In [13]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [14]:

```
#reading the data set
data = pd.read_csv('house_data.csv')
```

In [15]:

```
#exploring
data.head()
```

Out[15]:

	Rooms	Age	Distance	Accessibility	Tax	DisadvantagedPosition	Crime	NitricOxides	PupilTeacher	Residential	NonRe
0	5.565	70.6	2.0635	24	666	17.16	8.79212	0.584	20.2	0.0	18
1	6.879	77.7	3.2721	8	307	9.93	0.62356	0.507	17.4	0.0	6
2	5.972	76.7	3.1025	4	304	9.97	0.34940	0.544	18.4	0.0	9
3	6.943	97.4	1.8773	5	403	4.59	1.22358	0.605	14.7	0.0	19
4	5.926	71.0	2.9084	24	666	18.13	15.57570	0.580	20.2	0.0	18

In [16]:

```
# seperate the independant variables from the dependant variables
X = data.drop('Price', axis=1)
y = data['Price']
```

In [17]:

```
#standardizing x and y
#standardizing ensures that you data has a standard deviation=1 and mean=0
x_Standard = StandardScaler().fit_transform(X)
#transfroms y into a 2-d array
y_Standard = StandardScaler().fit_transform(y.values.reshape(-1,1))
```

In [18]:

```
# split the data into test and train sets and get their shape
X_train, X_test, y_train, y_test= train_test_split(x_Standard, y_Standard, test_size=0.2
, random_state=42)
#reshape  y_train and y_test to be 1-d arrays as it is required to be in that format for
some functions.
y_train=y_train.reshape(-1)
y_test=y_test.reshape(-1)
#get the shape of x train and test along with y train and test
print(y_train.shape)
print(X_train.shape)
print(y_test.shape)
print(X_test.shape)
```

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

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

In [19]:

```
# linear regression model
# the class LinearRegression is made to create an object lr_model which will be used to f
it the model to the training data
lr_model = LinearRegression()
```

In [20]:

```
# To train the model, finds co-efficient and intercept that have the best fit for the tra
ining data
lr_model.fit(X_train, y_train)
```

Out[20]:

▾ LinearRegression

LinearRegression()

In [21]:

```
# To evaluate the model calculate the MSE and R^2
rsquare=lr_model.score(X_train,y_train)
print("R square \t \t: \t ",rsquare)
#print the coefficients(slope and intercept)
print("intercept    \t \t:\t", lr_model.intercept_)
X_train = sm.add_constant(X_train) # to calculate B0(intercept)
OLS_m = sm.OLS(y_train, X_train)
results = OLS_m.fit()
print(results.summary())
```

```
R square     :     0.7312338225521164
intercept        :  0.009179769586453904
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.731
Model:                            OLS   Adj. R-squared:                  0.722
Method:                 Least Squares   F-statistic:                     75.93
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           9.02e-81
Time:                        08:03:32   Log-Likelihood:                -247.88
No. Observations:                 319   AIC:                             519.8
Df Residuals:                     307   BIC:                             564.9
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0092      0.030      0.305      0.761      -0.050       0.068
x1             0.2566      0.040      6.372      0.000       0.177       0.336
x2             0.0078      0.052      0.152      0.879      -0.094       0.109
x3            -0.3075      0.058     -5.265      0.000      -0.422      -0.193
x4             0.2408      0.077      3.117      0.002       0.089       0.393
x5            -0.1974      0.084     -2.354      0.019      -0.363      -0.032
x6            -0.4740      0.050     -9.519      0.000      -0.572      -0.376
x7            -0.1203      0.040     -2.993      0.003      -0.199      -0.041
x8            -0.2000      0.064     -3.120      0.002      -0.326      -0.074
x9            -0.2458      0.041     -6.061      0.000      -0.326      -0.166
x10            0.1009      0.044      2.298      0.022       0.014       0.187
x11            0.0527      0.061      0.858      0.391      -0.068       0.173
==============================================================================
Omnibus:                      121.977   Durbin-Watson:                   2.017
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              499.286
Skew:                           1.608   Prob(JB):                    3.81e-109
Kurtosis:                       8.218   Cond. No.                         9.19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifie
d.
```

The beta coefficients represent the weight of the feature in a linear regression model . The highest beta coefficient values are for rooms and accessibility. For rooms it is 0.25 which means that if there is one unit increase in rooms then price would also increase by 0.25 units and if Accessibility increases by one unit then price would also increase by 0.24 units.

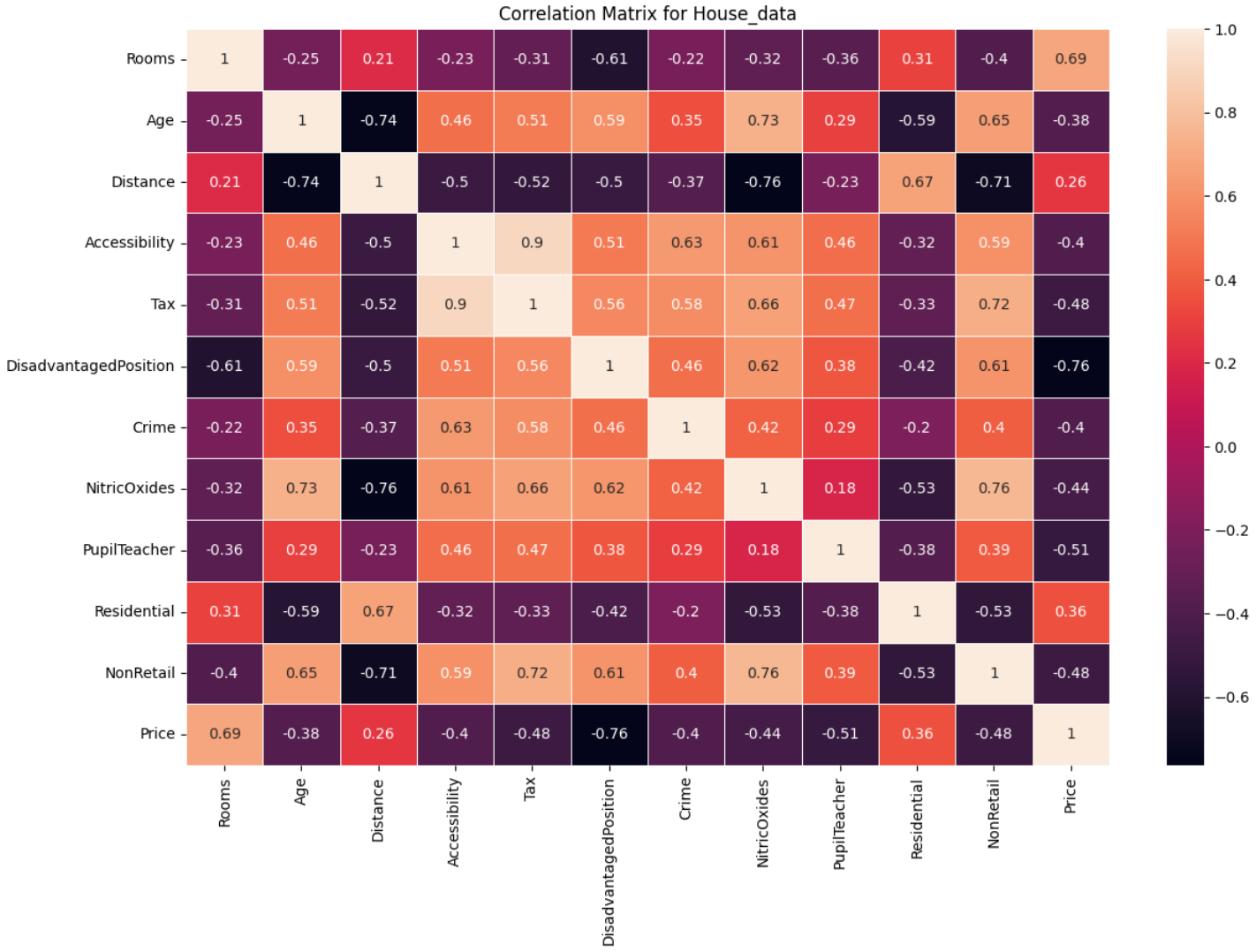
In [22]:

```
#price prediction using the dataset
y_predict = lr_model.predict(X_test)
#getting r^2 and Mean Squared Error
print("r-square \t:\t",lr_model.score(X_test,y_test))
e=y_test-y_predict
print("MSE\t\t:\t",sum(e**2)/80)
```

```
r-square    :    0.7489054841446297
MSE    :    0.2180334515605399
```

In [23]:

```
#to check the correlation using heat map.
c_m = data.corr()
# using matplotlb to generate a correlation matrix
plt.figure(figsize=(14, 9))
#generate a heat map to show significant correlations
sns.heatmap(c_m, annot=True, linewidths=.7)
plt.title('Correlation Matrix for House_data')
plt.show()
```



1. **Age has a high negative correlation with Distance of -0.74 and a high positive correlation with NitricOxides of 0.73**
2. **Distance has a high negative correlation with NitricOxides of -0.76 , NonRetail of -0.71 and a significant**

positive correlation with Residential of 0.67.
3. Accessibility has a significant positive correlation with tax of 0.9.
4. Tax has a strong positive correlation with NonRetail of 0.72.

A high negative and positive correlation between these variables tells us that these variables provide matching information to the model and this can lead to multicollinearity issues. It will be hard to differentiate between the individual effects of these variables on the dependant variable which is the price. The OLS(ordinary least squares) regression can predict the wrong values for the co-efficients and the std error when there is multicollinearity in the dataset.OLS estimation may not be accurate and cause several issues like :

1. **Sensitive coefficients: meaning that any change in the data will significantly affect the values of the coefficients for each of the variable.**
2. **increased standard errors: this can increase the standard errors which are calculated so when it will be increased it shows that the coefficients are less precise and not exaclty accurate.**

to interpret the multicollinearity in the dataset VIF(variance inflation factor) can be used :

1. **VIF>10: there is a high multicollinearity and might require attention.**
2. **VIF<5: this means that there is low multicollinearity.**
3. **VIF>5 and VIF<10: moderate multicollinearity.**

In our dataset although we have multicollinearity we can still use OLS regression. However the coefficients and the standard error will not be precise . To reduce this:

1. **remove Outliers in the dataset as this will improve the accuracy of the model and reduce the multicollinearity.**
2. **remove the variables that have high correlation between them.this will improve the accuracy and reduce the MSE.**

In [24]:

```
Q1 = data.quantile(0.25) #Calculating the upper and the lower Quartiles
Q3 = data.quantile(0.75)
interquart_range = Q3 - Q1
outlier_calc = ((data < (Q1 - 1.5 * interquart_range)) | (data > (Q3 + 1.5 * interquart_
range)))
#dropping the ouliers after iqr calculation while retaining the rows in the original data
set
newdata = data[~outlier_calc.any(axis=1)]
#seperating the independant and the dependant variables
X_wout = newdata.drop('Price', axis=1)
y_wout = newdata['Price']
xstandardw= StandardScaler().fit_transform(X_wout)
ystandardw= StandardScaler().fit_transform(y_wout.values.reshape(-1,1))

X_train_wout, X_test_wout, y_train_wout, y_test_wout = train_test_split(xstandardw, ysta
ndardw, test_size=0.2, random_state=42)

lr_model_out = LinearRegression()
lr_model_out.fit(X_train_wout, y_train_wout)

y_predn = lr_model_out.predict(X_test_wout)
#print the new r-square and MSE without outliers
noout_r2 = r2_score(y_test_wout, y_predn)
noout_mse = mean_squared_error(y_test_wout, y_predn)

print(f'new R-Square: {noout_r2}')
print(f'new MSE: {noout_mse}')
```

```
new R-Square: 0.8356886857512229
new MSE: 0.20546983419141424
```

In [ ]:

```
```