

# **Data and Artificial Intelligence**

## **Cyber Shujaa Program**

### **Week 7 Assignment** **Regression Model**

**Student Name:** John Brown Ouma

**Student ID:** CS-DA01-25030

## Table of Contents

Data and Artificial Intelligence .....	i
Cyber Shujaa Program.....	i
Week 7 Assignment Regression Model.....	i
Introduction .....	1
Tasks Completed .....	1
<b>1. Data loading and Exploration .....</b>	<b>1</b>
<b>2. Data Visualization.....</b>	<b>2</b>
<b>3. Data preparation.....</b>	<b>2</b>
<b>4. Model Training.....</b>	<b>3</b>
<b>5. Model Evaluation.....</b>	<b>3</b>
<b>6. Visualization of the Results.....</b>	<b>4</b>
Conclusion.....	5

## Introduction

This project demonstrates the implementation of a simple linear regression model using Python. The goal is to predict an outcome variable based on a single feature, following the standard machine learning workflow of data exploration, preparation, modelling, evaluation, and visualization.

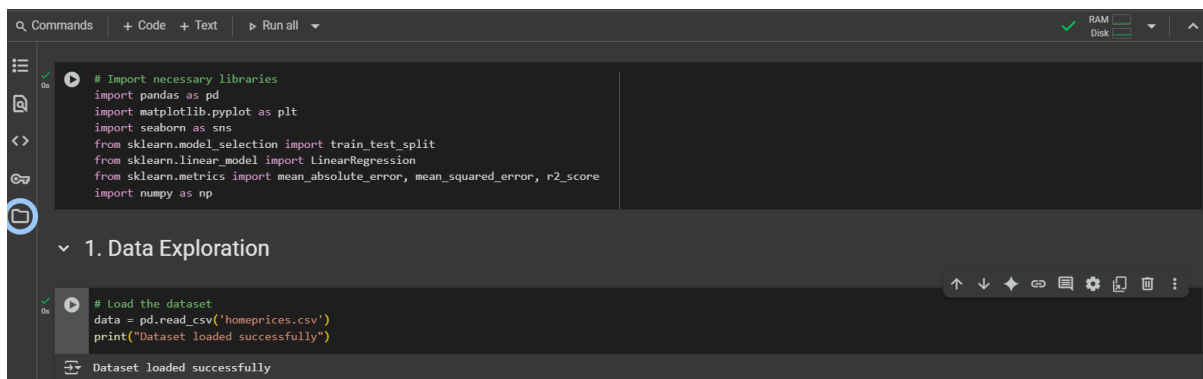
## Tasks Completed

### 1. Data loading and Exploration

**Objective:** Load the dataset and inspect its structure.

First i loaded the dataset using pandas and performed initial exploration.

Check for missing values and basic statistics.



```
# Import necessary libraries
import pandas as pd
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_absolute_error, mean_squared_error, r2_score
import numpy as np

# Load the dataset
data = pd.read_csv('homeprices.csv')
print("Dataset loaded successfully")
```

Dataset loaded successfully

Figure 1: Showing libraries imported and loading dataset



```
[5] # Display basic information about the dataset
print("Dataset Info:")
print(data.info())
print("\nFirst 5 rows:")
print(data.head())
print("\nSummary Statistics:")
print(data.describe())
```

Dataset Info:  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 5 entries, 0 to 4  
Data columns (total 2 columns):  
# Column Non-Null Count Dtype  
--- -  
0 area 5 non-null int64  
1 price 5 non-null int64  
dtypes: int64(2)  
memory usage: 212.0 bytes  
None

First 5 rows:  
 area price  
0 2600 550000  
1 3000 565000  
2 3200 610000  
3 3600 680000  
4 4000 725000

Figure 1.1: Showing basic information about the dataset



```
Summary Statistics:
```

	area	price
count	5.000000	5.000000
mean	3280.000000	626000.000000
std	540.370243	74949.983322
min	2600.000000	550000.000000
25%	3000.000000	565000.000000
50%	3200.000000	610000.000000
75%	3600.000000	680000.000000
max	4000.000000	725000.000000

```
[6] # Check for missing values
print("\nMissing Values:")
print(data.isnull().sum())
```

Missing Values:  
area 0  
price 0  
dtype: int64

Figure 1.2: Showing basic statistics and check missing values

## 2. Data Visualization

I created a scatter plot to visualize the relationship between the feature and target variable showing clear trend.

```
[7] # Visualize the data with a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(data['area'], data['price'], color='blue', alpha=0.5)
plt.xlabel('Area (sq ft)')
plt.ylabel('Price ($)')
plt.title('Scatter Plot of Area vs Price')
plt.show()
```

Figure 2: Code for Scatter plot Visualization

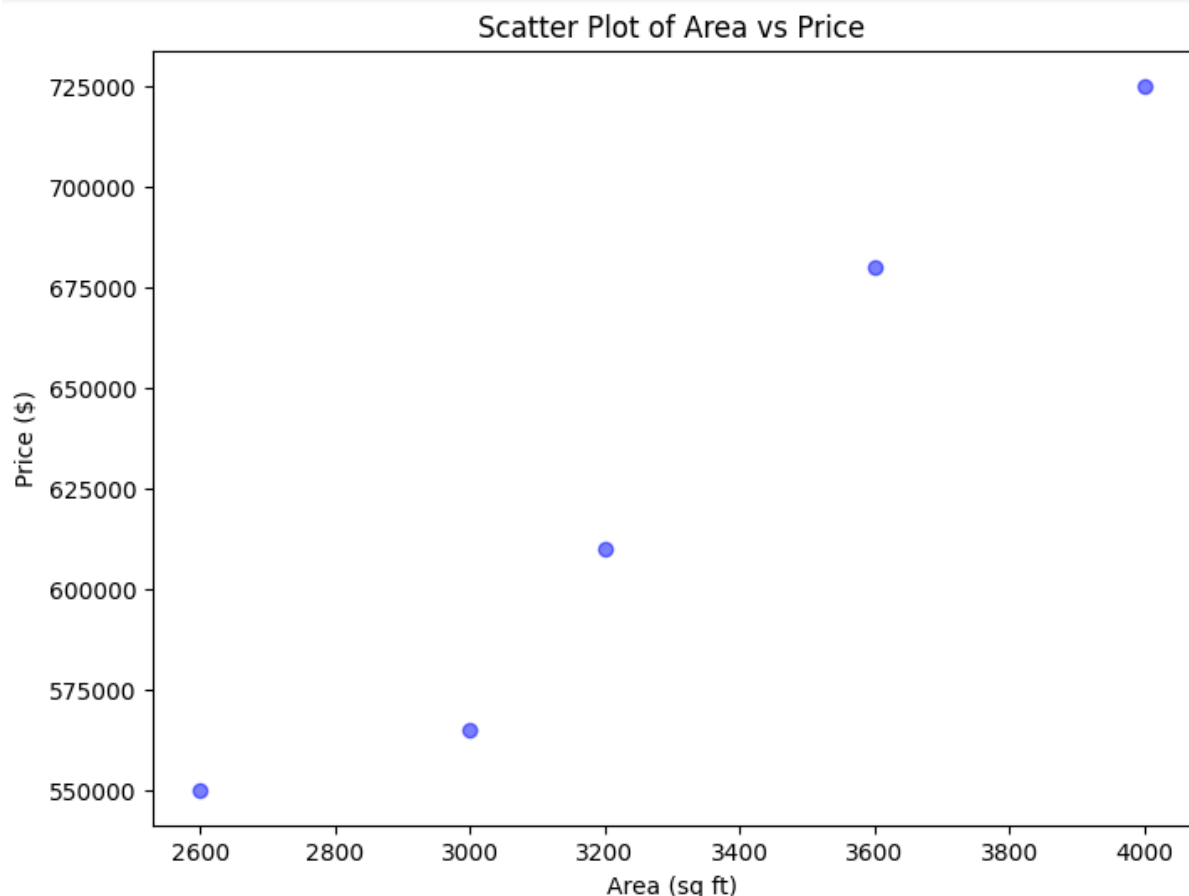


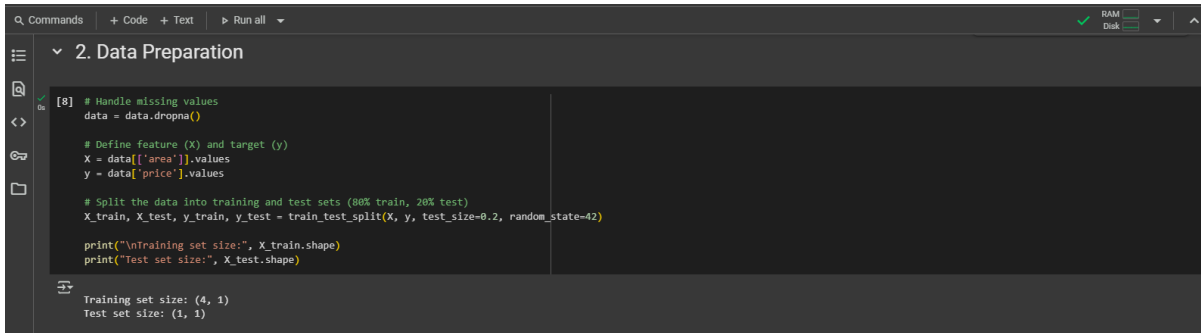
Figure 2.1: Scatter Plot visualization of prices against area

## 3. Data preparation

**Objective:** Handle missing values (if any) and split the data into training and test sets

**Steps:**

- If missing values exist, impute them (e.g., with mean/median) or drop rows. For our case here we did not have any missing values and so we did not necessarily have to handle them neither drop any rows.
- Split the data into features (x) and target (y).
- Use `train_test_split` to create training (e.g., 80%) and test (e.g., 20%) sets.



```
[8] # Handle missing values
data = data.dropna()

# Define feature (X) and target (y)
X = data[['area']].values
y = data['price'].values

# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("\nTraining set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

Training set size: (4, 1)  
Test set size: (1, 1)

Figure 3: Showing data split

## 4. Model Training

**Objective:** Train a linear Regression model using the training data.

**Steps:**

- Initialize the Linear Regression model.
- Fit the model on the training data.



```
[9] # Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Print the model parameters
print("Slope (Coefficient):", model.coef_)
print("Intercept:", model.intercept_)
```

Slope (Coefficient): [128.27102804]  
Intercept: 211542.05607476638

Figure 4: Showing model training

## 5. Model Evaluation

**Objective:** Assess the model's performance using Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R<sup>2</sup> Score.

**Steps:**

- Predict on the test sets.
- Calculate evaluation metrics.

I evaluated the model using multiple metrics.

```
[10] # Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print evaluation results
print("Evaluation Metrics:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")

Evaluation Metrics:
Mean Absolute Error (MAE): 31355.14
Mean Squared Error (MSE): 983144816.14
Root Mean Squared Error (RMSE): 31355.14
R² Score: nan
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R² score is not well-defined with less than two samples.
warnings.warn(msg, UndefinedMetricWarning)
```

Figure 5: Showing key evaluation metrics

## 6. Visualization of the Results

**Objective:** Plot the regression line against the actual data to visualize the model's fit.

I visualized the regression line against the actual data.

**Steps:**

- Create a scatter plot of the test data.
- Overlay the regression line using the model's predictions.

```
5. Visualization

[21] # Plot the regression line with training and test data
plt.figure(figsize=(8, 6))
plt.scatter(X_train, y_train, color='blue', marker='o', label='Training Data')
plt.scatter(X_test, y_test, color='green', marker='^', label='Test Data')
plt.plot(X, model.predict(X), color='red', linewidth=2, label='Regression Line')
plt.xlabel('Area (sq ft)')
plt.ylabel('Price ($)')
plt.title('Linear Regression: Area vs Price')
plt.legend()
plt.show()
```

Figure 6: Code for scatter plot visualization

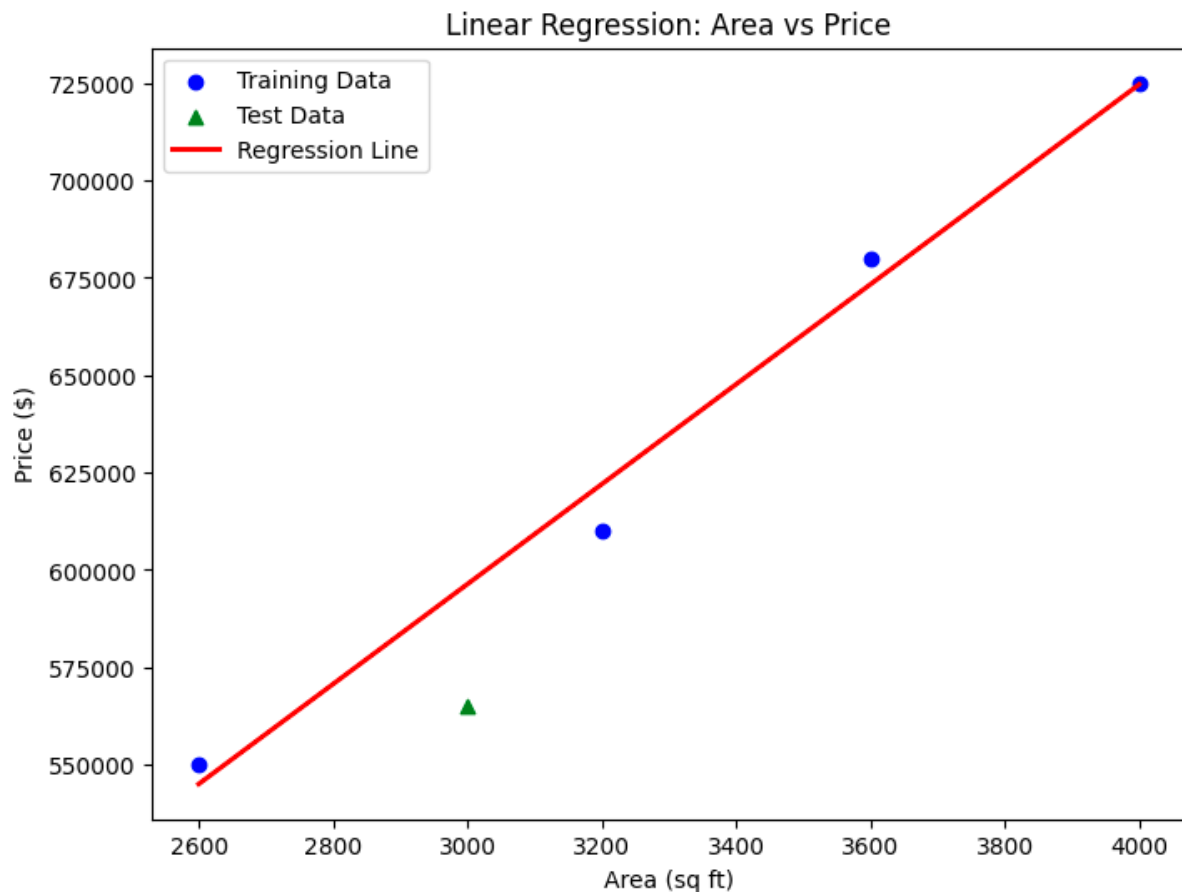


Figure 6.1: Scatter plot visualization for price against area

#### Link to Code:

[https://colab.research.google.com/drive/1k0zteNh3xNkoab\\_WdKFIdabJaUuTcg6e?usp=sharing](https://colab.research.google.com/drive/1k0zteNh3xNkoab_WdKFIdabJaUuTcg6e?usp=sharing)

## Conclusion

Through this project I learned:

- The importance of data exploration and visualization before model building.
- How to implement a simple linear regression model using scikit-learn.
- The interpretation of different evaluation metrics (MAE, MSE, RMSE, R-squared)
- The relationship between the actual data points and the regression line.
- Original train-test split: MAE = 31355.14, MSE = 983144816.14, RMSE = 31355.14,  $R^2 = \text{nan}$  (due to single test sample).