，占比 150，RAD 影响最小，占比为 1.

3． 获得判断矩阵的最大特征值和对应的特征向量，并进行最大特征值

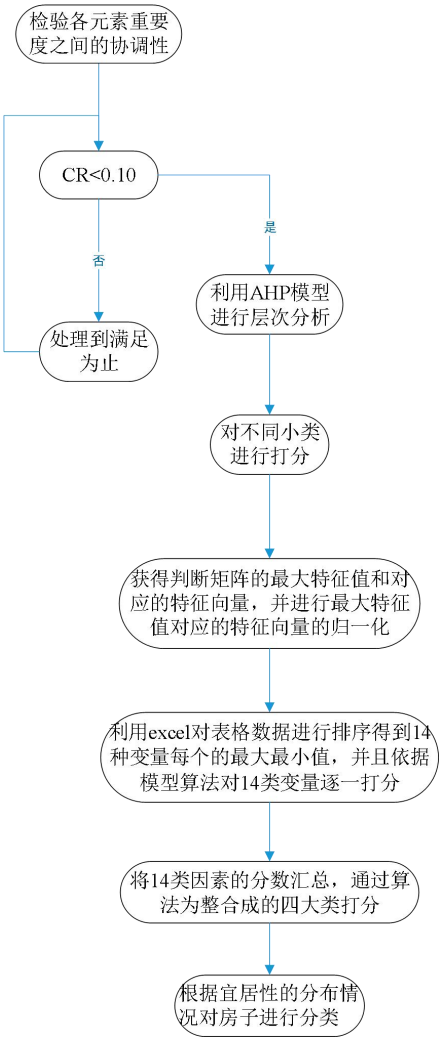
对应的特征向量的归一化

4. 接下来利用 excel 对表格数据进行排序得到 14 种变量每个的最大最小值，并且依据模型算法对 14 类变量逐一打分

5. 再将 14 类因素的分数汇总，通过算法为整合成的四大类打分，最后再将这四类整合为最终结果也就是宜居性打分

6. 根据宜居性的分布情况对房子进行分类。

模型二的求解流程如下图所示。（详细数据见文件包）



图表 5 模型二求解流程

2.4. 问题四分析

针对不同人群的需要给出买房建议。本文以新婚夫妇，投资购房人群，教育需求人群和改善型购房人群为代表，将他们的需要进行区分化，以决定性指标体系给出不同区域房屋的评分等级。

三、 模型假设与符号说明

3.1. 模型假设

1.房价可以作为评估宜居性的某种指标。

2.对于某个群体的购房者，只考虑某个最突出的决定性指标，其余数据不作参考。

3.评估宜居性仅考虑题目给出的 13 个指标，而不考虑其他因素。

4.经过预处理的数据认为可以直接使用。

3.2. 符号说明

符号	符号说明
CRIM	城镇的人均犯罪率
ZN	大于 25,000 平方英尺的地块的住宅用地比例
INDUS	每个镇的非零售业务英亩的比例
CHAS	查尔斯河虚拟变量（如果环河，则等于 1；否则等于 0）
NOX	一氧化氮的浓度（百万分之几）
RM	每个住宅的平均房间数
AGE	1940 年之前建造的自有住房的比例
DIS	到五个波士顿就业中心的加权距离
RAD	径向公路通达性的指标
TAX	每$ 10,000 的全值财产税率
PTRATIO	各镇的师生比率

B	计算方法为 1000（Bk-0.63）2，其中 Bk 是按城镇划分的非裔美国人的比例
LSTAT	底层人口的百分比
MEDV	**自有住房数的中位数，单位为$ 1000**

图表 6 符号说明

四、 模型建立与求解

4.1. 问题一模型建立与求解

4.1.1. 整体数据描述及数据预处理

首先对数据进行描述性统计如下（部分数据，详细数据见附件):

	CRIM	ZN	INDUS	CHAS	NOX	RM	...	RAD	TAX	PRTATTO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	...	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	...	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	...	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	...	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	...	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	...	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	...	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	...	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

[8 rows x 14 columns]

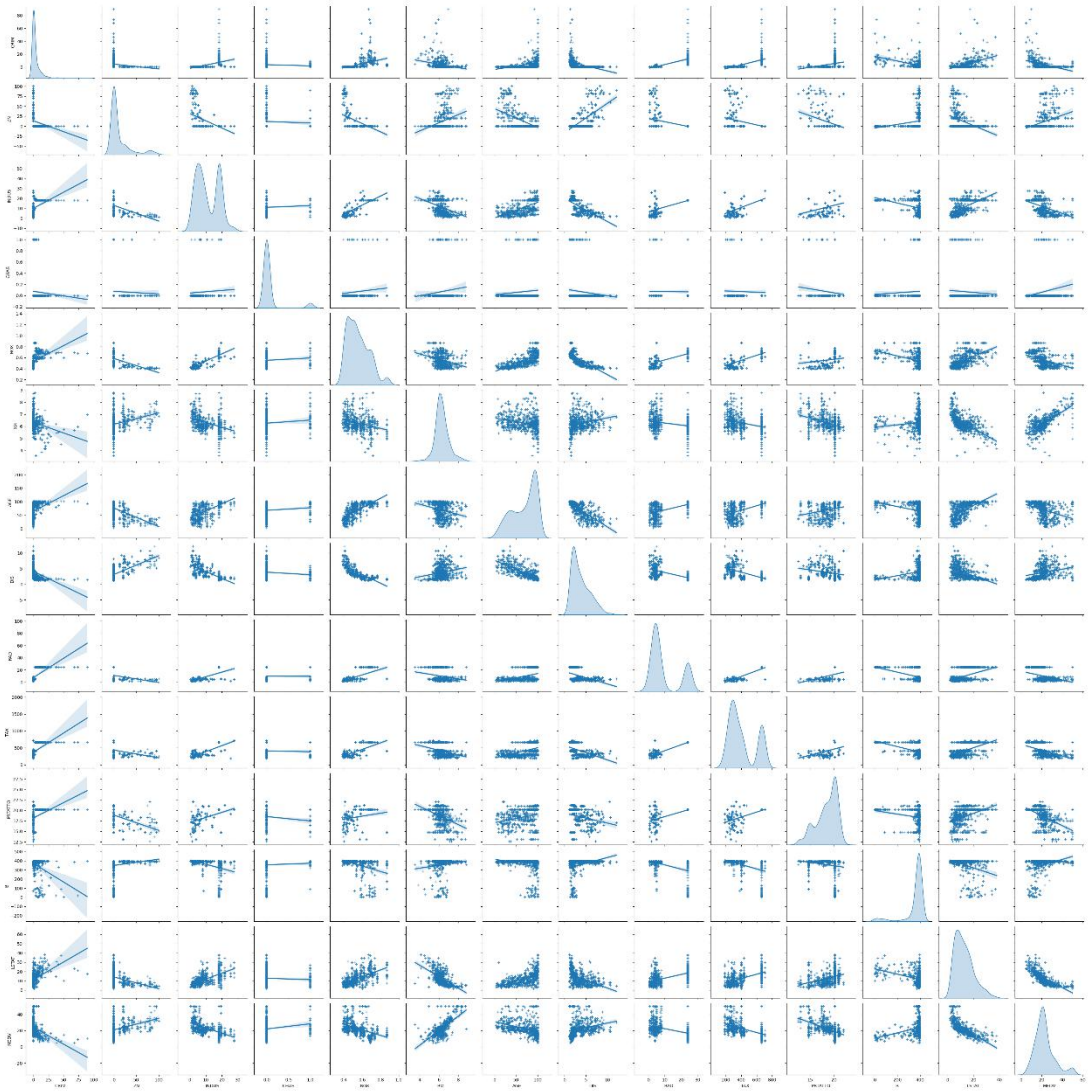
图表 7 描述性统计

数据预处理(又称数据清理、数据整理或数据处理)是指对数据进行各种检查和审查的过程，以纠正缺失值、拼写错误、使数值正常化/标准化以使其具有可比性、转换数据(如对数转换)等问题。

数据的质量将对生成模型的质量产生很大的影响经过完整性检验，结果显示：数据点完备，无需填补；数据皆为浮点型，无需处理类别变量。而上图统计特征显示，房价最大值是最小值的 10 倍。由于表格太大，其余特征与房价的关系及特征间交叉关系并不直观，故需要进行数据可视化。

4.1.2. 自变量间及自变量与因变量间关系分析

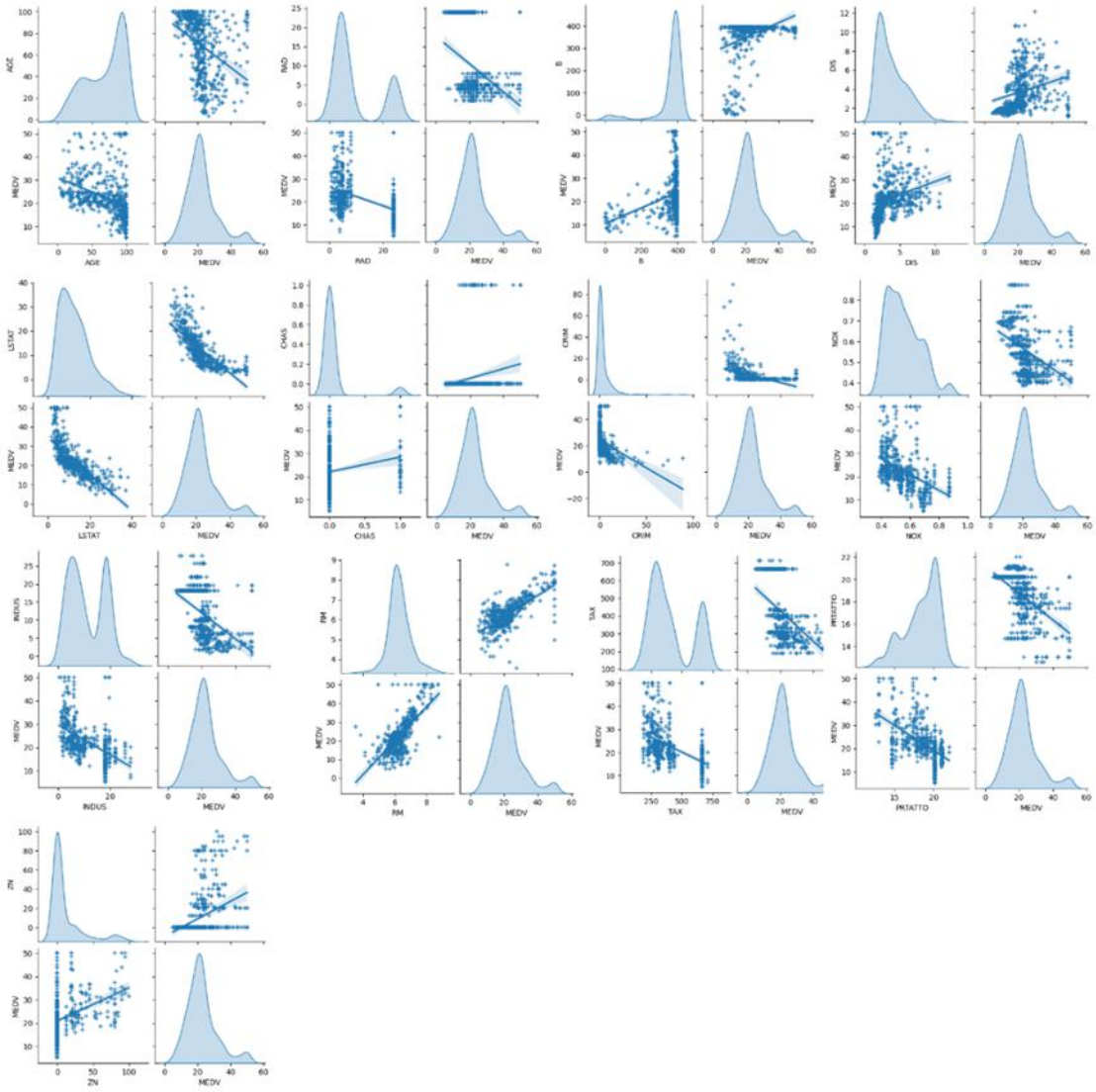
该过程的分析，我们采用 Seaborn 库对图像进行绘制，Seaborn 是基于 matplotlib 的 Python 可视化库，如下图 9：可以看到对角线上是各个属性的分布图，而非对角线上是两个不同属性之间的相关图，最后一行和一列为 price 与其他特征的相关关系图



图表 8　　Sum Housing Price Relation

接下来，重点探究 13 个因素与房价间的关系，分别绘制两两间关系图，得到组图如下图 10：

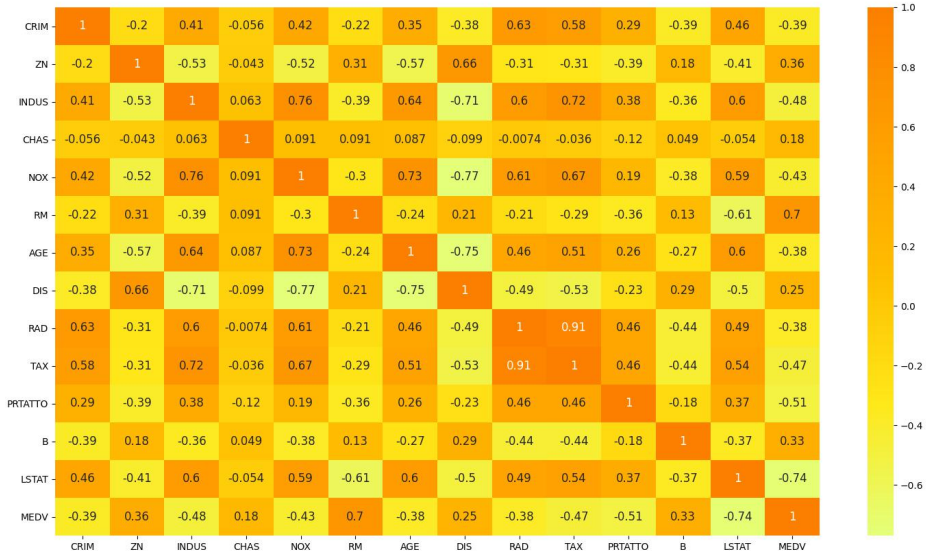
观察图中散点分布与直线趋势，由非对角线相关图，可以看出 NOX，RM，PTRATIO，LSTAT 与 price 有较为明显的线性相关性，再由对角线分布图可以看出：NOX(一氧化氮的浓度)，RM(每个住宅的平均房间数)，PTRATIO(各镇的师生比率),LSTAT(底层人口的百分比)与 price 的分布图较为接近，均呈现后尾性不明显的近似单峰分布



图表 9 Price relation to all factors

4.1.3. **相关系数分析**

对于各个特征间的相关系数，采用 seaborn 绘制的热力图进行分析，热力图在实际中常用于展示一组变量的相关系数矩阵，在展示列联表的数据分布上也有较大的用途，针对本题，通过热力图可以非常直观地感受到数值大小的差异状况，热力图的右侧是颜色带，上面代表了数值到颜色的映射，数值由小到大对应色彩由浅到深。从下面的 heatma（仅观察最后一列即可）中我们可以得到：LSTAT 与 price 的相关程度最高(r=-0.74)，其次是 RM,PTRAIO(|r|>=0.5), INDUS，TAX，NOX(|r|>=0.4)。



图表 10　　Housing Price Relation Heat

4.1.4. 综合因素分析

综合散点图&拟合直线，自身分布图，相关热力图的分析结果，可以定性地得出，房价可能与 LSTAT(底层人口的百分比)，RM(每个住宅的平均房间数)，PTRATIO(各镇的师生比率)，TAX(每$ 10,000 的全值财产税率)，NOX(一氧化氮的浓度)，CRIM（城镇的人均犯罪率）有较强的相关性，其余因素相关性较弱。

详细解释如下：

① CRIM（城镇的人均犯罪率）：与房价呈负相关，且相关系数约为 0.4

② ZN（大于 25,000 平方英尺的地块的住宅用地比例）：与房价无明显相关

③ INDUS（每个镇的非零售业务英亩的比例）：与房价呈较强负相关，且相关系数约为 0.48

④ CHAS（查尔斯河虚拟变量）：与房价无明显相关

⑤ NOX（一氧化氮的浓度）：与房价呈现较明显负相关，相关系数约为 0.43

⑥ RM（每个住宅的平均房间数）：与房价呈现明显正相关，相关系数约为 0.7

⑦ AGE（1940 年之前建造的自有住房的比例）：与房价无明显相关

⑧ DIS（到五个波士顿就业中心的加权距离）：与房价无明显相关

⑨ RAD（径向公路通达性的指标）：与房价无明显相关

⑩ TAX（每$ 10,000 的全值财产税率）：与房价呈现负相关，相关系数

约为 0.43

⑪ PTRATIO（各镇的师生比率）：与房价呈现较强正相关，相关系数约为 0.51

⑫ B（非裔美国人的比例）：与房价无明显相关

⑬ LSTAT（底层人口的百分比）：与房价呈现强负相关，相关系数约为 0.74

本次分析仅为定性分析，很可能不准确，甚至可能会与后期量化分析结果相矛盾，最终影响结果以后期结果为准。

4.2. 问题二模型的建立与求解

4.2.1. 主成分分析及特征提取

基于问题 1，考虑到不少特征与房价并无强相关性，若带入所有数据可能会对模型训练引入噪音，所以需要对原始数据进行主成分分析（PCA），对特征进行提取。依据 2.2 中的分析结果，选择 PCA 参数为 5 时的特征提取数据进行后续的分析，依据 2.2 图片，此时保留了原数据 85%以上的差异性，满足要求。

4.2.2. 模型的比较与选取

目前，常见的回归分析方法有：

LinearRegression()
Ridge(alpha=0.01)
Lasso()
ElasticNet( )
KNeighborsRegressor()
DecisionTreeRegressor()
SVR()
AdaBoostRegressor()
GBRr()
RandomForestRegressor()
ExtraTreesRegressor()

等十余种，我们重点依据以下指标对上述模型进行评估：

learning_rate: 学习率，模型是 0
n_estimators: 弱学习器的数目，默认值 100
max_depth: 每一个学习器的最大深度，默认为 3
min_samples_split: 可以划分为内部节点的最小样本数，默认为 2

min_samples_leaf: 叶节点所需的最小样本数，默认为 1

生成表格数据如下（详见文件包）

	estimator	min_score	mean_sco	max_score	std_score	C	gamma	kernel	n_neighbc	weights	max_dept	max_featu	min_samp	n_estimat	learning_r	min_samp	loss	alpha	l1_ratio	max_iter
3478	ExtraTreesRegr	0.78009621	0.825164	0.863984	0.034531	NAN	NAN	NAN				3								
3477	ExtraTreesRegr	0.78443863	0.809176	0.848525	0.028129	NAN	NAN	NAN				2								
3235	GradientBoosti	0.74632498	0.806997	0.849568	0.04405	NAN	NAN	NAN			6	3		100	0.1	3				
3231	RandomForest	0.75478822	0.805758	0.88148	0.054603	NAN	NAN	NAN					6	100						
3227	RandomForest	0.73645857	0.803882	0.876727	0.057393	NAN	NAN	NAN					3	50						
3230	RandomForest	0.73302662	0.802727	0.871765	0.056642	NAN	NAN	NAN					6	50						
3233	RandomForest	0.75303628	0.801743	0.873903	0.052056	NAN	NAN	NAN					9	50						
3229	RandomForest	0.75336651	0.801398	0.866356	0.047655	NAN	NAN	NAN					6	10						
3234	RandomForest	0.74187215	0.801081	0.876744	0.056277	NAN	NAN	NAN					9	100						
3228	RandomForest	0.74677936	0.800669	0.872898	0.053096	NAN	NAN	NAN					3	100						
3226	RandomForest	0.72323564	0.799369	0.872041	0.060799	NAN	NAN	NAN					3	10						
3260	AdaBoostRegr	0.73670404	0.78699	0.86007	0.05288	NAN	NAN	NAN						50	1		linear			
3265	AdaBoostRegr	0.72275538	0.786697	0.871378	0.062422	NAN	NAN	NAN						100	1		exponential			
3264	AdaBoostRegr	0.7439856	0.786627	0.854469	0.048497	NAN	NAN	NAN						50	1		exponential			
3261	AdaBoostRegr	0.72026339	0.784078	0.870666	0.063478	NAN	NAN	NAN						100	1		linear			
3255	AdaBoostRegr	0.74053261	0.783576	0.865575	0.058006	NAN	NAN	NAN						100	0.3		linear			
3232	RandomForest	0.69052104	0.782593	0.889823	0.082067	NAN	NAN	NAN					9	10						
3259	AdaBoostRegr	0.72676162	0.778962	0.857016	0.056231	NAN	NAN	NAN						100	0.3		exponential			
3249	AdaBoostRegr	0.73402542	0.777536	0.855567	0.055298	NAN	NAN	NAN						100	0.1		linear			
3258	AdaBoostRegr	0.73804632	0.776137	0.849817	0.052109	NAN	NAN	NAN						50	0.3		exponential			
3253	AdaBoostRegr	0.71607169	0.771934	0.851879	0.058	NAN	NAN	NAN						100	0.1		exponential			
3254	AdaBoostRegr	0.71553794	0.771157	0.850177	0.057403	NAN	NAN	NAN						50	0.3		linear			
3251	AdaBoostRegr	0.72151898	0.765082	0.844263	0.056083	NAN	NAN	NAN						100	0.1		square			
3256	AdaBoostRegr	0.71507806	0.764352	0.849385	0.06038	NAN	NAN	NAN						50	0.3		square			
3243	AdaBoostRegr	0.71051361	0.763857	0.83534	0.052549	NAN	NAN	NAN						100	0.05		linear			

由表格数据分析可得，GBRr 模型是十余种模型种综合指标最好的，确定模型。且依据上方表格数据，GBRr 模型参数为：learning_rate=0.1, max_depth=6, max_features=13, min_samples_leaf=3, n_estimators=100 时，训练效果达到最好。

4.2.3. **选定最终模型与参数，进行模型评估**

综上所述，选定 GBRr 模型，确定参数后，将数据按照 7:3 分为训练集与预测集进行房价预测，并依据：

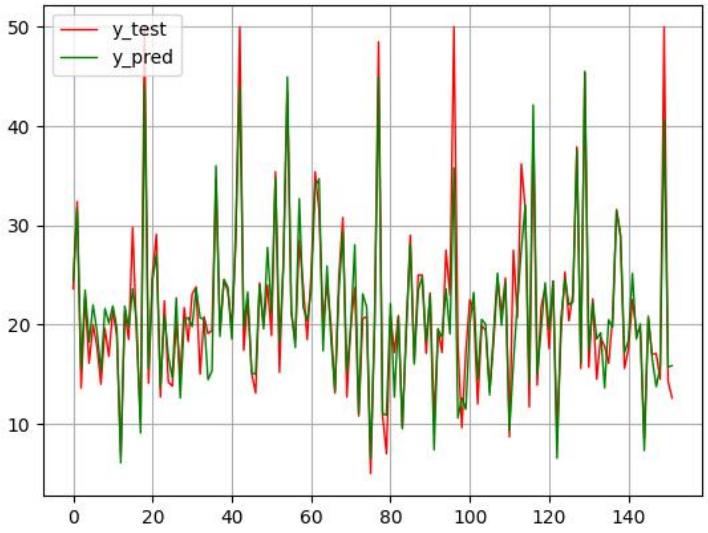
① 平均绝对误差 MAE

② 均方误差 MSE

③ 解释方差分 EVS

④ 拟合优度 R2

这四个指标对训练结果进行测评，其中某 100 次训练后完成测评结果如下，



图表 11 Test&Pred

```
  return f(**kwargs)
平均绝对误差MAE: 0.2015812010424482
均方误差MSE: 0.07881756213222756
解释方差分EVS: 0.9070936535346102
R2得分: 0.9070258708825613
(base) PS C:\Users\Administrator\Desktop\QuestionB>
```

图表 12 模型训练测试收敛情况

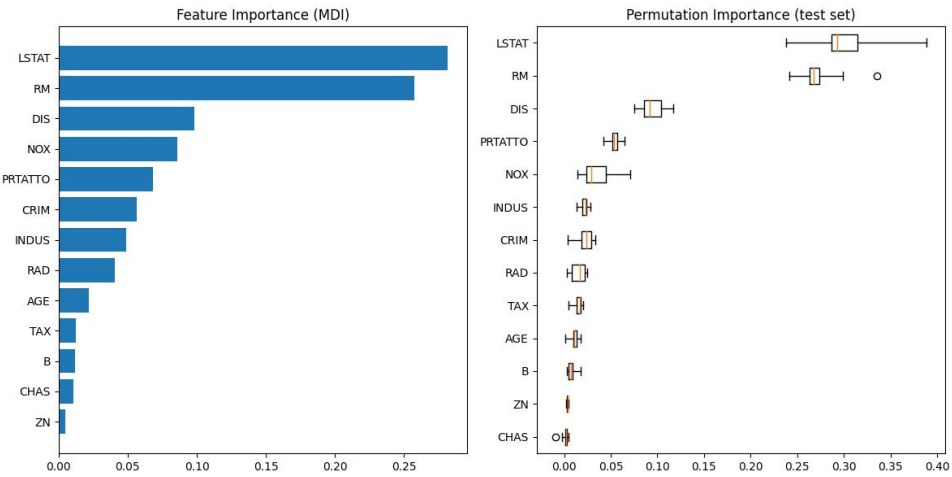
由上图数据可知，mae 为 0.2，这远小于波士顿各地区房价均值 22.3。这在一定程度上反映了此模型的预测具有较小的偏差，且拟合优度基本在 0.85~0.90 之间，拟合效果非常好。

4.2.4. 进行所有因素的重要性排序

在拟合完成的基础上，我们还对于除了房价本身在外的 13 个因素的重要性进行了探究，分别生成了具体数据，条形图，箱线图，如下所示，到这一步为止，我们已经得到了所有因素的定量化表示，如下图：

```
[0.08961794 0.01367896 0.06959024 0.00400217 0.0579043  0.27612356
 0.03383958 0.04672518 0.00246914 0.0524549  0.0306322  0.02485753
 0.2981043 ]
```

图表 13 所有因素的定量化表示



图表 14 Feature Importance

由上图分析可得：

① LSTAT(底层人口的百分比)对 price 有最强解释性(和之前的结论相符)，其次是 RM（每个住宅的平均房间数)，其他特征对 price 预测的重要程度则远小于前二者。

② DIS 城镇到五个波士顿就业中心的加权距离，NOX 一氧化氮的浓度在重要性排名中分列第三第四，符合常理。

4.3. 问题三 方法一模型的建立与求解

4.3.1. 方法一模型的建立

由问题 1 和问题 2 可求出房价与 13 个指标之间的关系。

针对问题 3，对某事物进行客观的评价时，评价因素可能很多，不能只根据一个指标的好坏作出判断，而是根据多个因素进行综合评价。所以对房子宜居性的分析采用多变量的评价法

具体步骤为：

(1) 建立房子宜居性的理想方案

房子的宜居性由 14 个变量影响，根据影响因素与宜居性的关系，建立理想方案为：

$$u = (u_1，u_2，u_3，u_4，u_5，u_6，u_7，u_8，u_9，u_{10}，u_{11}，u_{12}，u_{13}，u_{14})$$

(2) 构造相对偏差矩阵，进行数据的标准化

由于不同变量之间会存在量纲的差异，使得个变量之间没有可比性，无法体现个方案之间的优劣，相对偏差矩阵的元素，消除了量纲，解决这一问题。

相对偏差矩阵的公式为：

$$r_{ij} = \frac{|a_{ij} - u_j|}{max(a_{ij}) - mina(a_{ij})}$$

其中 i=1,2,…,506;j=1,2,…,14。r 为相对偏差矩阵的元素。

(3) 确定指标权重，确定指标的敏感度

多个因素进行综合评价时，通常有额权重问题。这里权重的确定原则为:如果某项指标的数值能明确区分开各个被评价的对象，说明该指标的评价信息丰富，给予较大的权重。区分度用方差来体现，方差大权重就大。

各指标的权重系数为：

$$v_j = \frac{s_j}{\overline{|r_j|}}$$

其中 j=1,2，…，14。

分母为各个因素的平均值，分子为标准差。

然后进行归一化处理：

$$w_j = \frac{v_j}{\sum_{j=1}^{14} v_j}$$

权重向量为

$W = (w_1, w_2, w_3, w_4, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}, w_{13}, w_{14})$

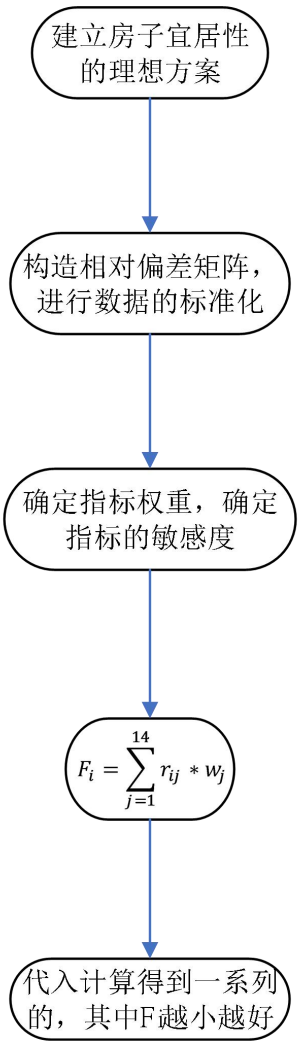
(4) 与理想方案越接近的方案宜居性越好

建立模型：

$$F_i = \sum_{j=1}^{14} r_{ij} * w_j$$

代入计算得到一系列的F_i，其中F_i越小越好。

流程如图 18 所示。



图表 15 方法一求解流程

4.3.2 **方法一模型的求解**

(1) 理想方案的求解

根据数据可以得到人均犯罪率 X1 的数值越小越好，所以理想方案中的 u1 取 X1 中最小的值，根据这一标准可以继续得到余下的。

最终的理想方案为：

u1	u2	u3	u4	u5	u6	u7	u8	u9	u10	u11	u12	u13	u14
0.0063	80	0.46	1	0.385	8.78	2.90	1.1296	24	187	12.6	0.32	1.73	5

(2) 权重系数求解：

先根据权重系数公式

$$v_j = \frac{s_j}{|\overline{r_j}|}$$

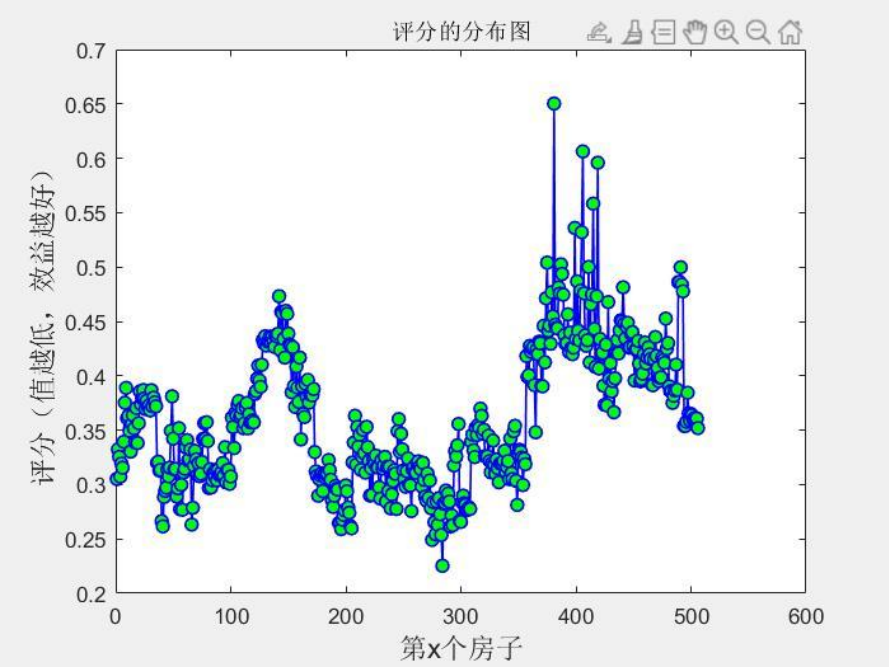
求得权重系数，然后有公式

$$w_j = \frac{v_j}{\sum_{j=1}^{14} v_j}$$

进行归一化得到 W，如下表格：

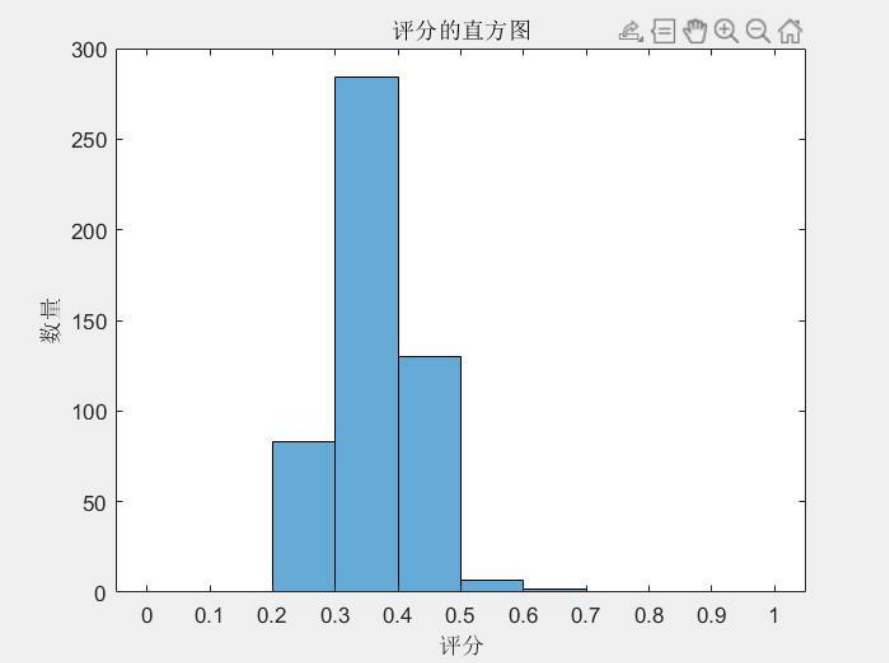
w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14
0.266	0.0347	0.071	0.030	0.076	0.031	0.047	0.088	0.067	0.084	0.041	0.028	0.072	0.058

(3) 将每个房子的评分 F 绘制如下图形



图表 16 评分分布图

评分的直方图如下：



图表 17 评分直方图

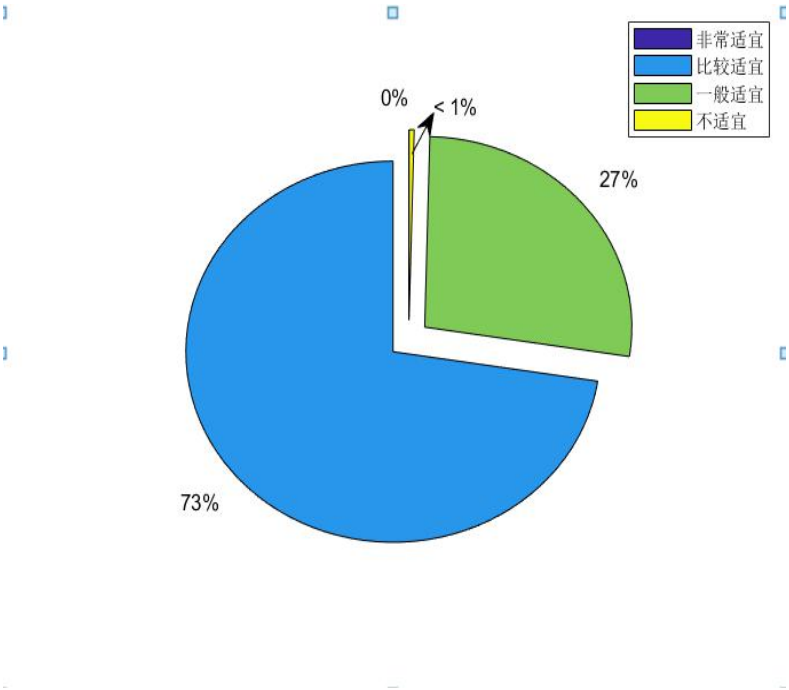
可见，506 组数据的评分集中在 0.2-0.5 之间，超出此范围的数据量较小。

(4) 评级

将房屋的宜居性由高到低划分为非常适宜、比较适宜、一般适宜和不适宜四个等级。

分数	等级
0.6-1	不适宜
0.4-0.6	一般适宜
0.2-0.4	比较适宜
0-0.2	非常适宜

图表 18 宜居性评估等级



图表 19 宜居性评估饼状图

由图可知，“非常适宜”和“不适宜”的房屋占比极低，不到 0.1%；73%的房屋属于“比较适宜”的等级；26.9%的房屋属于“一般适宜”的等级。

4.4. 问题三 方法二模型的建立与求解

4.4.1. AHP 模型层次分析法总述

人们在进行社会的、经济的以及科学管理领域问题的系统分析中，面临的常常是一个由相互关联、相互制约的众多因素构成的复杂而往往缺少定量数据的系统。层次分析法为这类问题的决策和排序提供了一种新的、简洁而实用的建模方法。

运用层次分析法建模，大体上可按下面四个步骤进行：

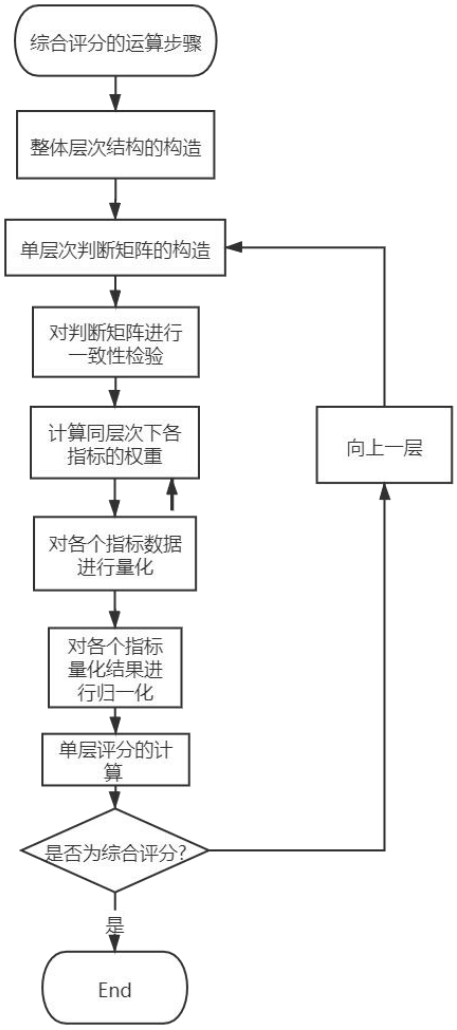
（i）建立递阶层次结构模型；

（ii ）构造出各层次中的所有判断矩阵；

（iii ）层次单排序及一致性检验；

（iv ）层次总排序及一致性检验。

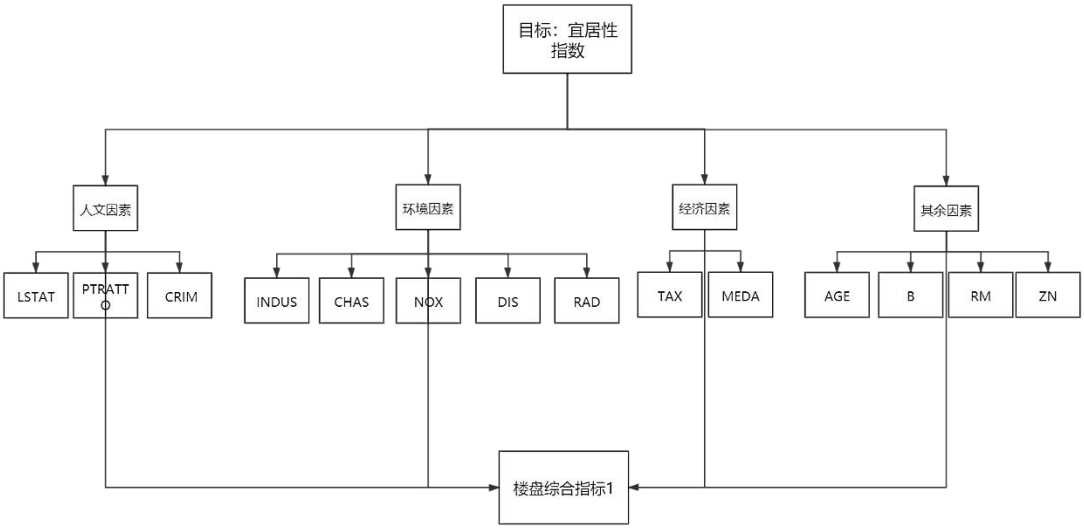
重构的层次分析法流程图如下：



图表 20 层次分析法流程

4.4.2. 建立递阶层次结构模型

层次分析法是用来根据多种准则，或是说因素从候选方案中选出最优的一种数学方法，最顶层是我们的目标，中间层是判断候选方物或人优劣的因素或标准，在分层以后，为了选出最优候选给目标层分配值 1.000，然后将这一值作为权重，分配给不同因素，对应因素的权重大小代表该因素在整个选择过程中的重要性程度。然后对于候选方案，每一个标准再将其权重值分配给所有的候选方案，每一方案获得权重值，来源于不同因素分得的权重值的和。针对本题目，我们依据对数据的理解建立层次如下图：

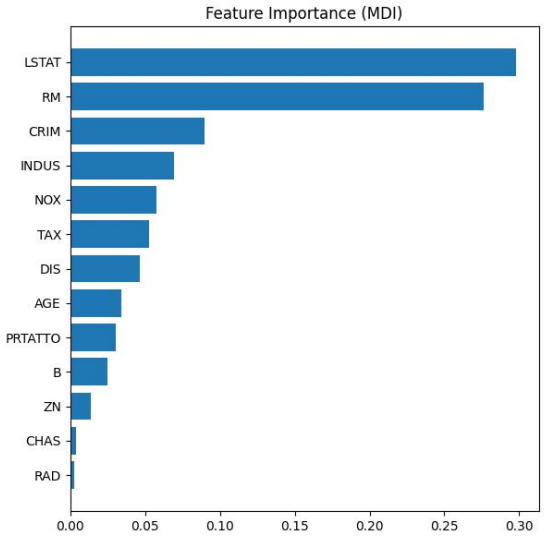


图表　AHP 层次分析结构

在分层以后，为了选出最优候选给目标层分配值，然后将这一值作为权重，分配给不同因素，对应因素的权重大小代表该因素在整个选择过程中的重要性程度然后对于候选方案，每一个标准再将其权重值分配给所有的候选方案，每一方案获得权重值，来源于不同因素分得的权重值的和。本次根据之前获得的数据范围，将，权重取在 1~150 范围之间。

4.4.3. **构造出各层次中的所有判断矩阵（**comparison matrix**）**

分析发现：直接要给各个因素分配权重比较困难，在不同因素之间两两比较其重要程度是相对容易的，所以，现在将不同因素两两作比获得的值 aij 填入到矩阵的 i 行 j 列的位置，则构造了所谓的比较矩阵，对角线上都是 1（因为是自己和自己比较），数值越大，意味着两个因素相比，该因素重要程度越大，矩阵参数大小关系依据下图所示：



计算原理及公式如下：

层次单排序是指，对于上一层某因素而言，本层次各因素的重要性的排序。具体计算是:对于判断矩阵 B，计算满足 BW = λ_{max}W 的特征根与特征向量。

式中λ_{max}为矩阵 B 的最大特征根，W 为对应于λ_{max}的正规化的特征向量，W 的分量ω_i即是相应元素单排序的权值。

利用判断矩阵计算各因素 C 对目标层 Z 的权重(权系数)

1.将 A 的每一列向量归一化得:$\omega_{ij}=\frac{a_{ij}}{\sum_{i=1}^{n} a_{ij}}$

2. 对而ω_{ij}按行求和得$\omega_i=\sum_{j=1}^{n}\omega_{ij}$

3. 将ω_{ij}归一化$\omega_i=\frac{\omega_i}{\sum_{i=1}^{n}\omega_i}$, $\omega=(\omega_1, \omega_2,\cdots\cdots,\omega_n)^T$，即为近似特征根（权向量)；

4.计算$\lambda=\frac{1}{n}\sum_{i=1}^{n}\frac{(A\omega)_i}{\omega_i}$，作为最大特征根的近似值。

4.4.4. 一致性检验

只有当矩阵满足一致性，即 aij*ajk = aik 的情况下，就是说如果 i 对 j 的重要程度是 a，j 对 k 的重要程度是 b，那么理所应当 i 对 k 的重要程度应该 a*b，有点符合“传递性"的感觉。但事实上不是这样的。所以需要进行一致性检验，如果在一定的合理范围之内，矩阵不需要修改，如果不在，则需要修改矩阵。

1.一致性指标：CI= $\frac{\lambda-n}{n-1}$, CI=0 时 A 一致，CI 越大，A 的不一致性程度越严重；

2.随机一致性指标 RI；

随机一致性指标 RI：

n	1	2	3	4	5	6	7	8	9	10	11
RI	0.00	0.00	0.58	0.90	1.12	1.24	1.32	1,41	1.45	1.49	.151

对矩阵一致性的测评我们采用 CR 这一标准 CR=CI/RI

当 CR<0.10 时，认为判断矩阵的一致性是可以接受的，否则应对判断矩阵作适当修正。

4.4.5. 输入子指标的打分向量，得到重要性权重向量

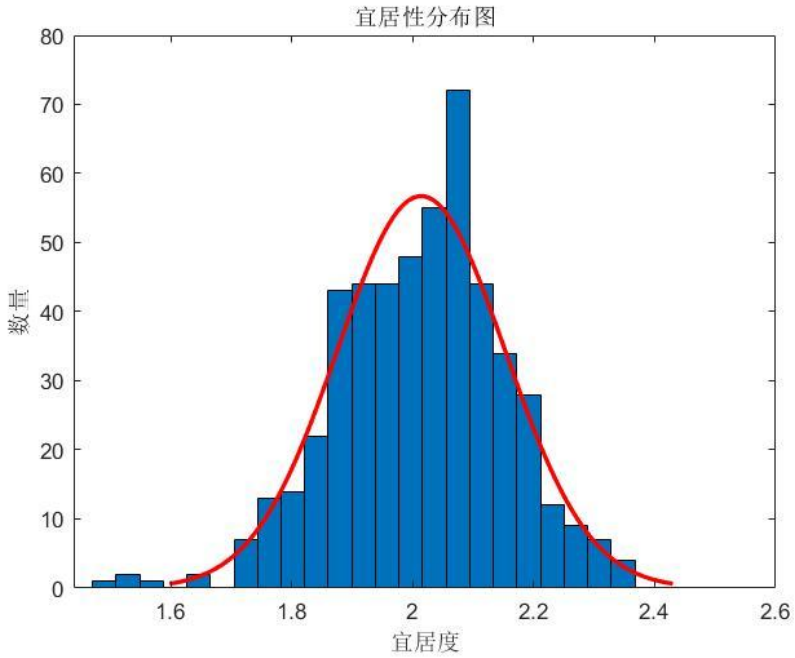
基于以上步骤，其次进行最大特征值对应的特征向量的归一化，输入子指标的打分向量，得到重要性权重向量，将 14 个因素打分，并通过层次提取分别对于人文，环境，经济，其他这四个方面进行打分，再将四个因素综合起来得到楼盘综合水平打分，最后得到宜居性的指标。（如下表，具体见文件包）

	房价中位数	人文指数	自然环境	消费水平	楼盘综合水平	宜居性
0	24	2.239097516	3.331454545	0.778142494	2.283767423	2.219534353
1	21.6	2.033447649	3.257525771	0.684310433	2.247495876	2.080773472
2	34.7	2.091847851	3.257525771	0.858977099	2.247495876	2.142456466
3	33.4	2.047411515	3.171706087	0.818743003	2.288873814	2.096873067
4	36.2	2.019726283	3.171706087	0.856076336	2.288873814	2.09041296
5	28.7	2.021494142	3.171706087	0.756076336	2.288873814	2.072314547
6	22.9	2.159441482	3.146897628	0.780651399	2.242319381	2.134526406
7	27.1	2.082072522	3.125624901	0.836651399	2.243123505	2.102443198
8	16.5	1.958198084	3.118446719	0.695318066	2.243394845	2.013419105
9	18.9	2.105243312	3.090628538	0.727318066	2.244446392	2.085039009
10	15	2.066402898	3.104013992	0.675318066	2.243940412	2.059244164
11	18.9	2.149546383	3.110559447	0.727318066	2.24369299	2.111263575
12	21.7	2.121900247	3.15287581	0.764651399	2.242093402	2.114499683
13	20.4	1.835306927	3.219511858	0.742737913	2.238400619	1.985317939
14	18.2	1.812366857	3.232908221	0.71340458	2.237894227	1.971600025
15	19.9	1.832930895	3.230906403	0.736071247	2.237969897	1.985497483

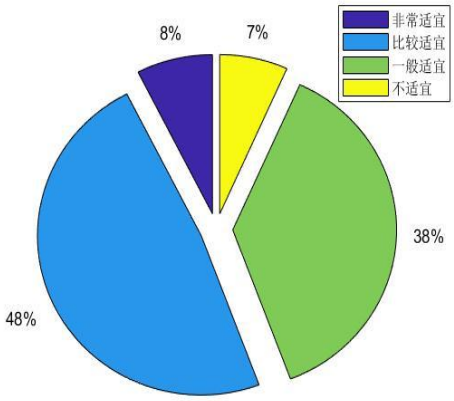
图表 21 宜居性指标

首先，依据表格可以得到，宜居性的的范围是 1.48~2.35，

最后借助 Matlab 对数据进行可视化分析，依据可视化得到的结果对波士顿房屋进行分类，可视化结果如图：



图表 22 宜居度可视化分类



图表 23 宜居性饼状图分类

基于 matlab 正态分布的结果，我们将房屋按照宜居性分为 4 类，1.8 以下属于不适宜，占比 7%，1.8~2 属于一般适宜，占比 38%，2~2.2 属于比较适宜，占比 48%，2.2 以上属于特别适宜，占比 8%。

4.5. 问题四 针对不同人群的购房建议

首先，不论是哪种类型的购房者，治安都是购房时要考虑的重要因素，即要求 CRIM 取值越小越好。

4.5.1. 新婚购房人群

新婚购房人群的特点：

1、新婚夫妇买房以小户型为主， RM 取值以 1 到 3 为宜。

2、新婚夫妇大多都在上班，一般要求交通方便，因此建议购买距离站或者公交站较近的住宅。即要求 DIS 尽可能小。

购房建议：这类人适合地区人均犯罪率低，每户平均房间数 1-3 间左右，与劳动力聚集区的加权距离小和辐射式公路接近指数大的房屋，方便工作上下班。

4.5.2. 投资购房人群

改善型购房人群的特点：

投资最看重的自然是房屋的升值潜力，知名开发商开发的品质楼盘、学校对应片区的户型房源、商业地段商铺和新开楼盘等，这都是很具有投资价值的房源。

购房建议：因此 DIS 取值越小越好，PTRATIO 取值越大越好。

4.5.3. 教育需求人群

教育需求人群的特点：

房产要求临近重点小学、中学。

购房建议：PTRATIO 取值越大越好，地区人均犯罪率低的房子

4.5.4. 改善型购房人群

改善型购房人群的特点：

改善型购房人群即收入颇丰，但仍不满意当前居住品质的购房者。一般为两代或者三代家庭成员住，对小区绿化和环境要求比较高。

购房建议：B 和 LSTAT 两项取值以小为宜。

购房建议整合为下表：

购房人群类型	特点	购房建议
新婚购房人群	买房以小户型为主，要求交通方便	每户平均房间数 1-3 间左右，地

		区人均犯罪率低，与劳动力聚集区的加权距离小和辐射式公路接近指数大的房屋
投资购房人群	看重房屋的升值潜力	与劳动力聚集区的加权距离小，地区人均犯罪率低，各镇的师生比率高的房屋
教育需求人群	房产要求临近重点小学、中学	各镇的师生比率越大越好，地区人均犯罪率低
改善型购房人群	两代或者三代家庭成员住，对小区绿化和环境要求比较高	底层人口的百分比低，地区人均犯罪率低，靠近查尔斯河的房屋

图表 24 针对不同人群的购房建议

五、 模型检验与评价

5.1 GBRr 模型的检验

5.1.1. 灵敏度分析

最佳模型不一定是鲁棒模型。 有时，模型太复杂或太简单而无法充分概括新数据。 有时，模型可能会使用不适用于给定数据结构的学习算法。 在其他时候，数据本身可能太嘈杂或包含的样本太少，以至于模型无法充分捕获目标变量，即模型拟合不足。 下面的代码单元使用不同的训练和测试集运行 fit_model 函数十次，以查看特定样本的预测如何随其训练的数据而变化。

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 1: $24.04

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 2: $24.17

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 3: $23.97

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 4: $24.01

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 5: $23.92

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 6: $23.99

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 7: $23.94

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 8: $24.19

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 9: $28.60

ShuffleSplit(n_splits=404, random_state=0, test_size=0.2, train_size=None)

Trial 10: $24.15

Range in prices: $4.68

以上 10 次预测除第 9 次出现较大偏差，其余预测值紧紧围绕 24 附近波动（该样本真实价格为 24.0），模型预测准确鲁棒性较强。（学生紧接着预测了再随机抽取了十个样本，结果仍表明最终模型有较强鲁棒性）

5.1.2. 模型优点

①可以灵活处理各种类型的数据，包括连续值和离散值。
②在相对少的调参时间情况下，预测的准备率相对于其他预测模型，准确性很高

5.1.3. 使用性探讨

模型优化上：

1) 本次训练数据并未出现异常数据，若以后数据中可以看到异常值，也可以依据上述方法对其进行处理。

2)后续可搭建神经网络模型与本次的训练模型进行评估比较。

数据搜集上：

尽管此模型最终预测效果良好，但在尝试使用给定参数预测给定新数据点的房价时，在实际环境中使用此模型之前，我们需要解决一些缺陷：

首先，以上预测使用的是 1978 年房价数据。由于通胀、动乱等政治经济社会因素的影响，若将该模型的特征用于如今的房价预测可能效果不会很显著，也许有一些与如今房价更紧密的特征（如：与学校的距离等）未被纳入此模型。

其次，一个社区内的房价也可能有很大差别（美国有不少贫民窟紧挨富人区），而我们的预测模型有将同一社区房价同质化的缺点。

再者，在城市中收集的数据不适用于农村城市，因为城市和农村都具有不同的特征，每个特征的值也不同。（总之，统计数据的采集过程可能有诸多细节有待优化。）

5.2. 综合指数法的分析与评价

5.2.1. 灵敏度分析

影响综合评价结果的因素有两个，一是指标的观测值．二是相应的权数。

线性加权模型属于主因素突出型的评价方法，其合成结果突出了较大评价值且权数较大者的作用，可以反映出指标重要程度的差异，适用于各评价指标间相互独立的情况，即各指标对综合水平的贡献彼此是没有什么影响的。由于线性评价模型只是实现"部分之和等于整体之和"思想，如果各指标之间不相互独立，则会造成一些指标信息的重复。不能反映客观情况。

各评价指标间可以线性地等量补偿，即此升彼降，任一指标评价分数的减少都可以用另一指标评价分数的相应增加来维持总评价分数的不变。因而这种合成方法对不同对象间指标评价值的差异反应不大敏感，从而使这种方法区分各评价对象的灵敏度相对于其他方法低一些。

5.2.2. 模型分析

从综合指数评价方法的性质看，它可以是分层处理，就是说，当被评价事物比较复杂，本身具有层次性，评价指标比较多时，可以先从被评价事物的低层次上进行综合评价。然后把低层次上的综合评价结果再综合起来。

5.2.3. 使用性探讨

模型优化上：

1、综合指数评价模型在多指标综合评价过程中没有考虑评价指标间相关作用对评价结果的影响．方法本身并不能够消除这种影响。因而采用以上方法时，在多指标综合评价的第一步——选取评价指标上．既要注意指标的全面性，又要把彼此相关的指标剔除。否则可能产生评价指标间信息重复的问题。

2、评价指标权数通常属于估价权数和信息量权数．即从评价者对指标的估价或者指标包含被评价对象差异信息多少的角度来衡量指标重要程度。因此选择

不同的方法，可能有不同的结果．即使采用同样的方法．由于各指标的赋值不同、权重不同等．也有可能使评价结果不同。

3、由于采用的无量纲化方法的不同．导致评价结果有时具有惟一性，有时则不能。这一性质会影响到评价结果的时间和空间可比性。因此当我们要比较不同时间和空间的诸多被评价对象的综合水平时，往往需要把他们放在同一对象集合中作综合评价，结果才是可比的。

数据搜集上：

本文仅使用了题目给出的 13 或 14 个数据来评估宜居性。在实际运用中，由于宜居性是一个较难考量的指标，应采用大数据调查的模式考察评估宜居性最重要的因素，有助于优化分析结果。

参考文献

[1]李东月.房价预测模型的比较研究[J].工业技术经济,2006:67-69.

[2]王世良,王世波.AHP 模型在风险投资项目评价中的应用[J].企业经济,2004:78-79+52.

[3]刘小虎,李生.决策树的优化算法[J].软件学报,1998:78-81.

[4]谢开贵,李春燕,周家启.基于神经网络的负荷组合预测模型研究[J].中国电机工程学报,2002:85-89.

程序附录

第一问代码

```
'''包的引入'''
import xlrd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

'''数据读入与描述'''
names=['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX',
'PRTATTO','B','LSTAT','MEDV']
data = pd.read_excel('./bostonh.xlsx',names=names)
print(data)
print(data.describe())

'''所有关系图的输出'''
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="A
ccent_r")
sns.color_palette("hls", 8)
plt.savefig('Sum Housing Price Relation')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["CRIM","MEDV"])
plt.savefig('Housing Price & Crime.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["ZN","MEDV"])
plt.savefig('Housing Price & Zone.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["INDUS","MEDV"])
plt.savefig('Housing Price & Retail.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["CHAS","MEDV"])
plt.savefig('Housing Price & ChasRiver.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["NOX","MEDV"])
plt.savefig('Housing Price & NOX.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["RM","MEDV"])
plt.savefig('Housing Price & Room.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["AGE","MEDV"])
```

```
plt.savefig('Housing Price & Individual.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["DIS","MEDV"])
plt.savefig('Housing Price & Length.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["RAD","MEDV"])
plt.savefig('Housing Price & Access.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["TAX","MEDV"])
plt.savefig('Housing Price & Tax.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["PRTATTO","MEDV"])
plt.savefig('Housing Price & Teacher.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["B","MEDV"])
plt.savefig('Housing Price & Black.png')
sns.pairplot(data,kind="reg",diag_kind="kde", markers="+",palette="h
usl",vars=["LSTAT","MEDV"])
plt.savefig('Housing Price & Layer.png')

'''热力图分布与输出'''
corrmat = data.corr()
fig, ax = plt.subplots(figsize = (18, 10))
sns.heatmap(corrmat, annot = True, annot_kws={'size': 12},cmap="Wisti
a")
#plt.show()
plt.savefig('Housing Price Relation Heat')
```

第二问代码

```
#考虑到不少特征与房价并无强相关性，若带入所有数据可能会对模型训练引入噪音，
# 故而对所有模型，学生将对全数据集与经特征提取过的数据集分别套用相同模型，
# 用mse、mae、r2_score作为evaluation metrics（以r2_score为主）。

'''导入库'''
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,e
xplained_variance_score,r2_score
```

```
from sklearn.svm import LinearSVR
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn import linear_model
from sklearn.linear_model import ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor,GradientBoostingRegre
ssor,ExtraTreesRegressor,RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import learning_curve
from sklearn.model_selection import GridSearchCV
from statistics import mean, median
from sklearn import ensemble
from matplotlib.colors import ListedColormap
import logging
import logging.config
from sklearn.metrics import mean_squared_log_error
from sklearn.inspection import permutation_importance
from sklearn import datasets

'''加载模型以及提前设置的一些超参数'''
class EstimatorSelectionHelper:
    # 初始化，加载模型以及提前设置的一些超参数
    def __init__(self, models, params):
        if not set(models.keys()).issubset(set(params.keys())):
            missing_params = list(set(models.keys()) - set(params.keys
()))
            raise ValueError("Some estimators are missing parameters:
 %s" % missing_params)
        self.models = models
        self.params = params
        self.keys = models.keys()
        self.grid_searches = {}
        self.best = {}
    # 对每个模型的每组超参数都进行交叉验证
    def fit(self, X, y, cv=3, n_jobs=3, verbose=1, scoring=None, refi
```

```
t=False):
        for key in self.keys:
            print("Running GridSearchCV for %s." % key)
            model = self.models[key]
            params = self.params[key]
            gs = GridSearchCV(model, params, cv=cv, n_jobs=n_jobs,
                              verbose=verbose, scoring=scoring, refit=r
efit,
                              return_train_score=True)
            gs.fit(X,y)
            self.best[key] = {'score':gs.best_score_,'params':gs.best
_params_}
            self.grid_searches[key] = gs
    # 对结果进行统计
    def score_summary(self, sort_by='mean_score'):
        def row(key, scores, params):
            d = {
                 'estimator': key,
                 'min_score': min(scores),
                 'max_score': max(scores),
                 'mean_score': np.mean(scores),
                 'std_score': np.std(scores),
            }
            return pd.Series({**params,**d})

        rows = []
        for k in self.grid_searches:
            print(k)
            params = self.grid_searches[k].cv_results_['params']
            scores = []
            for i in range(self.grid_searches[k].cv):
                key = "split{}_test_score".format(i)
                r = self.grid_searches[k].cv_results_[key]
                scores.append(r.reshape(len(params),1))

            all_scores = np.hstack(scores)
            for p, s in zip(params,all_scores):
                rows.append((row(k, s, p)))

        df = pd.concat(rows, axis=1).T.sort_values([sort_by], ascendi
ng=False)

        columns = ['estimator', 'min_score', 'mean_score', 'max_score
', 'std_score']
```

```
        columns = columns + [c for c in df.columns if c not in columns]


        return df[columns]
    # 最优超参数.
    def best_params(self):
        return self.best


'''读取.CSV 文件及处理'''
# boston = datasets.load_boston()
# diabetes = datasets.load_diabetes()

my_matrix = pd.read_csv("./bostonh.csv")
#对于矩阵而言，将矩阵倒数第一列之前的数值给了XPCA_（输入数据），将矩阵大最后一列的数值给了y（标签）
X, y = my_matrix.iloc[:,:-1].values,my_matrix.iloc[:,-1].values
#利用 train_test_split 方法，将X,y 随机划分问，训练集（X_train），训练集标签（X_test），测试卷（y_train），
#测试集标签（y_test），安训练集：测试集=7:3 的
#概率划分，到此步骤，可以直接对数据进行处理
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)

#np.column_stack 将两个矩阵进行组合连接
train= np.column_stack((X_train,y_train))
test = np.column_stack((X_test, y_test))
# print(my_matrix[1])
#数据预处理
Stand_X = StandardScaler()  # 特征进行标准化
Stand_Y = StandardScaler()  # 标签也是数值，也需要进行标准化
X_train = Stand_X.fit_transform(X_train)
X_test = Stand_X.transform(X_test)
y_train = Stand_Y.fit_transform(y_train.reshape(-1,1)) # reshape(-1,1)指将它转化为1 列，行自动确定
y_test = Stand_Y.transform(y_test.reshape(-1,1))

'''成分方差，看降维保留了多少差异性'''
X_std=StandardScaler().fit_transform(X_train)
pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,7,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')#主成分的方差,用于量化了这些
```

```
方向的重要程度
plt.savefig('Components variance')

#系数选择5，核密度估计(kernel density estimation)是在概率论中用来估计未知的密度函数，
# 属于非参数检验方法之一。通过核密度估计图可以比较直观的看出数据样本本身的分布特征。

sklearn_pca=PCA(n_components=5)
X_Train=sklearn_pca.fit_transform(X_std)

'''生成表格对比数据'''
models2 = {}
models2["Linear"]        = linear_model.LinearRegression()
models2["Ridge"]        = linear_model.Ridge(alpha=0.01)
models2["Lasso"]         = linear_model.Lasso()
models2["ElasticNet"]    = ElasticNet()
models2["KNN"]           = KNeighborsRegressor() #K-近邻算法
models2["DecisionTree"]  = DecisionTreeRegressor() #决策树
models2["SVR"]           = SVR()
models2["AdaBoost"]      = AdaBoostRegressor()
models2["GradientBoost"] = GradientBoostingRegressor()
models2["RandomForest"]  = RandomForestRegressor()
models2["ExtraTrees"]    = ExtraTreesRegressor()

models1 = {
    'SVR': SVR(),
    'KNeighborsRegressor': KNeighborsRegressor(),
    'DecisionTreeRegressor': DecisionTreeRegressor(),
    'RandomForestRegressor': RandomForestRegressor(),
    "GradientBoostingRegressor":GradientBoostingRegressor(),
    "AdaBoostRegressor":AdaBoostRegressor(),
    "ElasticNet":ElasticNet(),
    "ExtraTreesRegressor":ExtraTreesRegressor()
}

params1 = {
    'SVR': {'kernel':('linear', 'rbf'), 'C':[1, 2, 4], 'gamma':[0.125,
 0.25, 0.5 ,1, 2, 4]},
    'KNeighborsRegressor': {'weights': ['uniform', 'distance'],
                            'n_neighbors': range(2,100)
                            },
    'DecisionTreeRegressor':  {'max_features': ['sqrt', 'log2', None],
```

```
                            'max_depth': range(2,1000),
                            },
    'RandomForestRegressor': {
        'min_samples_split': list((3,6,9)),'n_estimators':list((10,50,
100))},
    "GradientBoostingRegressor":{'n_estimators':[100], 'learning_rate
': [0.1], 'max_depth':[6],'min_samples_leaf':[3], 'max_features':
[3]},

    "AdaBoostRegressor":{'n_estimators': [50, 100], 'learning_rate' :
 [0.01,0.05,0.1,0.3,1],'loss' : ['linear', 'square', 'exponential']},

    "ElasticNet": {"max_iter": [1, 5, 10],
                   "alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                   "l1_ratio": np.arange(0.0, 1.0, 0.1)},
    "ExtraTreesRegressor":{
        'max_features': range(1,4,1),
    }

}
helper1 = EstimatorSelectionHelper(models1, params1)
helper1.fit(X_train, y_train, scoring='r2', n_jobs=2)
helper1.score_summary(sort_by='max_score')
df_gridsearchcv_summary = helper1.score_summary()
#print(type(df_gridsearchcv_summary.iloc[:,0:]))
print(df_gridsearchcv_summary.iloc[:,0:])
df_gridsearchcv_summary.iloc[:,0:].to_csv("Comparison.csv")

# 综上所述，使用GradientBoostingRegressor: learning_rate=0.1, max_dept
h=6,
# max_features=13, min_samples_leaf=3, n_estimators=100 并结合所有特征
进行预测效果较佳。

'''选定最终模型与参数，进行模型评估'''
params = {'n_estimators': 100, 'max_depth': 6, 'max_features':3, 'min
_samples_split': 2,
          'learning_rate': 0.1, 'min_samples_leaf': 3,'loss': 'ls'}
gbr = ensemble.GradientBoostingRegressor(**params)
test_score = np.zeros((params['n_estimators'],), dtype=np.float64)
# 估计器拟合训练数据
gbr.fit(X_train, y_train)

# 训练完的估计器对测试数据进行预测
y_pred = gbr.predict(X_test)
```

```
'''参数输出'''
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("平均绝对误差 MAE：{}".format(mae))
print("均方误差 MSE：{}".format(mse))
print("解释方差分 EVS：{}".format(evs))
print("R2 得分：{}".format(r2))

'''训练集与测试集偏差曲线'''
fig = plt.figure(figsize=(6, 6))
plt.subplot(1, 1, 1)
plt.title('Deviance')
plt.plot(np.arange(params['n_estimators']) + 1,gbr.train_score_, 'b-
',
         label='Training Set Deviance')
plt.plot(np.arange(params['n_estimators']) + 1, test_score, 'r-',
         label='Test Set Deviance')
plt.legend(loc='upper right')
plt.xlabel('Boosting Iterations')
plt.ylabel('Deviance')
fig.tight_layout()
plt.show()
plt.savefig('Pred_Deviance.png')

'''模型的训练测试收敛情况和各特征的重要性图示'''
feature_importance = gbr.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, np.array(my_matrix.columns)[sorted_idx])
plt.title('Feature Importance (MDI)')

result = permutation_importance (gbr, X_test, y_test, n_repeats=10,
                                 random_state=42, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(result.importances[sorted_idx].T,
            vert=False, labels=np.array(my_matrix.columns)[sorted_id
x],whis=4)
```

```
plt.title("Permutation Importance (test set)")
fig.tight_layout()
plt.show()
plt.savefig("Importance.png")

'''模型评价，作图'''
t = np.arange(len(X_test))
plt.plot(t, y_test, color='red', linewidth=1.0, linestyle='-', label=
'y_test')
plt.plot(t, y_pred, color='green', linewidth=1.0, linestyle='-', labe
l='y_pred')
plt.legend()
plt.grid(True)
plt.show()
plt.savefig('Test&Pred.png')
```

第三问代码

```
'''导入库'''
import pandas as pd
from math import *
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from scipy.optimize import  fsolve
import math

mpl.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
mpl.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
mpl.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
mpl.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

'''规定了加权方式,生成判断矩阵的简单方法'''
def get_judgement_matrix(scores):
    '''
    get judgement matrix  according to personal score.
    :param scores: a list, the item is the score range 1 to 10 means t
he importance of each sub-indicator.
    :return: judgement matrix, item range 1 to 9.

    - more: in judgement matrix:
    1 means two sub-indicators are the same important.

    3 means the first sub-indicator is a little important than anothe
```

```
r one.

    5 means the first sub-indicator is apparently important than anot
her one.

    7 means the first sub-indicator is strongly significant than anot
her one.

    9 means the first sub-indicator is extremely significant than ano
ther one.

    and 2, 4, 6, 8 are in the middle degree.
    '''

    # 评分1——150
    length = len(scores)

    array = np.zeros((length, length))
    for i in range(0, length):
        for j in range(0, length):
            point1 = scores[i]
            point2 = scores[j]
            deta = point1 - point2
            if deta < 0:
                continue
            elif deta == 0 or deta == 1:
                array[i][j] = 1
                array[j][i] = 1
            else:
                array[i][j] = deta
                array[j][i] = 1 / deta

    return array

'''获得判断矩阵的最大特征值和对应的特征向量'''
def get_tezheng(array):
    '''
    get the max eigenvalue and eigenvector
    :param array: judgement matrix
    :return: max eigenvalue and the corresponding eigenvector
    '''
    # 获取最大特征值和对应的特征向量
    te_val, te_vector = np.linalg.eig(array)
    list1 = list(te_val)
```

```
    max_val = np.max(list1)
    index = list1.index(max_val)
    max_vector = te_vector[:, index]

    return max_val, max_vector

'''获取RI值,随机一致性指标之一,另一个是CI'''
def RImatrix(n):
    '''
    get RI value according the the order
    :param n: matrix order
    :return: Random consistency index RI of a n order matrix
    '''
    n1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
    n2 = [0, 0, 0.52, 0.89, 1.12, 1.26, 1.36, 1.41, 1.46, 1.49, 1.52,
1.54, 1.56, 1.58, 1.59, 1.60]
    d = dict(zip(n1, n2))
    return d[n]

'''计算一致性,进行一致性检验'''
def consitstence(max_val, RI, n):
    '''
    use the CR indicator to test the consistency of a matrix.
    :param max_val: eigenvalue
    :param RI: Random consistency index
    :param n: matrix order
    :return: true or false, denotes whether it meat the validation of
consistency
    '''
    CI = (max_val - n) / (n - 1)
    if RI == 0:

        return True
    else:
        CR = CI / RI
        if CR < 0.10:
            #print("hhhhhhhh")
            return True
        else:
            #print("ZZZZZZZZZZZ")
            return False
```

```
def minMax(array):
    result = []
    for x in array:
        x = float(x - np.min(array)) / (np.max(array) - np.min(array))

        result.append(x)
    return np.array(result)

'''最大特征值对应的特征向量的归一化'''
def normalize_vector(max_vector):
    '''
    normalize the vector, the sum of elements is 1.0
    :param max_vector: a eigenvector
    :return: normalized eigenvector
    '''
    vector = []
    for i in max_vector:
        vector.append(i.real)
    vector_after_normalization = []
    sum0 = np.sum(vector)
    for i in range(len(vector)):
        vector_after_normalization.append(vector[i] / sum0)
    vector_after_normalization = np.array(vector_after_normalization)


    return vector_after_normalization

'''综合以上，输入子指标的打分向量，得到重要性权重向量。'''
def get_weight(score):
    '''
    get weight vector according to personal score.
    :param score: a list, the item is the score range 1 to 10 means th
e importance of each sub-indicator.
    :return: a list, the item is the weight range 0.0 to 1.0.
    '''
    n = len(score)
    array = get_judgement_matrix(score)
    max_val, max_vector = get_tezheng(array)
    RI = RImatrix(n)
    if consitstence(max_val, RI, n) == True:
        feature_weight = normalize_vector(max_vector)
        return feature_weight
    else:
        return [1 / n] * n
```

```
def getScore(array, point1, point2):
    '''
    a normalization function based on Human psychological satisfactio
n
    :param array: list, element is indicator's original value
    :param point1: the left expectation point, a list, [x1,y1]
    :param point2: the right expectation point, a list, [x2,y2]
    :return: normalized array
    '''
    x1 = point1[0]
    x2 = point2[0]
    y1 = point1[1]
    y2 = point2[1]

    def f1(a):
        equation1 = 1 / (1 + math.exp(-a * x1)) - y1
        return equation1

    def f2(a):
        equation1 = 1 / (1 + math.exp(-a * x2)) - y2
        return equation1

    # 存储归一化后的值
    values = []

    for i in array:
        try:
            i=i[0]
        except:
            pass
        if i < x1:
            sol3_fsolve = fsolve(f1, [0])
            a = sol3_fsolve[0]
            value = 1 / (1 + math.exp(a * (i - 2 * x1)))
        elif x1 <= i and i <= x2:
            value = (i - x1) * (y2 - y1) / (x2 - x1) + y1
        else:
            sol3_fsolve = fsolve(f2, [0])
            a = sol3_fsolve[0]
            value = 1 / (1 + math.exp(-a * i))
        values.append(value)

    # plt.scatter(array, values)
```

```
    # plt.show()
    return values

def show_score(value, title=''):
    x = np.linspace(1, len(value) + 1, len(value))
    plt.scatter(x, value)
    plt.title(title)
    #plt.show()
    plt.savefig("score.png")


def result(dataDict):
    '''人群打分'''
    def Layer_score():
        # 底层人口的百分比
        #data = pd.read_excel('./bostonh.xlsx',names=names)
        df = pd.read_excel('bostonh.xlsx')
        LSTAT_score_array = df.iloc[:, 12].values
        # print('LSTAT_score_array')
        # print(LSTAT_score_array)
        # 底层人口的百分比分数
        #print(type(LSTAT_score_array))
        LSTAT_score_array.resize(506,1)
        #print(LSTAT_score_array)
        LSTAT_score = getScore(LSTAT_score_array[:, 0], [dataDict['la
yer[LSTAT_score_min]'], 0.7],
                               [dataDict['layer[LSTAT_score_max]'],
 0.3])
        #print(LSTAT_score_array)
        # 各镇的师生比率
        PTRATTO_score_array = df.iloc[:, 10].values
        PTRATTO_score_array.resize(506,1)
        #print(PTRATTO_score_array)
        # 各镇的师生比率分数
        PTRATTO_score = getScore(PTRATTO_score_array[:, 0], [dataDict
['layer[PTRATTO_score_min]'], 0.8],
                                 [dataDict['layer[PTRATTO_score_max]
'], 0.2])

        # 城镇的人均犯罪率
        CRIM_score_array = df.iloc[:, 0].values
        CRIM_score_array.resize(506,1)
        #print(CRIM_score_array)
        # 城镇的人均犯罪率分数
```

```
        CRIM_score  = getScore(CRIM_score_array[:, 0], [dataDict['lay
er[CRIM_score_min]'],0.95],
                           [dataDict['layer[CRIM_score_max]'],0.05])


        # 综合考虑
        Layer_score = [dataDict['layer[LSTAT_score]'], dataDict['laye
r[PRTATTO_score]'],dataDict['layer[CRIM_score]']]
        layer_weight = get_weight(Layer_score)
        #print(type(layer_weight))
        score_values = []
        # layer_weight_list = []
        # for i in layer_weight:
        #     layer_weight_list.append(round(i))
        for i in range(0, len(PTRATTO_score_array)):
            a1 = LSTAT_score[i]
            a2 = PTRATTO_score[i]
            a3 = CRIM_score[i]
            # print(type([a1,a2,a3]))
            # print(type(layer_weight))
            score_value = np.convolve(layer_weight,[a1, a2, a3])
            # score_value = layer_weight*[a1, a2, a3]
            score_values.append(sum(score_value))

        show_score(score_values,'人群指数')
        return score_values

    '''环境打分'''
    def environmental_score():
        df = pd.read_csv('bostonh.csv', index_col=0)
        array = df.loc[:, ['INDUS','CHAS', 'NOX', 'DIS', 'RAD']].valu
es

        # 每个镇的非零售业务英亩的比例打分
        INDUS_score = getScore(array[:, 0], [dataDict['environment[IN
DUS_score_min]'], 0.5],
                                       [dataDict['environment[INDUS_scor
e_max]'], 0.5])

        # 查尔斯河虚拟变量打分
        CHAS_score = getScore(array[:, 1], [dataDict['environment[CHA
S_score_min]'], 0.9],
                                    [dataDict['environment[CHAS_score_m
ax]'], 0.1])
```

```
        # 一氧化氮的浓度
        NOX_score = getScore(array[:, 2], [dataDict['environment[NOX_
score_min]'], 0.5],
                                           [dataDict['environment[NOX_score_
max]'], 0.5])

        # 到五个波士顿就业中心的加权距离打分
        DIS_score = getScore(array[:, 3], [dataDict['environment[DIS_
score_min]'], 0.8],
                                         [dataDict['environment[DIS_score_m
ax]'], 0.2])

        # 径向公路通达性的指标打分
        RAD_score = getScore(array[:, 4], [dataDict['environment[RAD_
score_min]'], 0.8],
                                           [dataDict['environment[RAD_score_
max]'], 0.2])

        #综合五个因素
        environmental_score = [dataDict['environment[score_INDUS]'],

                               dataDict['environment[score_CHAS]'],
                               dataDict['environment[score_NOX]'],
                               dataDict['environment[score_DIS]'],
                               dataDict['environment[score_RAD]']]
        # 交通各个因素的权重为:
        environmental_weight = get_weight(environmental_score)

        environmental_scores = []
        for i in range(0, len(array)):
            a1 = INDUS_score[i]
            a2 = CHAS_score[i]
            a3 = NOX_score[i]
            a4 = DIS_score[i]
            a5 = RAD_score[i]
            environment_value = np.convolve(environmental_weight ,[a1,
 a2, a3, a4, a5])
            environmental_scores.append(sum(environment_value))
        show_score(environmental_scores,'环境指数')
        return environmental_scores

    '''经济打分'''
    def price_score():
```

```
        df = pd.read_csv('bostonh.csv')

        # 每$10,000 的全值财产税率
        TAX_score_array = df.iloc[:, 9].values
        #print(TAX_score_array)
        TAX_score_array.resize(506,1)
        TAX_score = getScore(TAX_score_array[:, 0], [dataDict['price
[TAX_score_min]'], 0.2],
                             [dataDict['price[TAX_score_max]'], 0.
8])

        #拥有住房价值的中位数
        MEDA_score_array = df.iloc[:, 13].values
        MEDA_score_array.resize(506,1)
        MEDA_score = getScore( MEDA_score_array[:, 0], [dataDict['pri
ce[MEDA_score_min]'], 0.2],
                              [dataDict['price[MEDA_score_max]'],
0.8])

        # 综合考虑
        price_score = [dataDict['price[TAX_score]'], dataDict['price
[MEDA_score]']]
        price_weight = get_weight(price_score)

        price_values = []
        for i in range(0, len(TAX_score_array)):
            a1 = TAX_score[i]
            a2 = MEDA_score[i]
            price_value = np.convolve(price_weight,[a1, a2])
            price_values.append(sum(price_value))
        show_score(price_values,'经济支出')
        return price_values

    '''其余因素打分'''
    def muti_score():
        df = pd.read_csv('bostonh.csv', index_col=0)

        # 非裔美国人的比例
        B_score_array = df.iloc[:, 11].values
        B_score_array.resize(506,1)
        B_score = getScore(B_score_array[:, 0], [dataDict['muti[B_sco
re_min]'], 0.5],
                             [dataDict['muti[B_score_max]'], 0.5])
```

```
        # 1940 年之前建造的自有住房的比例
        AGE_score_array = df.iloc[:, 6].values
        AGE_score_array.resize(506,1)
        AGE_score = getScore( AGE_score_array[:, 0], [dataDict['muti
[AGE_score_min]'], 0.4],
                                    [dataDict['muti[AGE_score_max]'], 0.
6])

        # 每个住宅的平均房间数
        RM_score_array = df.iloc[:, 5].values
        RM_score_array.resize(506,1)
        RM_score = getScore( AGE_score_array[:, 0], [dataDict['muti[R
M_score_min]'], 0.5],
                                    [dataDict['muti[RM_score_max]'], 0.
5])

        #大于 25,000 平方英尺的地块的住宅用地比例
        ZN_score_array = df.iloc[:, 1].values
        ZN_score_array.resize(506,1)
        ZN_score = getScore( ZN_score_array[:, 0], [dataDict['muti[ZN
_score_min]'], 0.9],
                                    [dataDict['muti[ZN_score_max]'], 0.
1])

        # 综合考虑
        muti_score = [dataDict['muti[B_score]'], dataDict['muti[AGE_s
core]'],
                      dataDict['muti[RM_score]'],dataDict['muti[ZN_sco
re]']]
        muti_weight = get_weight(muti_score)

        muti_values = []
        for i in range(0, len(B_score_array)):
            a1 = B_score[i]
            a2 = AGE_score[i]
            a3 = RM_score[i]
            a4 = ZN_score[i]
            muti_value = np.convolve(muti_weight,[a1, a2, a3,a4])
            muti_values.append(sum(muti_value))
        show_score(muti_values,'其余因素')
        return muti_values
```

```
    ps = Layer_score()
    ts = environmental_score()
    cs = price_score()
    ms = muti_score()
    ps=np.array(ps).reshape(506,)

    V = [dataDict['final[final_price]'],
         dataDict['final[final_traffic]'],
         dataDict['final[final_community]'],
         dataDict['final[final_muti]']]
    W = []
    W = get_weight(V)

    M = []
    data = []
    df = pd.read_csv('bostonh.csv', index_col=0)
    #names = df.loc[:, ['楼盘名称']].values[:, 0]
    prices = df.loc[:, ['MEDV']].values[:, 0]
    for i in range(0, len(ps)):
        a1 = ps[i]
        a2 = ts[i]
        a3 = cs[i]
        a4 = ms[i]
        q = W * [a1, a2, a3, a4]
        m = sum(q)
        M.append(m)
        #name = names[i]
        price = prices[i]
        data.append([ price, a1, a2, a3, a4, m])
    df = pd.DataFrame(data, columns=['房价中位数','人文指数', '自然环境
', '消费水平','楼盘综合水平', '宜居性'])
    df.to_excel('comprehensive evaluation.xlsx')
    return data


if __name__ == '__main__':
    '''
    you can mark your personal score in daraDict
    '''
    dataDict = {'final[final_price]': 8,
                'final[final_traffic]': 6,
                'final[final_community]': 5,
                'final[final_muti]': 3,
```

```
                'final[final_location]': 4,
                'layer[LSTAT_score_min]': 2,
                'layer[LSTAT_score_max]': 37,
                'layer[PTRATTO_score_min]': 12.5,
                'layer[PTRATTO_score_max]': 22,
                'layer[CRIM_score_min]':0.006,
                'layer[CRIM_score_max]':89,
                'environment[INDUS_score_min]': 0.5,
                'environment[INDUS_score_max]': 27.5,
                'environment[CHAS_score_min]': 0,
                'environment[CHAS_score_max]': 1,
                'environment[NOX_score_min]': 0.4,
                'environment[NOX_score_max]': 0.9,
                'environment[DIS_score_min]': 1,
                'environment[DIS_score_max]': 12,
                'environment[RAD_score_min]': 1,
                'environment[RAD_score_max]': 24,
                'price[TAX_score_min]': 187,
                'price[TAX_score_max]': 711,
                'price[MEDA_score_min]': 5,
                'price[MEDA_score_max]': 50,
                'muti[B_score_min]': 0.3,
                'muti[B_score_max]':396,
                'muti[AGE_score_min]':3,
                'muti[AGE_score_max]':100,
                'muti[RM_score_min]':3.6,
                'muti[RM_score_max]':8.8,
                'muti[ZN_score_min]':0,
                'muti[ZN_score_max]':100,
                }

dataDict['layer[LSTAT_score]']=150
dataDict['layer[PRTATTO_score]']=15
dataDict['layer[CRIM_score]']=45
dataDict['environment[score_INDUS]']=35
dataDict['environment[score_CHAS]'] = 2
dataDict['environment[score_NOX]']=30
dataDict['environment[score_DIS]']=22.5
dataDict['environment[score_RAD]']= 1
dataDict['price[TAX_score]'] = 25
dataDict['price[MEDA_score]'] = 150
dataDict['muti[B_score]'] = 12.5
dataDict['muti[AGE_score]'] = 17.5
dataDict['muti[RM_score]'] = 140
```

```
dataDict['muti[ZN_score]'] = 7.5

data = result(dataDict)
#print(data)
```