波士顿房价的预测分析

运行环境python3.11

IDE：jupyter notebook

1.数据集介绍

本文所用《波士顿房价数据集》数据来源于UCI的《Boston House Prices dataset》的经典数据集。

名称	波士顿房价数据集（Boston House Prices dataset）
特征简介	CRIM - 城镇人均犯罪率 ZN - 占地面积超过25,000平方英尺的住宅用地比例 INDUS - 每个城镇非零售业务的比例 CHAS - Charles River虚拟变量 NOX - 一氧化氮浓度（每千万份） RM - 每间住宅的平均房间数 AGE - 1940年以前建造的自住单位比例 DIS - 波士顿的五个就业中心加权距离 RAD - 径向高速公路的可达性指数 TAX - 每10,000美元的全额物业税率 PTRATIO - 城镇的学生与教师比例 B - $1000 * (Bk - 0.63)^2$其中Bk是城镇黑人的比例 LSTAT - 区域中被认为是低收入阶层的比率 MEDV - 自有住房的中位数报价, 单位1000美元
记录数	506条
分析目标	建立一个从已知特征预测波士顿未知房价的模型
分析思路和方法	采用线性回归和随机森林对房价进行预测，同时筛选出影响波士顿房价的主要因素

CRIM - 城镇人均犯罪率
ZN - 占地面积超过25,000平方英尺的住宅用地比例
INDUS - 每个城镇非零售业务的比例
CHAS - Charles River虚拟变量
NOX - 一氧化氮浓度（每千万份）
RM - 每间住宅的平均房间数
AGE - 1940年以前建造的自住单位比例
DIS - 波士顿的五个就业中心加权距离
RAD - 径向高速公路的可达性指数
TAX - 每10,000美元的全额物业税率

PTRATIO - 城镇的学生与教师比例
B - $1000 * (Bk - 0.63)^2$其中Bk是城镇黑人的比例
LSTAT - 区域中被认为是低收入阶层的比率
MEDV - 自有住房的中位数报价, 单位1000美元

2.数据预处理

sklearn库版本较高，故直接采用github下载的方式，下载后数据分成两部分，，每部分十一个指标需要整合

```
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
```

之后给数据加上标签如图

```
att =['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT']
boston = {'data':data,'target':target,'feature_names':np.array(att)}
```

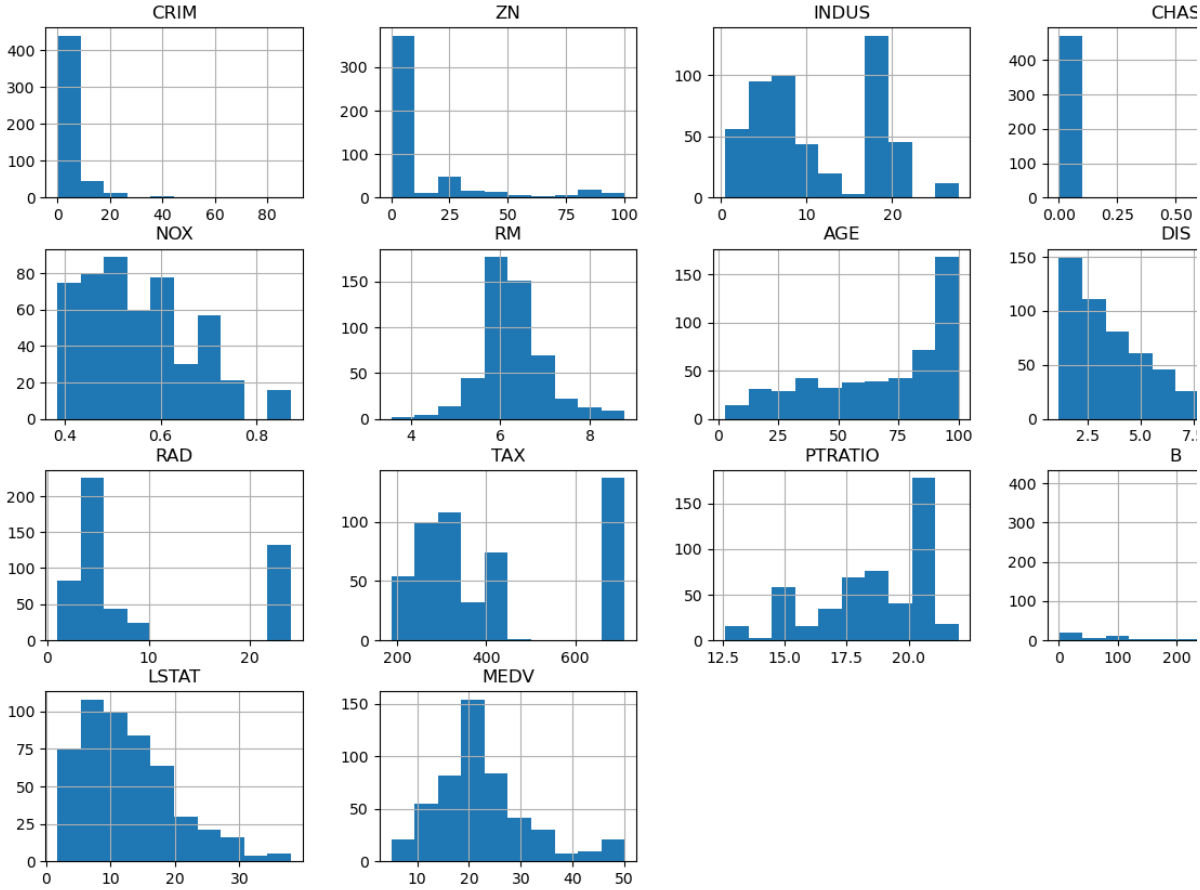
最终得到Boston_df为pandas的dataframe格式，如图

]:

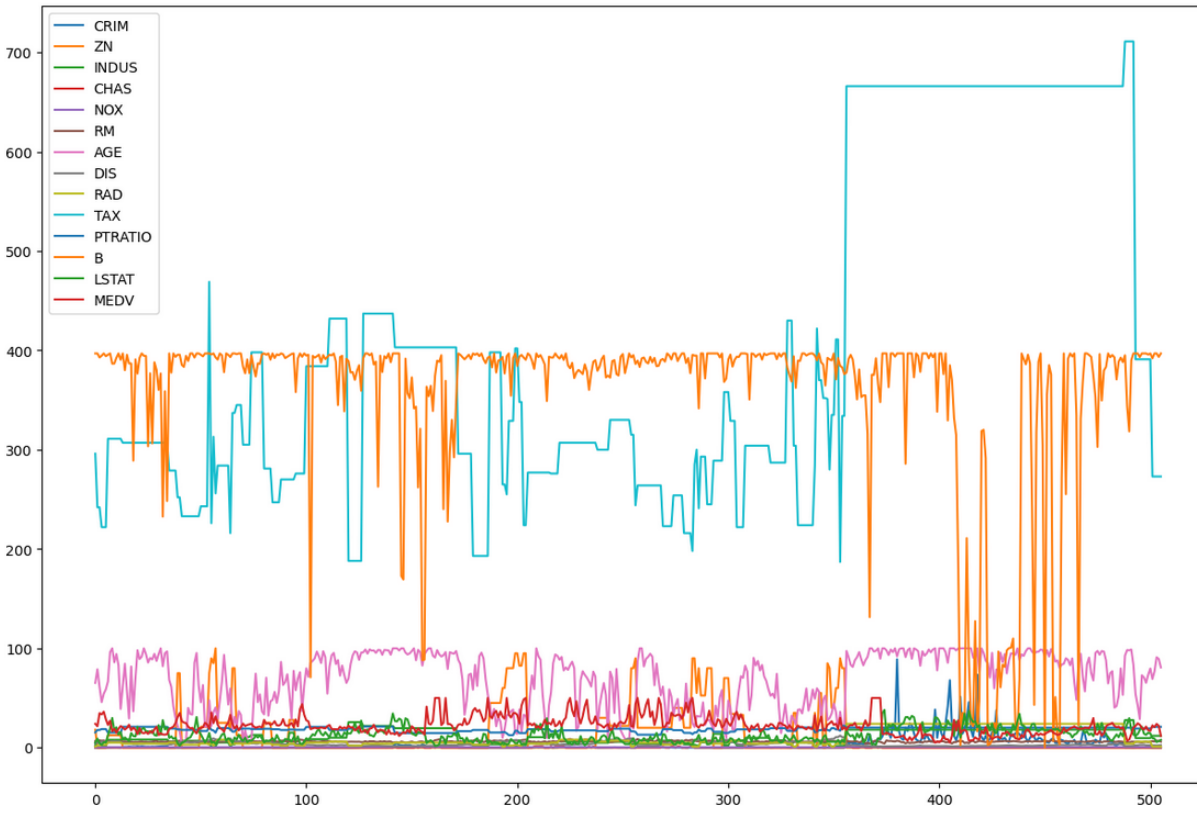
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	M
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.53
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.19
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.00
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.02
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.00
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.00

3.数据可视化

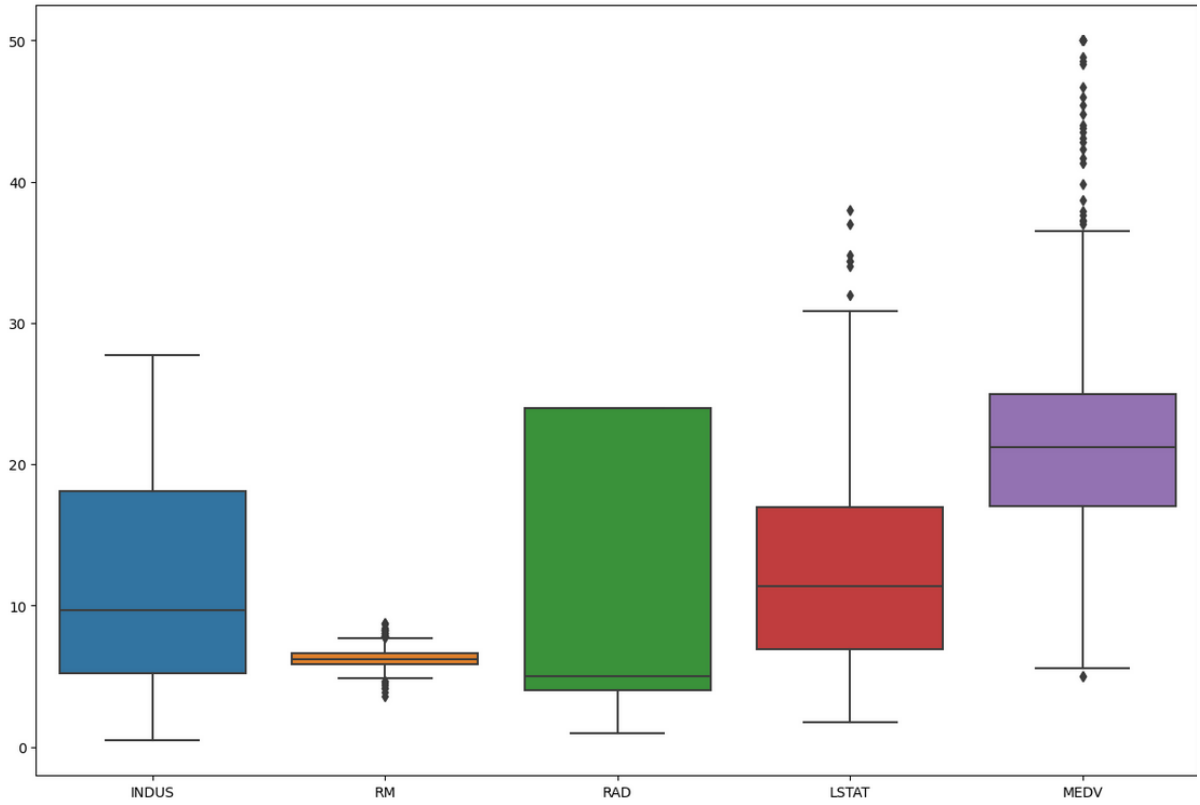
频率分布直方图


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
MEDV

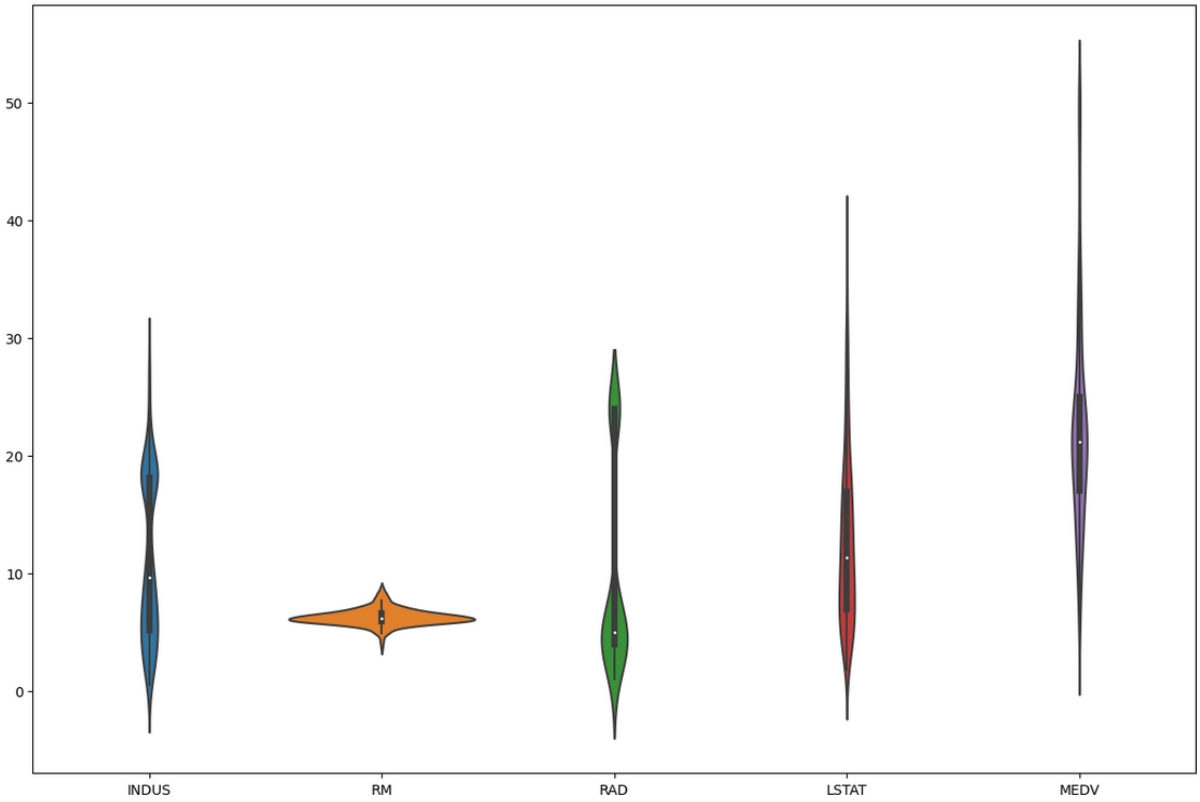
数据分布图


CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
MEDV

箱型图



小提琴图



4.划分测试集和验证集

使用sklearn库中的方法，按照2：8的比例划分测试集和数据集，并对X进行标准化

```
from sklearn.model_selection import train_test_split
# 数据二八分测试和训练

data_df = boston_df
```

```
X = data_df.drop(['MEDV'],axis=1)
y = data_df['MEDV']


X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=42,test_size =
0.2)

#标准化
from sklearn.preprocessing import StandardScaler
preprocess = StandardScaler()
X_train = preprocess.fit_transform(X_train)
X_test = preprocess.fit_transform(X_test)
```

5.测试几种回归方法并评价好坏

选取五种回归方法

```
#岭回归

from sklearn.linear_model import RidgeCV

reg = RidgeCV(alphas=(0.1, 1.0, 10.0), fit_intercept=True,
              scoring=None, cv=5, gcv_mode=None, store_cv_values=False)
reg.fit(X_train, y_train)
Ridge_s = reg.score(X_test, y_test)
```

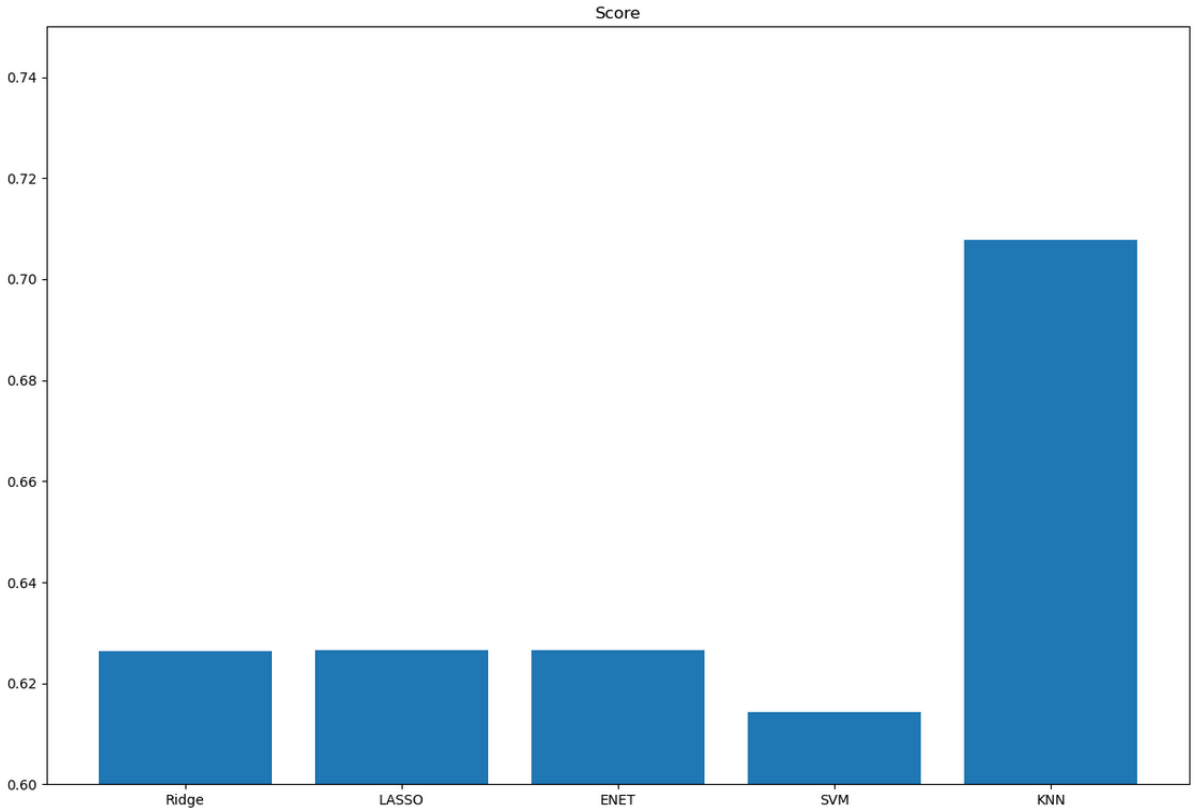
```
#Lasso
from sklearn.linear_model import LassoCV
reg = LassoCV(eps=0.001, n_alphas=100, alphas=None, fit_intercept=True,
precompute="auto", max_iter=1000, tol=0.0001,
              copy_X=True, cv=5, verbose=False, n_jobs=None,
              positive=False, random_state=None, selection="cyclic")
reg.fit(X_train, y_train)
Lasso_s = reg.score(X_test, y_test)
```

```
#弹性网络

from sklearn.linear_model import ElasticNetCV
reg = ElasticNetCV(l1_ratio=0.5, eps=0.001, n_alphas=100, alphas=None,
                   fit_intercept=True, precompute="auto",
                   max_iter=1000, tol=0.0001, cv=5, copy_X=True, verbose=0,
                   n_jobs=None, positive=False, random_state=None,
selection="cyclic")
reg.fit(X_train, y_train)
ENet_s = reg.score(X_test, y_test)
```

```
#SVM

from sklearn.svm import SVR
reg = SVR(kernel="rbf", degree=3, gamma="auto", coef0=0.0,
          tol=0.001, C=1.0, epsilon=0.1, shrinking=True,
          cache_size=200, verbose=False, max_iter=-1)
reg.fit(X_train, y_train)
SVM_s = reg.score(X_test, y_test)
```

```
#KNN
from sklearn.neighbors import KNeighborsRegressor
reg = KNeighborsRegressor(n_neighbors=5, weights="uniform", algorithm="auto",
                          leaf_size=30, p=2, metric="minkowski",
metric_params=None)
reg.fit(X_train, y_train)
KNN_s = reg.score(X_test, y_test)
```

对评分画柱形图



可以发现在Boston房价数据集上，使用K最近邻算法进行回归得到的效果最好，而SVM的效果较差

但五种方法评分都较高，因此都可以用于该数据集的预测