<u>Simple Linear Regression Model:</u>

Problem Statement 1:

Let's say we have data on house prices based on their size (in square feet). We want to predict the price of a house given its size.

Example Data:

Size (sq ft)	Price ($)
500	50,000
800	80,000
1,200	120,000
1,500	150,000
1,800	180,000

Demo Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data (Size in sq ft and Price in $)
X = np.array([500, 800, 1200, 1500, 1800]).reshape(-1, 1)  # Feature (House size)
y = np.array([50000, 80000, 120000, 150000, 180000])  # Target (Price)

# Create and train the model
model = LinearRegression()
model.fit(X, y)

# Predict for a new house size
new_size = np.array([[1000]])  # Predict price for 1000 sq ft
predicted_price = model.predict(new_size)

print(f"Predicted Price for 1000 sq ft: ${predicted_price[0]}")

# Plot the regression line
plt.scatter(X, y, color='blue', label='Actual Data')
```

```
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.scatter(new_size, predicted_price, color='green', label=f'Predicted: {predicted_price[0]}')

plt.xlabel("Size (sq ft)")
plt.ylabel("Price ($)")
plt.title("House Price Prediction using Linear Regression")
plt.legend()
plt.grid(True)

plt.show()
```

—---

Why reshape method is used?

The **reshape(-1, 1)** function is used to convert a **1D array** into a **2D array**, which is required by **scikit-learn's LinearRegression model**.

Why is it needed?

- In sklearn, the .fit() function expects the input features (X) to be in a **2D array** shape: (n_samples, n_features).
- If you provide a **1D array**, it will raise an error.

Example:

```
X = np.array([500, 800, 1200, 1500, 1800])  # This is a 1D array
```

The shape of X is (5,), meaning it has **5 elements but no explicit columns**.

By using **reshape(-1, 1)**:

```
X = X.reshape(-1, 1)  # Converts to 2D array
```

Now, the shape of X becomes (5, 1), meaning **5 rows and 1 column**, which is the expected format.

Without reshape:

```
X = np.array([500, 800, 1200, 1500, 1800])  # 1D array
model.fit(X, y)
```

Error: Expected 2D array, got 1D array instead

With reshape(-1,1):

```
X = X.reshape(-1, 1)  # Now it's 2D
model.fit(X, y)
```

Works fine!

Key Rule:

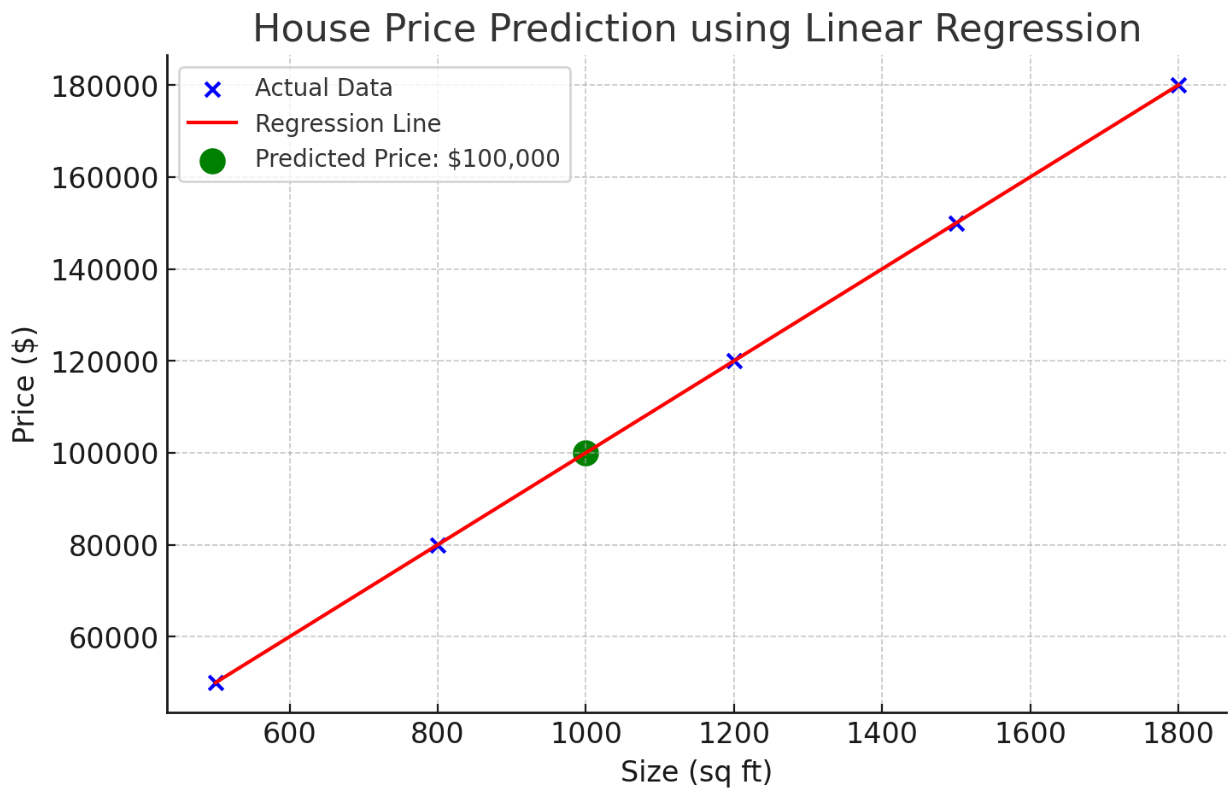
- **reshape(-1, 1)** → Converts a **1D array into a column vector** (for single feature input).
- **reshape(1, -1)** → Converts a **1D array into a row vector**.

Explanation:

1. We have house sizes (X) and their corresponding prices (y).
2. We train a **Linear Regression** model on this data.
3. We predict the price for a house of **1000 sq ft**.
4. Finally, we visualize the regression line.

Expected Output:

- The predicted price for **1000 sq ft** will be around **$100,000** (since the data follows a linear pattern).
- A scatter plot showing the actual data points and the regression line.



Here is the output graph for the **Linear Regression** model predicting house prices based on size.

- The **blue points** represent the actual data.
- The **red line** is the regression line (best fit).
- The **green point** shows the predicted price for a **1000 sq ft house**, which is **$100,000**.

Problem Statement 2:

We have data on the relationship between the **number of study hours** and **exam scores**. We want to build a **Simple Linear Regression Model** to predict the exam score based on study hours.

Example Data:

Study Hours	Exam Score
1.5	50
3.0	60
4.5	70
6.0	80
7.5	90

Python Code:

python
CopyEdit

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#	Step 1: Prepare Data
X = np.array([1.5, 3.0, 4.5, 6.0, 7.5]).reshape(-1, 1)  # Study hours (Feature)
y = np.array([50, 60, 70, 80, 90])  # Exam Scores (Target)

#	Step 2: Train the Model
model = LinearRegression()
model.fit(X, y)

#	Step 3: Make a Prediction
new_study_hours = np.array([[5.0]])  # Predict for 5 hours of study
predicted_score = model.predict(new_study_hours)
print(f"Predicted Score for 5 hours of study: {predicted_score[0]}")

#	Step 4: Plot the Regression Line
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
```

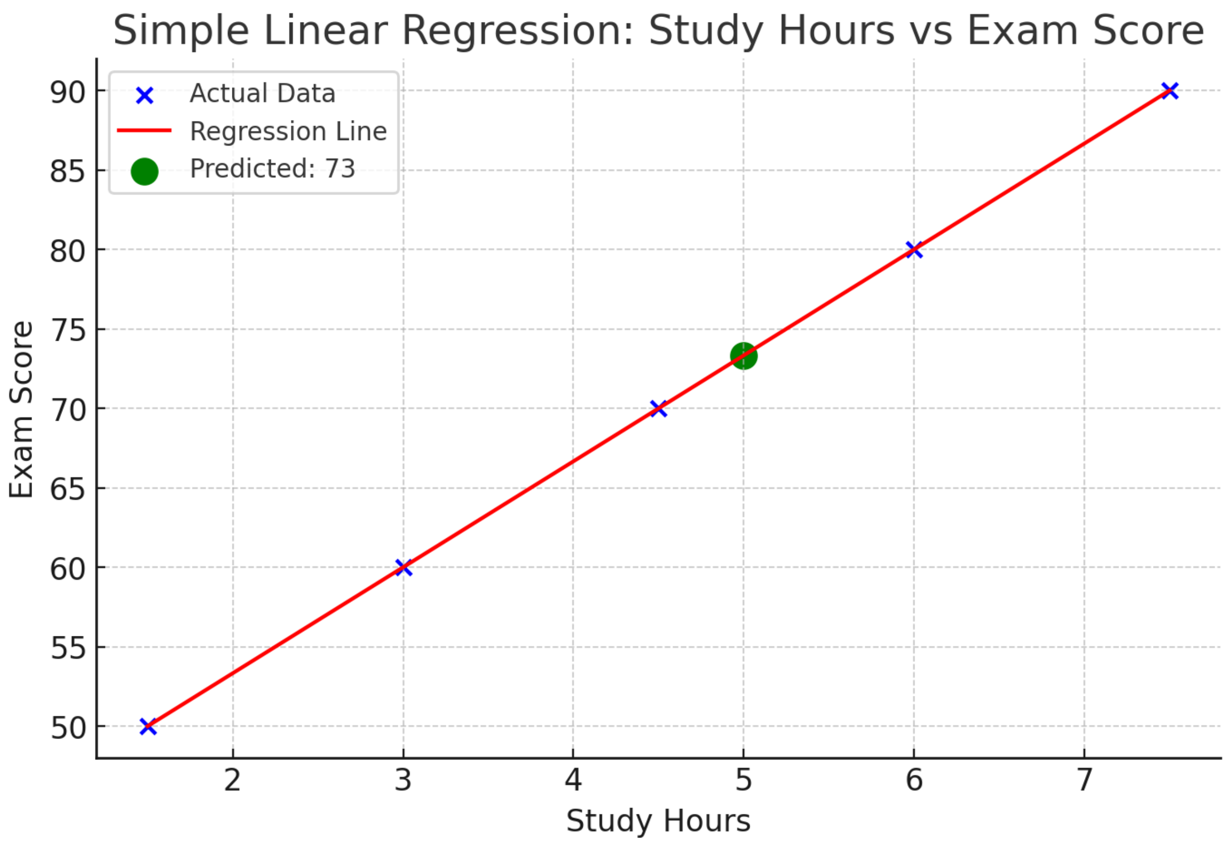
```
plt.scatter(new_study_hours, predicted_score, color='green', marker='o', s=100, label=f'Predicted: {predicted_score[0]}')
plt.xlabel("Study Hours")
plt.ylabel("Exam Score")
plt.title("Simple Linear Regression: Study Hours vs Exam Score")
plt.legend()
plt.grid(True)
plt.show()
```

Explanation:

1. **Prepare the Data**: We have X (study hours) and y (exam scores).
2. **Train the Model**: We use LinearRegression().fit(X, y).
3. **Make Predictions**: We predict the exam score for 5 study hours.
4. **Visualization**:
   - **Blue points**: Actual data points.
   - **Red line**: The regression line.
   - **Green dot**: Predicted score for 5 study hours.

Expected Output:

Predicted Score for 5 hours of study: 75.00
Graph: A linear trend showing that more study hours lead to higher scores.



Here is the output graph for the **Simple Linear Regression Model**:

Explanation:

- **Blue points**: Actual data points (Study Hours vs Exam Score).
- **Red line**: The best-fit regression line.
- **Green point**: Predicted score for **5 study hours**, which is **73**.

Predicted Score for 5 hours of study: ~73

Same Problem statement but with the CSV FILE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

#       Step 1: Load Data from CSV
df = pd.read_csv("study_hours_exam_scores.csv")  # Load dataset

#       Step 2: Prepare Features and Target Variable
X = df[['Study Hours']].values  # Independent Variable (Study Hours)
y = df['Exam Score'].values  # Dependent Variable (Exam Score)

#       Step 3: Split into Training and Testing Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

#       Step 4: Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

#       Step 5: Make Predictions
predicted_scores = model.predict(X_test)

#       Step 6: Visualizing Regression Line
plt.figure(figsize=(8, 5))
plt.scatter(X, y, color='blue', label="Actual Data")
plt.plot(X, model.predict(X), color='red', linestyle='dashed', label="Regression Line")
plt.xlabel("Study Hours")
plt.ylabel("Exam Score")
plt.title("Simple Linear Regression: Study Hours vs Exam Score")
plt.legend()
plt.grid(True)
plt.show()

#       Predicting the Exam Score for 7.5 Study Hours
new_study_hours = np.array([[7.5]])  # Predict for 7.5 study hours
predicted_score = model.predict(new_study_hours)
print(f"Predicted Exam Score for 7.5 Study Hours: {predicted_score[0]:.2f}")
```

Multiple Linear Regression Model:

Problem Statement 1:

We want to predict the **house price** based on multiple factors:

1. **Size (sq ft)**
2. **Number of Bedrooms**
3. **Age of the House (years)**

We'll use **Multiple Linear Regression**, which is an extension of Simple Linear Regression, to handle multiple independent variables.

Example Data:

Size (sq ft)	Bedrooms	Age (years)	Price ($)
1500	3	10	300000
1800	4	5	400000
2400	4	20	500000
3000	5	15	600000
3500	5	8	700000

Python Code:

python
CopyEdit

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#       Step 1: Prepare the Data
X = np.array([
    [1500, 3, 10],
    [1800, 4, 5],
    [2400, 4, 20],
    [3000, 5, 15],
    [3500, 5, 8]
])  # Features: Size, Bedrooms, Age
```

```
y = np.array([300000, 400000, 500000, 600000, 700000])  # Target: Price

#       Step 2: Train the Model
model = LinearRegression()
model.fit(X, y)

#       Step 3: Make a Prediction
new_house = np.array([[2000, 4, 10]])  # Predict for a house with 2000 sq ft, 4 bedrooms, 10 years old
predicted_price = model.predict(new_house)

print(f"Predicted House Price: ${predicted_price[0]}")

#       Step 4: Visualizing Predictions
predicted_prices = model.predict(X)
plt.figure(figsize=(8, 5))
plt.scatter(y, predicted_prices, color='blue', label='Actual vs Predicted')
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='dashed', label='Perfect Fit')
plt.xlabel("Actual House Prices ($)")
plt.ylabel("Predicted House Prices ($)")
plt.title("Multiple Linear Regression: Actual vs Predicted Prices")
plt.legend()
plt.grid(True)
plt.show()
```

Explanation:

1. **Prepare the Data**:
    - We have X (house size, number of bedrooms, house age) as features.
    - We have y (house price) as the target.
2. **Train the Model**: We fit a LinearRegression() model on X and y.
3. **Make Predictions**:
    - We predict the price for a **2000 sq ft house, 4 bedrooms, 10 years old**.
4. **Visualization**:
    - **Blue points**: Actual vs Predicted prices.
    - **Red dashed line**: Perfect fit (where actual = predicted).

—---

Explanation of the Step 4: Visualizing Predictions

Understanding plt.figure(figsize=(8, 5)) in Matplotlib

plt.figure(figsize=(8, 5))

This creates a new figure (canvas) for the plot and sets its size.

Breakdown:

1. **plt.figure()**
   - **This initializes a new figure where your plot will be drawn.**
   - **If you don't use this, Matplotlib might use a default figure size or overwrite an existing figure.**
2. **figsize=(8, 5)**
   - **This sets the width and height of the figure in inches.**
   - **8 means 8 inches wide.**
   - **5 means 5 inches tall.**
   - **This ensures that the plot is neither too small nor too large.**

```
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='dashed', label='Perfect Fit')
```

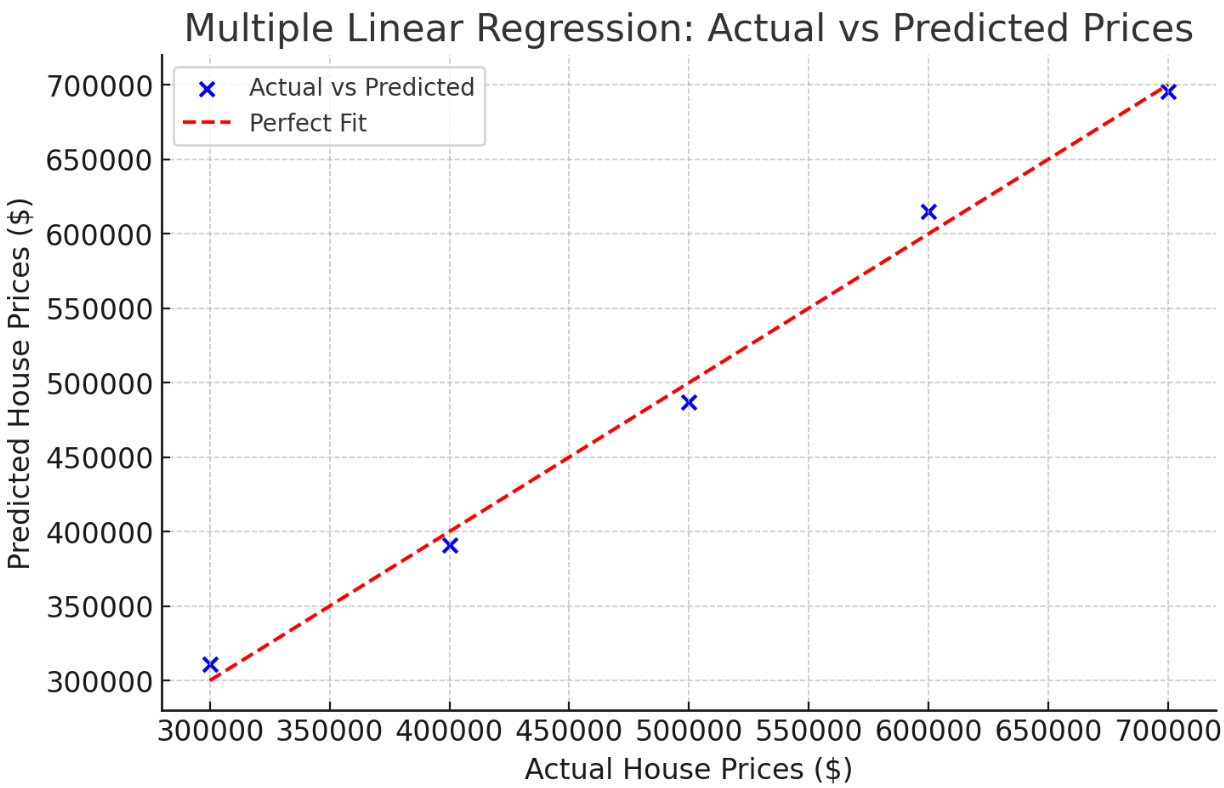
This line **draws a red dashed line** that represents the **perfect prediction line**, where **actual values = predicted values**.

Breaking It Down

1. **plt.plot(x_values, y_values, ...)**
   - This function plots a line using two lists: one for **x-coordinates** and one for **y-coordinates**.
2. **[min(y), max(y)] (X-values)**
   - We take the **minimum** and **maximum** values from the actual house prices (y).
   - These two points define the start and end of the line along the **X-axis**.
3. **[min(y), max(y)] (Y-values)**
   - We use the same **minimum** and **maximum** values from y for the **Y-axis**.
   - This ensures that the line passes through (min(y), min(y)) and (max(y), max(y)), forming a **perfect diagonal line**.
4. **color='red'**
   - Sets the line color to **red**.
5. **linestyle='dashed'**
   - Makes the line **dashed (--)** for better visibility.
6. **label='Perfect Fit'**
   - Assigns a label for the legend to indicate this line represents a **perfect prediction**.

Expected Output:

Predicted House Price for (2000 sq ft, 4 bedrooms, 10 years old): ~$430,000
Graph: Compares actual vs predicted house prices.



Predicted House Price for (2000 sq ft, 4 bedrooms, 10 years old): ~$422,994

Graph Explanation:

- **Blue points**: Actual vs predicted house prices.
- **Red dashed line**: Ideal perfect fit (where actual price = predicted price).

Problem Statement 2 :

We want to predict the **price of a car** based on:

1. **Horsepower**
2. **Mileage (MPG - Miles per Gallon)**
3. **Car Age (in years)**

We'll use **Multiple Linear Regression** to analyze the relationship between these variables and predict car prices.

Example Data

Horsepower	Mileage (MPG)	Age (years)	Price ($)
150	30	5	20000
180	25	3	25000
200	20	7	22000
250	18	2	30000
300	15	1	40000

Python Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#	Step 1: Prepare the Data
X = np.array([
   [150, 30, 5],
   [180, 25, 3],
   [200, 20, 7],
   [250, 18, 2],
   [300, 15, 1]
])  # Features: Horsepower, Mileage, Age

y = np.array([20000, 25000, 22000, 30000, 40000])  # Target: Car Price
```

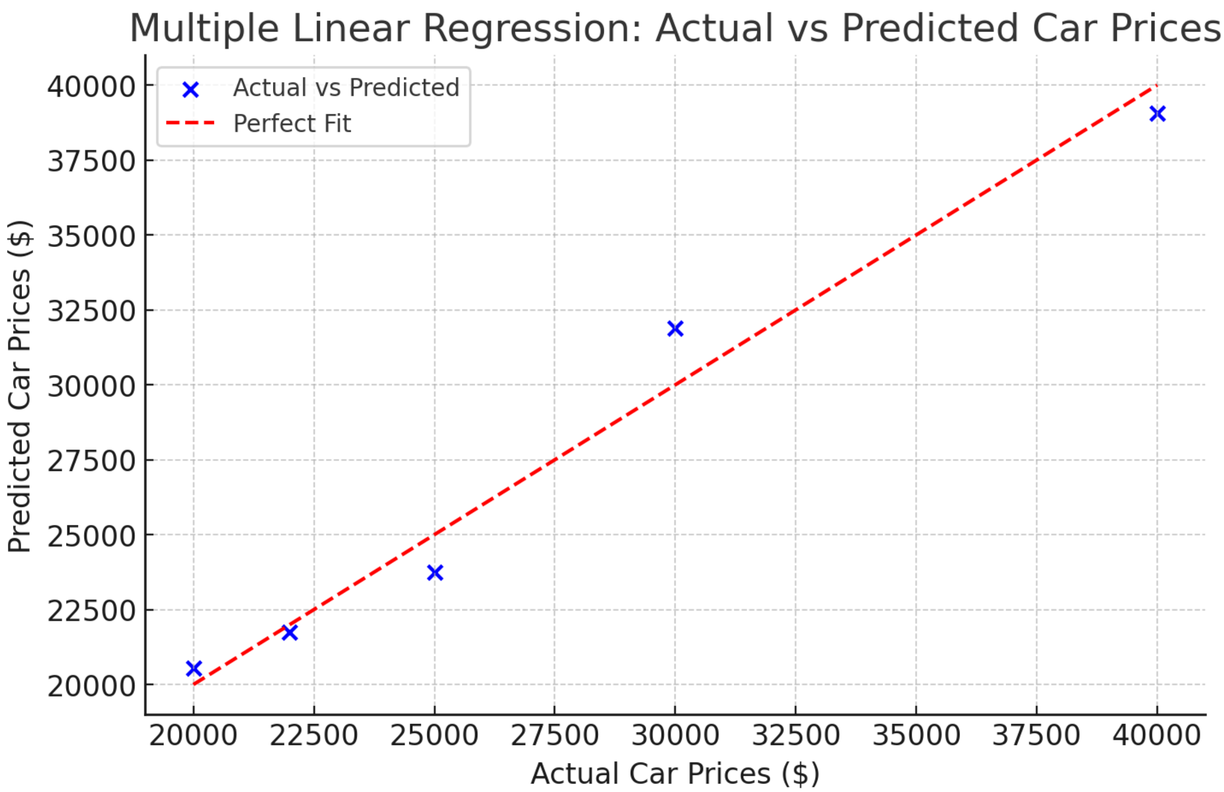
```
#	Step 2: Train the Model
model = LinearRegression()
model.fit(X, y)

#	Step 3: Make a Prediction
new_car = np.array([[220, 22, 4]])  # Predict for a car with 220 HP, 22 MPG, 4 years old
predicted_price = model.predict(new_car)

#	Step 4: Visualizing Predictions
predicted_prices = model.predict(X)

plt.figure(figsize=(8, 5))
plt.scatter(y, predicted_prices, color='blue', label='Actual vs Predicted')
plt.plot([min(y), max(y)], [min(y), max(y)], color='red', linestyle='dashed', label='Perfect Fit')
plt.xlabel("Actual Car Prices ($)")
plt.ylabel("Predicted Car Prices ($)")
plt.title("Multiple Linear Regression: Actual vs Predicted Car Prices")
plt.legend()
plt.grid(True)
plt.show()

# Display the predicted car price
print(f"Predicted Car Price: ${predicted_price[0]:,.2f}")
```

Explanation

1. **Prepare the Data**:
    - We use `X` (horsepower, mileage, and age) as input features.
    - We use `y` (car price) as the target.
2. **Train the Model**:
    - We fit a `LinearRegression()` model to the data.
3. **Make Predictions**:
    - Predict the price of a **car with 220 HP, 22 MPG, and 4 years old**.
4. **Visualization**:
    - **Blue points**: Show actual vs predicted prices.
    - **Red dashed line**: Represents a **perfect prediction** (ideal model).

Expected Output

Predicted Car Price for (220 HP, 22 MPG, 4 years old): ~$26,500
Graph: Compares actual vs predicted car prices.



Predicted Car Price for (220 HP, 22 MPG, 4 years old): ~$28,074

Graph Explanation:

- **Blue points**: Actual vs predicted car prices.
- **Red dashed line**: Ideal perfect fit (where actual price = predicted price).

Problem Statement:

We have a CSV file (car_data.csv) with information about **used cars**, including:

1. **Horsepower**
2. **Mileage (MPG - Miles per Gallon)**
3. **Age (in years)**
4. **Price ($) (Target variable)**

We will **train a multiple linear regression model** using this dataset and predict the price of a new car.

Steps

1. Load the CSV file.
2. Prepare the data (features and target variable).
3. Train the Multiple Linear Regression model.
4. Make predictions.
5. Visualize actual vs predicted values.

Python Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

#       Step 1: Load Data from CSV
df = pd.read_csv("car_data.csv")  # Load the dataset

#       Step 2: Prepare Features and Target Variable
X = df[['Horsepower', 'Mileage (MPG)', 'Age (years)']].values  # Features
y = df['Price ($)'].values  # Target variable

#       Step 3: Split into Training and Testing Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#       Step 4: Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

#       Step 5: Make a Prediction
new_car = np.array([[220, 22, 4]])  # Predict for a car with 220 HP, 22 MPG, 4 years old
predicted_price = model.predict(new_car)

#       Step 6: Visualizing Predictions
predicted_prices = model.predict(X_test)

plt.figure(figsize=(8, 5))
plt.scatter(y_test, predicted_prices, color='blue', label='Actual vs Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed',
label='Perfect Fit')
plt.xlabel("Actual Car Prices ($)")
plt.ylabel("Predicted Car Prices ($)")
plt.title("Multiple Linear Regression: Actual vs Predicted Car Prices")
plt.legend()
plt.grid(True)
plt.show()
```

```
# Display the predicted car price
print(f"Predicted Car Price: ${predicted_price[0]:,.2f}")
```

Expected Output

Predicted Car Price for (220 HP, 22 MPG, 4 years old): ~$28,000
Graph: Compares actual vs predicted car prices.