linear_regrerssion_corrected_100%

November 24, 2024

[80]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[82]:
```python
data = pd.read_csv("boston.csv")
```

[84]:
```python
data.head()
```

[84]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	\
0	0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	
1	1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	
2	2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	
3	3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	
4	4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	

	TAX	PTRATIO	B	LSTAT	Price
0	296.0	15.3	396.90	4.98	24.0
1	242.0	17.8	396.90	9.14	21.6
2	242.0	17.8	392.83	4.03	34.7
3	222.0	18.7	394.63	2.94	33.4
4	222.0	18.7	396.90	5.33	36.2

[86]:
```python
data.describe()
```

[86]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	252.500000	3.613524	11.363636	11.136779	0.069170	0.554695	
std	146.213884	8.601545	23.322453	6.860353	0.253994	0.115878	
min	0.000000	0.006320	0.000000	0.460000	0.000000	0.385000	
25%	126.250000	0.082045	0.000000	5.190000	0.000000	0.449000	
50%	252.500000	0.256510	0.000000	9.690000	0.000000	0.538000	
75%	378.750000	3.677083	12.500000	18.100000	0.000000	0.624000	
max	505.000000	88.976200	100.000000	27.740000	1.000000	0.871000	

	RM	AGE	DIS	RAD	TAX	PTRATIO	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	
std	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	

```
min       3.561000    2.900000    1.129600    1.000000  187.000000   12.600000
25%       5.885500   45.025000    2.100175    4.000000  279.000000   17.400000
50%       6.208500   77.500000    3.207450    5.000000  330.000000   19.050000
75%       6.623500   94.075000    5.188425   24.000000  666.000000   20.200000
max       8.780000  100.000000   12.126500   24.000000  711.000000   22.000000

                B       LSTAT       Price
count  506.000000  506.000000  506.000000
mean   356.674032   12.653063   22.532806
std     91.294864    7.141062    9.197104
min      0.320000    1.730000    5.000000
25%    375.377500    6.950000   17.025000
50%    391.440000   11.360000   21.200000
75%    396.225000   16.955000   25.000000
max    396.900000   37.970000   50.000000
```

[88]:
```
data.isnull().sum()
```

[88]:
```
Unnamed: 0    0
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
```

[90]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  506 non-null    int64
 1   CRIM        506 non-null    float64
 2   ZN          506 non-null    float64
 3   INDUS       506 non-null    float64
 4   CHAS        506 non-null    float64
 5   NOX         506 non-null    float64
```

```
 6   RM          506 non-null    float64
 7   AGE         506 non-null    float64
 8   DIS         506 non-null    float64
 9   RAD         506 non-null    float64
 10  TAX         506 non-null    float64
 11  PTRATIO     506 non-null    float64
 12  B           506 non-null    float64
 13  LSTAT       506 non-null    float64
 14  Price       506 non-null    float64
dtypes: float64(14), int64(1)
memory usage: 59.4 KB
```

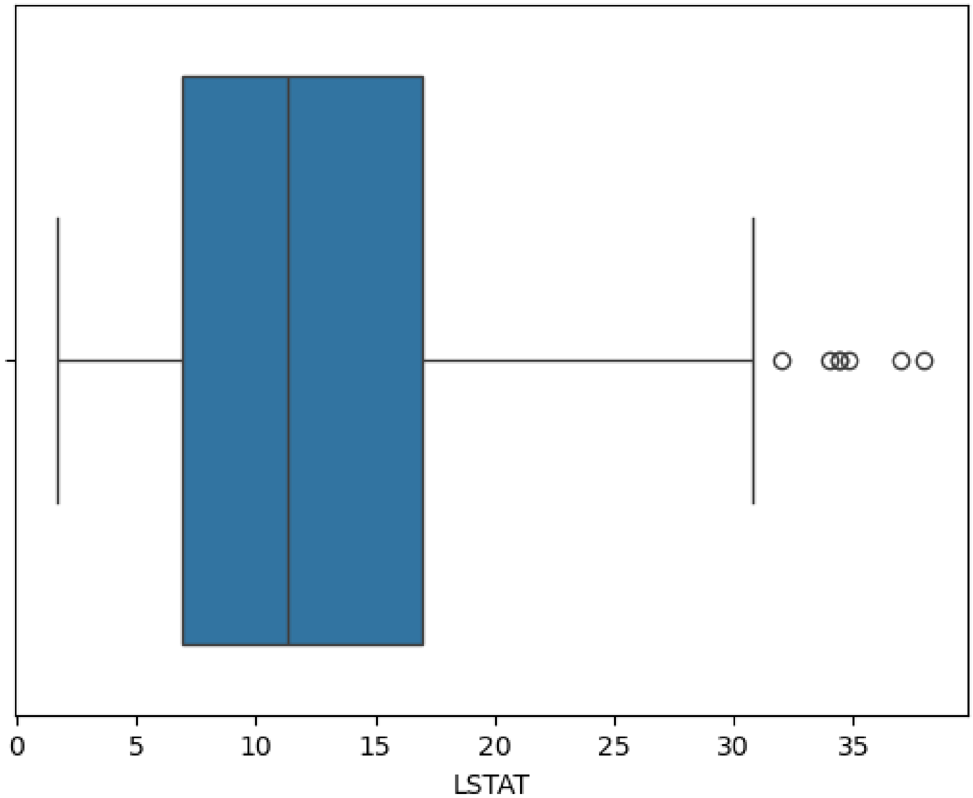
[92]:
```
data.drop('Unnamed: 0', axis = 1,inplace=True)
```

[94]:
```
data.head()
```

[94]:
```
      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  Price
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

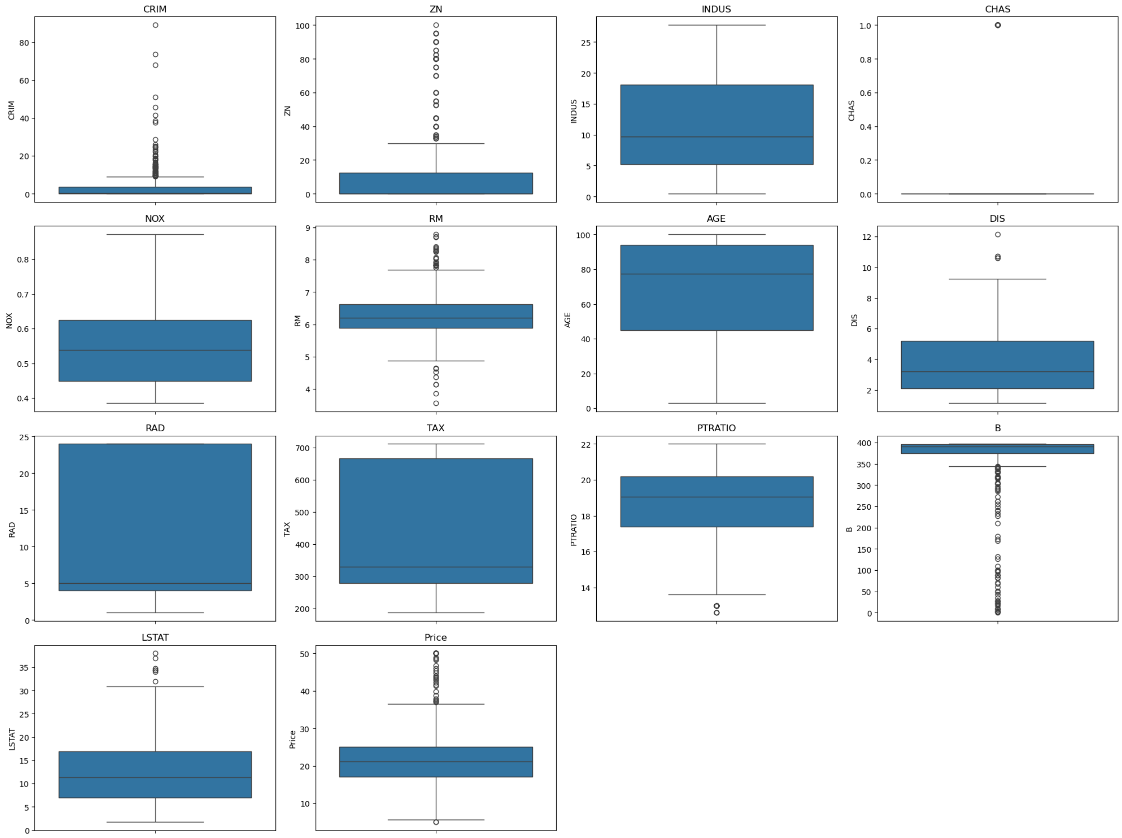
[99]:
```
sns.boxplot(x='LSTAT', data=data)
```

[99]:
```
<Axes: xlabel='LSTAT'>
```



[105]:

```
plt.figure(figsize=(20, 15))  # Adjust figure size as needed
for i, column in enumerate(data.columns, 1):
    plt.subplot(4, 4, i)  # Adjust grid dimensions based on the number of␣
 ↪columns
    sns.boxplot(y=data[column])
    plt.title(column)

plt.tight_layout()
plt.show()
```

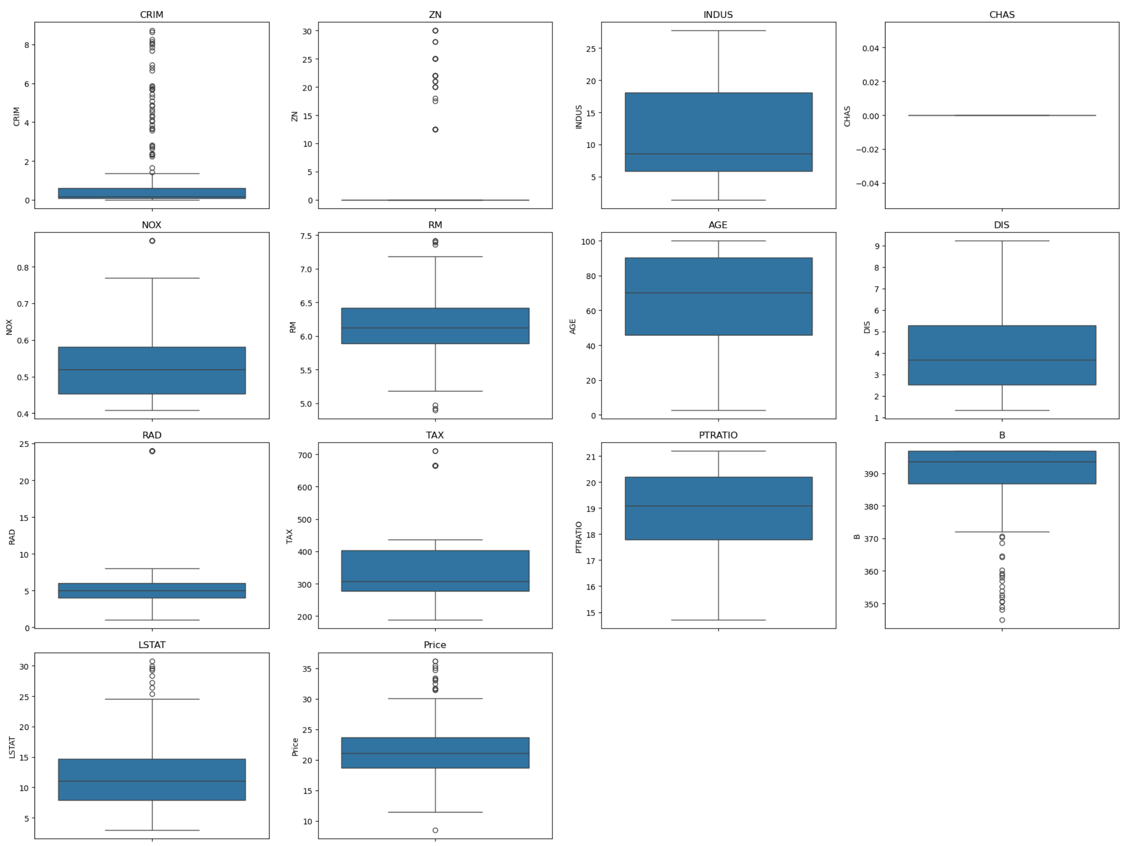


[107]:
```
import pandas as pd

# Function to remove outliers using the IQR method
def remove_outliers(df):
    numeric_columns = df.select_dtypes(include='number').columns
    cleaned_df = df.copy()
    for column in numeric_columns:
        Q1 = df[column].quantile(0.25)  # First quartile (25th percentile)
        Q3 = df[column].quantile(0.75)  # Third quartile (75th percentile)
        IQR = Q3 - Q1  # Interquartile range
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # Filter the dataset to include only data within bounds
        cleaned_df = cleaned_df[(cleaned_df[column] >= lower_bound) &␣
↪(cleaned_df[column] <= upper_bound)]
    return cleaned_df

# Apply the function to the dataset
data = remove_outliers(data)
```

```python
# Check the resulting dataset
print("Original dataset shape:", data.shape)
print("Cleaned dataset shape:", data.shape)
```

```
Original dataset shape: (268, 14)
Cleaned dataset shape: (268, 14)
```

[109]:
```python
plt.figure(figsize=(20, 15))  # Adjust figure size as needed
for i, column in enumerate(data.columns, 1):
    plt.subplot(4, 4, i)  # Adjust grid dimensions based on the number of columns
    sns.boxplot(y=data[column])
    plt.title(column)

plt.tight_layout()
plt.show()
```

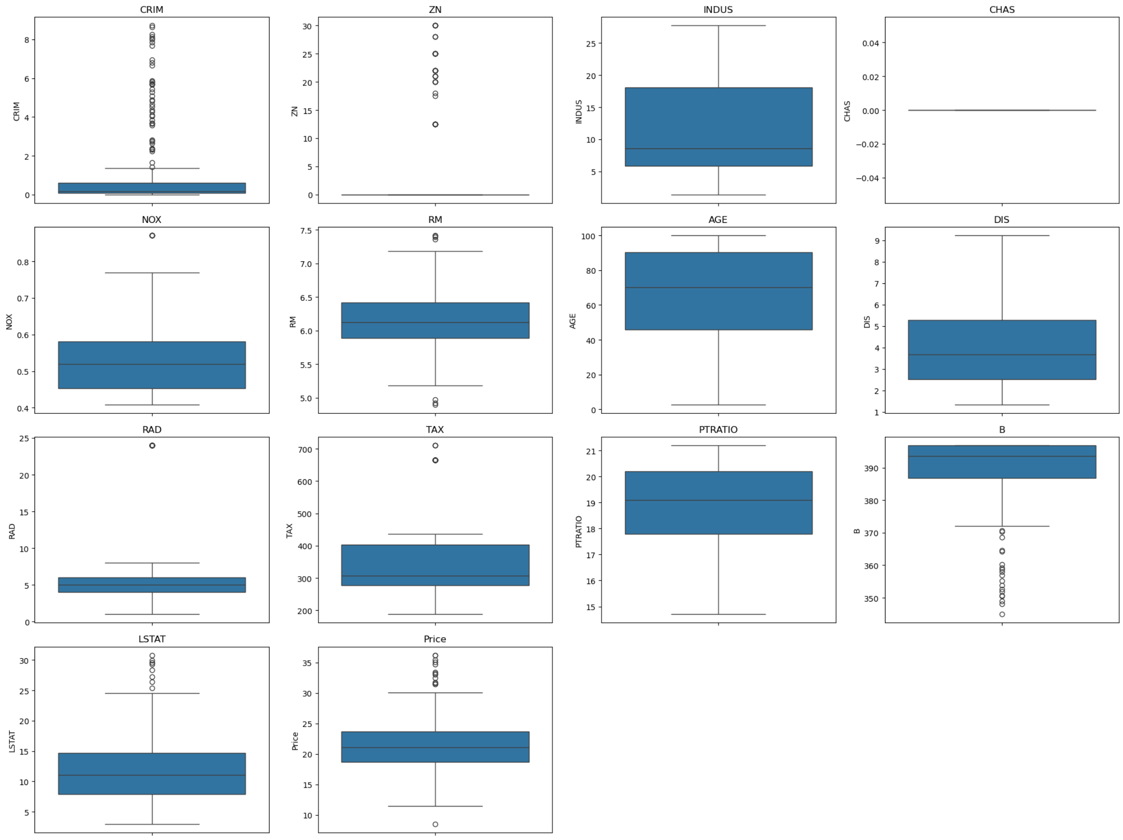


[115]:
```python
# Calculate the IQR for the 'AGE' column
Q1 = data['AGE'].quantile(0.25)  # 25th percentile
Q3 = data['AGE'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1
```

```
# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with 0 in the 'AGE' column
data['AGE'] = data['AGE'].apply(lambda x: 0 if x < lower_bound or x >␣
↪upper_bound else x)

# Check the updated column
data['AGE'].head()
```

[115]:
```
0    65.2
1    78.9
2    61.1
3    45.8
4    54.2
Name: AGE, dtype: float64
```

[117]:
```
plt.figure(figsize=(20, 15))  # Adjust figure size as needed
for i, column in enumerate(data.columns, 1):
    plt.subplot(4, 4, i)  # Adjust grid dimensions based on the number of␣
↪columns
    sns.boxplot(y=data[column])
    plt.title(column)

plt.tight_layout()
plt.show()
```



[123]:
```
X = data[['AGE', 'RM', 'LSTAT']]
y = data['Price']
```

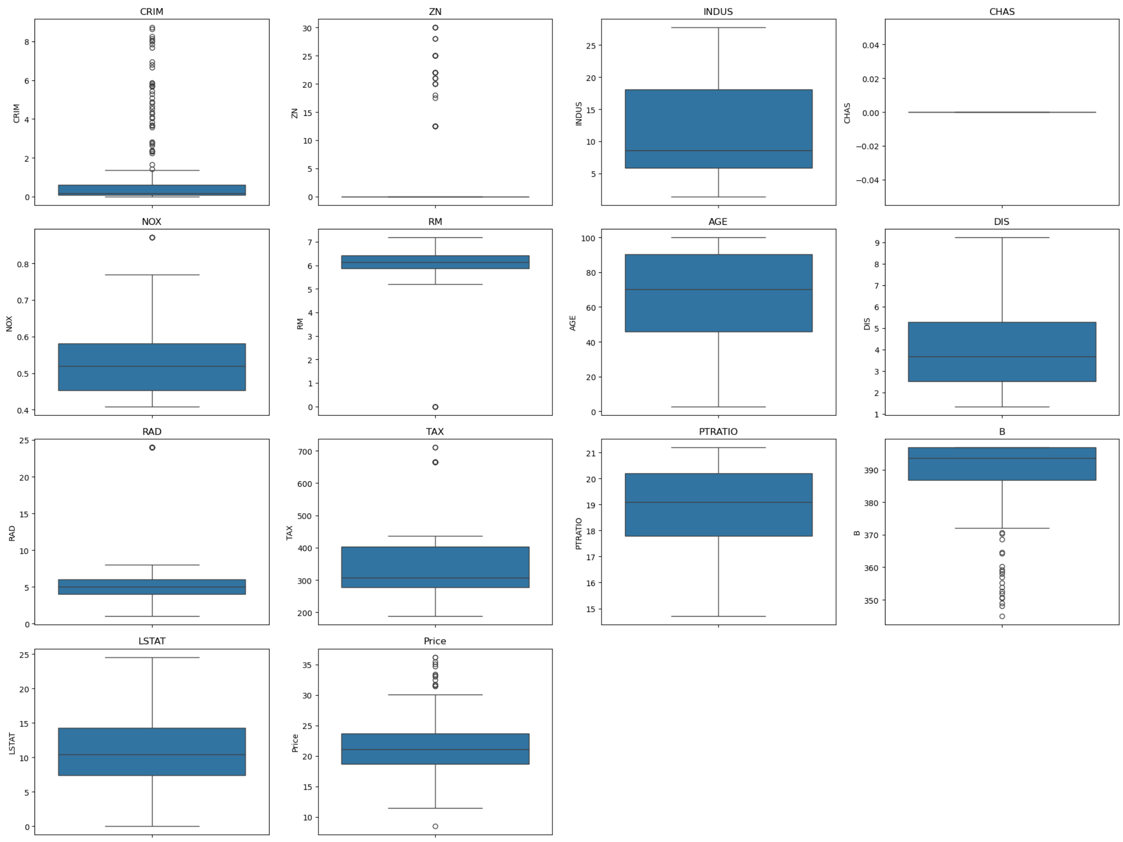
[125]:
```
# Calculate the IQR for the 'AGE' column
Q1 = data['RM'].quantile(0.25)  # 25th percentile
Q3 = data['RM'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with 0 in the 'AGE' column
data['RM'] = data['RM'].apply(lambda x: 0 if x < lower_bound or x > upper_bound␣
 ↪else x)

# Check the updated column
data['RM'].head()
```

[125]:
```
0    6.575
1    6.421
2    7.185
3    6.998
4    7.147
Name: RM, dtype: float64
```

[127]:
```
# Calculate the IQR for the 'AGE' column
Q1 = data['LSTAT'].quantile(0.25)  # 25th percentile
Q3 = data['LSTAT'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with 0 in the 'AGE' column
data['LSTAT'] = data['LSTAT'].apply(lambda x: 0 if x < lower_bound or x >␣
↪upper_bound else x)

# Check the updated column
data['RM'].head()
```

[127]:
```
0    6.575
1    6.421
2    7.185
3    6.998
4    7.147
Name: RM, dtype: float64
```

[129]:
```
plt.figure(figsize=(20, 15))  # Adjust figure size as needed
for i, column in enumerate(data.columns, 1):
    plt.subplot(4, 4, i)  # Adjust grid dimensions based on the number of␣
↪columns
    sns.boxplot(y=data[column])
    plt.title(column)

plt.tight_layout()
plt.show()
```



[131]:
```python
X = data[['AGE', 'RM', 'LSTAT']]
y = data['Price']
```

[133]:
```python
from sklearn.model_selection import train_test_split
```

[135]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state=40)
```

[139]:
```python
X_train.shape
```

[139]: (214, 3)

[141]:
```python
X_test.shape
```

[141]: (54, 3)

[143]:
```python
y_train.shape
```

[143]: (214,)

[145]:
```python
y_test.shape
```

[145]: (54,)

[149]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
```

[151]:
```
model = LinearRegression()
```

[153]:
```
model.fit(X_train, y_train)
```

[153]: LinearRegression()

[158]:
```
y_pred = model.predict(X_test)
```

[160]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")
```

```
Mean Squared Error: 12.2986764374419
R-squared Score: 0.4727964006739217
```

[ ]:
```

```