In [26]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

In [27]:
```python
# Load the dataset
# Replace 'your_dataset.csv' with the actual file name
df = pd.read_csv(r"C:\Users\junai\OneDrive - Middlesex University\ML, Regression\We
```

In [28]:
```python
df.head(5)
```

Out[28]:

	Rooms	Age	Distance	Accessibility	Tax	DisadvantagedPosition	Crime	NitricOxides	Pupil
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 [29]:
```python
# Task 1: Train a linear regression model
# Assume 'Price' is the dependent variable
X = df.drop('Price', axis=1)
y = df['Price']
```

X contains all the independent variables (features), and y contains the dependent variable ('Price'). The goal is to train a linear regression model that can predict 'Price' based on the other features in the dataset.

In [30]:
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

#test_size=0.2 means that 20% of the data will be reserved for testing, and the rem
#random_state to a specific value (here, 42) means that the random split will be th
```

In [31]:
```python
# Fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[31]:

```
▾ LinearRegression
LinearRegression()
```

During the training process, the model estimates the coefficients for each independent variable in the training set. These coefficients are used to create a linear equation that can predict the target variable ('Price' in this case) based on the values of the independent variables.

In [32]:
```
#The correlation between the predictions produced by the linear regression model ar
#values of house prices was measured.

# Task 2: Measure the correlation between predictions and actual values
y_pred = model.predict(X_test)
#uses the trained linear regression model (model) to make predictions on the testir

correlation = np.corrcoef(y_pred, y_test)[0, 1]
#calculates the correlation coefficient between the predicted values (y_pred) and t

print(f'Correlation between predictions and actual values: {correlation:.2f}')
```

```
Correlation between predictions and actual values: 0.87
```

In [33]:
```
# Task 3: Interpret the regression model
# Print coefficients and their significance
X_train_with_const = sm.add_constant(X_train)
model_ols = sm.OLS(y_train, X_train_with_const).fit()
#creates an ordinary least squares (OLS) regression model using the training data.
# fits the model to find the coefficients that minimize the sum of squared residual

print(model_ols.summary())

#The correlation between the predictions produced by the linear regression model ar
#the actual values of house prices was measured.
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.731
Model:                            OLS   Adj. R-squared:                  0.722
Method:                 Least Squares   F-statistic:                     75.93
Date:                Sun, 12 Nov 2023   Prob (F-statistic):           9.02e-81
Time:                        20:46:42   Log-Likelihood:                -971.73
No. Observations:                 319   AIC:                             1967.
Df Residuals:                     307   BIC:                             2013.
Df Model:                          11
Covariance Type:            nonrobust
==================================================================================
=======
                            coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------------
-------
const                    44.4169      6.654      6.675      0.000      31.323
57.510
Rooms                     3.5006      0.549      6.372      0.000       2.420
4.582
Age                       0.0027      0.018      0.152      0.879      -0.032
0.037
Distance                 -1.3847      0.263     -5.265      0.000      -1.902
-0.867
Accessibility             0.2655      0.085      3.117      0.002       0.098
0.433
Tax                      -0.0112      0.005     -2.354      0.019      -0.021
-0.002
DisadvantagedPosition    -0.6425      0.067     -9.519      0.000      -0.775
-0.510
Crime                    -0.1235      0.041     -2.993      0.003      -0.205
-0.042
NitricOxides            -16.6021      5.321     -3.120      0.002     -27.072
-6.132
PupilTeacher             -1.0925      0.180     -6.061      0.000      -1.447
-0.738
Residential               0.0403      0.018      2.298      0.022       0.006
0.075
NonRetail                 0.0729      0.085      0.858      0.391      -0.094
0.240
==============================================================================
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.                     1.19e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 1.19e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

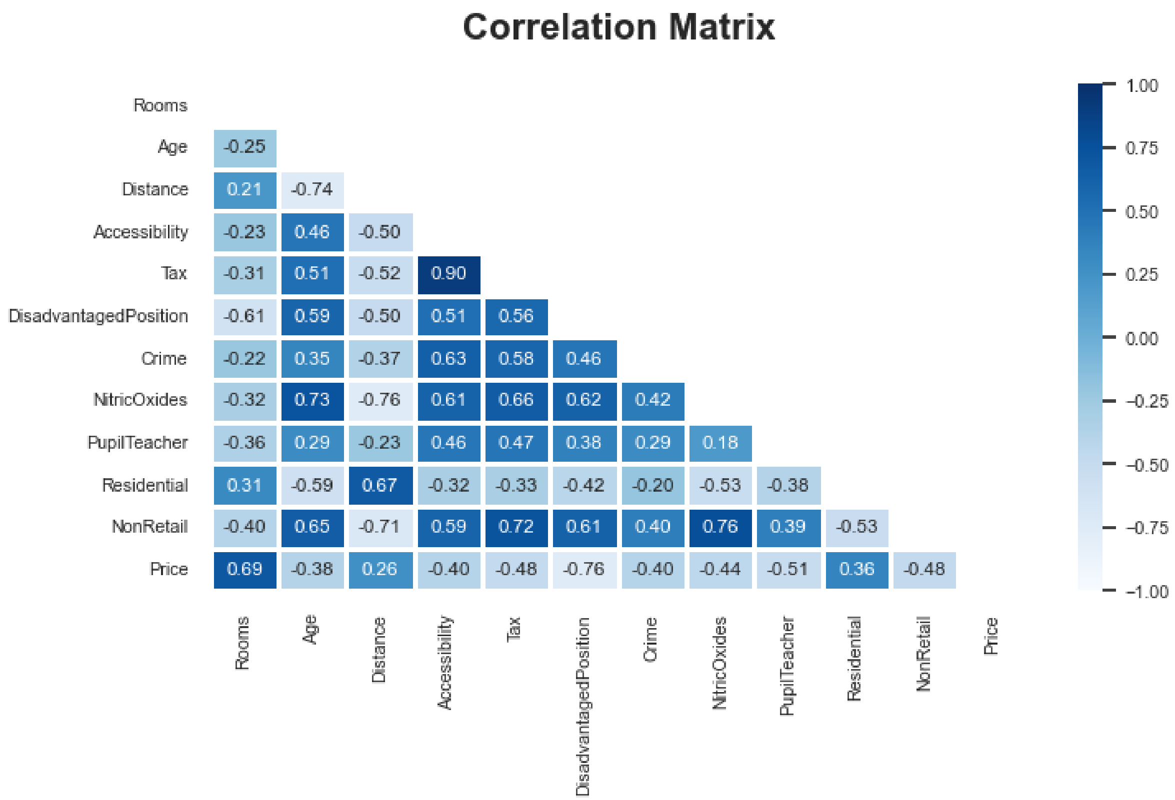
Standard Errors (std err): These represent the standard deviation of the estimated coefficients The t-statistic is the coefficient divided by its standard error. It is used to test the hypothesis that the coefficient is different from zero. P-Values (P>|t|): The p-value associated with each coefficient tests the null hypothesis that the coefficient is equal to zero. Small p-values (typically less than 0.05) suggest that the coefficient is statistically significant. Confidence Intervals ([0.025 0.975]): These intervals provide a range within which the true population parameter is likely to fall.

In [ ]:

In [34]:
```
#To remove multicollinearity
corr = df.corr()

# create mask for upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Create heatmap
plt.figure(figsize=(8, 4))
# set theme to change overall style
sns.set_theme(style="white", font_scale=0.7)
sns.heatmap(corr, cmap="Blues", linewidths=2, mask=mask, vmin=-1, vmax=1, annot=Tru
plt.title("Correlation Matrix", fontsize=16, weight = 'bold', pad=20)
plt.show()
```



In [37]:
```
sc = StandardScaler()
df_scaled = sc.fit_transform(df)
df_scaled = pd.DataFrame(df_scaled)
```

In [38]:
```
X = df.drop('Price', axis=1)
y = df['Price']
```

In [39]:
```
X_scaled = sc.fit_transform(X)
XX = sm.add_constant(X_scaled)
model = sm.OLS(y, XX).fit()
print(model.summary())
```

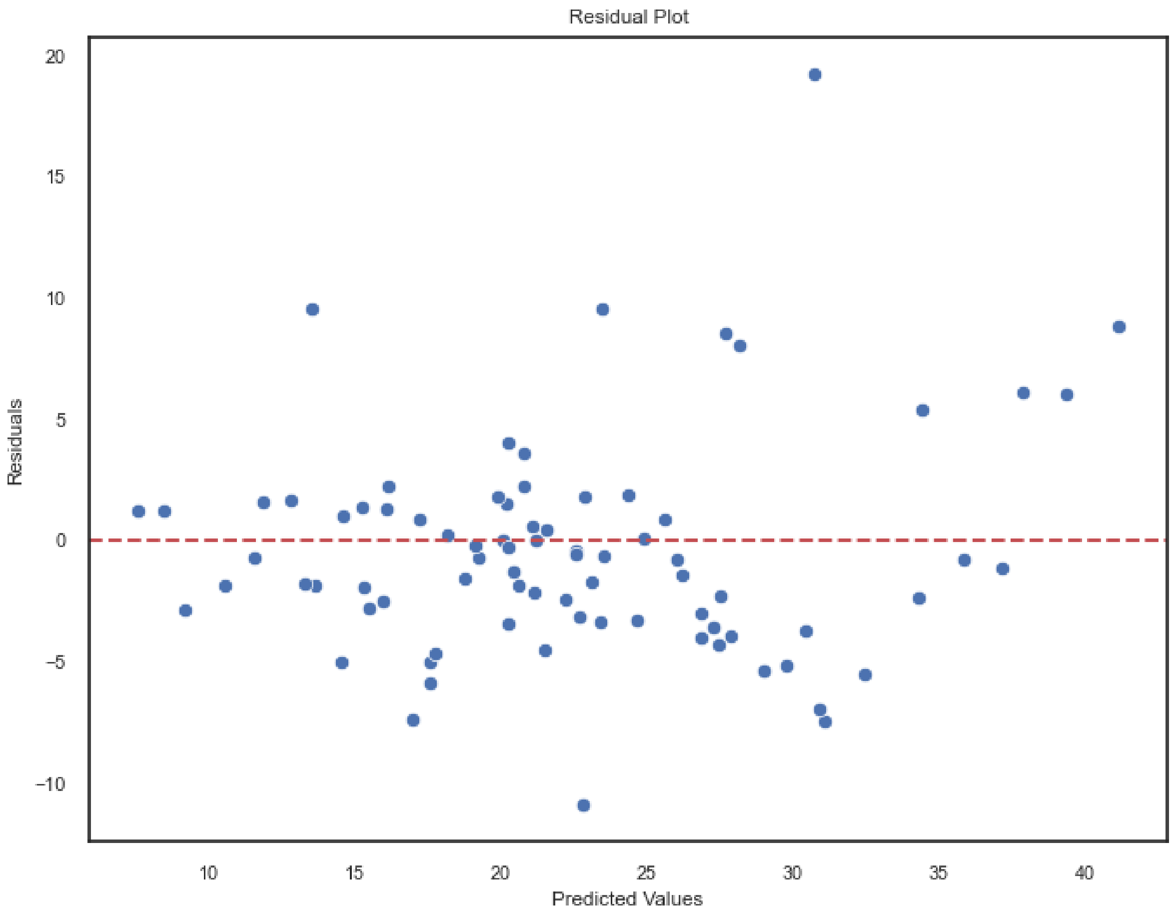
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.736
Model:                            OLS   Adj. R-squared:                  0.728
Method:                 Least Squares   F-statistic:                     97.86
Date:                Sun, 12 Nov 2023   Prob (F-statistic):          1.67e-104
Time:                        20:47:42   Log-Likelihood:                -1206.2
No. Observations:                 399   AIC:                             2436.
Df Residuals:                     387   BIC:                             2484.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         22.7035      0.253     89.810      0.000      22.206      23.201
x1             2.5249      0.345      7.312      0.000       1.846       3.204
x2             0.2810      0.440      0.639      0.523      -0.583       1.145
x3            -2.8694      0.500     -5.739      0.000      -3.852      -1.886
x4             2.5692      0.663      3.876      0.000       1.266       3.872
x5            -2.1799      0.726     -3.002      0.003      -3.608      -0.752
x6            -4.6742      0.424    -11.022      0.000      -5.508      -3.840
x7            -1.0833      0.336     -3.222      0.001      -1.744      -0.422
x8            -2.0741      0.537     -3.864      0.000      -3.130      -1.019
x9            -2.3321      0.340     -6.863      0.000      -3.000      -1.664
x10            0.9836      0.382      2.576      0.010       0.233       1.734
x11            0.4983      0.503      0.991      0.322      -0.490       1.487
==============================================================================
Omnibus:                      145.656   Durbin-Watson:                   2.033
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              598.842
Skew:                           1.571   Prob(JB):                    9.18e-131
Kurtosis:                       8.113   Cond. No.                         9.25
==============================================================================

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

Now there is no multicollinearity

In [40]:

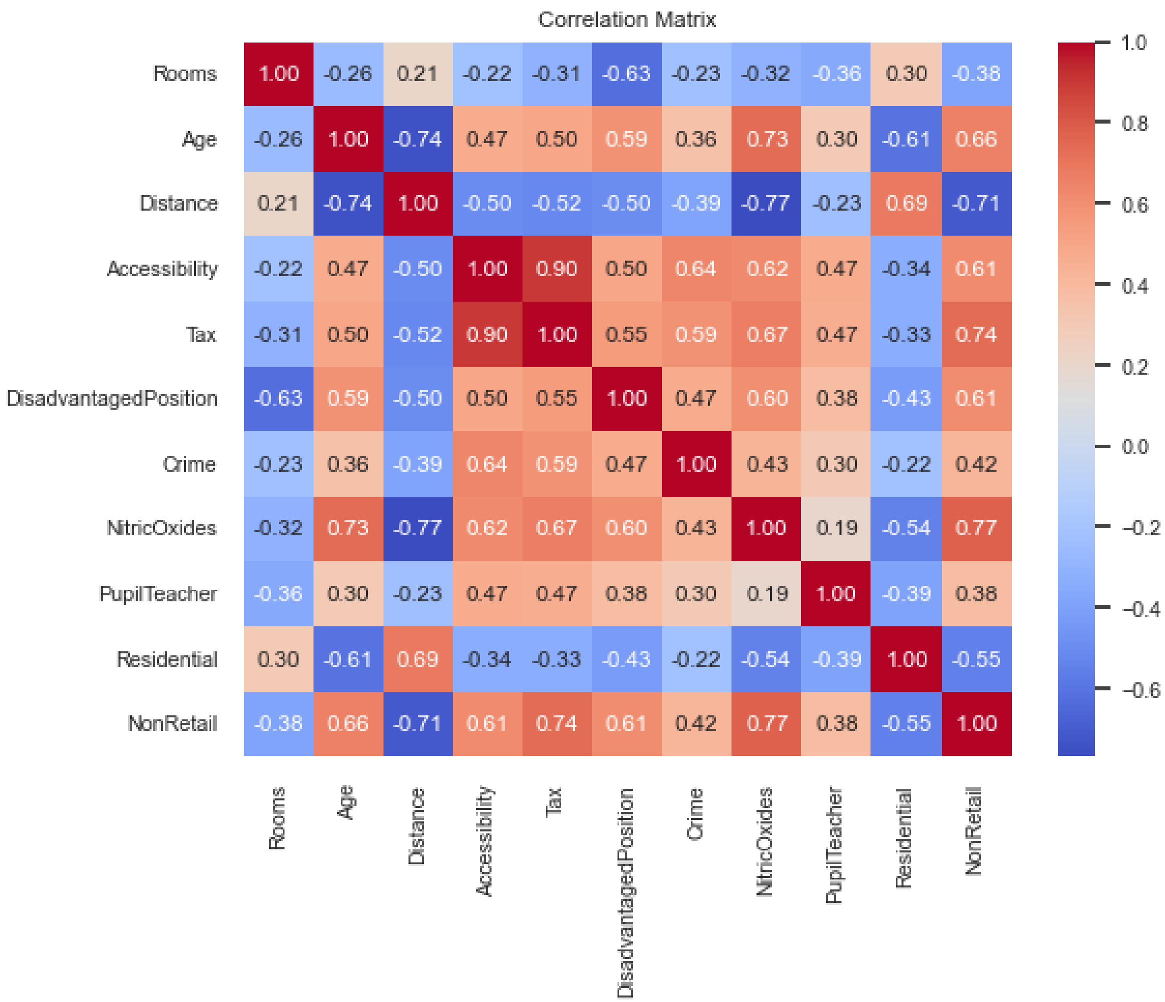
```
# Task 4: Conduct residual analysis
# Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```



Equal Spread: Ideally, you want to see a random scatter of points around the horizontal dashed line at y=0. This indicates that the residuals have an equal spread across different predicted values, supporting the assumption of homoscedasticity.

We can see that our residuals are random around zero that means aur data is linear

In [41]:
```
# Check for OLS conditions
# i) Non-multicollinearity
correlation_matrix = X_train.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



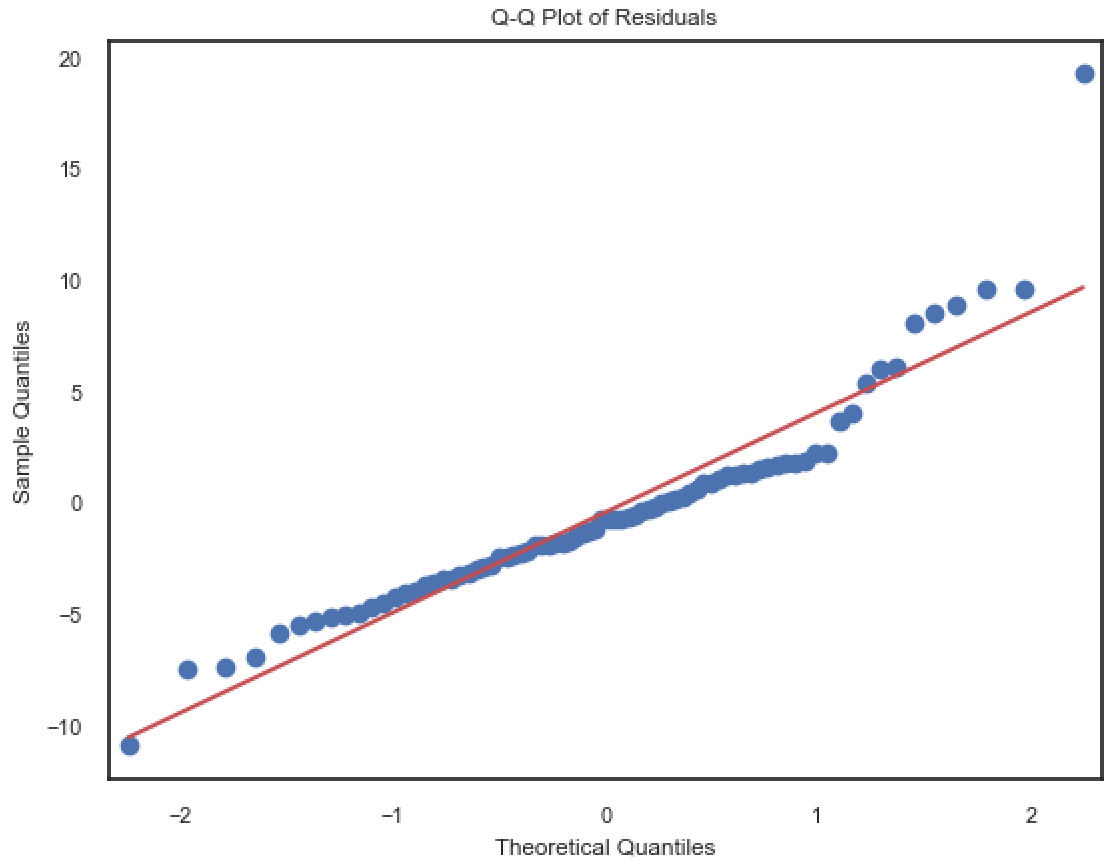
The correlation coefficient between predicted and actual values is extracted from the correlation matrix using [0, 1]

The correlation coefficient ranges from -1 to 1. A value of 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation. In this context, a positive correlation close to 1 would suggest that the model's predictions align well with the actual values in the testing set.

ii. there is no clear pattern in the residuals and they are spread randomly, it suggests that the independence assumption is reasonable.

iii. the spread remains fairly consistent in the above residual plot, homoscedasticity is likely present.

In [42]:
```
# iv) Normality of residuals - Check normality using Q-Q plot
sm.qqplot(residuals, line='s')  #is a function from the Statsmodels library that ge
plt.title('Q-Q Plot of Residuals')
plt.show()
```



The Q-Q (Quantile-Quantile) plot is a graphical tool used to assess whether a given set of data follows a theoretical distribution, such as the normal distribution.

Departures from linearity at the tails might suggest skewness or heavy-tailed distributions.

If the Q-Q plot shows a clear departure from the line, it could imply that the residuals do not follow a normal distribution.

In [ ]: