In [1]:

```
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.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
```

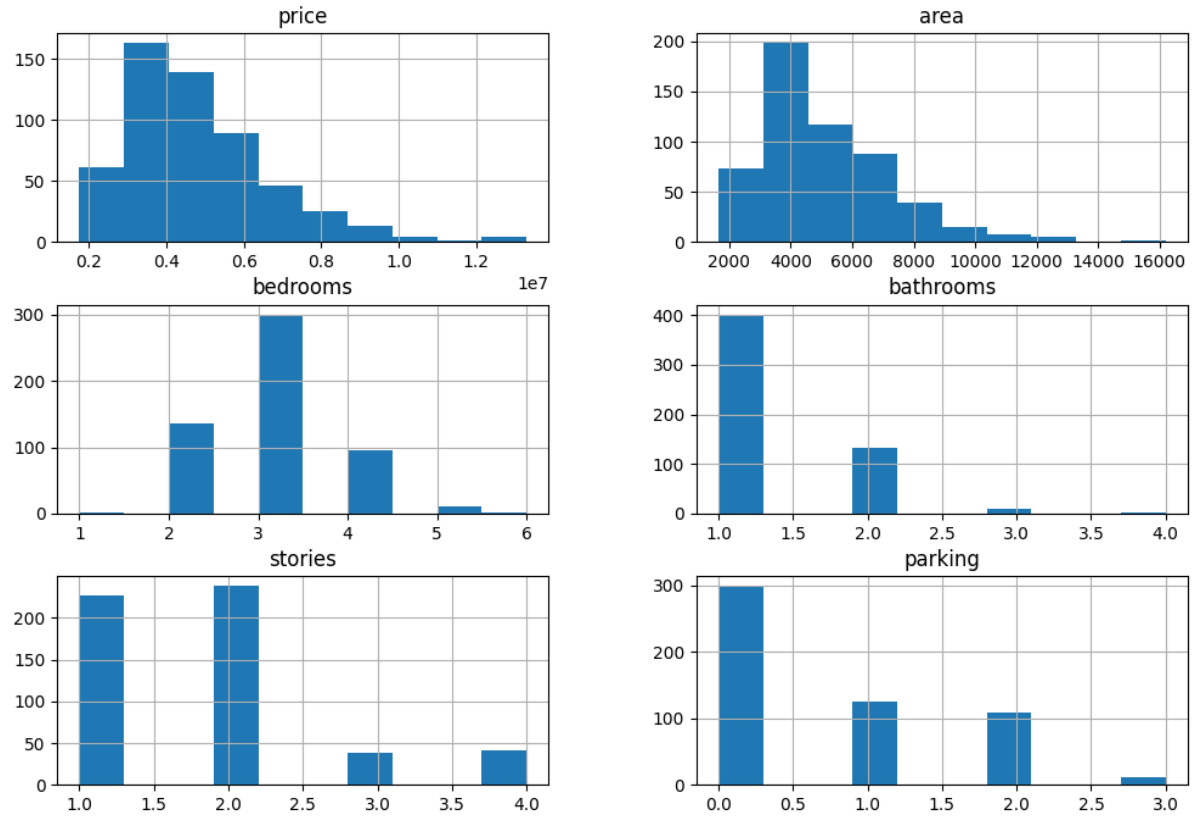
```
C:\Users\Tia\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\_dist
ributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\Tia\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs
\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\Tia\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs
\libopenblas64__v0.3.21-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [2]:

```
# Step 1: Download and load the dataset
data = pd.read_csv('Housing.csv')
```

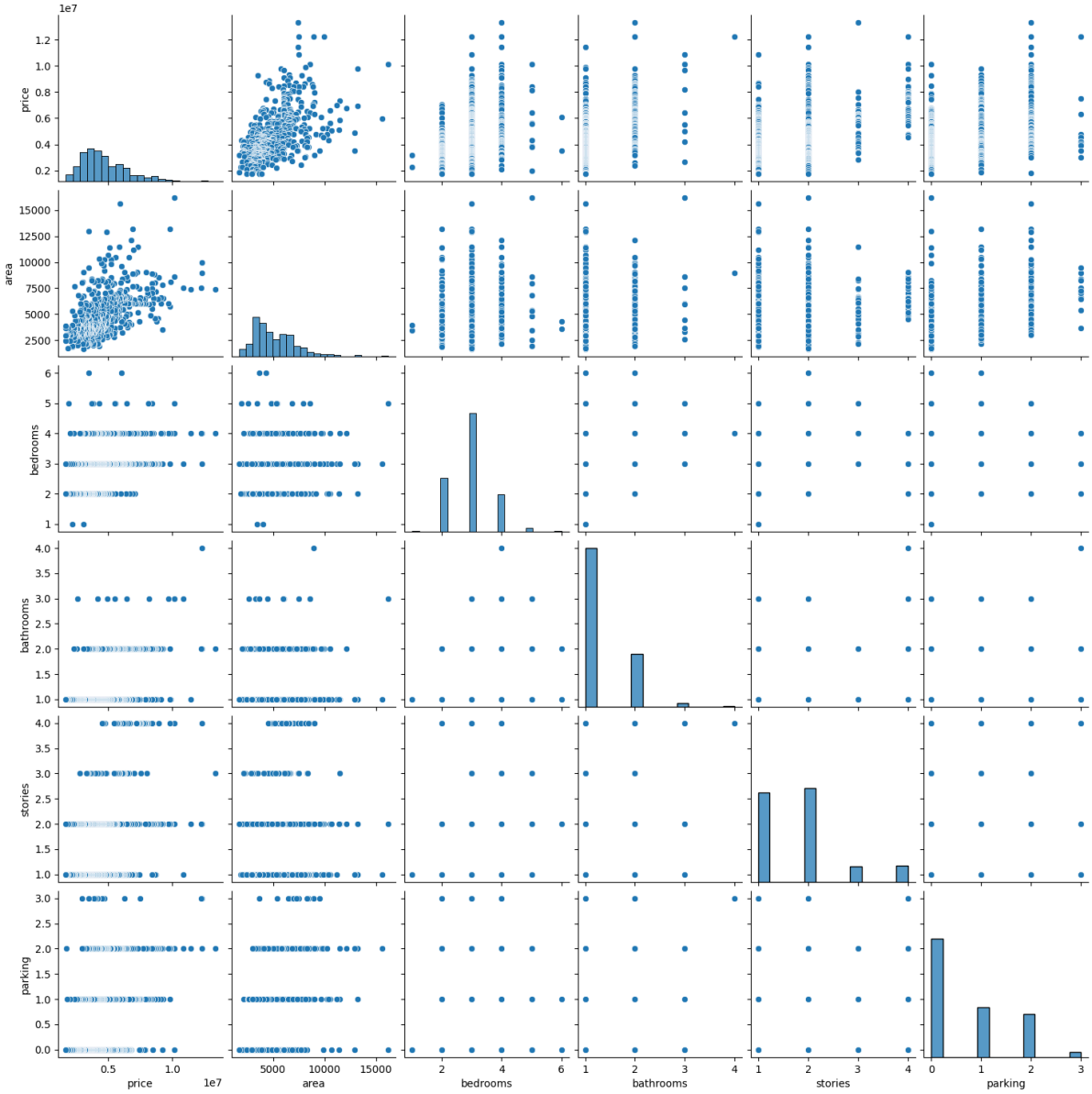
In [3]:

```
# Step 3: Perform visualizations
# Univariate Analysis
data.hist(figsize=(12, 8))
plt.show()
```



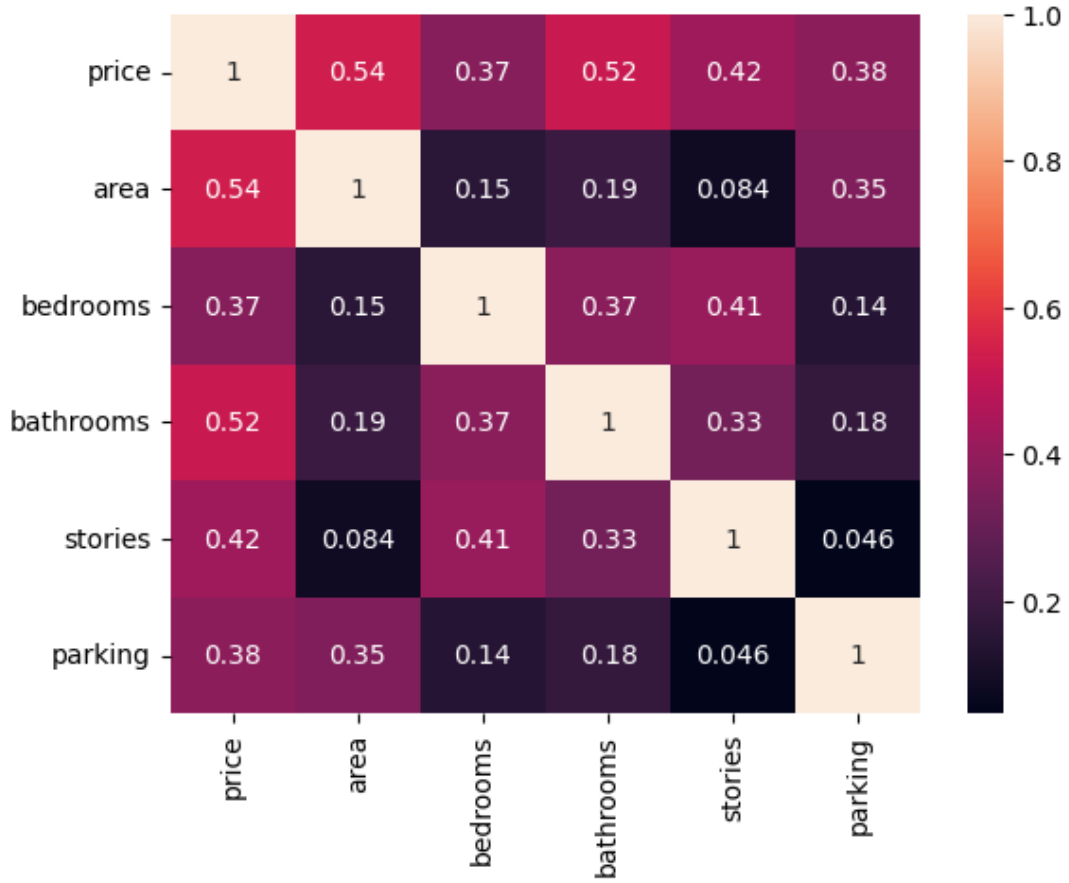
In [4]:

```
# Bi-Variate Analysis
sns.pairplot(data)
plt.show()
```



In [7]:

```
# Multi-Variate Analysis
numeric_cols = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
data_numeric = data[numeric_cols]
correlation_matrix = data_numeric.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.show()
```



In [8]:

```
# Step 4: Perform descriptive statistics
statistics = data.describe()
print(statistics)
```

```
              price          area    bedrooms   bathrooms     stories  \
count  5.450000e+02    545.000000  545.000000  545.000000  545.000000   \
mean   4.766729e+06   5150.541284    2.965138    1.286239    1.805505   
std    1.870440e+06   2170.141023    0.738064    0.502470    0.867492   
min    1.750000e+06   1650.000000    1.000000    1.000000    1.000000   
25%    3.430000e+06   3600.000000    2.000000    1.000000    1.000000   
50%    4.340000e+06   4600.000000    3.000000    1.000000    2.000000   
75%    5.740000e+06   6360.000000    3.000000    2.000000    2.000000   
max    1.330000e+07  16200.000000    6.000000    4.000000    4.000000   

          parking  
count  545.000000  
mean     0.693578  
std      0.861586  
min      0.000000  
25%      0.000000  
50%      0.000000  
75%      1.000000  
max      3.000000  
```

In [9]:

```
# Step 5: Check for missing values and deal with them
missing_values = data.isnull().sum()
print(missing_values)
```

```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
furnishingstatus  0
dtype: int64
```

In [11]:

```
# Step 7: Check for categorical columns and perform encoding
categorical_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])
```

In [13]:

```
# Perform one-hot encoding for categorical variables
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

column_transformer = ColumnTransformer([('encoder', OneHotEncoder(), [11])], remainder='passthrou
data = column_transformer.fit_transform(data)
```

In [14]:

```
# Step 8: Split the data into dependent and independent variables
X = data[:, 1:]  # Independent variables
y = data[:, 0]  # Dependent variable
```

In [15]:

```
# Step 9: Scale the independent variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [16]:

```
# Step 10: Split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [17]:

```
# Step 11: Build the Model (Linear Regression)
model = LinearRegression()
```

In [18]:

```
# Step 12: Train the Model
model.fit(X_train, y_train)
```

Out[18]:

▾ LinearRegression

LinearRegression()

In [19]:

```
# Step 13: Test the Model
y_pred = model.predict(X_test)
```

In [22]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Step 14: Measure the performance using Metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:", mae)
```

```
Mean Squared Error: 7.602511275517291e-31
Root Mean Squared Error: 8.719238083409175e-16
Mean Absolute Error: 7.33358328192764e-16
```

In [ ]:

```

```