In [64]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [65]:
```
# Load the dataset
file_path = r"C:\Users\junai\OneDrive - Middlesex University\ML, Regression\Week8\h
df = pd.read_csv(file_path)
```

In [66]:
```
df.head(7)
```

Out[66]:

	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	
5	6.251	96.6	2.1980	24	666	16.44	9.92485	0.740	
6	5.605	70.2	7.9549	7	330	18.46	0.19133	0.431	

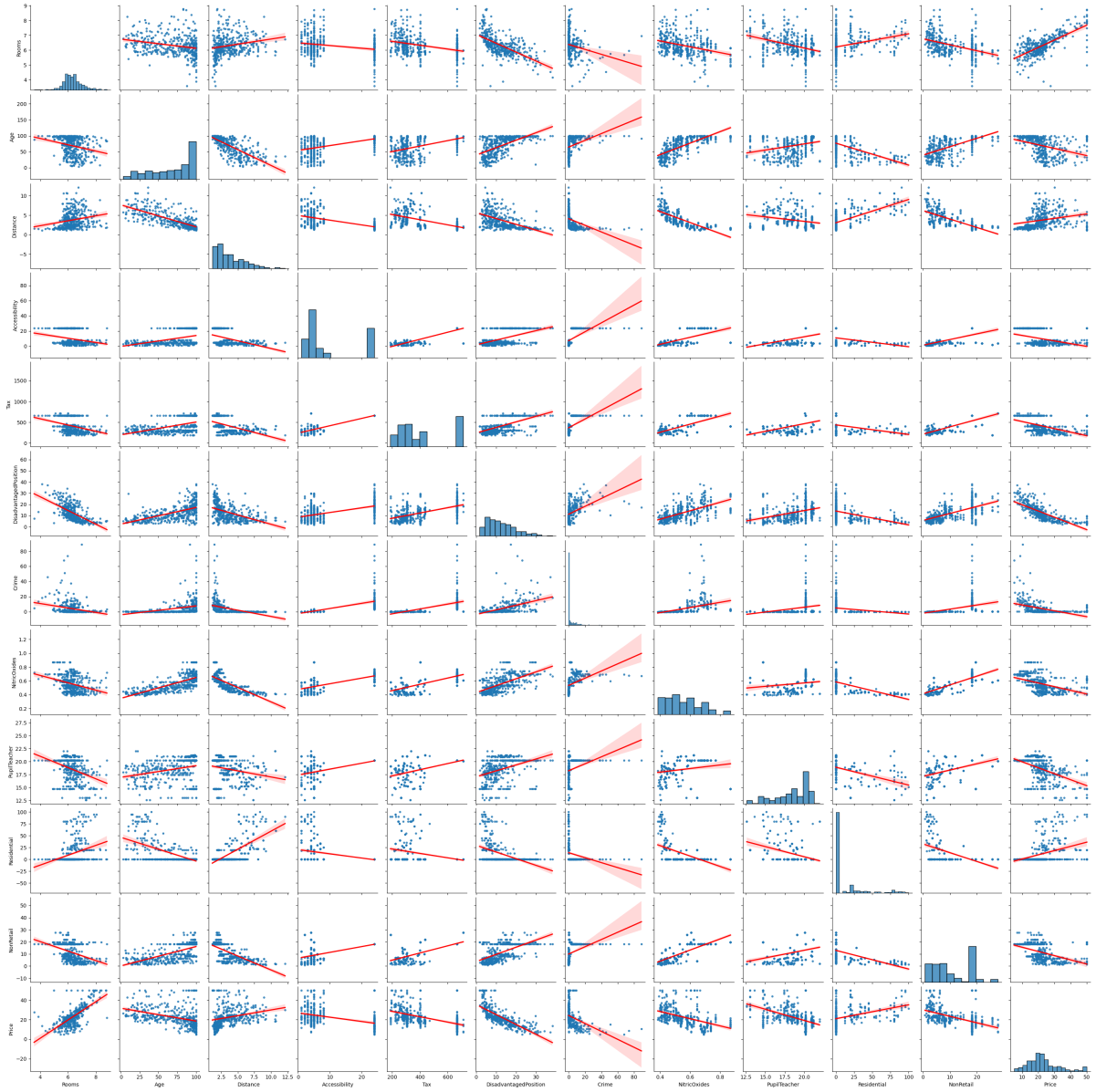
In [67]:
```
# Separate features and target variable
X = df.drop('Price', axis=1)
y = df['Price']
```

In [68]:
```
 #Print the coefficients (slope and intercept)
print("intercept:", linear_model.intercept_)
print("beta coefficients:", linear_model.coef_)
```

```
intercept: 44.41694811634807
beta coefficients: [ 3.50057923e+00  2.66105881e-03 -1.38472923e+00  2.65453772e-0
1
 -1.12181255e-02 -6.42486576e-01 -1.23515170e-01 -1.66020901e+01
 -1.09254310e+00  4.03208920e-02  7.28825944e-02]
```

In [69]:
```
# Create a pairwise scatterplot with correlation lines
sns.pairplot(df, kind="reg", plot_kws={'line_kws': {'color': 'red'}, 'scatter_kws':
# Show the plot
plt.show()
```

```
C:\Users\junai\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: T
he figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [70]:
```
# Convert DataFrame to NumPy array
X_np = X.values
y_np = y.values
```

In [71]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_np, y_np, test_size=0.2, rand
```

In [87]:
```
# Initialize the linear regression model
linear_model = LinearRegression()

# Train the linear regression model
linear_model.fit(X_train, y_train)
```

Out[87]:
```
▾ LinearRegression

LinearRegression()
```

In [88]:
```
# Make predictions on training and testing sets
y_train_pred = linear_model.predict(X_train)
y_test_pred = linear_model.predict(X_test)
```

In [89]:
```
# Evaluate the model
mse_train = mean_squared_error(y_train, y_train_pred)
```

```
mse_test = mean_squared_error(y_test, y_test_pred)
```

In [90]:
```
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
```

In [91]:
```
print("Linear Regression Training MSE:", mse_train)
print("Linear Regression Testing MSE:", mse_test)
```

```
Linear Regression Training MSE: 25.9056516815301
Linear Regression Testing MSE: 20.391567676138614
```

In [ ]:

In [92]:
```
print("Linear Regression Training R-squared:", r2_train)
print("Linear Regression Testing R-squared:", r2_test)
```

```
Linear Regression Training R-squared: 0.7312338225521164
Linear Regression Testing R-squared: 0.7489054841446298
```

In [ ]:

In [79]:
```
from sklearn.pipeline import make_pipeline

# Initialize polynomial regression models with different degrees
degrees = [2, 3, 4]
poly_models = {}
```

In [80]:
```
for degree in degrees:
    poly_model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    poly_model.fit(X_train, y_train)
    poly_models[degree] = poly_model

    # Make predictions on training and testing sets
    y_train_pred_poly = poly_model.predict(X_train)
    y_test_pred_poly = poly_model.predict(X_test)

    # Evaluate the model
    mse_train_poly = mean_squared_error(y_train, y_train_pred_poly)
    mse_test_poly = mean_squared_error(y_test, y_test_pred_poly)

    r2_train_poly = r2_score(y_train, y_train_pred_poly)
    r2_test_poly = r2_score(y_test, y_test_pred_poly)

    print(f"\nPolynomial Regression (Degree {degree}) Training MSE:", mse_train_pol
    print(f"Polynomial Regression (Degree {degree}) Testing MSE:", mse_test_poly)

    print(f"Polynomial Regression (Degree {degree}) Training R-squared:", r2_train_
    print(f"Polynomial Regression (Degree {degree}) Testing R-squared:", r2_test_pc
```

```
Polynomial Regression (Degree 2) Training MSE: 9.299561822563346
Polynomial Regression (Degree 2) Testing MSE: 15.612979108226284
Polynomial Regression (Degree 2) Training R-squared: 0.9035188261728764
Polynomial Regression (Degree 2) Testing R-squared: 0.8077473251442309

Polynomial Regression (Degree 3) Training MSE: 0.02193048942917856
Polynomial Regression (Degree 3) Testing MSE: 3133834.9817933305
Polynomial Regression (Degree 3) Training R-squared: 0.9997724753700119
Polynomial Regression (Degree 3) Testing R-squared: -38587.92999407813

Polynomial Regression (Degree 4) Training MSE: 5.2106724457354885e-18
Polynomial Regression (Degree 4) Testing MSE: 53563.08283040595
Polynomial Regression (Degree 4) Training R-squared: 1.0
Polynomial Regression (Degree 4) Testing R-squared: -658.5567621198547
```

3 point

In [84]:
```
# Initialize the polynomial regression model within a pipeline
degrees = (2,3,4)
for degree in degrees:
    poly_model = make_pipeline(PolynomialFeatures(degree), LinearRegression())

# Function to evaluate stability by training the model on different subsets
def evaluate_stability(model, X_train, y_train, num_iterations=100):
    coefficients = []
    for _ in range(num_iterations):
        # Create a random subset of the training data
        random_indices = np.random.choice(len(X_train), size=len(X_train), replace=
        X_subset = X_train[random_indices]
        y_subset = y_train[random_indices]

        # Train the model on the subset
        model.fit(X_subset, y_subset)

        # Store the beta coefficients
        coefficients.append(model.named_steps['linearregression'].coef_)

    return np.array(coefficients)

# Evaluate stability for polynomial regression model
poly_coefficients = evaluate_stability(poly_model, X_train, y_train)

# Calculate stability metrics for polynomial regression
poly_stability = np.std(poly_coefficients, axis=0)

# Print stability metrics
print(f"Polynomial Regression (Degree {degree}) Coefficients Stability:")
print(poly_stability)
```

```
Polynomial Regression (Degree 4) Coefficients Stability:
[4.00226221e-07 2.03356532e-07 3.91403637e-07 ... 2.82295864e-05
 1.54427222e-05 4.75421009e-05]
```

In [ ]:

In [99]:
```
from sklearn.linear_model import LinearRegression

# Assuming X_train is a DataFrame
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

Out[99]:
```
▾ LinearRegression
LinearRegression()
```

In [100…
```
# Load the new data for prediction
new_data = pd.read_csv(r"C:\Users\junai\OneDrive - Middlesex University\ML, Regress
```

In [101…
```
# Assuming new_data is a DataFrame with the same column names as X_train
predictions = linear_model.predict(new_data)
```

```
C:\Users\junai\anaconda3\Lib\site-packages\sklearn\base.py:457: UserWarning: X has
feature names, but LinearRegression was fitted without feature names
  warnings.warn(
```

In [107…

```
# Make predictions using the linear model
predictions = linear_model.predict(new_data)
```

```
C:\Users\junai\anaconda3\Lib\site-packages\sklearn\base.py:457: UserWarning: X has
feature names, but LinearRegression was fitted without feature names
  warnings.warn(
```

In [108…

```
# Print or use the predictions as needed
print("Predictions for New Data:")
print(predictions)
```

```
Predictions for New Data:
[25.84923637 41.05945579 17.24124895 27.29282474 30.87799868 32.86802656
 17.37090511 24.58391406 25.16387978 25.35125415]
```

In [ ]: