Minor Project in Machine learning

Topic-Boston Housing Price Prediction

Done by – Paavana M Rao

Introduction

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA.

The Boston Dataset-

https://docs.google.com/spreadsheets/d/1UAUiPOs8dQXZ0_r4_SEYEq1O4ue_KGbr/edit?usp=drivesdk&ouid=105619591429527999775&rtpof=true&sd=true

The features of the given dataset are as follows

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq. ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centers
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO - pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's

Task

The task is to predict the house prices based on the given features i.e 'medv'

Approach

My approach is first doing the preprocessing and doing the exploratory data analysis (EDA) and then applying the Linear Regression algorithm and fitting the data into the model

Preprocessing

First the dataset will be loaded and then we will print the first 5 rows of the and last 5 rows of the dataset by using head() and tail() function

```
In [26]: df.head()
Out[26]:
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
In [8]: df.tail()
Out[8]:
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

As we can observe that we have 506 rows and 15 columns but there s an one unnecessary column ie 'unnamed' as it displays the index values which already set in the dataframe,hence we will drop by using drop() function available in pandas library.

```
In [6]:  #Displays the dataframe without the unnamed column
         boston
```

Out[6]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 14 columns

Now we will get the brief information about the dataset by using info() function from

```
In [11]:  # provides the all the information of the data frame
          boston.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
```

We can observe here that the column names and there datatypes with non-null count, there are 11 columns with float data type and 3 integer type

Next up we have description of the dataset which holds standard deviations, mean, minimum value and other statistical information of each column .

```
In [12]:  #Describes the statistical info
          boston.describe()
```

Out[12]:

	crim	zn	indus	chas	nox	rm	age
count	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	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

And we can also get to know the count of NaN values of a given dataset by using isna().sum() function available

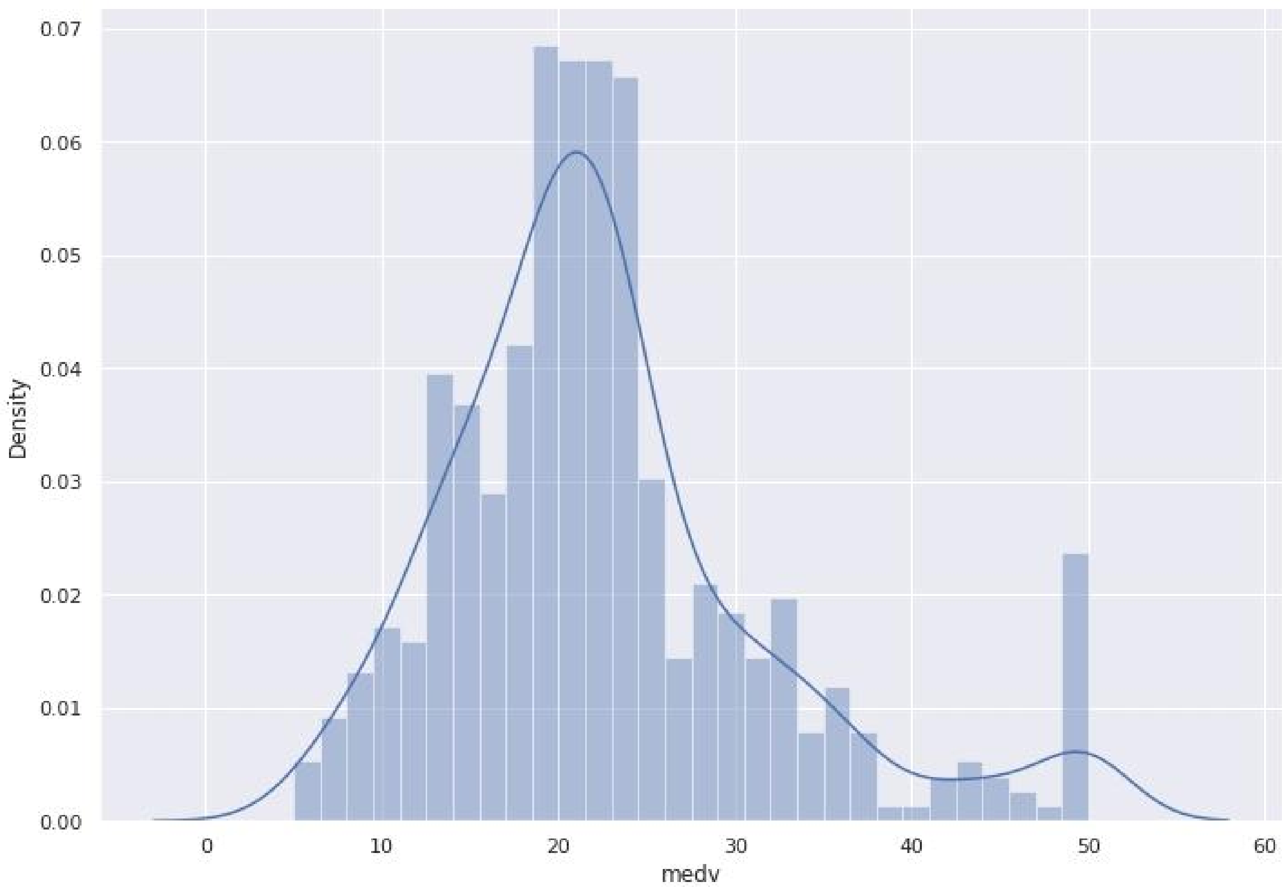
```
In [13]:  #Gives the sum count of Nan ( Not an integer)
          boston.isna().sum()

Out[13]:  crim       0
          zn         0
          indus      0
          chas       0
          nox        0
          rm         0
          age        0
          dis        0
          rad        0
          tax        0
          ptratio    0
          black      0
          lstat      0
          medv       0
          dtype: int64
```

Exploratory Data Analysis

Exploratory Data Analysis is a very important step before training the model. In this section, we will use some visualizations to understand the relationship of the target variable with other features.

Let's first plot the distribution of the target variable `'medv'` We will use the `distlpot` function from the `seaborn` library



We see that the values of MEDV are distributed normally with few outliers.

Next, we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be formed by using the corr function from the pandas data frame library. We will use the heatmap function from the seaborn library to plot the correlation matrix.

Observations

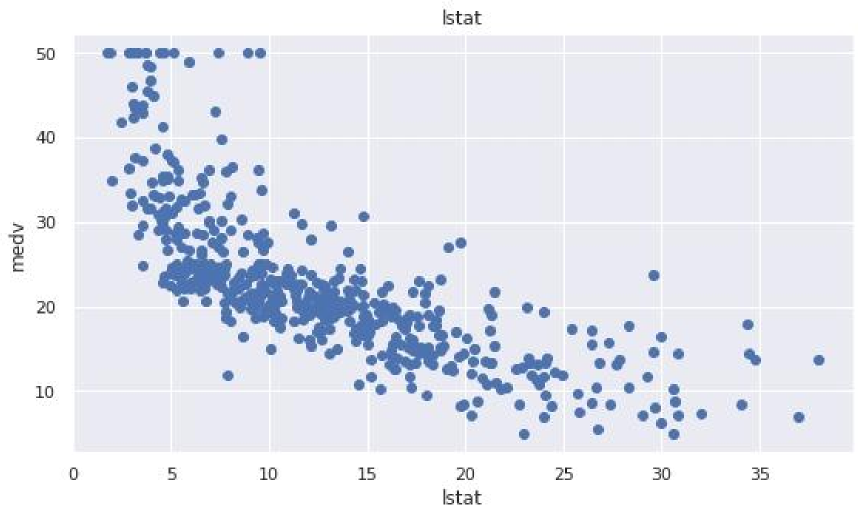
1. To fit a linear regression model, we select those features which have a high correlation with our target variable medv. By looking at the correlation matrix we can see that

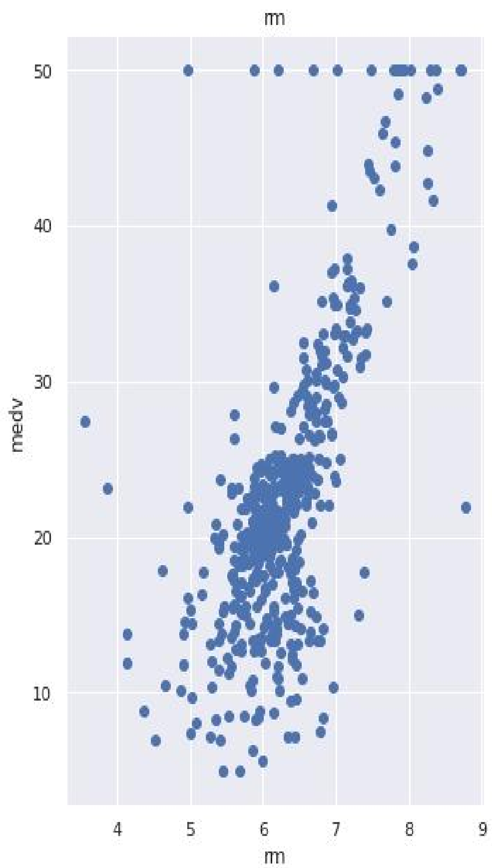
rm has a strong positive correlation with medv(0.7) where as LSTAT has a high negative correlation with medv(-0.74).

2. An important point in selecting features for a linear regression model is to check for multi-co-linearity. The features rad, tax have a correlation of 0.91. These feature pairs are strongly correlated to each other.

Visualization

Based on the above observations we will rm and lstat as our features. Using a scatter plot let's see how these features vary with medv





Observations

- The prices increase as the value of RM increases linearly. There are few outliers and the data seems to be capped at 50.
- The prices tend to decrease with an increase in LSTAT. Though it doesn’t look to be following exactly a linear line.

Preparing the dataset

Now We will concatenate the lstat and rm columns using np.c_ provided by the numpy library.

Splitting the data

Next, we split the data into training and testing sets. We train the model with 80% of the samples and test with the remaining 20%.To split the data we use train_test_split function provided by scikit-learn library. We will print the sizes of our training

```
In [15]: from sklearn.model_selection import tr
         X_train,X_test,Y_train,Y_test=train_te
         #The shape of the 4 splited sets Will
         print(X_train.shape)
         print(X_test.shape)
         print(Y_train.shape)
         print(Y_test.shape)

(404, 2)
(102, 2)
(404,)
(102,)
```

The shape of the 4 splited sets Will be returned

Fitting the model-

```
In [21]: from sklearn.linear_model import Linea
         lm=LinearRegression()
         lm.fit(X_train,Y_train)

Out[21]: LinearRegression()
```

Evaluation

Now we will evaluate our model by importing metrics library from sklearn and doing the R2 score from the same library.

According to the r2_score we can say that the model predicts moderately

```
In [18]: from sklearn import metrics
         print('Mean absolute error:',metrics.m
         print('Mean squared error:',metrics.me
         print('RMSE:',np.sqrt(metrics.mean_squ
```

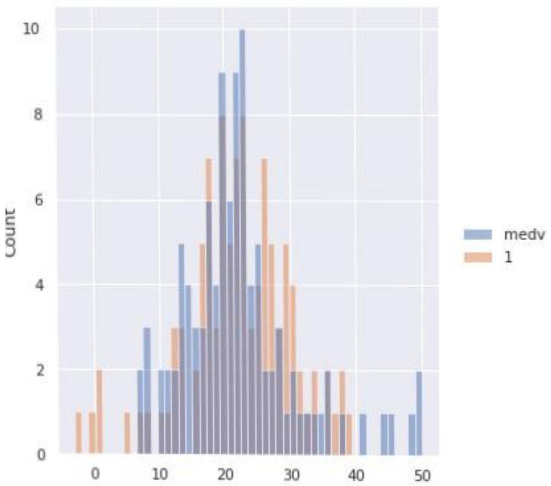
```
Mean absolute error: 3.79131021334310
56
Mean squared error: 26.39288682266607
8
RMSE: 5.13740078470291
```

Considering the RMSE: we can conclude that this model average error is RMSE at medv

```
In [25]: from sklearn.metrics import r2_score
         print('r2_score:',r2_score(Y_test, pre
```

```
r2_score: 0.6628996975186954
```

According to the r2_score we can say that the model predicts moderately



Resources

https://stackoverflow.com/
https://www.kaggle.com/code/henriqueyamahata/boston-housing-with-linear-regression