Question #02: Linear Regression

1. Data Preprocessing and Exploration

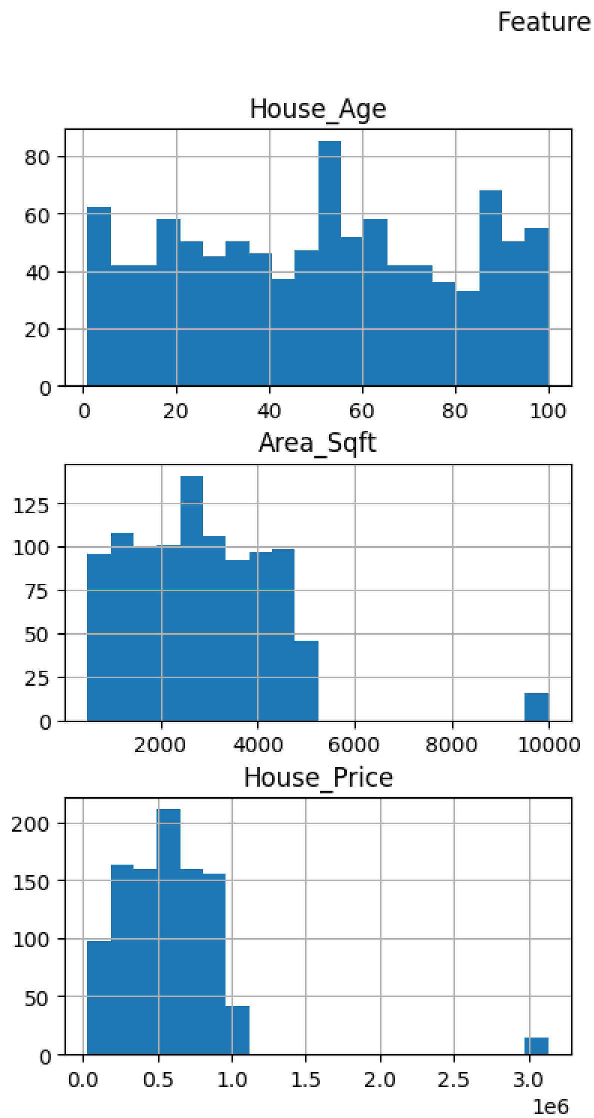
Load the Dataset and Handle Missing Values

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler


df = pd.read_csv("linear_regression_dataset.csv")


print(df.info())
print(df.isnull().sum())


df.fillna(df.median(numeric_only=True), inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   House_Age                 968 non-null    float64
 1   Num_Bedrooms              973 non-null    float64
 2   Area_Sqft                 965 non-null    float64
 3   Distance_to_City_Center   981 non-null    float64
 4   House_Price               968 non-null    float64
dtypes: float64(5)
memory usage: 39.2 KB
None
House_Age                  32
Num_Bedrooms               27
Area_Sqft                  35
Distance_to_City_Center    19
House_Price                32
dtype: int64
```

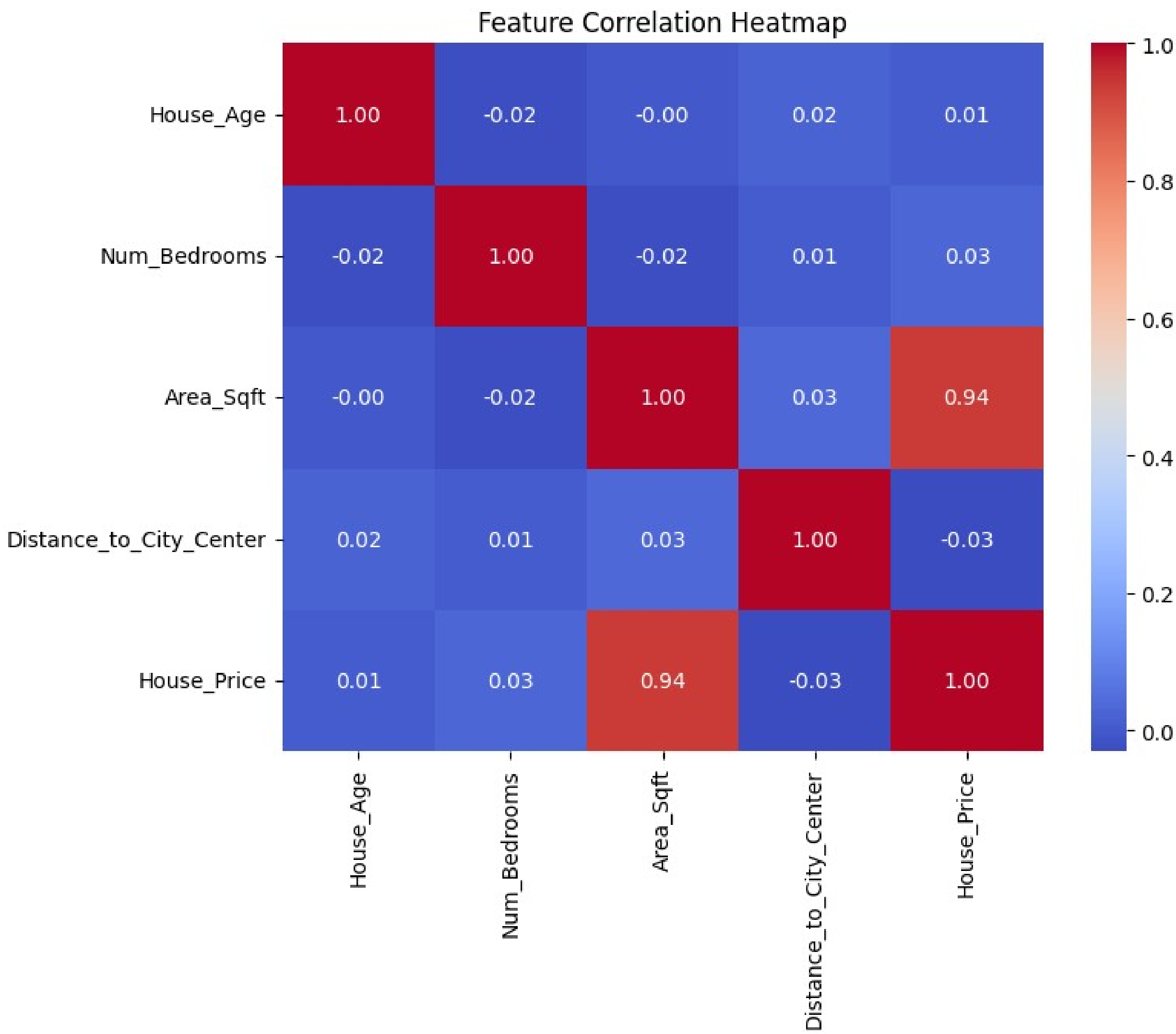
Visualizing Feature Distributions

```
df.hist(figsize=(10, 8), bins=20)
plt.suptitle("Feature Distributions")
plt.show()
```



Feature Correlation Analysis

```
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



2. Feature Engineering and Selection

Feature Normalization & Correlation Analysis

```
from sklearn.preprocessing import PolynomialFeatures


scaler = StandardScaler()
X_scaled =
pd.DataFrame(scaler.fit_transform(df.drop(columns=['House_Price'])),
columns=df.columns[:-1])


correlations = df.corr()['House_Price'].drop('House_Price')
print("Feature Correlations with House_Price:\n", correlations)
```

```
selected_features = correlations[abs(correlations) >
0.2].index.tolist()
X_selected = X_scaled[selected_features]


Feature Correlations with House_Price:
 House_Age                   0.005886
Num_Bedrooms                0.032417
Area_Sqft                   0.939058
Distance_to_City_Center    -0.031151
Name: House_Price, dtype: float64
```

Create Polynomial Features

```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X_selected)

print("Polynomial Features Shape:", X_poly.shape)

Polynomial Features Shape: (1000, 2)
```

3. Train a Linear Regression Model

Splitting Data into Training & Testing Sets

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score


X = X_poly

y = df['House_Price']


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Training & Evaluating Linear Regression Model

```
model = LinearRegression()
model.fit(X_train, y_train)


y_pred = model.predict(X_test)
```

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Performance:\nMAE: {mae}\nMSE: {mse}\nR²
Score: {r2}")
```

```
Linear Regression Performance:
MAE: 47552.10709897928
MSE: 7035494815.528621
R² Score: 0.9723984550181082
```

4. Implement Linear Regression using Gradient Descent

Gradient Descent Implementation

```
import numpy as np


m, n = X_train.shape
theta = np.zeros(n)
alpha = 0.01
iterations = 1000


def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    for i in range(iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradient = (1/m) * X.T.dot(errors)
        theta -= alpha * gradient
    return theta
```

Training Gradient Descent Model

```
# Add bias column (intercept) to X
X_train_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]


y_train_np = y_train.values


theta_final = gradient_descent(X_train_bias, y_train_np,
np.zeros(X_train_bias.shape[1]), alpha, iterations)
```

```
y_pred_gd = X_test_bias.dot(theta_final)


mae_gd = mean_absolute_error(y_test, y_pred_gd)
mse_gd = mean_squared_error(y_test, y_pred_gd)
r2_gd = r2_score(y_test, y_pred_gd)

print(f"Gradient Descent Regression Performance:\nMAE: {mae_gd}\nMSE:
{mse_gd}\nR² Score: {r2_gd}")
```

```
Gradient Descent Regression Performance:
MAE: 47602.07332171253
MSE: 7030830459.975481
R² Score: 0.9724167541460285
```

5. Predict House Prices for New Data

Scaling and Transformation for New Data

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, PolynomialFeatures


scaler_single = StandardScaler()
scaler_single.fit(df[['Area_Sqft']])

poly_single = PolynomialFeatures(degree=2, include_bias=False)
poly_single.fit(df[['Area_Sqft']])

def predict_house_price(new_data):

    if not isinstance(new_data, list):
        raise ValueError("Input must be a list containing one feature
value.")

    # Convert new_data into a DataFrame with correct column name
    new_data_df = pd.DataFrame([new_data], columns=['Area_Sqft'])


    new_data_scaled = scaler_single.transform(new_data_df)


    new_data_poly = poly_single.transform(new_data_scaled)

    predicted_price = model.predict(new_data_poly)[0]

    return predicted_price
```

Predicting House Prices

```python
print("Feature used during training:", ['Area_Sqft'])


new_house_1 = [2000]
new_house_2 = [1500]
new_house_3 = [3000]


predicted_price_1 = predict_house_price(new_house_1)
predicted_price_2 = predict_house_price(new_house_2)
predicted_price_3 = predict_house_price(new_house_3)

print(f"Predicted House Price for 2000 sqft: $
{predicted_price_1:.2f}")
print(f"Predicted House Price for 1500 sqft: $
{predicted_price_2:.2f}")
print(f"Predicted House Price for 3000 sqft: $
{predicted_price_3:.2f}")
```

```
Feature used during training: ['Area_Sqft']
Predicted House Price for 2000 sqft: $380809.10
Predicted House Price for 1500 sqft: $302954.74
Predicted House Price for 3000 sqft: $568047.87
```

```
C:\Users\HP 840G4\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
C:\Users\HP 840G4\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
C:\Users\HP 840G4\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```