**🟦 Detailed Step-by-Step Explanation of the Practical**

**1. Load the Dataset**

python

CopyEdit

data = pd.read\_csv("USA\_Housing.csv")

data.head()

✅ Here, pandas (pd) is used to load the **CSV file** into a **DataFrame** (a table-like structure).

* data.head() shows the **first 5 rows** — helps you quickly check if the data is loaded properly.

🔵 **Important:**  
At this point, the data has **6 columns**:

* 5 independent (input) features
* 1 dependent (target) feature: Price

**2. Feature Selection**

python

CopyEdit

X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',

'Avg. Area Number of Bedrooms', 'Area Population']]

y = data['Price']

✅ Here:

* X = inputs (features used for prediction)
* y = output (what we want to predict — the **house price**)

🔵 **Important Concept:**  
You **manually** select only the useful columns for X — this is called **Feature Selection**.  
You don't use unwanted columns like Address.

**3. Splitting the Data (Training and Testing Sets)**

python

CopyEdit

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

✅ Here:

* train\_test\_split (from sklearn.model\_selection) splits your data into:
  + **Training set (80%)**: for learning
  + **Testing set (20%)**: for checking performance
* random\_state=42 ensures the split is **reproducible** (you get the same split every time).

🔵 **Important Concept:**  
We **train** on one part and **test** on unseen data — simulates real-world prediction scenarios.

**4. Training the Model (Linear Regression)**

python

CopyEdit

model = LinearRegression()

model.fit(X\_train, y\_train)

✅ Here:

* LinearRegression() (from sklearn.linear\_model) creates the **regression model**.
* .fit(X\_train, y\_train) teaches the model:
  + It finds the **best line/plane** that **minimizes the error** between predicted and actual house prices.

🔵 **Important Concept:**  
Linear Regression learns the **best weights (coefficients)** for each feature.

**5. Making Predictions**

python

CopyEdit

y\_pred = model.predict(X\_test)

✅ After training, now the model **predicts house prices** for the **test set**.

🔵 **Important:**  
The predicted values (y\_pred) will not be exactly the same as y\_test, but they should be **close** if the model is good.

**6. Evaluating the Model**

python

CopyEdit

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

✅ Here:

* mean\_squared\_error calculates the **average of the squares of the errors**.
* r2\_score measures **how well the model explains the variability**.

✅ Then we print:

python

CopyEdit

print("\nModel Evaluation:")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"R² Score: {r2:.2f}")

🔵 **Interpretation:**

* **MSE**: Lower is better. (Your MSE was around 10 billion — big because prices are in dollars)
* **R²**: Closer to 1 is better. (Your R² = 0.92 = **92% accuracy** → very strong model!)

**📊 Now Let's Explain the Graphs**

**7. Actual vs Predicted Prices Scatter Plot**

python

CopyEdit

plt.figure(figsize=(8,6))

plt.scatter(y\_test, y\_pred, color='blue', alpha=0.6)

plt.title("Actual vs Predicted House Prices")

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', linestyle='--') # Perfect line

plt.show()

✅ **Explanation:**

* Blue dots represent **actual vs predicted prices**.
* Red dashed line = **perfect prediction line** (where prediction = actual).
* **If points are close to the red line, model is good!**

🔵 **Important Observations:**

* Your points are **very close** to the red line → prediction is strong.
* Some small spread is natural.

**8. Residuals (Errors) Plot**

python

CopyEdit

residuals = y\_test - y\_pred

plt.figure(figsize=(8,6))

sns.histplot(residuals, kde=True, color='purple', bins=30)

plt.title("Residuals Plot")

plt.xlabel("Residuals (Errors)")

plt.ylabel("Frequency")

plt.show()

✅ **Explanation:**

* Residuals = **Actual - Predicted** (errors).
* We plot a histogram to check:
  + Are most errors around **zero**?
  + Are errors **symmetrically distributed**?

🔵 **Important Observations:**

* If residuals are centered at 0 and shaped like a bell (normal distribution), the model is **unbiased**.
* Your plot probably shows most errors around 0 → good.

**9. Residuals vs Fitted (Predicted) Values Plot**

python

CopyEdit

plt.figure(figsize=(8,6))

plt.scatter(y\_pred, residuals, color='green', alpha=0.6)

plt.title("Residuals vs Fitted (Predicted) Values")

plt.xlabel("Predicted Prices")

plt.ylabel("Residuals (Errors)")

plt.axhline(y=0, color='red', linestyle='--') # Zero line

plt.show()

✅ **Explanation:**

* Scatter plot of **residuals vs predicted values**.
* Horizontal red line at **0** means "no error".

🔵 **Important Observations:**

* You want **random scatter** (no pattern).
* If there is a funnel shape or curve → model is bad.
* Flat scatter around the red line → **good model**.