In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Read the file
url = "house_data.csv"
df = pd.read_csv(url, header=0, names=['Rooms', 'Age', 'Distance', 'Accessibil
```

In [3]:
```
df.head() # prints the first five rows
```

Out[3]:

	Rooms	Age	Distance	Accessibility	Tax	DisadvantagedPosition	Crime	NiticOxides	Pup
0	5.565	70.6	2.0635	24	666	17.16	8.79212	0.584	
1	6.879	77.7	3.2721	8	307	9.93	0.62356	0.507	
2	5.972	76.7	3.1025	4	304	9.97	0.34940	0.544	
3	6.943	97.4	1.8773	5	403	4.59	1.22358	0.605	
4	5.926	71.0	2.9084	24	666	18.13	15.57570	0.580	

In [4]:
```
# Saved the independent variables in x and dependent variable in y.
features = ['Rooms', 'Age', 'Distance', 'Accessibility', 'Tax', 'Disadvantaged
x = df[features].values
y = df['Price'].values
```

In [5]:
```
print(x.shape)
print(y.shape)
```

```
(399, 11)
(399,)
```

In [6]:
```
# Standardized the data (x and y) using the StandardScaler().fit_transform fun
x_Stand = StandardScaler().fit_transform(x)
y_Stand = StandardScaler().fit_transform(y.reshape(-1, 1))
```

In [7]:
```
# Split the data into train and test datasets with test size = 20%
X_train, X_test, y_train, y_test = train_test_split(x_Stand, y_Stand, test_siz
y_train = y_train.reshape(-1) # used to flatten the array elements to a 1D arr
y_test = y_test.reshape(-1)
print (X_train.shape)
print(y_train.shape)
print (X_test.shape)
print( y_test.shape)
```

```
(319, 11)
(319,)
(80, 11)
(80,)
```

In [8]:
```
# Apply LR model on xTrain, yTrain, print the r_squared, the intercept, and th
model = LinearRegression() # this is the OLS model
model.fit(X_train, y_train)
r_sq = model.score(X_train, y_train)
print('R squared:', r_sq)

print('intercept:', model.intercept_)

print('slope:', model.coef_)
```

```
R squared: 0.710424607703857
intercept: -0.010633473551120154
slope: [ 0.22499014  0.00857913 -0.30043321  0.253541   -0.21230575 -0.47439
113
 -0.11684349 -0.23366174 -0.21449969  0.09478876  0.0443366 ]
```

```
 -R squared 0.7104 means that 71% of the variance in the dependent variable
is explained by the independent variables in the model. this is a reasonably
good fit because it is closer to 1.
 -Intercept -0.011 explains that when all independent variables equal zero,
the dependent variable also tend towards zero.
 -Coefficients - Rooms with positive coefficent 0.2535 shows the most
substantial influence on house price predictions in this model. Smaller
variables are less likely to be statistically significant.
```

In [9]:
```
# Use the model to predict the output for the test data set (xTest),
# then find the error (MSE) and r^2
y_pred = model.predict(X_test)
print('R squared:', model.score(X_test, y_test))

y_pred= y_pred.reshape(-1)
e = y_test - y_pred

print("MSE = ",sum(e**2)/83)
```

```
R squared: 0.7740839250436754
MSE =  0.2668190698198442
```

The MSE tells us the average squared difference between the predicted and actual value of the dependent variable. The lower the better and this MSE shows a reasonably good fit, it indicates a relatively small errors on the average.

In [12]:
```
# summarize results
from mlxtend.evaluate import bias_variance_decomp

mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test,
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

```
MSE: 0.294
Bias: 0.277
Variance: 0.016
```

Multicollinearity occurs when two or more independent variable in a dataframe have a high correlation with one another in a regression model.This can be examined by calculating the correlation matrix of coefficients.

```
High correlation of coefficients suggest multicollinearity which can affect
the interpretablity and stability of the regression model.
```

In [14]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [15]:

```
# Calculate VIF for each variable
def calc_vif(X):
    vif = pd.DataFrame()
    vif["Variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shap

    return(vif)
X = df.iloc[:,:-1]
calc_vif(X)


# Display the results
```

Out[15]:

	Variables	VIF
0	Rooms	73.140393
1	Age	20.530213
2	Distance	14.100498
3	Accessibility	13.804679
4	Tax	55.873128
5	DisadvantagedPosition	10.773437
6	Crime	2.083883
7	NiticOxides	72.374336
8	PupilTeacher	76.705813
9	Residential	2.834652
10	NonRetail	13.862675

```
.Multicollinearity is best assessed by calculating the Variance Inflation
Factor (VIF) for each coefficient.
.VIF measures the extent of multicollinearity in the model.
.VIF less than 5 shows moderate correlation which is generally acceptable.
.VIF greater than 5 shows high correlation which indicates potential
multicollinearity.
.The high VIF's suggest that the variables are highly correlated with
eachother in the model, potentially causing multicollinearity issues.
.Some of the effects of multicollinearity is that the coefficients become
less reliable and might change dramatically with small changes in the data.
Also, it becomes extremely difficult to interpret the individual effects of
each variable due to the strong interdependence.
```