College of Computer and Information Sciences
Computer Science Department

CSC 462
Machine learning

HW1: Univariate Linear Regression Report

Prepared by:

Name: Sarah Alajlan

Student ID: 442202314

Dr. Sarab AlMuhaideb

Table of contents

Dataset Description and Scatterplots:

- **Dataset Description:** The dataset consists of two columns: "area" and "price," primarily focused on housing data. The "area" column represents a fundamental feature related to the houses, while the "price" column denotes the monetary value or selling price of these houses. In the context of machine learning and data analysis, "area" serves as the primary feature used to predict or understand variations in house prices, with "price" serving as the target variable under examination.

- **scatterplots required:**

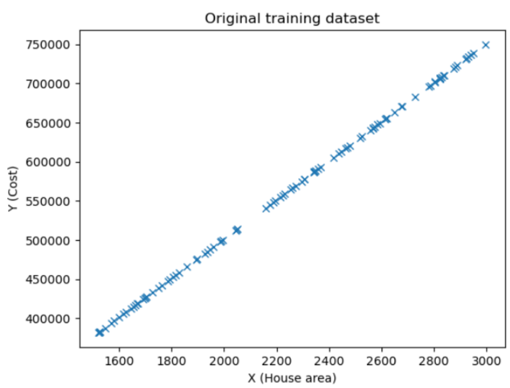


Figure 1

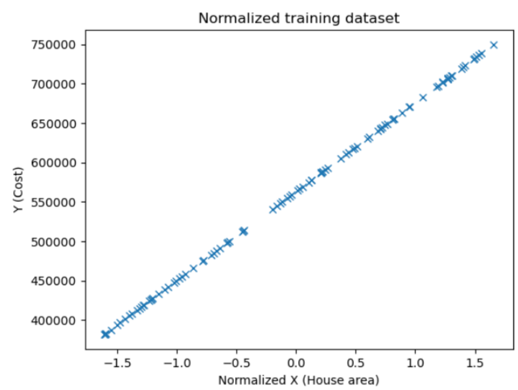


Figure 2

Data Preprocessing:

Loading and Extracting the Dataset: The code begins by importing necessary libraries, including NumPy, Pandas, Matplotlib, and scikit-learn's LinearRegression. We then proceed to extract the dataset from a CSV file named 'HW1_house_data(1).csv' using the pd.read_csv function. The dataset is loaded into a Pandas DataFrame and stored in the variable Hw1_data. To prepare the data for modeling, we extract the 'area' feature into a variable X and the 'price' into a variable Y. This step ensures that we have the raw data readily available for analysis and modeling, with X representing the input feature and Y representing the target variable.

Data Exploration: In Figures 3, 4, 5, and 6, we obtained summary statistics and information about the dataset using Hw1_data.head(), Hw1_data.tail(), Hw1_data.describe(), and Hw1_data.info(), respectively. These functions provide insights into the structure and content of the dataset.

Data Visualization: The original dataset is visualized using a scatter plot with house area ('X') on the x-axis and price ('Y') on the y-axis as showen in figure 1.

Data Normalization: After visualizing the data, it became apparent that the 'area' feature and the 'price' feature have different scales. The 'area' values are relatively large compared to the 'price' values, which can lead to difficulties during model training, particularly for machine learning algorithms sensitive to scale. To address this issue, we perform data normalization.

Normalization is a crucial preprocessing step that standardizes the scale of the features. In this case, the 'area' feature is normalized. The process involves subtracting the mean value of 'area' from each data point and then dividing by the standard deviation. This results in a new variable, 'X_normalized,' which represents the normalized 'area' feature.

The reason behind normalization is twofold:

Scale Consistency: It ensures that all features have the same scale, which can help prevent certain features from dominating the learning process during model training. It's especially important for algorithms like linear regression, where the scale of features affects the coefficients.

Improved Model Convergence: Many machine learning algorithms, including gradient-based optimization methods, converge faster and more reliably when features are within a similar scale. Normalization assists in smoother and faster convergence during model training.

After normalization, we can visualize the 'X_normalized' data using a scatter plot, as shown in Figure 3. This plot allows us to see the relationships between the normalized 'area' and 'price' features more clearly, facilitating further analysis and model development.

```
In [58]: Hw1_data.head()
Out[58]:
```

	area	price
0	2231.88	558852.17
1	2524.92	632260.29
2	1527.65	382994.25
3	1986.09	497607.86
4	2920.16	730870.84

Figure 3

```
In [59]: Hw1_data.tail()
Out[59]:
```

	area	price
95	2345.42	587342.25
96	1701.95	426321.81
97	2880.39	721159.46
98	2273.60	569238.36
99	2923.85	732054.33

Figure 4

```
In [57]: Hw1_data.describe()
Out[57]:
```

	area	price
count	100.00000	100.000000
mean	2247.97960	562990.340700
std	453.86445	113464.467239
min	1522.16000	381657.070000
25%	1812.84750	454164.525000
50%	2284.15000	571887.305000
75%	2617.72750	655407.227500
max	2995.40000	749686.490000

Figure 5

```
In [60]: Hw1_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   area    100 non-null    float64
 1   price   100 non-null    float64
dtypes: float64(2)
memory usage: 1.7 KB
```

Figure 6

Model Parameters and Predicted Values:

Table 1: Model Parameters and Predicted Values

Method	Theta0	Theta1	Predicted value
Gradient Descent	562966.0357047117	112890.79348255106	500974.7396107308
Scikit-Learn Regression	562990.3407000001	112895.66732934	500996.3682489
Normal Equation	562990.3407000001	112895.6673293369	500996.36824889656

Actual vs. Predicted Prices:

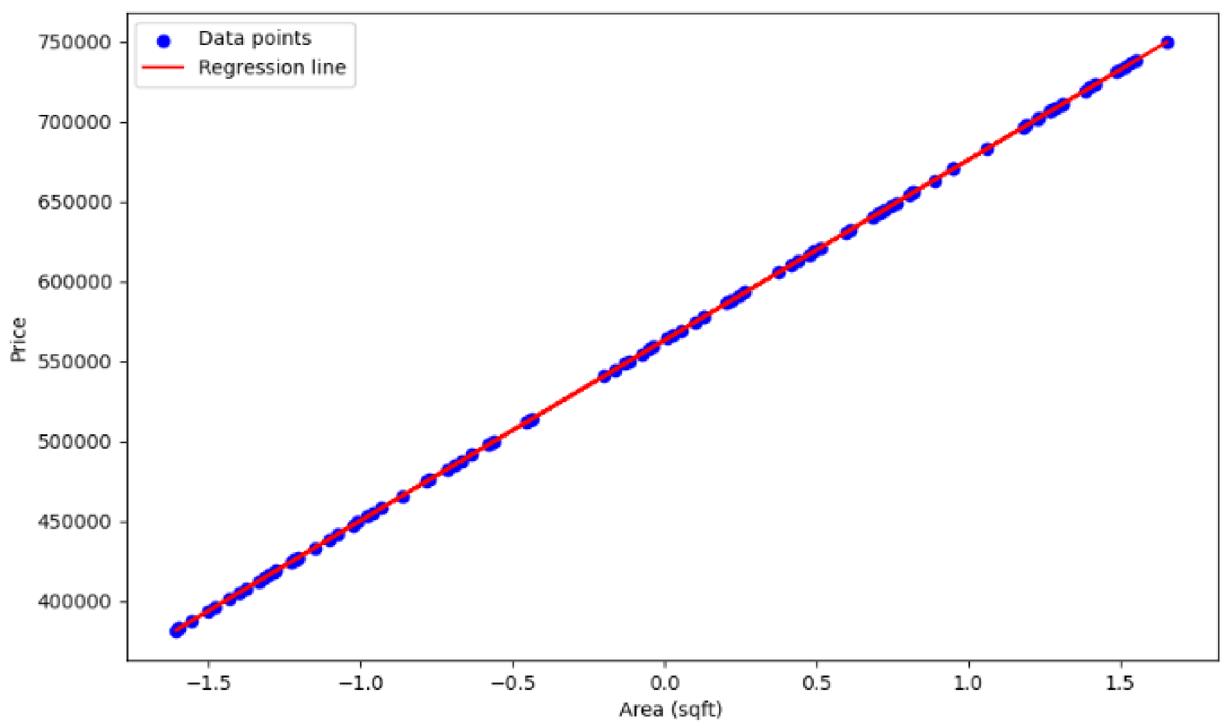


Figure 7

Code and evaluation results:

In [55]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [56]:
```
# Step 1: Load the dataset
Hw1_data = pd.read_csv('HW1_house_data(1).csv')
```

In [57]:
```
Hw1_data.describe()
```

Out[57]:

	area	price
count	100.00000	100.000000
mean	2247.97960	562990.340700
std	453.86445	113464.467239
min	1522.16000	381657.070000
25%	1812.84750	454164.525000
50%	2284.15000	571887.305000
75%	2617.72750	655407.227500
max	2995.40000	749686.490000

In [58]:
```
Hw1_data.head()
```

Out[58]:

	area	price
0	2231.88	558852.17
1	2524.92	632260.29
2	1527.65	382994.25
3	1986.09	497607.86
4	2920.16	730870.84

In [59]:
```
Hw1_data.tail()
```

Out[59]:

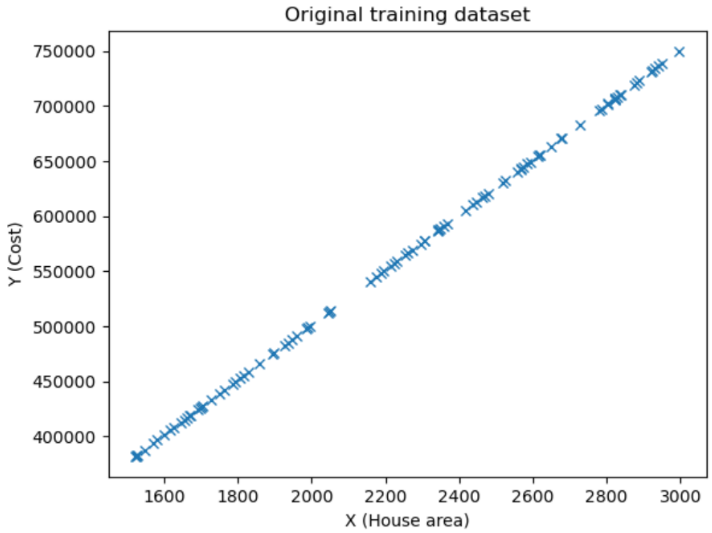
	area	price
95	2345.42	587342.25
96	1701.95	426321.81
97	2880.39	721159.46
98	2273.60	569238.36
99	2923.85	732054.33

In [60]:
```
Hw1_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   area    100 non-null    float64
 1   price   100 non-null    float64
dtypes: float64(2)
memory usage: 1.7 KB
```

In [61]:
```
X = Hw1_data['area'].values
y = Hw1_data['price'].values
```

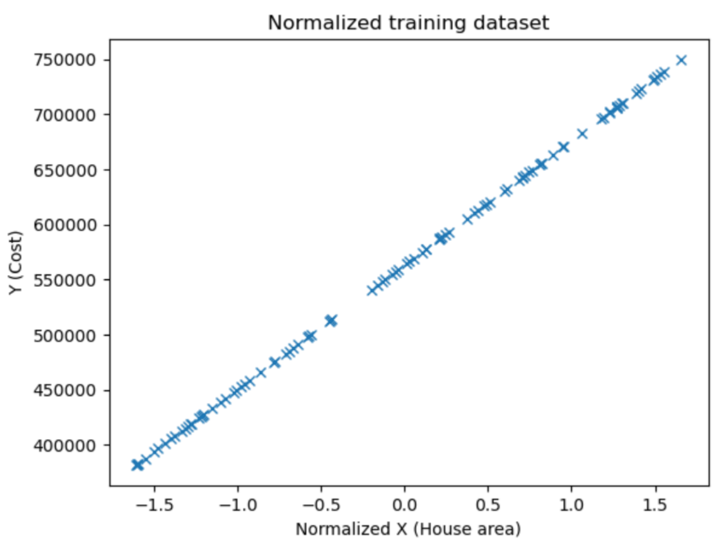
In [62]:
```
plt.plot(X, y, 'x')
plt.title("Original training dataset")
plt.xlabel("X (House area)")
plt.ylabel("Y (Cost)")
plt.show()
```



In [63]:
```
# Normalizing x
```

In [63]:
```
# Normalizing x
Xnormalized = (X - X.mean()) / X.std()
```

In [64]:
```
plt.plot(Xnormalized, y, 'x')  # Plot the normalized data
plt.title("Normalized training dataset")
plt.xlabel("Normalized X (House area)")
plt.ylabel("Y (Cost)")
plt.show()
```



In [65]:
```
LearningRate = 0.01
```

In [65]:
```python
LearningRate = 0.01
iterations = 1000
theta0 = 0
theta1 = 0
m = len(y)
```

In [66]:
```python
for _ in range(iterations):
    yPred = theta0 + theta1 * Xnormalized
    gradient_theta0 = (1 / m) * np.sum(yPred - y)
    gradient_theta1 = (1 / m) * np.sum((yPred - y) * Xnormalized)
    theta0 -= LearningRate * gradient_theta0
    theta1 -= LearningRate * gradient_theta1
```

In [67]:
```python
print("Gradient Descent Theta0:", theta0)
print("Gradient Descent Theta1:", theta1)
```

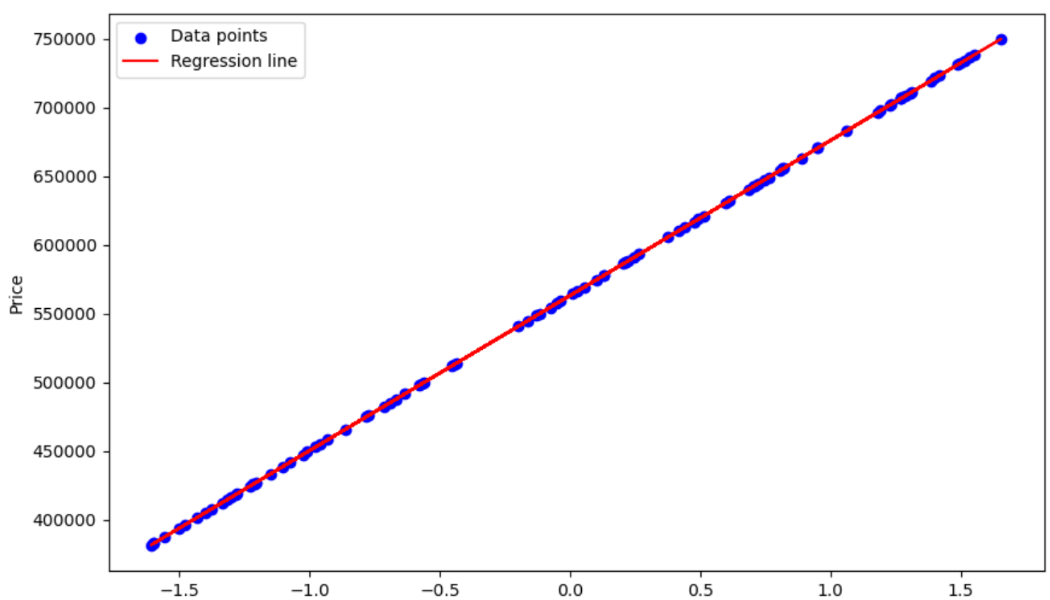
```
Gradient Descent Theta0: 562966.0357047117
Gradient Descent Theta1: 112890.79348255106
```

In [68]:
```python
areaPredict = 2000
normalizedArea = (areaPredict - X.mean()) / X.std()
ypred = theta0 + theta1 * normalizedArea
print("Gradient Descent prediction: " , ypred)
```

```
Gradient Descent prediction:  500974.7396107308
```

In [69]:
```python
# Step 8: Plot predicted vs. actual values
plt.figure(figsize=(10, 6))
plt.scatter(Xnormalized, y, marker='o', color='blue', label='Data points')
plt.plot(Xnormalized, theta0 + theta1 * Xnormalized, color='red', label='Regression line')
plt.xlabel('Area (sqft)')
plt.ylabel('Price')
plt.legend()
plt.show()
```

In [69]:
```python
# Step 8: Plot predicted vs. actual values
plt.figure(figsize=(10, 6))
plt.scatter(Xnormalized, y, marker='o', color='blue', label='Data points')
plt.plot(Xnormalized, theta0 + theta1 * Xnormalized, color='red', label='Regression line')
plt.xlabel('Area (sqft)')
plt.ylabel('Price')
plt.legend()
plt.show()
```

In [70]:
```
model = LinearRegression()
Xnormalized = Xnormalized.reshape(-1, 1)
```

In [71]:
```
model.fit(Xnormalized, y)
```

Out[71]:
```
▾ LinearRegression
LinearRegression()
```

In [72]:
```
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

```
Intercept: 562990.3407000001
Coefficient: [112895.66732934]
```

In [73]:
```
# Useing the trained model to make predictions
predictions = model.predict(normalizedArea.reshape(-1 , 1))

# Print the predictions
print("Linear Regression Predictions:", predictions)
```

```
Linear Regression Predictions: [500996.3682489]
```

In [74]:
```
#11
X_b = np.c_[np.ones(Xnormalized.shape[0]), Xnormalized]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
print("Theta0 from Normal Equation:", theta_best[0])
print("Theta1 from Normal Equation:", theta_best[1])
predicted_price_normal_equation = theta_best[0] + theta_best[1] * normalizedArea
print("Predicted Price for an area of 2000 using Normal Equation:",predicted_price_normal_equation)
```

```
Theta0 from Normal Equation: 562990.3407000001
Theta1 from Normal Equation: 112895.6673293369
Predicted Price for an area of 2000 using Normal Equation: 500996.36824889656
```