

Data and Artificial Intelligence

Cyber Shujaa Program

Week 1 Assignment

Web Scraping and Data Handling in Python

Student Name: Deborah Kwamboka Omae

Student ID: CS-DA02-25075

Table of Contents

Data and Artificial Intelligence	1
Cyber Shujaa Program.....	1
Week 1 Assignment Web Scraping and Data Handling in Python.....	1
Introduction	4
Tasks Completed	4
Part 1: simple linear regression.....	4
Step 1: exploring the data set given.....	4
Step 2: visualizing my data	5
Step 3: preparing and splitting the data for training and testing.....	6
Step 4: visualizing predictions and regression lines.....	8
Step 5: evaluating my model using key metrics	9
Evaluation.....	10
Part 2: linear regression with multiple variables	11
Step 1: exploring the dataset	11
Step 2: data preparation and cleaning	11
step 3: splitting data into training and testing and fitting the regression model.....	12
Step 4: evaluating the model	14
Evaluation.....	15
Link to code	15
Conclusion.....	16

Table of figures

Figure 1: screenshot showing the imported dataset as a pandas data frame	5
Figure 2: A scatter plot showing the area vs price	6
Figure 3: screenshot showing the code for splitting the data set. It can be seen that my testing data is 3200	7
Figure 4: screenshot showing the predicted price for the area Of 3200.....	7
Figure 5: screenshot showing the visualized predictions.....	8
Figure 6: graph showing predicted and actual prices	9
Figure 7: screenshot of model evaluation.....	10
Figure 8: screenshot showing the dataset imported as pandas DataFrame.....	11
Figure 9: screenshot showing the handled missing value	12
Figure 10: screenshot showing the evaluation of the model.....	15

Introduction

This week's assignment was to develop hands-on experience in building a **Linear Regression model** using Python. I explored a dataset, trained and tested my model, and visualized my results.

The purpose of the assignment was to gain hands-on practice in:

- Exploring a real-world dataset
- Preparing and splitting data for training and testing
- Building a simple linear regression model
- Evaluating the model using key metrics
- Visualizing predictions and regression lines
- Publishing your project as part of your portfolio collection

Tasks Completed

Part 1: simple linear regression

Step 1: exploring the data set given

In this step I imported the homeprices.csv data set as pandas' data frame. To get an overview of the data I ran the df. head() command.

Code

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import linear_model

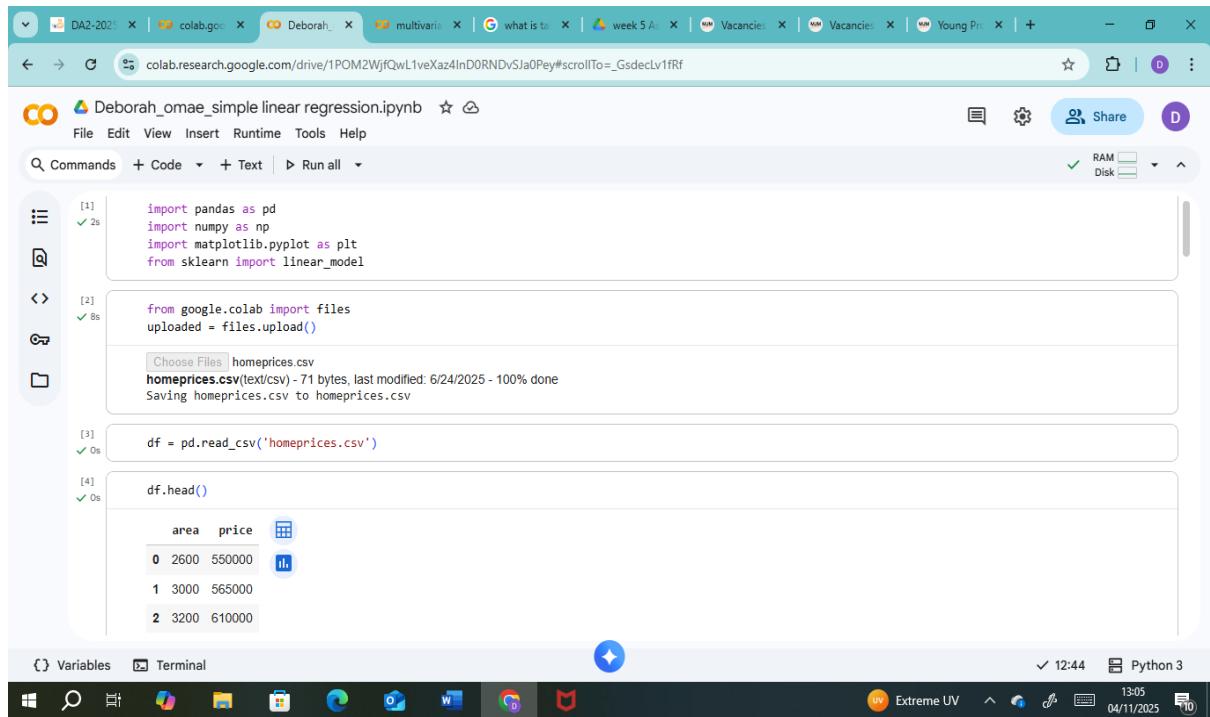
from google.colab import files

uploaded = files.upload()

df = pd.read_csv('homeprices.csv')
```

`df.head()`

Screenshot:



The screenshot shows a Google Colab notebook titled "Deborah_omae_simple linear regression.ipynb". The code cell [4] contains the command `df.head()`. The output shows the first three rows of a DataFrame:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000

Figure 1: screenshot showing the imported dataset as a pandas data frame

Step 2: visualizing my data

I visualized the distribution of my dataset through a scatter plot.

Code:

```
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
```

Screenshot:

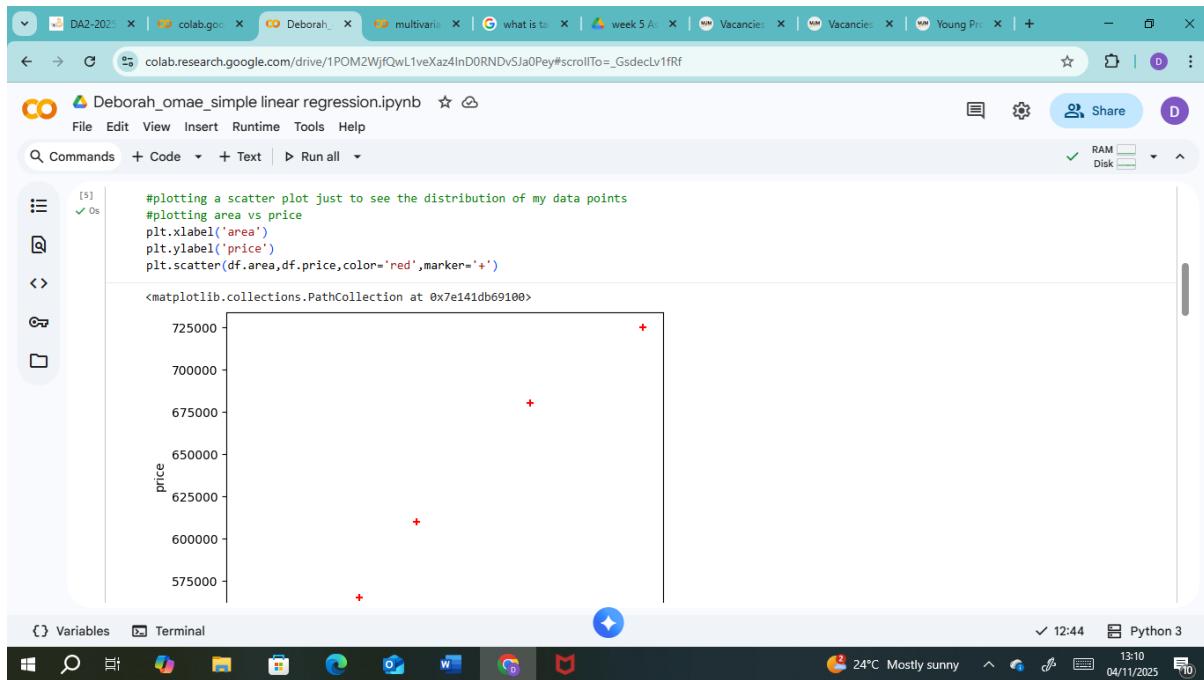


Figure 2: A scatter plot showing the area vs price

Step 3: preparing and splitting the data for training and testing

I used a linear regression model from Scikitlearn library. In order to train my regression model, I had to split my data set into 80% training and 20% testing. This is because we have to see how the model works on unseen data.

Code:

```
from sklearn.model_selection import train_test_split

# drawing the linear regression line after seeing the distribution

reg = linear_model.LinearRegression()

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(df[['area']], df.price, test_size=0.2, random_state=0)

print('X_train', X_train)

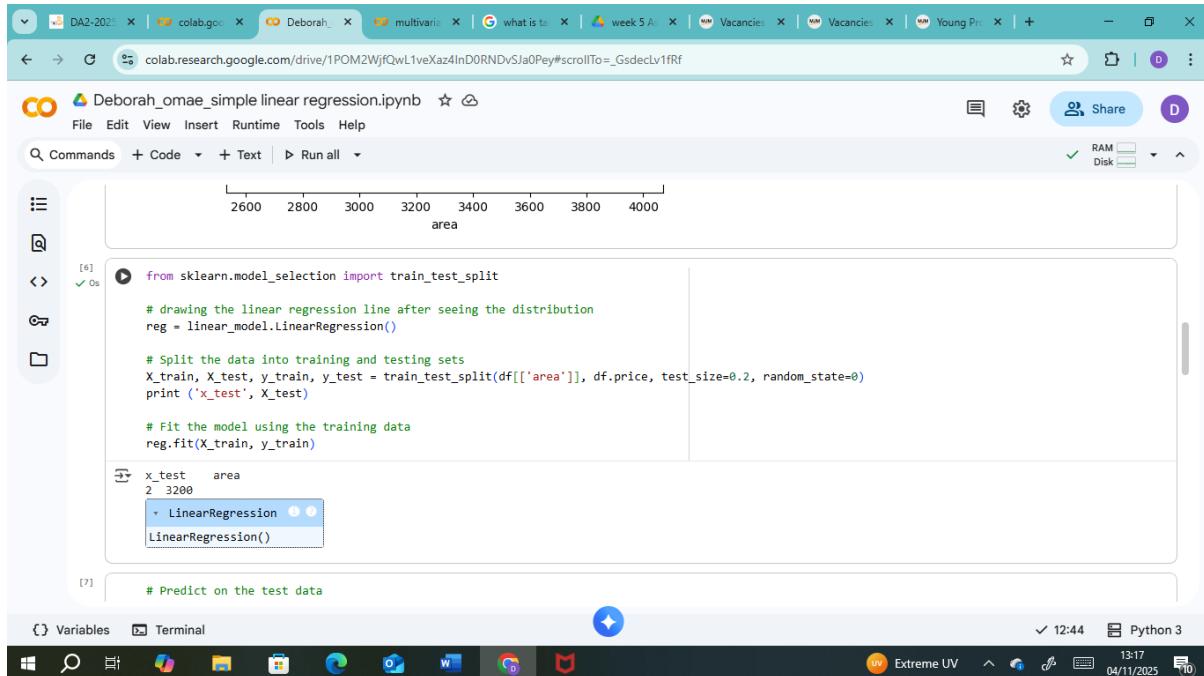
# Fit the model using the training data

reg.fit(X_train, y_train)

y_pred = reg.predict(X_test)

display(y_pred)
```

Screenshot:



The screenshot shows a Google Colab notebook titled "Deborah_omae_simple linear regression.ipynb". The code in cell [6] demonstrates how to split the data into training and testing sets using `train_test_split` from `sklearn.model_selection`. It prints the test set area as 3200. The code then fits a Linear Regression model to the training data and makes a prediction for the test data.

```

from sklearn.model_selection import train_test_split
# drawing the linear regression line after seeing the distribution
reg = LinearRegression()

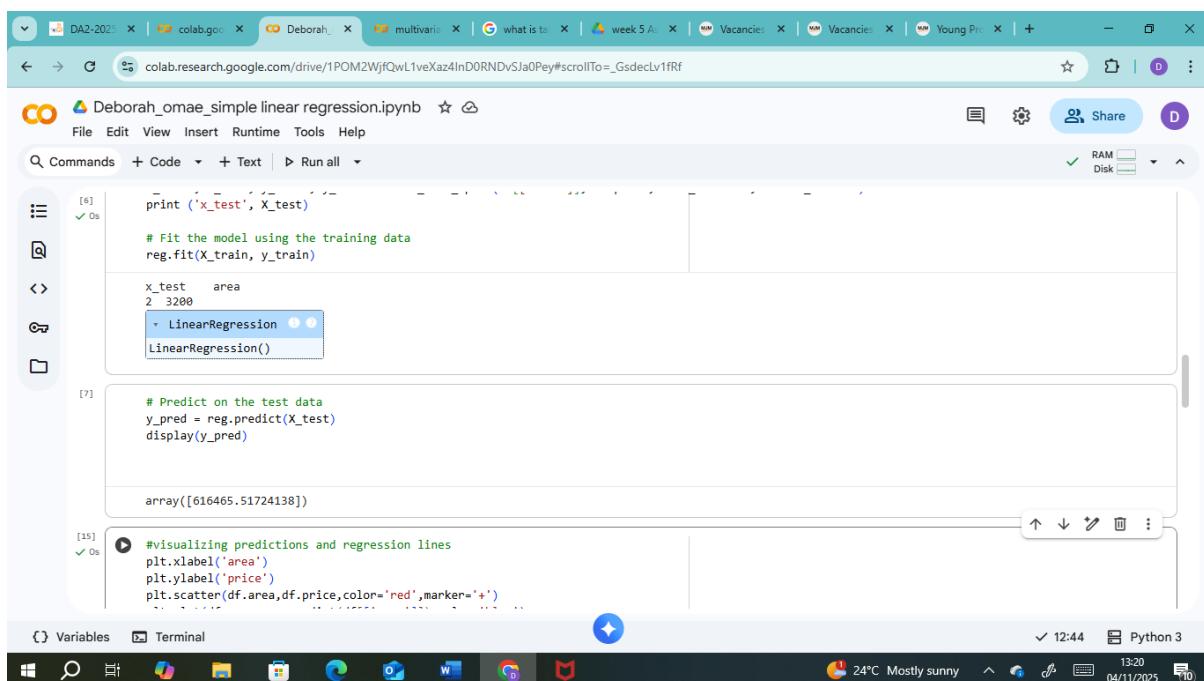
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['area']], df.price, test_size=0.2, random_state=0)
print ('x_test', X_test)

# Fit the model using the training data
reg.fit(X_train, y_train)

# Predict on the test data
y_pred = reg.predict(X_test)
display(y_pred)

```

Figure 3: screenshot showing the code for splitting the data set. It can be seen that my testing data is 3200



The screenshot shows the continuation of the Google Colab notebook. The code in cell [7] prints the test set area as 3200 and fits the model to the training data. In cell [15], it makes a prediction for the test data and displays the result. The output shows an array containing a single value: [616465.51724138]. The code in cell [16] then visualizes the predictions and regression lines.

```

print ('x_test', X_test)
# Fit the model using the training data
reg.fit(X_train, y_train)

# Predict on the test data
y_pred = reg.predict(X_test)
display(y_pred)

array([616465.51724138])

#visualizing predictions and regression lines
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
plt.plot(X_test,y_pred,color='blue', linewidth=2)

```

Figure 4: screenshot showing the predicted price for the area Of 3200

Step 4: visualizing predictions and regression lines

I visualized the predicted prices against the original prices

Code

```
#visualizing predictions and regression lines
```

```
plt.xlabel('area')
```

```
plt.ylabel('price')
```

```
plt.scatter(df.area,df.price,color='red',marker='+')
```

```
plt.plot(df.area,reg.predict(df[['area']]),color='blue')
```

Plot the predicted prices against the original areas

```
plt.scatter(df.area, reg.predict(df[['area']]), color='green', marker='x', label='Predicted Price')
```

```
plt.legend()
```

Screenshot:

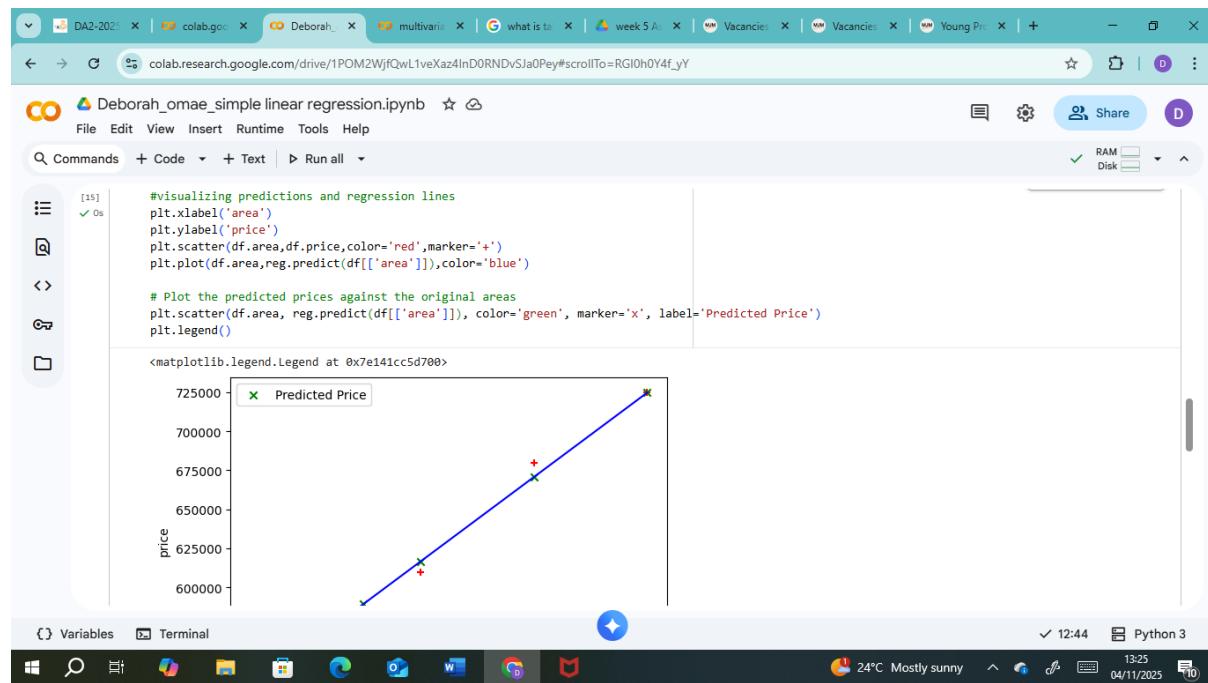


Figure 5: screenshot showing the visualized predictions

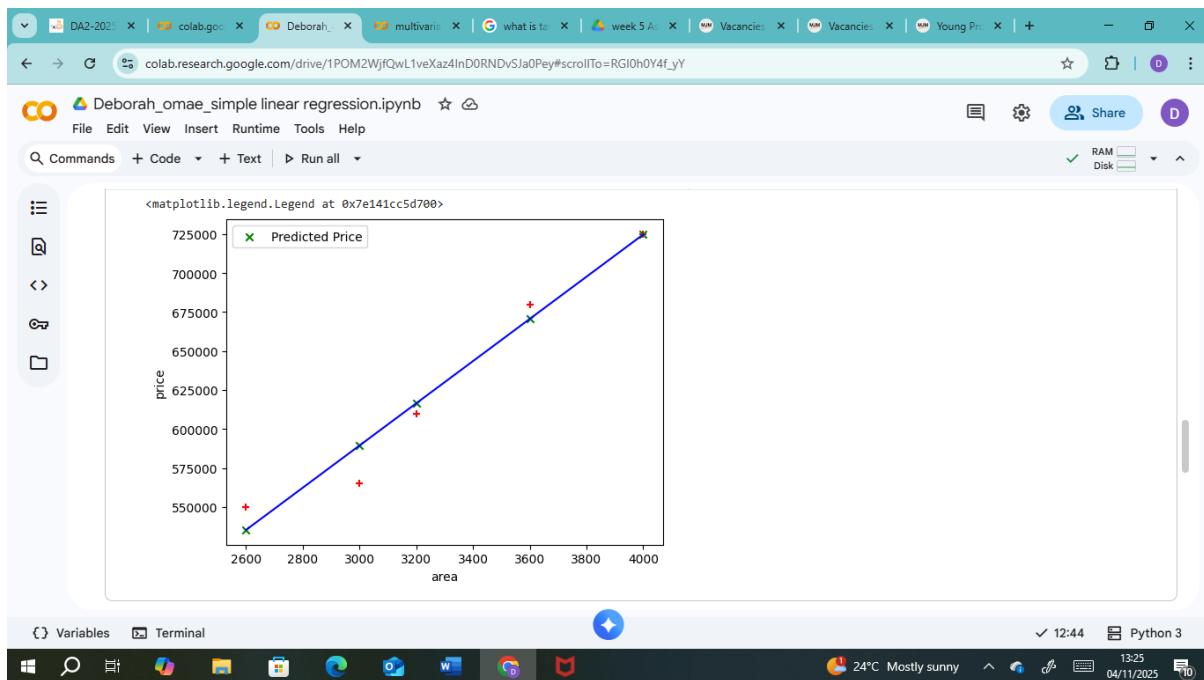


Figure 6: graph showing predicted and actual prices

Step 5: evaluating my model using key metrics

In this step I evaluated my model using regression metrics such as mean absolute error, mean squared error, root mean squared error, and R^2

MAE is the **average of the absolute differences** between actual and predicted values. It measures how far, on average, the predictions are from the real values.

MSE is the average of squared differences between actual and predicted values. By squaring the errors, it gives more weight to larger mistakes, making it useful when large errors are especially undesirable.

RMSE is the square root of the MSE, converting the result back to the same unit as the target variable. It represents the standard deviation of prediction errors. In other words, how much the predictions typically deviate from the actual values.

R^2 (coefficient of determination) measures the **proportion of the variance in the dependent variable** that is explained by the independent variables in the model. It compares how well your model performs against a simple baseline model that always predicts the mean.

Code

```
#evaluating this model using the regression metrics; MSE, RMSE, AME, R2
```

```

from sklearn.metrics import mean_squared_error, mean_absolute_error, root_mean_squared_error,
r2_score

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = root_mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print('Mean Absolute Error:', mae)

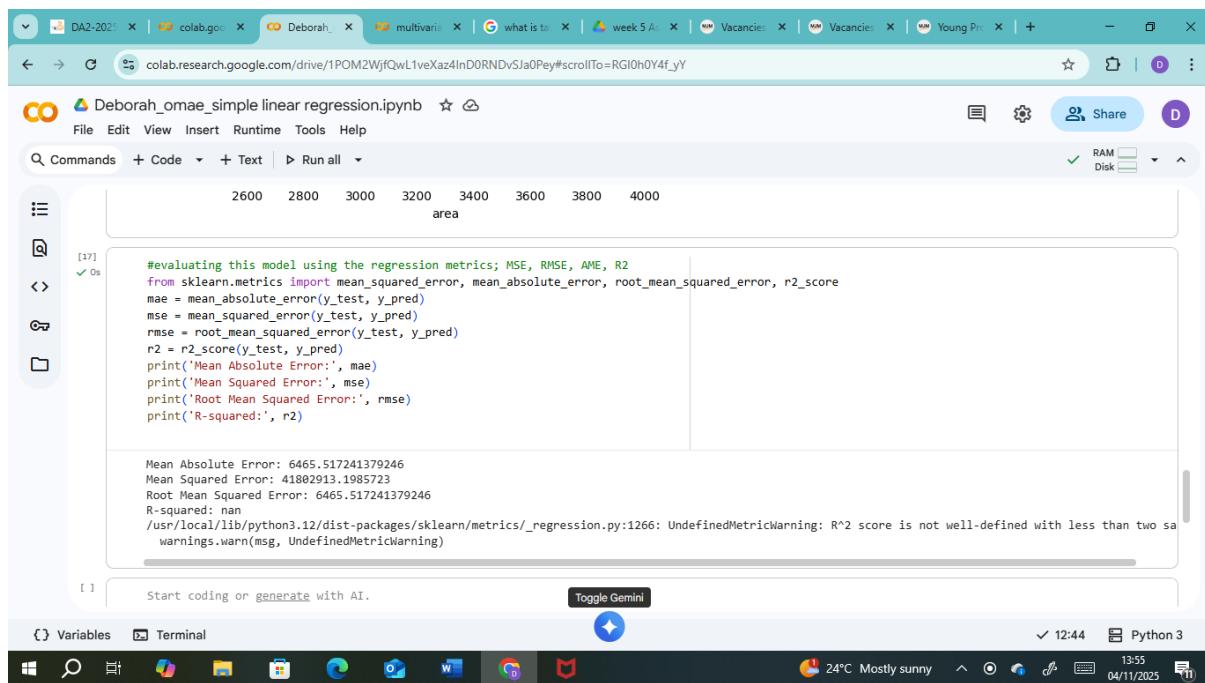
print('Mean Squared Error:', mse)

print('Root Mean Squared Error:', rmse)

print('R-squared:', r2)

```

Screenshot:



The screenshot shows a Google Colab notebook titled "Deborah_omae_simple linear regression.ipynb". The code cell contains the same Python script as above. The output pane shows the results of the calculations:

```

#evaluating this model using the regression metrics; MSE, RMSE, AME, R2
from sklearn.metrics import mean_squared_error, mean_absolute_error, root_mean_squared_error, r2_score
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
print('R-squared:', r2)

```

```

Mean Absolute Error: 6465.517241379246
Mean Squared Error: 41802913.1985723
Root Mean Squared Error: 6465.517241379246
R-squared: nan
/usr/local/lib/python3.12/dist-packages/scikit-learn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two sa
  warnings.warn(msg, UndefinedMetricWarning)

```

Figure 7: screenshot of model evaluation

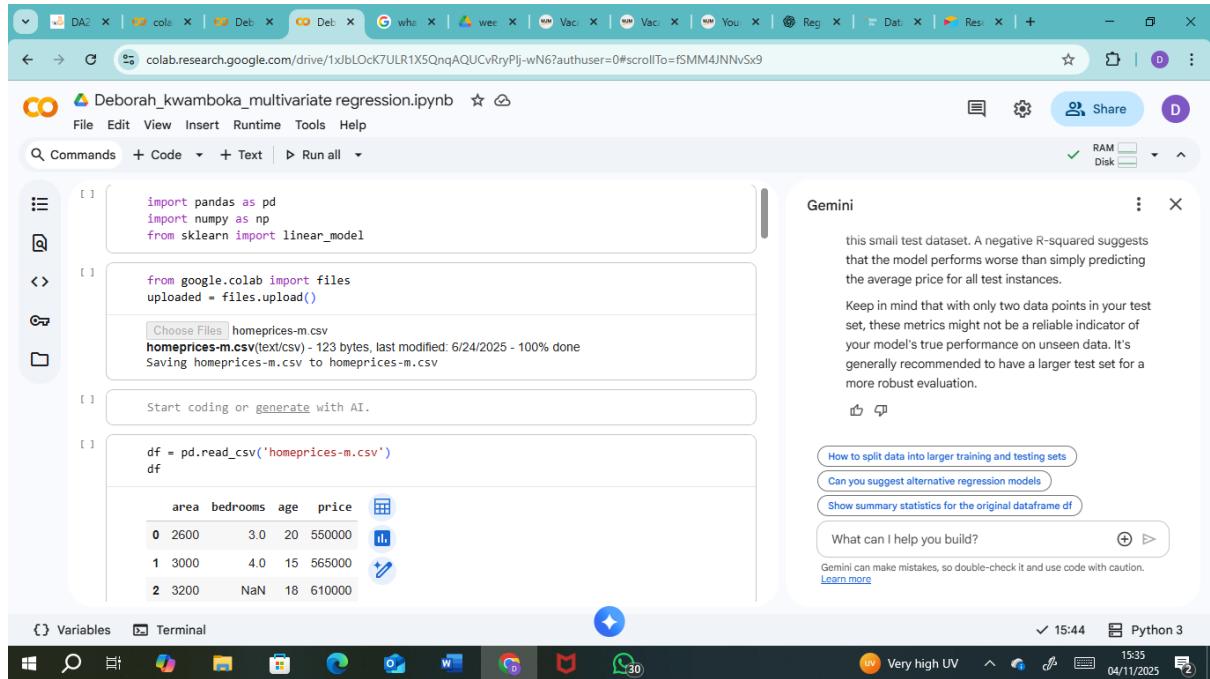
Evaluation

The model has a mean absolute error and root mean square error of approximately 6465. This shows the error margin between our predicted value and actual value. Therefore, the predictions are reliable.

Part 2: linear regression with multiple variables

Step 1: exploring the dataset

I imported the homeprices-m.csv into a pandas' data frame and got an overview of the data.



The screenshot shows a Google Colab notebook titled "Deborah_kwamboka_multivariate regression.ipynb". The code cell imports pandas, numpy, and linear_model from sklearn. It then uses files.upload() to upload a CSV file named "homeprices-m.csv". A message indicates the file was uploaded successfully. Below the code, a DataFrame df is created by reading the CSV file. The DataFrame has four columns: area, bedrooms, age, and price. The preview shows three rows of data:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000

Figure 8: screenshot showing the dataset imported as pandas DataFrame

Step 2: data preparation and cleaning

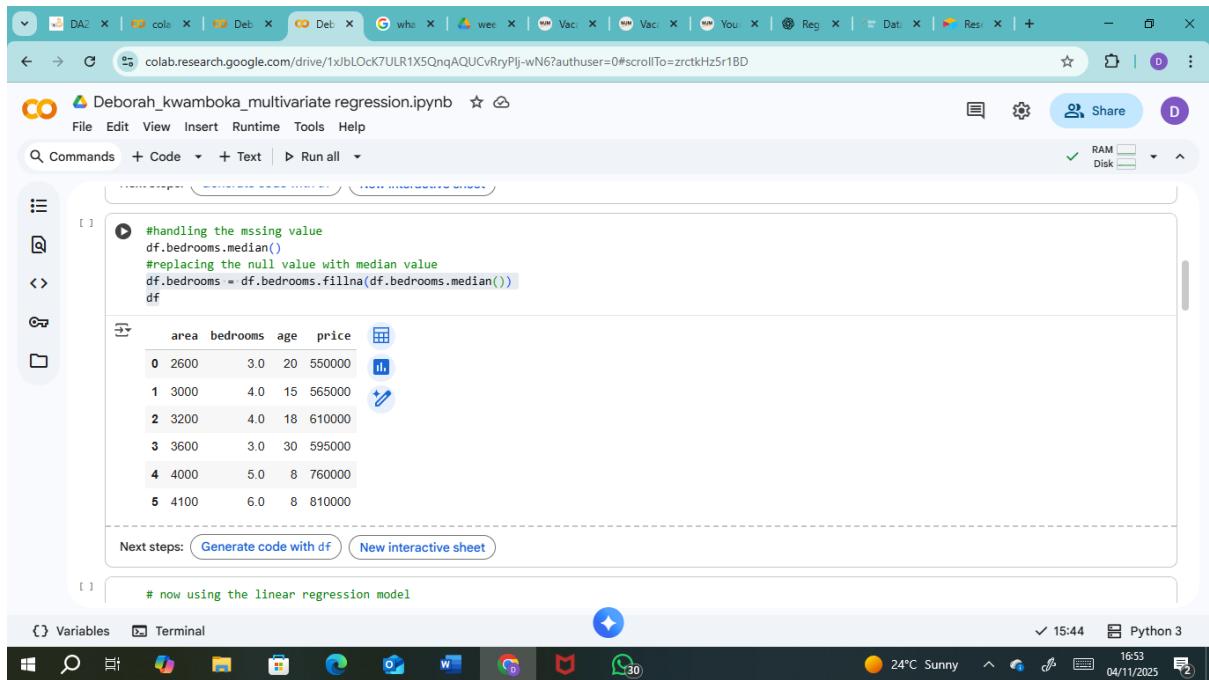
My data set had one missing value in the bedroom's column. I imputed the missing value using the median of the bedroom's column.

The code:

```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
```

df

screenshot:



The screenshot shows a Google Colab notebook titled "Deborah_kwamboka_multivariate regression.ipynb". The code cell contains:

```
#handling the missing value
df.bedrooms.median()
#replacing the null value with median value
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
```

Below the code is a preview of a DataFrame:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	4.0	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

Next steps: [Generate code with df](#) [New interactive sheet](#)

The code cell below contains:

```
# now using the linear regression model
```

The Colab interface includes a sidebar with "Variables" and "Terminal" tabs, and a status bar at the bottom showing "Python 3", "15:44", "24°C Sunny", "16:53", and "04/11/2025".

Figure 9: screenshot showing the handled missing value

step 3: splitting data into training and testing and fitting the regression model

In this step I had to split my data homeprices-m into training and testing data.

#80% training and 20% testing. I manually selected the data points for training and testing.

Code:

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error, root_mean_squared_error,
r2_score

p = df[['area', 'bedrooms', 'age']]

t = df['price']

print(p)

print("")

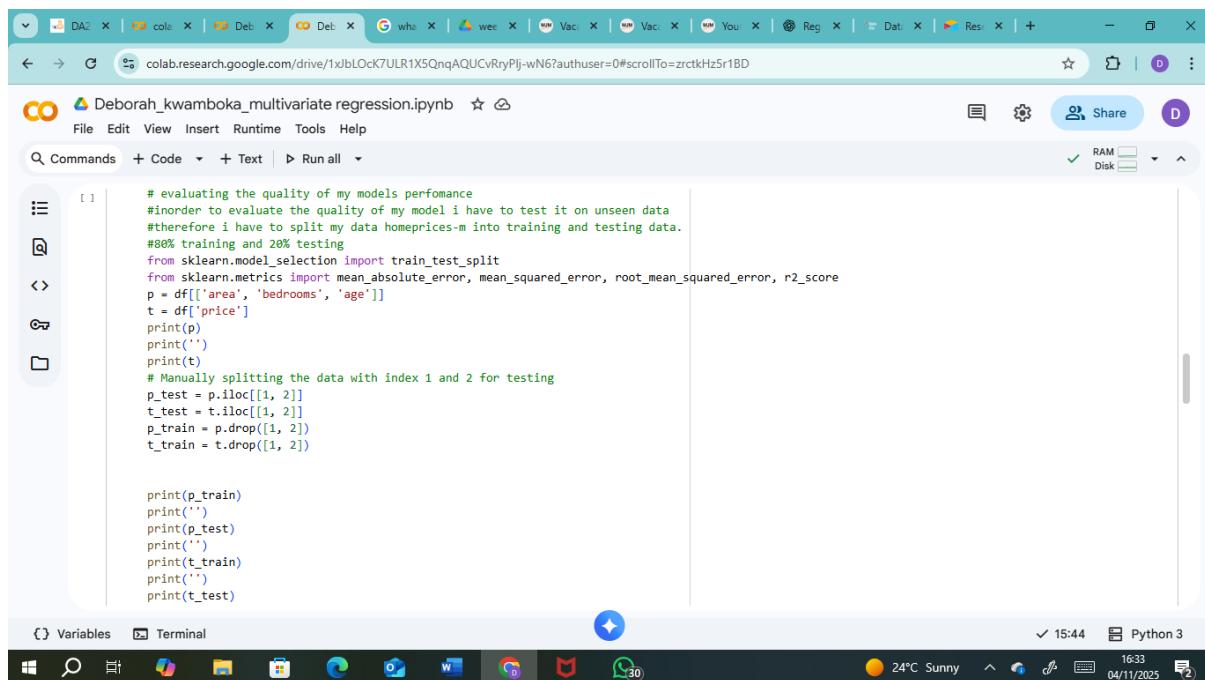
print(t)

# Manually splitting the data with index 1 and 2 for testing

p_test = p.iloc[[1, 2]]
```

```
t_test = t.iloc[[1, 2]]  
  
p_train = p.drop([1, 2])  
  
t_train = t.drop([1, 2])  
  
  
  
print(p_train)  
  
print()  
  
print(p_test)  
  
print()  
  
print(t_train)  
  
print()  
  
print(t_test)  
  
model = linear_model.LinearRegression()  
  
model.fit(p_train, t_train)
```

screenshot:

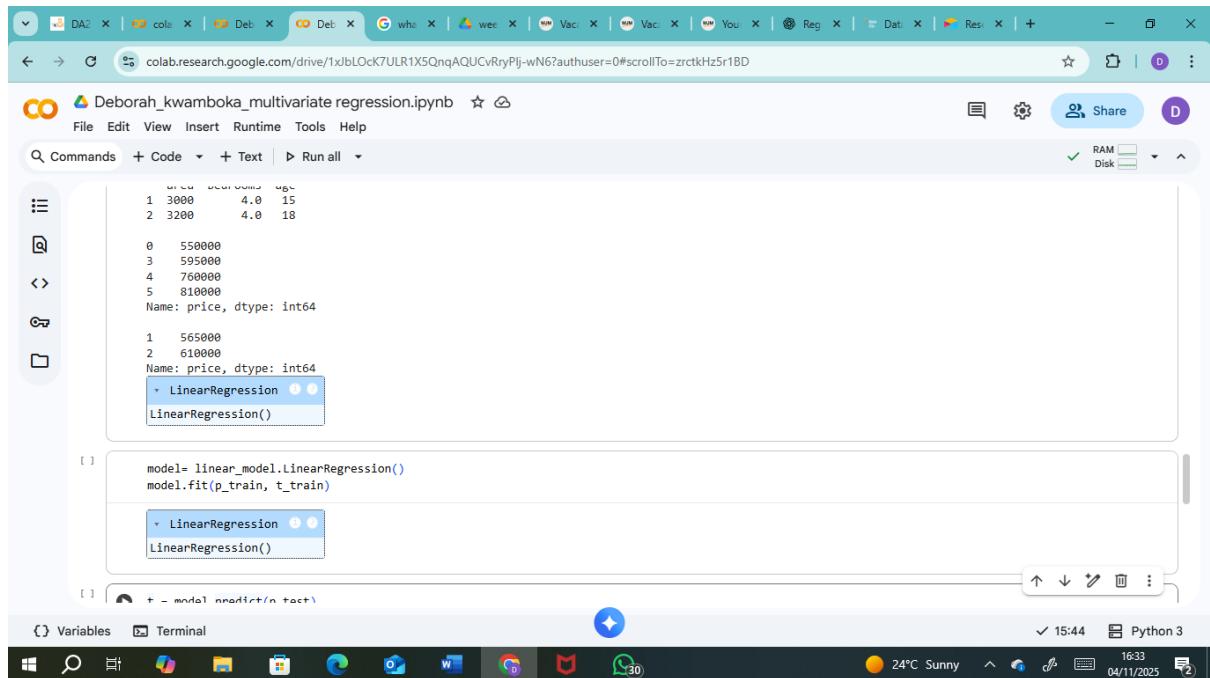


The screenshot shows a Google Colab notebook titled "Deborah_kwamboka_multivariate regression.ipynb". The code cell contains the following Python script:

```
# evaluating the quality of my models performance
#inorder to evaluate the quality of my model i have to test it on unseen data
#therefore i have to split my data homeprices-m into training and testing data.
#80% training and 20% testing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, root_mean_squared_error, r2_score
p = df[['area', 'bedrooms', 'age']]
t = df['price']
print(p)
print()
print(t)
# Manually splitting the data with index 1 and 2 for testing
p_test = p.iloc[[1, 2]]
t_test = t.iloc[[1, 2]]
p_train = p.drop([1, 2])
t_train = t.drop([1, 2])

print(p_train)
print()
print(p_test)
print()
print(t_train)
print()
print(t_test)
```

The notebook interface includes tabs for "Variables" and "Terminal". The status bar at the bottom shows "15:44", "Python 3", "24°C Sunny", "16:33", and the date "04/11/2025".



The screenshot shows a Google Colab notebook titled "Deborah_kwamboka_multivariate regression.ipynb". The code cell contains the following Python code:

```

1 3000    4.0    15
2 3200    4.0    18

0  550000
3  595000
4  760000
5  810000
Name: price, dtype: int64

1  565000
2  610000
Name: price, dtype: int64
+ LinearRegression
LinearRegression()

```

Below the code cell, the output shows the creation of a LinearRegression model object.

Step 4: evaluating the model

In this step I evaluated our regression model through regression metrics such as mean square error and mean absolute error.

Code:

```

# getting the regression metrics

mae = mean_absolute_error(t_test, t)

mse = mean_squared_error(t_test, t)

rmse = root_mean_squared_error(t_test, t)

r2 = r2_score(t_test, t)

print(f"Mean Absolute Error (MAE): {mae}")

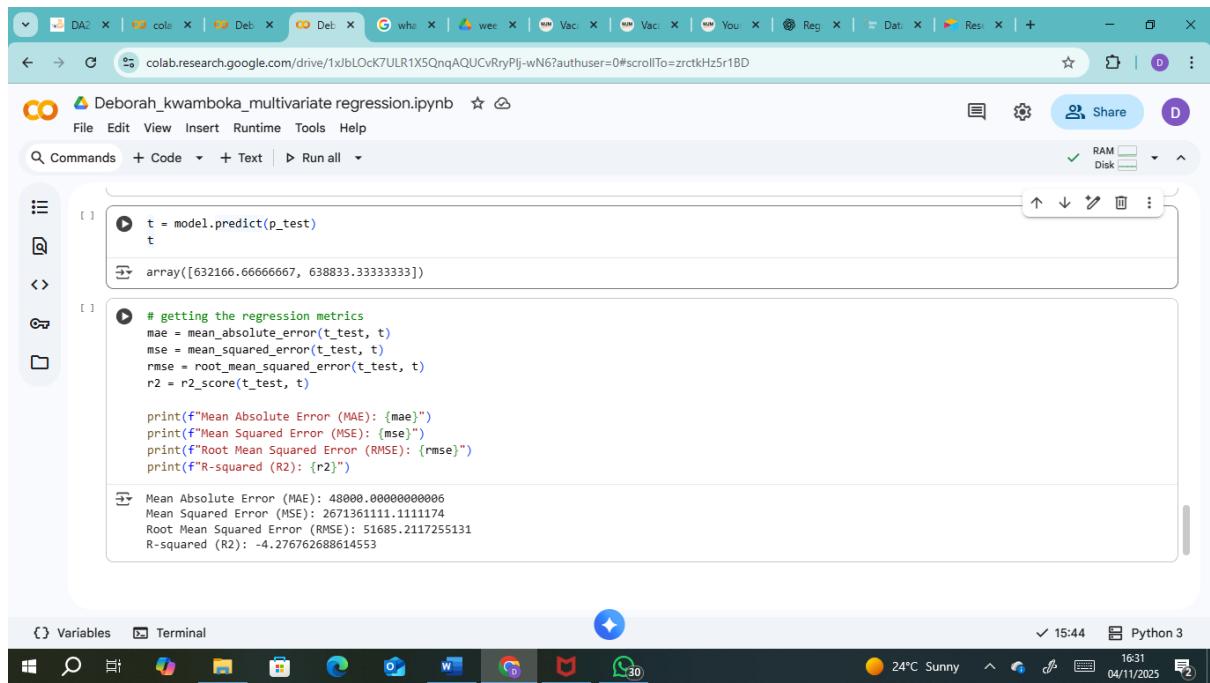
print(f"Mean Squared Error (MSE): {mse}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R-squared (R2): {r2}")

```

screenshot:



```

Deborah_kwamboka_multivariate_regression.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[ ] t = model.predict(p_test)
t
array([632166.66666667, 638833.33333333])

[ ] # getting the regression metrics
mae = mean_absolute_error(t_test, t)
mse = mean_squared_error(t_test, t)
rmse = root_mean_squared_error(t_test, t)
r2 = r2_score(t_test, t)

print("Mean Absolute Error (MAE): {mae}")
print("Mean Squared Error (MSE): {mse}")
print("Root Mean Squared Error (RMSE): {rmse}")
print("R-squared (R2): {r2}")

Mean Absolute Error (MAE): 48000.0000000006
Mean Squared Error (MSE): 2671361111.1111174
Root Mean Squared Error (RMSE): 51685.2117255131
R-squared (R2): -4.276762688614553

```

Variables Terminal ✓ 15:44 Python 3
 24°C Sunny 16:31 04/11/2025

Figure 10: screenshot showing the evaluation of the model

Evaluation

The data set has a mean absolute error of approximately 48000 and R^2 is -4. This shows that our model is making poor predictions. Perhaps it needs training with a larger dataset.

Link to code

Link to Code:

Link 1(for simple linear regression):

<https://colab.research.google.com/drive/1POM2WjfQwL1veXaz4InD0RNDvSJa0Pey?usp=sharing>

Link 2(for linear regression with multiple variables):

<https://colab.research.google.com/drive/1xJbLOcK7ULR1X5QnqAQUCVrryPlj-wN6?usp=sharing>

Conclusion

In this task, a linear regression model was developed to predict house prices based on key features such as area (in square feet), number of bedrooms, and age of the house. The model was trained and evaluated using standard regression performance metrics including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the R² score. I can confidently say that a linear regression model is a good fit to make predictions and is more reliable with a larger dataset.