Predict Boston housing prices using Linear Regression analysis in Python

1. **Summary:**

- Found Boston housing prices using the features with the help of Linear Regression.
- For this analysis we have taken Boston housing Data. We fill find prices of houses in Boston using the feature given to us in the data.
- The Boston housing market is highly competitive, and you want to be the best real estate agent in the area. To compete with your peers, you decide to leverage a few basic machine learning concepts to assist you and a client with finding the best-selling price for their home.
- Luckily, we have come across the Boston Housing dataset which contains aggregated data on various features for houses in Greater Boston communities, including the median value of homes for each of those areas.
- The task is to build an optimal model based on a statistical analysis with the tools available. This model will then be used to estimate the best-selling price for your clients' homes.

2. **Data:**

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 oxide 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 [people of African American descent] by town

LSTAT: Percentage of lower status of the population

MEDV: Median value of owner-occupied homes in $1000s

Below is the screenshot of the Data used in the analysis

0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	1
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	1
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	1
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	1
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	1
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	1
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	1
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	1
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	1
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	1
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	1
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	1
0.62739	0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21	395.62	8.47	19.9	1
1.05393	0	8.14	0	0.538	5.935	29.3	4.4986	4	307	21	386.85	6.58	23.1	1
0.7842	0	8.14	0	0.538	5.99	81.7	4.2579	4	307	21	386.75	14.67	17.5	1
0.80271	0	8.14	0	0.538	5.456	36.6	3.7965	4	307	21	288.99	11.69	20.2	1
0.7258	0	8.14	0	0.538	5.727	69.5	3.7965	4	307	21	390.95	11.28	18.2	1
1.25179	0	8.14	0	0.538	5.57	98.1	3.7979	4	307	21	376.57	21.02	13.6	1
0.85204	0	8.14	0	0.538	5.965	89.2	4.0123	4	307	21	392.53	13.83	19.6	1
1.23247	0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21	396.9	18.72	15.2	1
0.98843	0	8.14	0	0.538	5.813	100	4.0952	4	307	21	394.54	19.88	14.5	1
0.75026	0	8.14	0	0.538	5.924	94.1	4.3996	4	307	21	394.33	16.3	15.6	1
0.84054	0	8.14	0	0.538	5.599	85.7	4.4546	4	307	21	303.42	16.51	13.9	1
0.67191	0	8.14	0	0.538	5.813	90.3	4.682	4	307	21	376.88	14.81	16.6	1
0.95577	0	8.14	0	0.538	6.047	88.8	4.4534	4	307	21	306.38	17.28	14.8	1

The prices of the house indicated by the variable MEDV is our target variable and the remaining are the feature variables based on which we will predict the value of a house.

Data Preprocessing:

1. Missing values in important columns;

We count the number of missing values for each feature using isnull()

However, there are no missing values in this dataset

- Let's first plot the distribution of the target variable MEDV. We will use the distplot function from the seaborn library.

```
sns.set(rc={'figure.figsize':(11.7,8.27)})

sns.distplot(boston['MEDV'], bins=30)

plt.show()
```

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 dataframe library. We will use the **heatmap function** from the seaborn library to plot the correlation matrix.

```
correlation_matrix = boston.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation

Based on the observations we will **RM and LSTAT** as our features. Using a scatter plot let's see how these features vary with MEDV.

```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']

target = boston['MEDV']

for i, col in enumerate(features):

    plt.subplot(1, len(features) , i+1)

    x = boston[col]

    y = target

    plt.scatter(x, y, marker='o')

    plt.title(col)

    plt.xlabel(col)

    plt.ylabel('MEDV')
```

We found that:

- 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

3. Linear Regresssion on Boston Housing Data

- Prepare the data for training the model

We concatenate the LSTAT and RM columns using np.c_ provided by the numpy library.

```
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])

Y = boston['MEDV']
```

- Splitting the data into training and testing sets

  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 finally print the sizes of our training and test set to verify if the splitting has occurred properly.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
(404, 2)
(102, 2)
(404,)
(102,)
```

- Training and testing the model
  We use scikit-learn's LinearRegression to train our model on both the training and test sets.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)
```

4. Model Evaluation:

We will evaluate our model using RMSE and R2-score.

 # model evaluation for training set

```
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)


print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

 # model evaluation for testing set

```
y_test_predict = lin_model.predict(X_test)

rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

r2 = r2_score(Y_test, y_test_predict)


print("The model performance for testing set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))

print('R2 score is {}'.format(r2))
```

The model performance for training set

RMSE is 5.6371293350711955

R2 score is 0.6300745149331701

The model performance for testing set

RMSE is 5.137400784702911

R2 score is 0.6628996975186952