VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Fall Semester 2024 –2025
PMDTS 504L – Regression Analysis and Predictive Modelling
Experiment 1

Study and Implementation of Simple Linear Regression in Python programming language

Objective:
To build a simple linear regression model that can predict the price of a house based on its area.

Problem 1: Estimate the prices of the houses having an area 3300 sq. feet and 5000 sq. ft based on the data given below:

area	price
2600	550000
3000	565000
3200	610000
3600	680000
4000	725000

Step 1: Import Required Libraries

First, we need to import the necessary Python libraries for data handling, visualization, and modeling.

```
import numpy as np  # Import NumPy for numerical operations (e.g., arrays and mathematical functions)
import pandas as pd  # Import pandas for data manipulation and analysis
import matplotlib.pyplot as plt  # Import Matplotlib for creating visualizations (plots)
from sklearn.model_selection import train_test_split  # Import function to split the dataset into training and testing sets
from sklearn.linear_model import LinearRegression  # Import the LinearRegression class to build the linear regression model
from sklearn.metrics import mean_squared_error, r2_score  # Import evaluation metrics to assess the model's performance
```

Step 2: Load the Provided Data

We'll input the provided data into a pandas DataFrame, which makes it easier to handle.

```
# Create a pandas DataFrame with the provided data. This is a table-like structure with two columns: 'Area' and 'Price'.
data = pd.DataFrame({
    'Area': [2600, 3000, 3200, 3600, 4000],  # List of house areas in square feet
    'Price': [550000, 565000, 610000, 680000, 725000]  # List of house prices in USD corresponding to the areas
})

# Display the DataFrame to verify the data is loaded correctly
print(data)  # This will print the table containing the data with columns 'Area' and 'Price'
```

```
   Area   Price
0  2600  550000
1  3000  565000
2  3200  610000
3  3600  680000
4  4000  725000
```

Step 3: Understand the Data

Before we build the model, it's important to understand the data. Here, we have two variables:

- **Area**: The independent variable (input) that represents the size of the house in square feet.
- **Price**: The dependent variable (output) that represents the price of the house in USD.

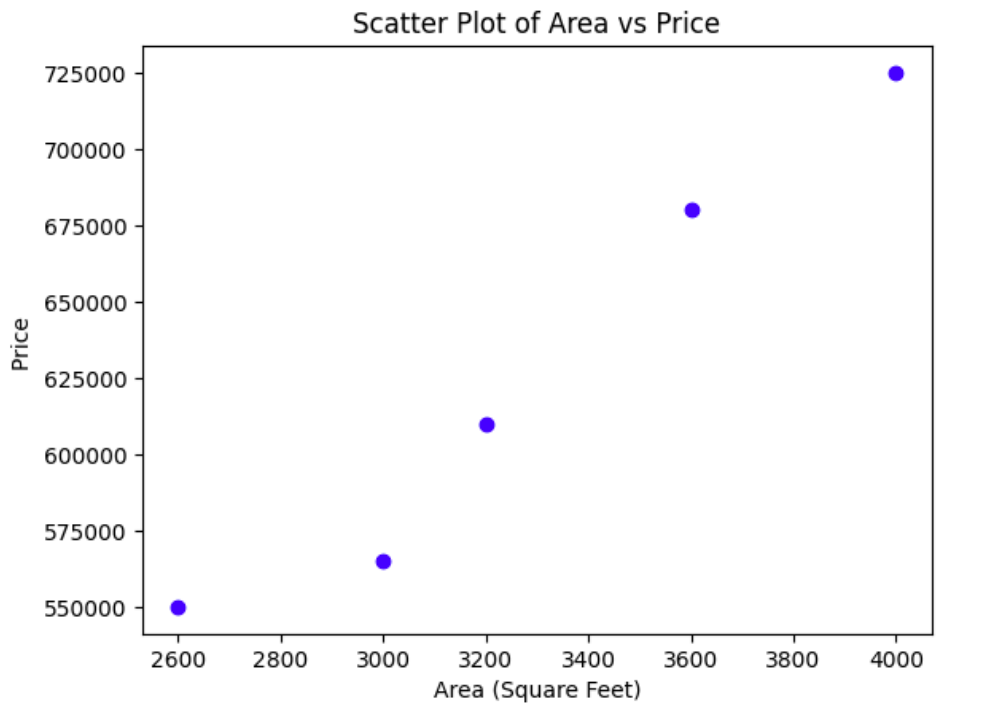
We want to find a relationship between the area and the price, i.e., how the price changes as the area changes.

Step 4: Visualize the Data

Let's plot the data points to visually see if there's a relationship between the area of the house and its price.

```
# The DataFrame contains two columns: 'Area' (the independent variable or input)
# and 'Price'(the dependent variable or output).
# The goal is to understand how 'Area' affects 'Price'.
# Create a scatter plot to visualize the relationship between Area and Price.
plt.scatter(data['Area'], data['Price'], color='blue')  # Plot the data points (Area on the x-axis, Price on the y-axis)

plt.xlabel('Area (Square Feet)')  # Label the x-axis as 'Area (Square Feet)'
plt.ylabel('Price ')  # Label the y-axis as 'Price'
plt.title('Scatter Plot of Area vs Price')  # Give the plot a title to describe what it shows
plt.show()  # Display the plot
```



- **Interpretation**: The scatter plot shows a positive relationship between area and price, indicating that as the area increases, the price generally increases.

Step 5: Prepare the Data for Modeling

Now, we'll prepare the data for modeling. This involves separating the input (X) and output (y) variables.

```
# Extract the feature (X) and target variable (y) from the DataFrame.

X = data['Area'].values.reshape(-1, 1)  # Convert the 'Area' column to a NumPy array and reshape it to be a 2D array with one column
y = data['Price'].values  # Convert the 'Price' column to a NumPy array (1D array)

# Explanation:
# X is the independent variable (input) that the model will use to make predictions.
# y is the dependent variable (output) that the model will predict.
# The 'reshape(-1, 1)' ensures that X is a 2D array with one column, which is the required format for scikit-learn models.
```

+ Code + Text

```
X
```

```
array([[2600],
       [3000],
       [3200],
       [3600],
       [4000]])
```

```
y
```

```
array([550000, 565000, 610000, 680000, 725000])
```

Step 6: Split the Data into Training and Testing Sets

To evaluate the performance of our model, we split the data into a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance.

```
# Split the data into training and testing sets to evaluate the model's performance.

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

# Explanation:
# train_test_split function splits the data into two sets: training and testing.
# 'test_size=0.2' means 20% of the data will be used for testing, and 80% will be used for training.
# 'random_state=42' ensures that the split is the same every time you run the code, making the results reproducible.
# X_train, y_train: The training data used to train the model.
# X_test, y_test: The testing data used to evaluate the model.
X_train
```

```
array([[4000],
       [3200],
       [2600],
       [3600]])
```

```
X_test
```

```
array([[3000]])
```

```
y_train
```

```
array([725000, 610000, 550000, 680000])
```

```
y_test
```

```
array([565000])
```

Step 7: Train the Simple Linear Regression Model

Now, let's build and train the linear regression model using the training data.

```
# Create a LinearRegression model object.
model = LinearRegression()  # Initialize the LinearRegression model from scikit-learn

# Train the model using the training data.
model.fit(X_train, y_train)  # Fit the model to the training data (find the best-fitting line for the data)

# Explanation:
# The 'fit' function finds the best-fit line by calculating the intercept
#and slope that minimize the error between the predicted and actual prices.
```

```
▾ LinearRegression
LinearRegression()
```

```
# Print the model's coefficients to understand the linear equation.
print(f"Intercept: {model.intercept_}")  # Print the intercept of the regression line (the value of y when X = 0)
print(f"Slope (Coefficient): {model.coef_[0]}")  # Print the slope of the regression line (the change in y for a one-unit change in X)

# Explanation:
# The intercept and slope are the parameters of the linear equation: y = intercept + slope * X.
# The intercept is the starting value of the line, and the slope indicates how much the price changes as the area changes.
```

```
Intercept: 211542.05607476638
Slope (Coefficient): 128.27102803738316
```

Step 8: Make Predictions

Once the model is trained, we can use it to predict prices for the testing data.

```
# Use the trained model to make predictions on the test data.
y_pred = model.predict(X_test)  # Predict the house prices for the test data (X_test)

# Explanation:
# The 'predict' function uses the learned linear relationship (from training) to predict the price of houses in the testing set.
# The output, y_pred, contains the predicted prices.
```

```
y_pred
```

```
array([596355.14018692])
```

Step 9: Evaluate the Model

To evaluate how well our model performs, we'll calculate two key metrics:

- **Mean Squared Error (MSE)**: Measures the average squared difference between actual and predicted values (lower is better). A measure of the quality of the estimator.
- **R-squared (R^2)**: Represents the proportion of the variance in the dependent variable that is predictable from the independent variable (closer to 1 is better). The proportion of variance explained by the model; it gives us an idea of how well the model fits the data.

```
# Evaluate the model's performance using two metrics: Mean Squared Error (MSE) and R-squared (R²).

mse = mean_squared_error(y_test, y_pred)  # Calculate the Mean Squared Error (MSE) between actual and predicted prices
r2 = r2_score(y_test, y_pred)  # Calculate the R-squared (R²) score, which represents the proportion of variance explained by the model

print(f"Mean Squared Error: {mse}")  # Print the Mean Squared Error (lower values indicate better fit)
print(f"R-squared: {r2}")  # Print the R-squared score (closer to 1 indicates a better fit)

# Explanation:
# MSE measures the average squared difference between actual and predicted prices; a lower MSE means better predictions.
# R-squared indicates how well the model fits the data; an R-squared close to 1 means the model explains most of the variance in the data.
```

```
Mean Squared Error: 983144816.1411468
R-squared: nan
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py:996: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

Step 10: Visualize the Regression Line

Finally, let's visualize the linear regression line on top of the data points to see how well the model fits the data.

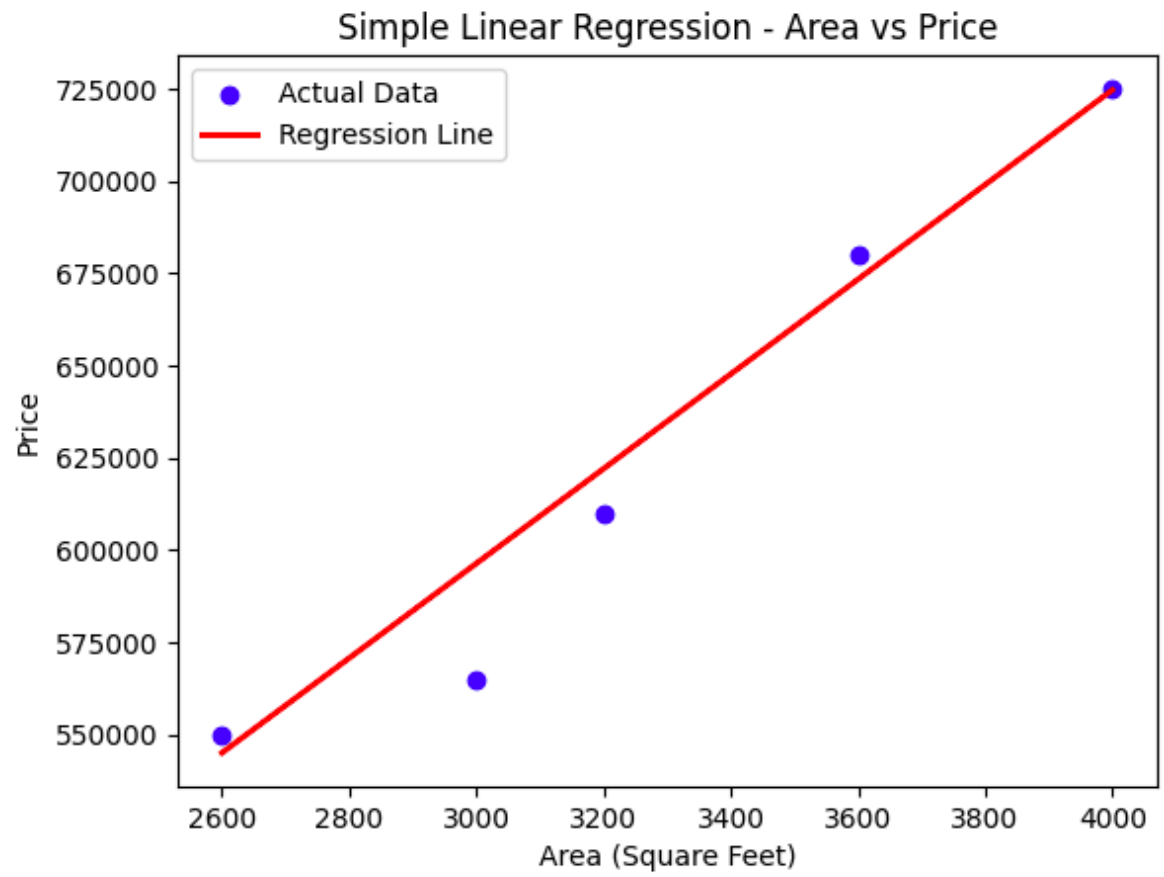
```
# Visualize the original data points along with the regression line.

plt.scatter(X, y, color='blue', label='Actual Data')  # Plot the original data points in blue

plt.plot(X, model.predict(X), color='red', linewidth=2, label='Regression Line')  # Plot the regression line (predicted values) in red

plt.xlabel('Area (Square Feet)')  # Label the x-axis as 'Area (Square Feet)'
plt.ylabel('Price')  # Label the y-axis as 'Price (USD)'
plt.title('Simple Linear Regression - Area vs Price')  # Give the plot a title
plt.legend()  # Show the legend to differentiate between actual data and the regression line
plt.show()  # Display the plot

# Explanation:
# The blue points represent the actual data (area vs price).
# The red line represents the regression line predicted by the model.
# If the model is good, the red line should fit the blue points well.
```



Data Summary:

- We have a dataset consisting of 5 data points, where `Area` (in square feet) is the independent variable, and `Price` (in USD) is the dependent variable.
- The goal was to build a simple linear regression model to predict `Price` based on `Area`.

Model Summary:

- **Intercept**: The predicted starting price for a house, even with 0 square feet, is about **211,542**.
- **Slope**: For every extra square foot of house area, the price increases by about **128.27**.

Model Evaluation:

- **Mean Squared Error (MSE)**: The average error between the predicted and actual prices is quite high (**983,144.82**). This means the model's predictions aren't very accurate.
- **R-squared**: We couldn't calculate the R-squared value (which tells how well the model explains the data) because there wasn't enough test data.

Conclusion:

In this experiment, we built a simple linear regression model to predict house prices based on the area of the house. We visualized the data, trained the model, made predictions, and evaluated the model using MSE and R-squared metrics. The final visualization shows how well the model's predictions align with the actual data.

- **Small Dataset**: The dataset is too small to draw strong conclusions. We need more data to improve the model and better evaluate its performance.