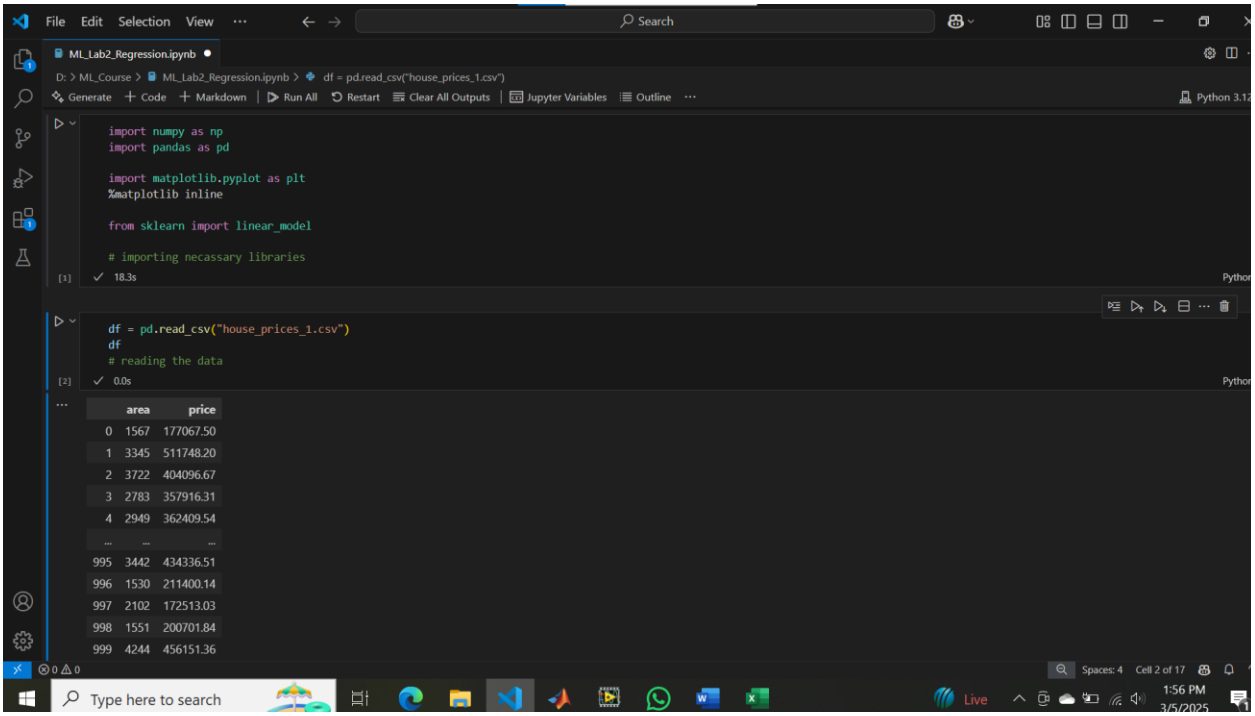
ML_Lab2_Regression

Student Name: Abdul Malik

Student Number: 2211011098

Objective:

Make a Linear Regression model that predicts the house price given a specific feature i.e. area in square feet.



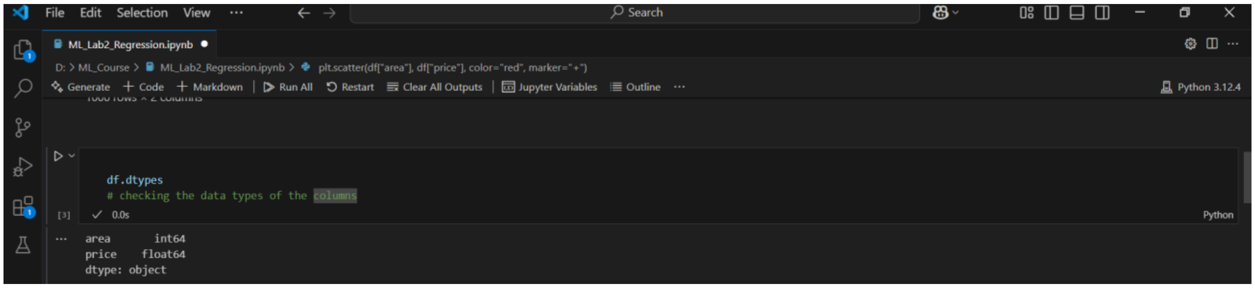
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import linear_model

# importing necassary libraries
```

```
df = pd.read_csv("house_prices_1.csv")
df
# reading the data
```

```
     area      price
0    1567  177067.50
1    3345  511748.20
2    3722  404096.67
3    2783  357916.31
4    2949  362409.54
...   ...        ...
995  3442  434336.51
996  1530  211400.14
997  2102  172513.03
998  1551  200701.84
999  4244  456151.36
```

The above two cells of jupyter notebook show the necessary libraries which are firstly the famous ones, numpy, panda, matplotlib (for plotting the data) and scikit learn (famous for having packages for linear regression, logistic regression, classification and clustering etc) (scikit-learn: machine learning in Python — scikit-learn 1.6.1 documentation).

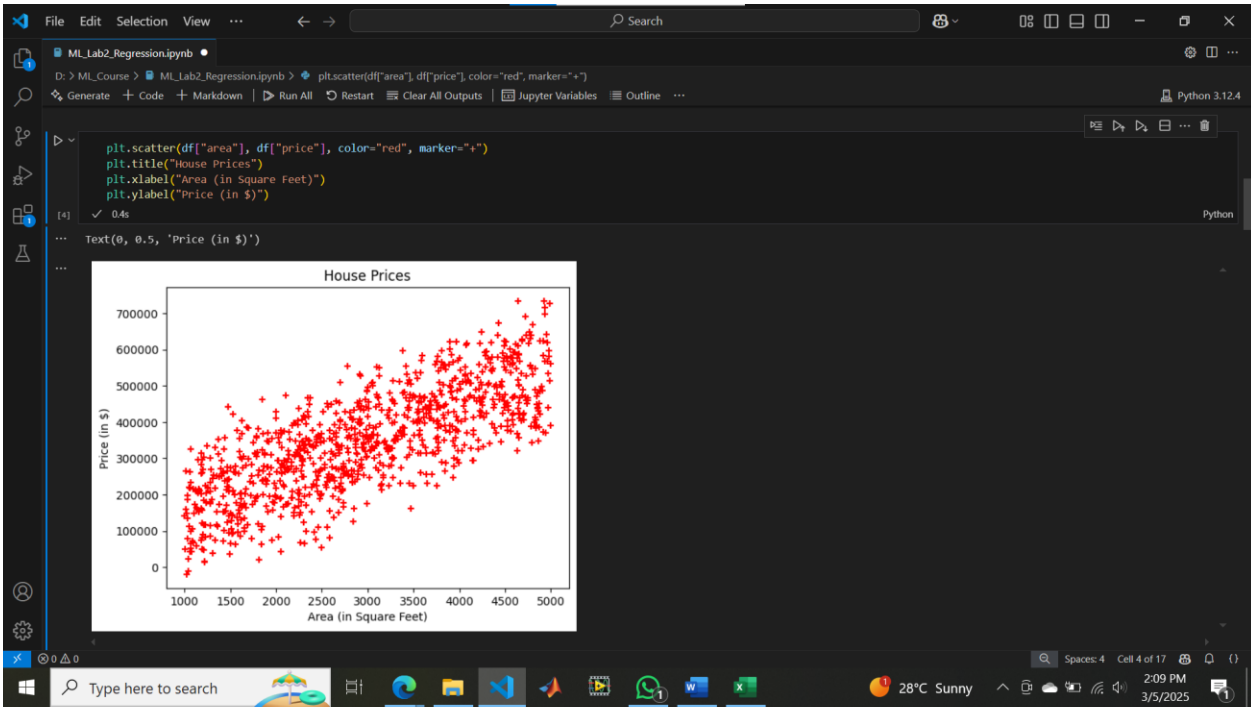
Panda's **read_csv** reads the data from the saved **dataset house_prices_1.csv** and then prints the elements of csv file.


File Edit Selection View ···
Search
ML_Lab2_Regression.ipynb
D: > ML_Course > ML_Lab2_Regression.ipynb > plt.scatter(df["area"], df["price"], color="red", marker="+")
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ···
Python 3.12.4
[3]
0.0s
Python

```
df.dtypes
# checking the data types of the columns
```

```
area        int64
price     float64
dtype: object
```

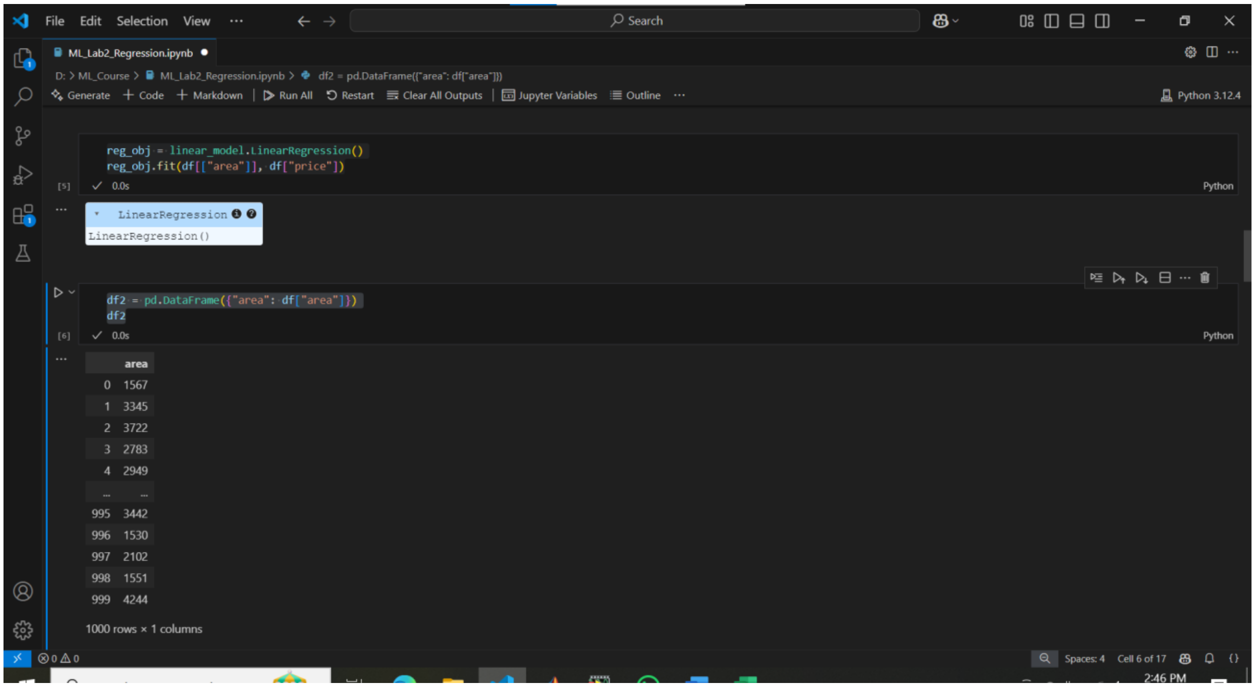
Now the datatypes of the elements inside csv (df variable defined) are shown using **dtypes** from Pandas.



```
plt.scatter(df["area"], df["price"], color="red", marker="+")
plt.title("House Prices")
plt.xlabel("Area (in Square Feet)")
plt.ylabel("Price (in $)")
```

```
Text(0, 0.5, 'Price (in $)')
```

Here we scatter plot the elements of datasets which are area (square feet) and price ($) using **plt** function from **matplotlib.**


File Edit Selection View ···
Search
ML_Lab2_Regression.ipynb
D: > ML_Course > ML_Lab2_Regression.ipynb > df2 = pd.DataFrame({"area": df["area"]})
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ···
Python 3.12.4
Python
Spaces: 4 Cell 6 of 17
2:46 PM

```
reg_obj = linear_model.LinearRegression()
reg_obj.fit(df[["area"]], df["price"])
```

[5] ✓ 0.0s

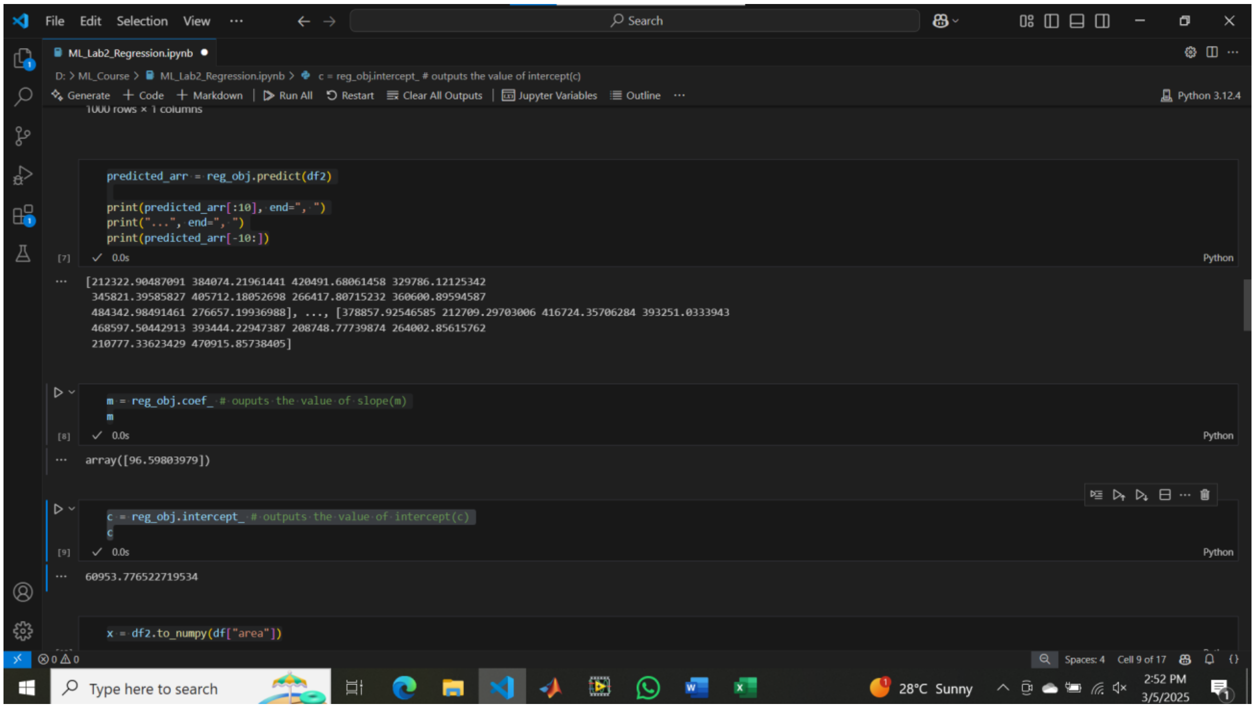
```
LinearRegression
LinearRegression()
```

```
df2 = pd.DataFrame({"area": df["area"]})
df2
```

[6] ✓ 0.0s

```
      area
  0   1567
  1   3345
  2   3722
  3   2783
  4   2949
 ...   ...
995   3442
996   1530
997   2102
998   1551
999   4244

1000 rows × 1 columns
```

Here we load the features (area in our case) to our linear regression model form scikit learn.


ML_Lab2_Regression.ipynb
D: > ML_Course > ML_Lab2_Regression.ipynb > c = reg_obj.intercept_ # outputs the value of intercept(c)
Generate + Code + Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.4
1000 rows × 1 columns
Python
28°C Sunny
2:52 PM
3/5/2025

```
predicted_arr = reg_obj.predict(df2)

print(predicted_arr[:10], end=", ")
print("...", end=", ")
print(predicted_arr[-10:])
```

[7] ✓ 0.0s

```
[212322.90487091 384074.21961441 420491.68061458 329786.12125342
 345821.39585827 405712.18052698 266417.80715232 360600.89594587
 484342.98491461 276657.19936988], ..., [378857.92546585 212709.29703006 416724.35706284 393251.0333943
 468597.50442913 393444.22947387 208748.77739874 264002.85615762
 210777.33623429 470915.85738405]
```

```
m = reg_obj.coef_ # ouputs the value of slope(m)
m
```

[8] ✓ 0.0s

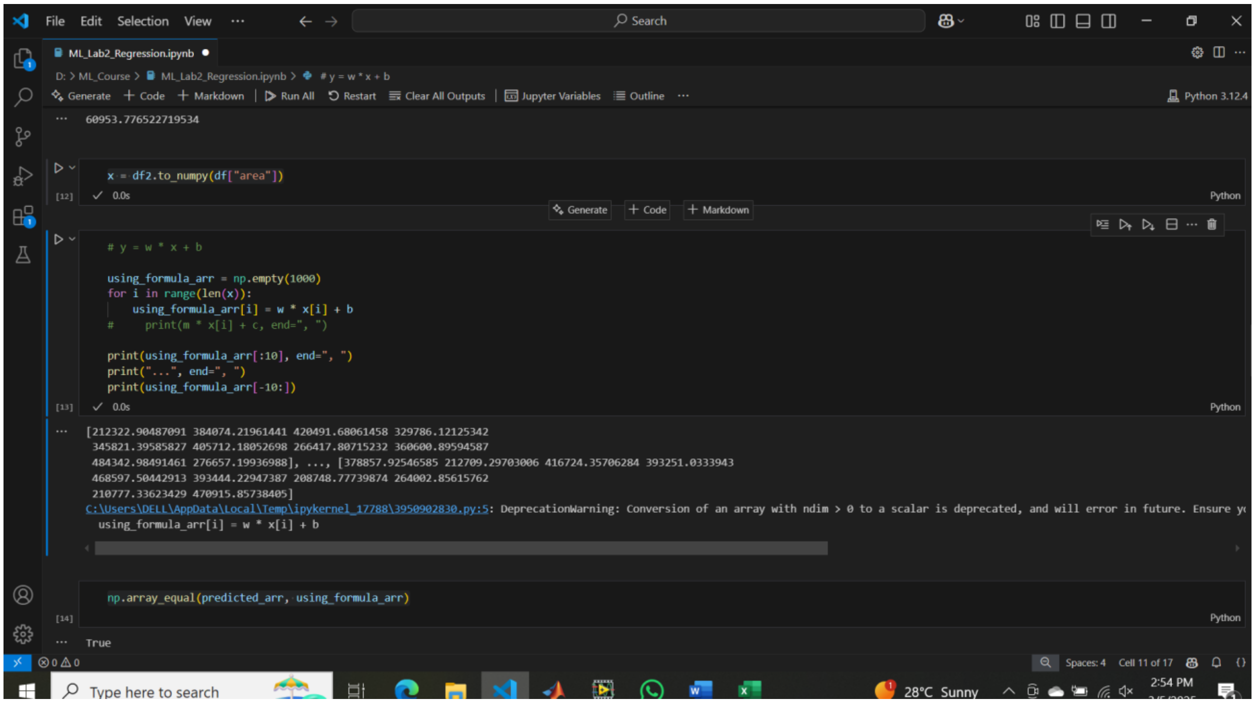
```
array([96.59803979])
```

```
c = reg_obj.intercept_ # outputs the value of intercept(c)
c
```

[9] ✓ 0.0s

```
60953.776522719534
```

```
x = df2.to_numpy(df["area"])
```


ML_Lab2_Regression.ipynb
D: > ML_Course > ML_Lab2_Regression.ipynb > # y = w * x + b
Generate + Code + Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.12.4
Python
28°C Sunny
2:54 PM

```
60953.776522719534
```

```
x = df2.to_numpy(df["area"])
```

[12] ✓ 0.0s

```
# y = w * x + b

using_formula_arr = np.empty(1000)
for i in range(len(x)):
    using_formula_arr[i] = w * x[i] + b
#     print(m * x[i] + c, end=", ")

print(using_formula_arr[:10], end=", ")
print("...", end=", ")
print(using_formula_arr[-10:])
```

[13] ✓ 0.0s

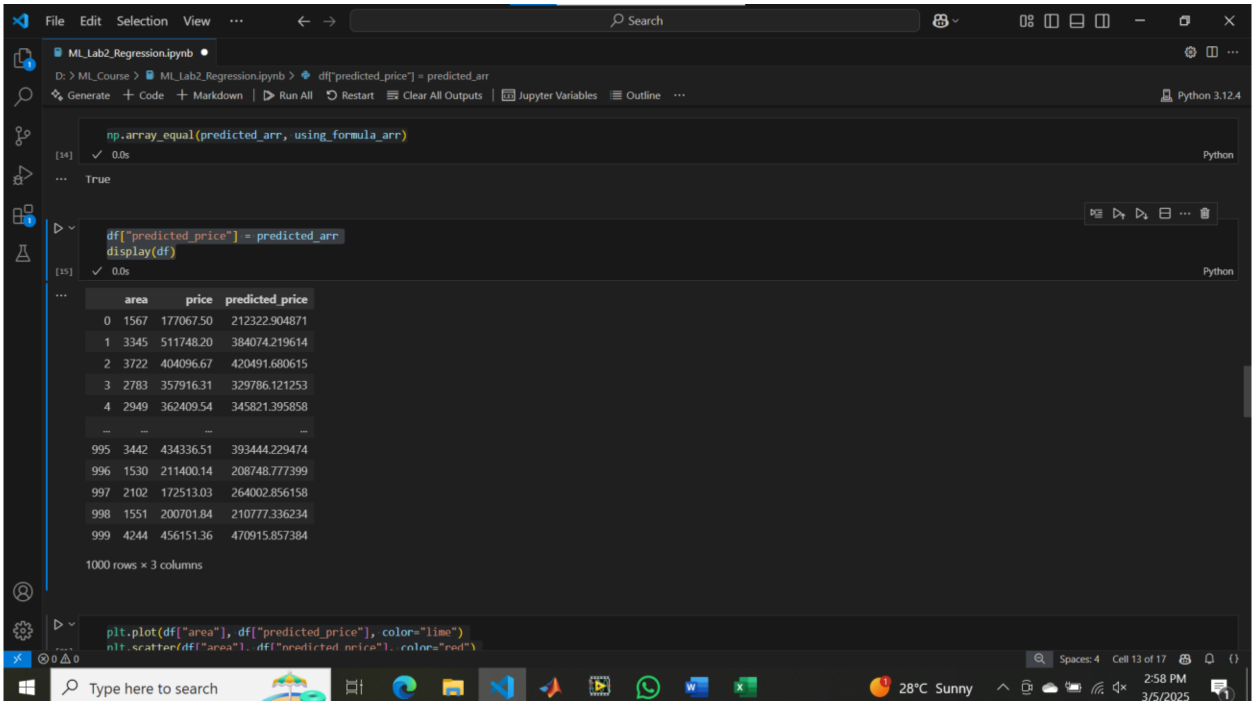
```
[212322.90487091 384074.21961441 420491.68061458 329786.12125342
 345821.39585827 405712.18052698 266417.80715232 360600.89594587
 484342.98491461 276657.19936988], ..., [378857.92546585 212709.29703006 416724.35706284 393251.0333943
 468597.50442913 393444.22947387 208748.77739874 264002.85615762
 210777.33623429 470915.85738405]
C:\Users\DELL\AppData\Local\Temp\ipykernel_17788\3950902830.py:5: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure yo
  using_formula_arr[i] = w * x[i] + b
```

```
np.array_equal(predicted_arr, using_formula_arr)
```

[14]

```
True
```

Using the prediction equation y’ = w*x + b, we calculate prediction inside a for loop by running it from i to number of features which is **len(x)** and saving each iteration inside using_formula_arr[i] array.



```
np.array_equal(predicted_arr, using_formula_arr)
```

[14] ✓ 0.0s

```
True
```

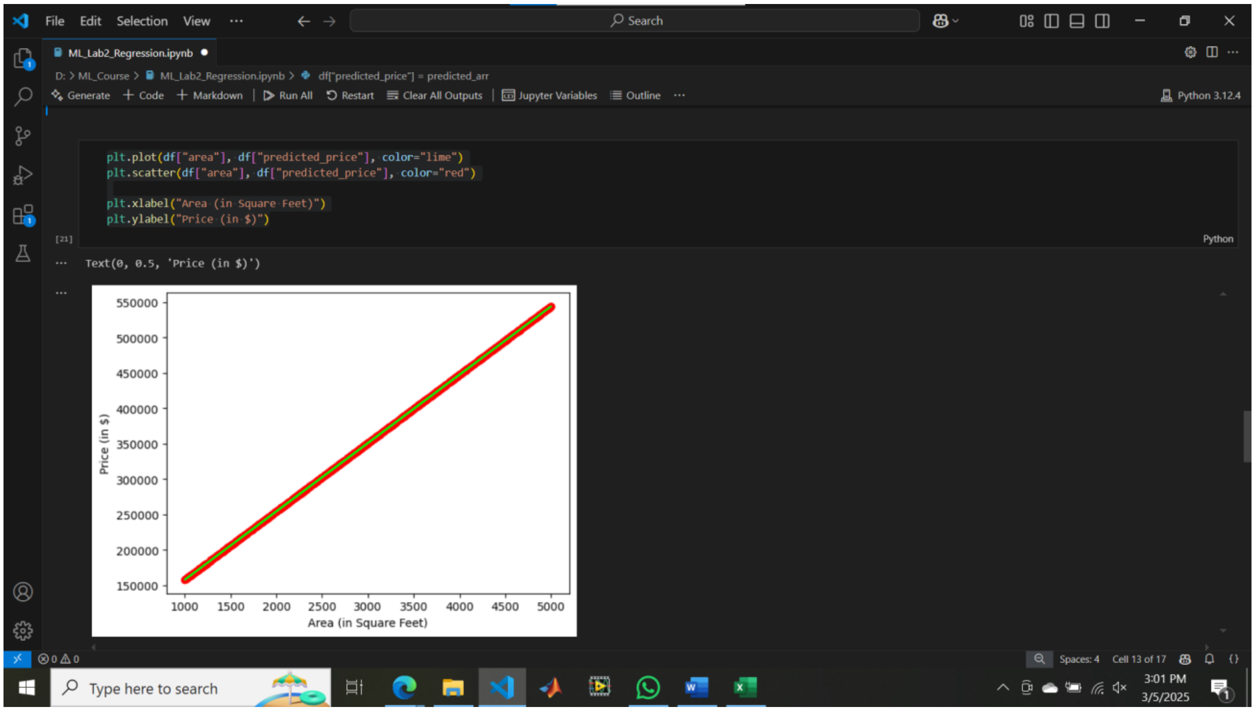
```
df["predicted_price"] = predicted_arr
display(df)
```

[15] ✓ 0.0s

```
     area      price   predicted_price
0    1567  177067.50     212322.904871
1    3345  511748.20     384074.219614
2    3722  404096.67     420491.680615
3    2783  357916.31     329786.121253
4    2949  362409.54     345821.395858
...   ...        ...               ...
995  3442  434336.51     393444.229474
996  1530  211400.14     208748.777399
997  2102  172513.03     264002.856158
998  1551  200701.84     210777.336234
999  4244  456151.36     470915.857384

1000 rows × 3 columns
```

```
plt.plot(df["area"], df["predicted_price"], color="lime")
plt.scatter(df["area"], df["predicted_price"], color="red")
```

Now we pass the using_formula_arr[i] and predicted_arr (predicted value) to a numpy array and display the predicted_prices ($) together with our feature (area) and price ($).

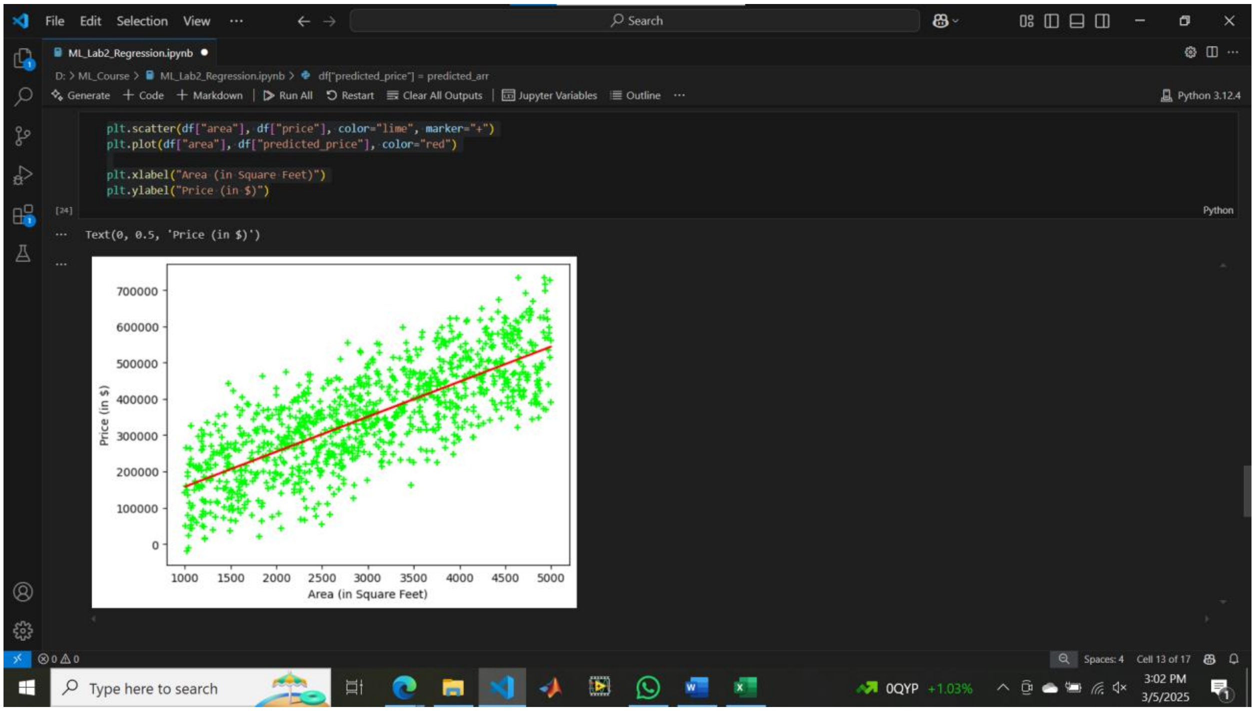


```
plt.plot(df["area"], df["predicted_price"], color="lime")
plt.scatter(df["area"], df["predicted_price"], color="red")

plt.xlabel("Area (in Square Feet)")
plt.ylabel("Price (in $)")
```

[21]

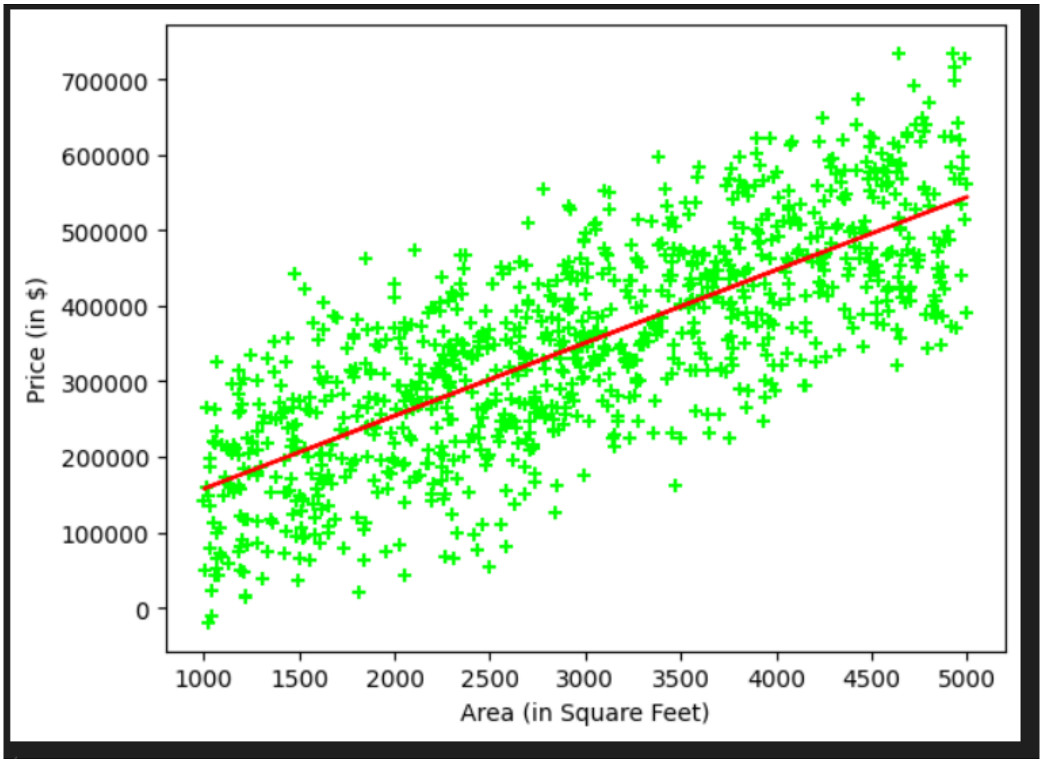
```
Text(0, 0.5, 'Price (in $)')
```

As shown above, plotting the linear regression best fit line. Why we choose the lime-colored line here? It is because we are calculating errors which is the vertical distance between our actual values and predicted values. We squared the errors that we found, taking their squares and summing them up and making sure we minimize that error. So my lime-colored line actually represents the minimum sum value.

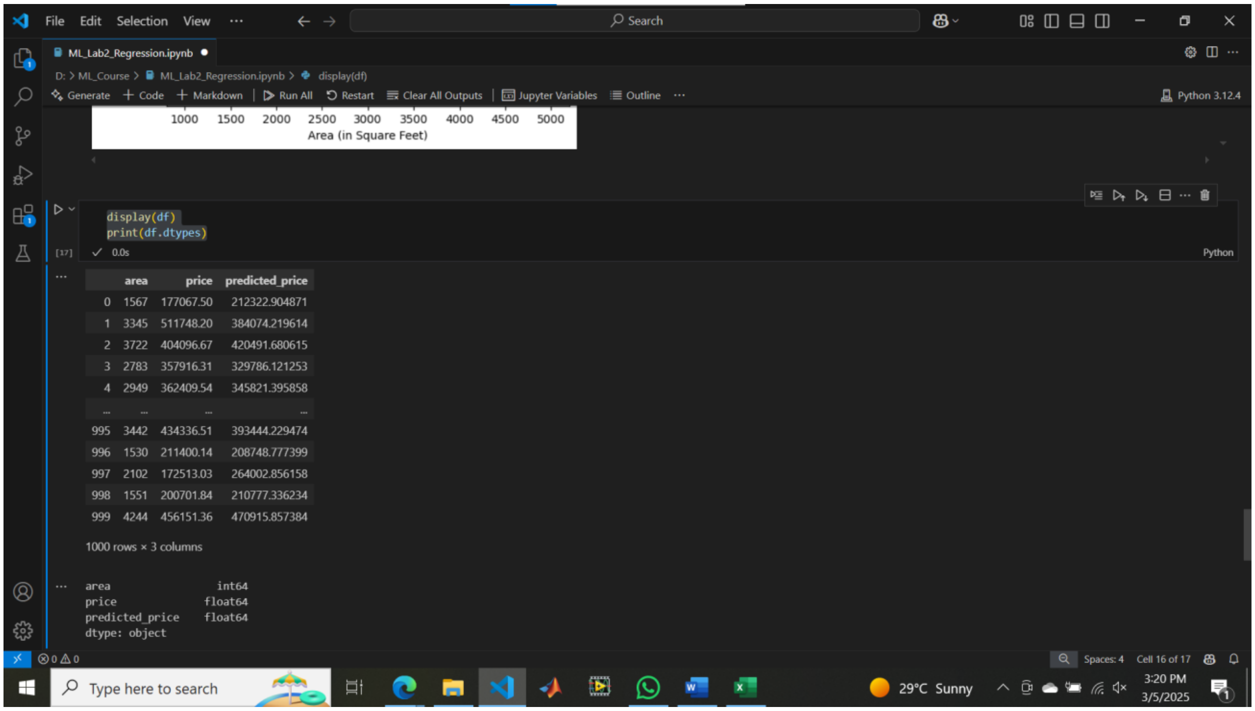


```
plt.scatter(df["area"], df["price"], color="lime", marker="+")
plt.plot(df["area"], df["predicted_price"], color="red")

plt.xlabel("Area (in Square Feet)")
plt.ylabel("Price (in $)")
```

```
Text(0, 0.5, 'Price (in $)')
```



Now in a single scatter plot, we combine the regression line and the dataset values that we have to get the prediction model.



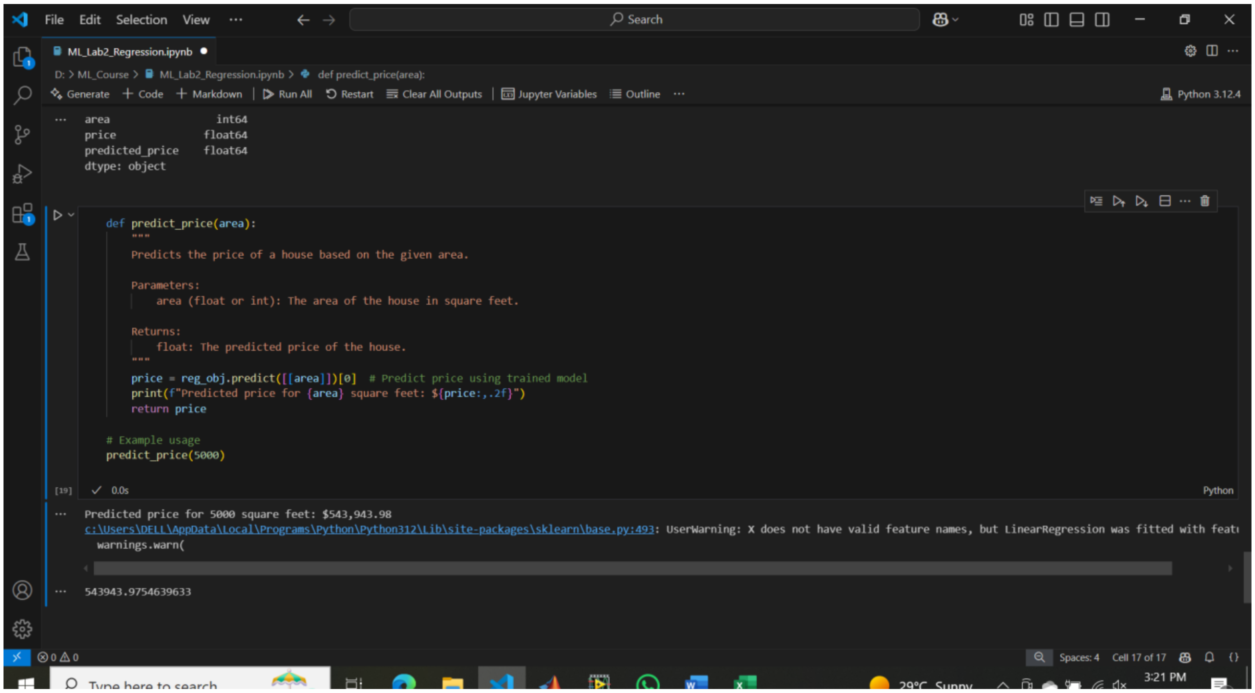
```python
display(df)
print(df.dtypes)
```

	area	price	predicted_price
0	1567	177067.50	212322.904871
1	3345	511748.20	384074.219614
2	3722	404096.67	420491.680615
3	2783	357916.31	329786.121253
4	2949	362409.54	345821.395858
...	...	...	...
995	3442	434336.51	393444.229474
996	1530	211400.14	208748.777399
997	2102	172513.03	264002.856158
998	1551	200701.84	210777.336234
999	4244	456151.36	470915.857384

1000 rows × 3 columns

```
area                int64
price             float64
predicted_price   float64
dtype: object
```

Displaying predicted price.



```
area                int64
price             float64
predicted_price   float64
dtype: object
```

```python
def predict_price(area):
    """
    Predicts the price of a house based on the given area.

    Parameters:
        area (float or int): The area of the house in square feet.

    Returns:
        float: The predicted price of the house.
    """
    price = reg_obj.predict([[area]])[0]  # Predict price using trained model
    print(f"Predicted price for {area} square feet: ${price:,.2f}")
    return price

# Example usage
predict_price(5000)
```

```
Predicted price for 5000 square feet: $543,943.98
c:\Users\DELL\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but LinearRegression was fitted with featu
  warnings.warn(
```

```
543943.9754639633
```

Here we are creating a function **predict_price** and giving it the **area** as a parameter. Here we can pass a certain value of our choice for the area that we want to predict the price for. As we can see when an area of **5000 square feet** is passed to the function, it predicts the price of **543943.98$.** It can be verified with the table given above as well.

Conclusion:

Our model gave us good results at the end. We implemented the simple linear regression model and got pretty close values to our actual values with almost 82% accuracy. If we train it with more datasets and different features we will get a more accurate result.