

Boston_house_Linear_Regression

```
[1]: import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
[2]: import warnings
warnings.filterwarnings("ignore")
```

```
[3]: from sklearn.datasets import load_boston
boston = load_boston()
```

```
[4]: data = pd.DataFrame(boston.data)
```

```
[5]: data.head()
```

```
[5]:
```

	0	1	2	3	4	5	6	7	8	9	10	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	

	11	12
0	396.90	4.98
1	396.90	9.14
2	392.83	4.03
3	394.63	2.94
4	396.90	5.33

```
[6]: #Adding the feature names to the dataframe
data.columns = boston.feature_names
data.head()
```

```
[6]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	

2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0

	PTRATIO	B	LSTAT
0	15.3	396.90	4.98
1	17.8	396.90	9.14
2	17.8	392.83	4.03
3	18.7	394.63	2.94
4	18.7	396.90	5.33

```
[7]: #Adding target variable to dataframe
data['PRICE'] = boston.target
```

```
[8]: #Check the shape of dataframe
data.shape
```

```
[8]: (506, 14)
```

```
[9]: data.columns
```

```
[9]: Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
          'PTRATIO', 'B', 'LSTAT', 'PRICE'],
          dtype='object')
```

```
[10]: data.dtypes
```

```
[10]: CRIM      float64
      ZN       float64
      INDUS    float64
      CHAS     float64
      NOX      float64
      RM       float64
      AGE      float64
      DIS      float64
      RAD      float64
      TAX      float64
      PTRATIO  float64
      B        float64
      LSTAT    float64
      PRICE    float64
      dtype: object
```

```
[11]: # Identifying the unique number of values in the dataset
data.nunique()
```

```
[11]: CRIM      504
      ZN        26
      INDUS    76
      CHAS      2
      NOX      81
      RM      446
      AGE     356
      DIS     412
      RAD       9
      TAX      66
      PTRATIO  46
      B       357
      LSTAT   455
      PRICE   229
      dtype: int64
```

```
[12]: # Check for missing values
      data.isnull().sum()
```

```
[12]: CRIM      0
      ZN        0
      INDUS    0
      CHAS      0
      NOX      0
      RM      0
      AGE     0
      DIS     0
      RAD      0
      TAX      0
      PTRATIO  0
      B        0
      LSTAT    0
      PRICE    0
      dtype: int64
```

```
[13]: # See rows with missing values
      data[data.isnull().any(axis=1)]
```

```
[13]: Empty DataFrame
      Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT, PRICE]
      Index: []
```

```
[14]: # Viewing the data statistics
      data.describe()
```

```
[14]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM \
count	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
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

	AGE	DIS	RAD	TAX	PTRATIO	B \
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032
std	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864
min	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000
75%	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000
max	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000

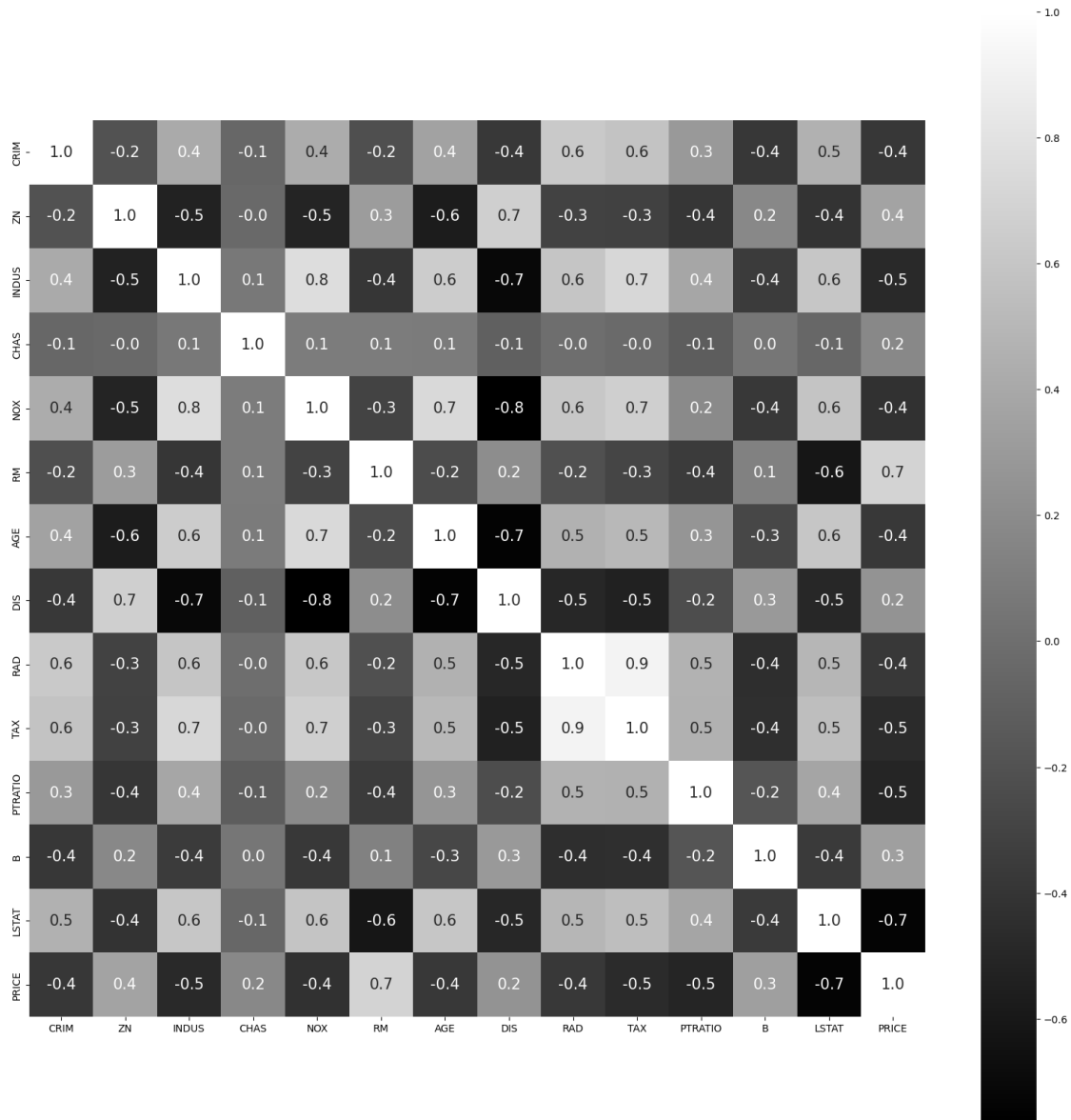
	LSTAT	PRICE
count	506.000000	506.000000
mean	12.653063	22.532806
std	7.141062	9.197104
min	1.730000	5.000000
25%	6.950000	17.025000
50%	11.360000	21.200000
75%	16.955000	25.000000
max	37.970000	50.000000

```
[15]: # Finding out the correlation between the features
corr = data.corr()
corr.shape
```

```
[15]: (14, 14)
```

```
[16]: # Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square=True, fmt='.1f', annot=True,
            annot_kws={'size':15}, cmap='gray')
```

```
[16]: <AxesSubplot:>
```



```
[17]: # Splitting target variable and independent variables
```

```
X = data.drop(['PRICE'], axis = 1)
y = data['PRICE']
```

```
[18]: # Splitting to training and testing data
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,
↳ random_state = 4)
```

```
[19]: # Import library for Linear Regression
from sklearn.linear_model import LinearRegression
```

```
[20]: # Create a Linear regressor
lm = LinearRegression()

# Train the model using the training sets
lm.fit(X_train, y_train)
```

```
[20]: LinearRegression()
```

```
[21]: # Value of y intercept
lm.intercept_
```

```
[21]: 36.357041376595205
```

```
[22]: #Converting the coefficient values to a dataframe
coefficients = pd.DataFrame([X_train.columns,lm.coef_]).T
coefficients = coefficients.rename(columns={0: 'Attribute', 1: 'Coefficients'})
coefficients
```

```
[22]:
```

	Attribute	Coefficients
0	CRIM	-0.12257
1	ZN	0.055678
2	INDUS	-0.008834
3	CHAS	4.693448
4	NOX	-14.435783
5	RM	3.28008
6	AGE	-0.003448
7	DIS	-1.552144
8	RAD	0.32625
9	TAX	-0.014067
10	PTRATIO	-0.803275
11	B	0.009354
12	LSTAT	-0.523478

```
[23]: # Model prediction on train data
y_pred = lm.predict(X_train)
```

```
[24]: # Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train,
↪y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

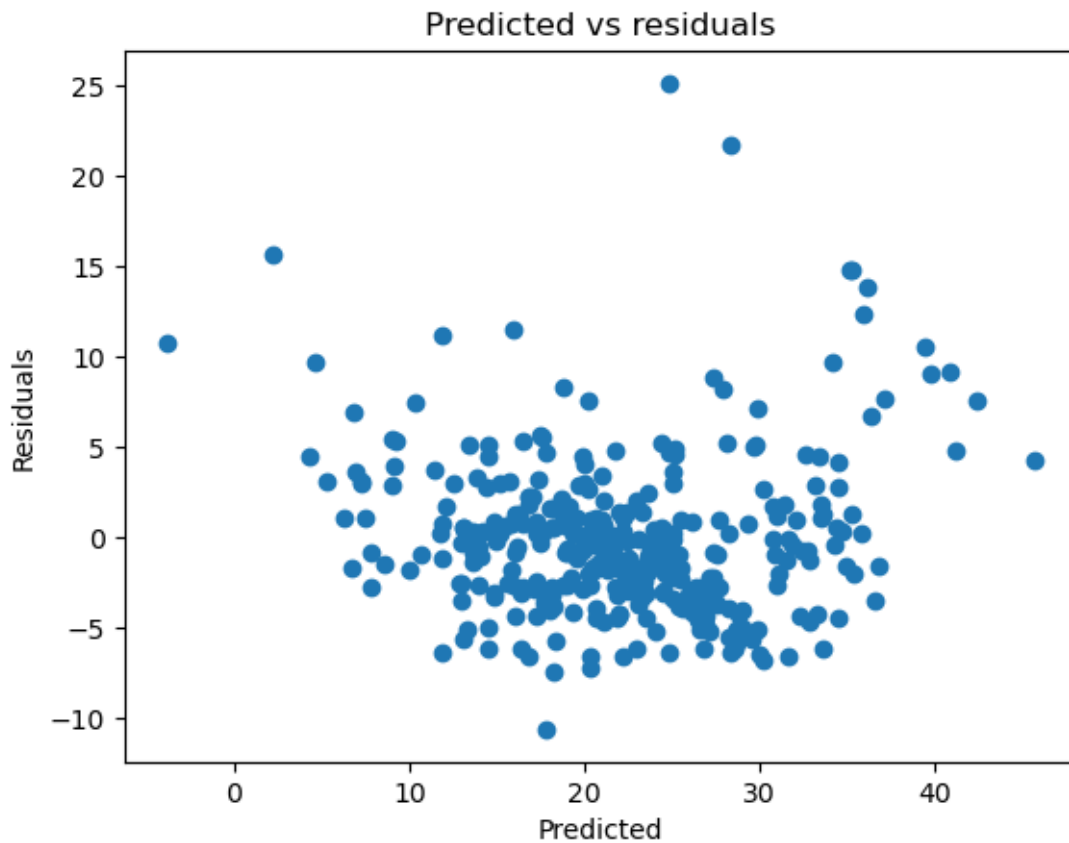
```
R^2: 0.7465991966746854
```

Adjusted R²: 0.736910342429894
MAE: 3.08986109497113
MSE: 19.07368870346903
RMSE: 4.367343437774162

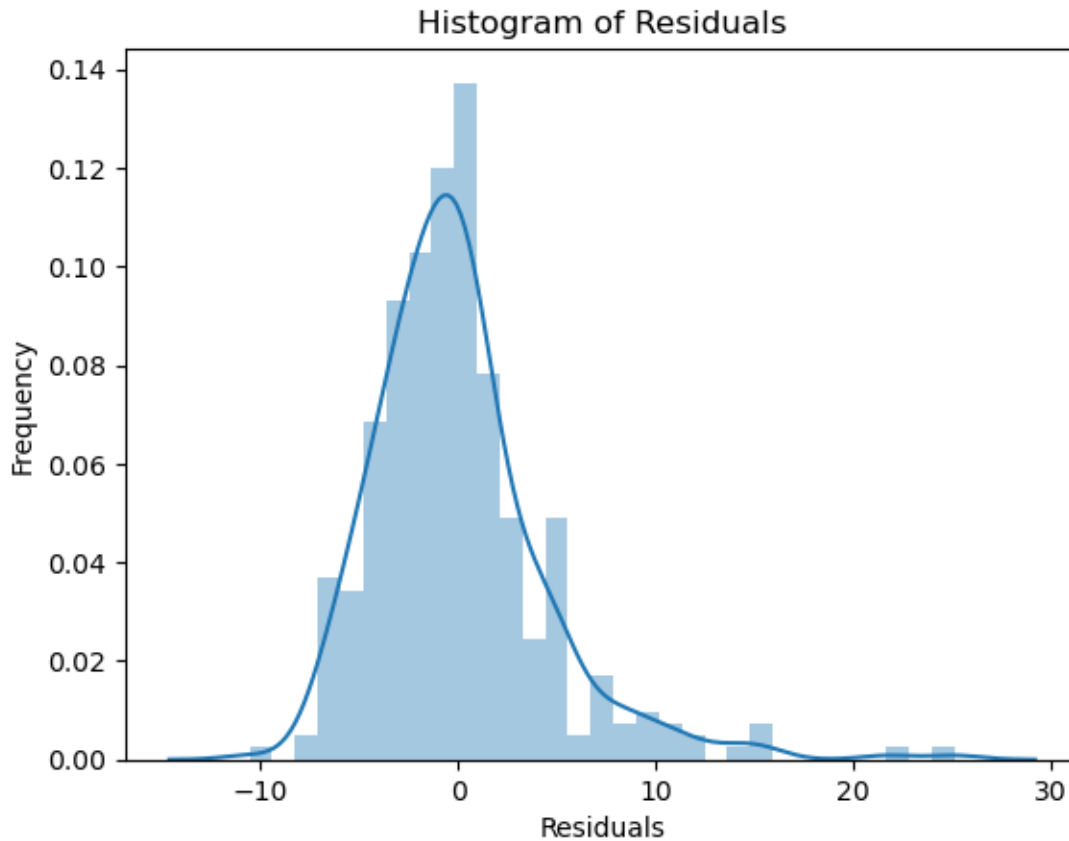
```
[25]: # Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



```
[26]: # Checking residuals
plt.scatter(y_pred, y_train - y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



```
[27]: # Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```

```
[28]: # Predicting Test data with the model
y_test_pred = lm.predict(X_test)
```

```
[29]: # Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:', 1 - (1-metrics.r2_score(y_test, y_test_pred)) * (len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('MAE:', metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7121818377409195
Adjusted R^2: 0.6850685326005713
MAE: 3.8590055923707407
MSE: 30.053993307124127
RMSE: 5.482152251362974
```