**Data Analytics-1**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

\_

df= pd.read\_csv("Assignment-A3-BostonHousing.csv")

df.head() # Prints first 5 rows

\_

print("Columns:\n", df.columns)

print("Info:\n", df.info())

print("Description:\n", df.describe())

\_

print(df.isnull().sum())

\_  
plt.figure(figsize=(12, 10))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')# Check **how much two columns are related** (called correlation).

plt.title("Correlation Matrix")

plt.show()

\_

X = df.drop('medv', axis=1) # Deleted/Dropped "medv" (median value) column from dataset

y = df['medv'] # Target (Median value of owner-occupied homes)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Split data into 80% training and 20% testing

# X is independent variable; y is dependent variable

\_  
lr = LinearRegression() # Create linear regression model object "lr"

lr.fit(X\_train, y\_train) # Train linear regression model using "X\_train" and "y\_train"

y\_pred = lr.predict(X\_test) # Make prediction on test case (X\_train); predicated value stored in variable (y\_pred)

# Evaluation

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

print("R-squared (R²):", r2\_score(y\_test, y\_pred))

\_

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

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

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red')

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.grid(True)

plt.show()

✅ **In this practical, you are:**

* Building a **Linear Regression Model**.
* Using the **Boston Housing dataset**.
* **Target**: Predict the **house prices**.

### ****Target/Objective****:

👉 **Predict the house price** based on other features like:

* number of rooms,
* location quality,
* crime rate,
* distance to employment centers,
* property tax rate, etc.

House price is the dependent variable (target) you are predicting.  
All other columns (features) are independent variables used to make that prediction.

### ****In very simple words****:

* Dataset has information about houses.
* You are building a model that can **guess the price of a house** based on house characteristics.
* You are using **Linear Regression** because price prediction is a **continuous** value (not categories like yes/no).

### ****What is a Correlation Matrix?****

* It shows **how strongly two variables are related**.
* Value is between **-1 and 1**:
  + **+1** → Perfect positive correlation (both increase together)
  + **-1** → Perfect negative correlation (one increases, other decreases)
  + **0** → No relation

Code Explanation:  
1). plt.figure(figsize=(12, 10))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title("Correlation Matrix")

plt.show()

explanation:

Got it — you want **super basic explanation**, like you are starting from **zero**.

Let’s go **very very simple**, like how a teacher explains on board:

### ****What are we doing in this part?****

✅ We are **drawing a chart** to see **which columns (features) are related** to each other.  
✅ Especially we want to find **which features** are related to **House Price**.  
✅ This will help our model to **make better predictions**.

### ****Now line by line:****

1. **plt.figure(figsize=(12, 10))**

➔ Create a **big empty whiteboard** where we will draw the heatmap.  
➔ Size is 12 (width) and 10 (height).

1. **sns.heatmap(df.corr(), annot=True, cmap='coolwarm')**

➔ **sns.heatmap** means "draw a colorful table (heatmap)".

➔ df.corr() means:

* + Check **how much two columns are related** (called correlation).
  + Example:
    - Number of rooms ↑ → House price ↑ = Positive relation (+ value)
    - Crime rate ↑ → House price ↓ = Negative relation (- value)

➔ annot=True means:

* + Write **numbers** inside each box (like 0.7, -0.5).

➔ cmap='coolwarm' means:

* + Use **blue for negative** and **red for positive** relations.

1. **plt.title("Correlation Matrix")**

➔ Add a **title** on top: "Correlation Matrix"  
(So we know what this chart is showing.)

1. **plt.show()**

➔ **Show** the heatmap on the screen.

### ****In short:****

| **We are drawing a colorful table to check:** |
| --- |
| Which features (columns) are strongly related to house price. |
| Which features are not related. |
| Then we can decide which features to use for our prediction model. |

### ****Simple Example:****

Imagine you are checking your exam marks:

| **Subject** | **Marks** |
| --- | --- |
| Maths | 90 |
| Science | 85 |
| History | 40 |

Now you see:

* If you study more maths ➔ Science marks also increase ➔ (positive relation)
* Studying history doesn't affect science ➔ (no relation)

Same way here:  
We are checking **which house features** are **helpful for price prediction**.

### ⭐ Important thing to remember:

* **Correlation value**:
  + Close to **+1** → Strong positive relation
  + Close to **-1** → Strong negative relation
  + Close to **0** → No relation

2).X = df.drop('medv', axis=1) # Deleted/Dropped "medv" (median value) column from dataset

y = df['medv'] # Target (Median value of owner-occupied homes)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Split data into 80% training and 20% testing

# X is independent variable; y is dependent variable  
explanation:  
Perfect! Let’s go **very very simple** again 🔥:

### ****What are we doing here overall?****

✅ We are **preparing the data** for training our machine learning model.  
✅ We **separate features** and **target**.  
✅ Then we **split the data** into **training** and **testing** parts.

### ****Line-by-Line Explanation:****

### X = df.drop('medv', axis=1)

* **df** = our full dataset.
* **drop('medv', axis=1)** = **remove the "medv" column** (medv = median value of house).
* So **X** now contains **only input features** (like rooms, age, tax, etc.).

**➔ In simple words:**  
✅ X = **All the information except the price**.

### y = df['medv']

* **y** = **only the "medv" column**.
* **"medv"** is the house **price** (the thing we want to predict).

**➔ In simple words:**  
✅ y = **Only the house prices** (target/output).

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

* **train\_test\_split** = a function that **divides the data** into two parts:
  + 80% for **training** → (X\_train, y\_train)
  + 20% for **testing** → (X\_test, y\_test)
* **test\_size=0.2** ➔ 20% data will be used for testing.
* **random\_state=42** ➔ To make sure every time we run the code, we get the same split (for consistency).

**➔ In simple words:**  
✅ We **teach** the model on 80% data (training)  
✅ We **test** how well it learned on 20% data (testing)

### ****Meaning of variables:****

| **Variable** | **Meaning** |
| --- | --- |
| X\_train | Features for training (example: rooms, tax, age...) |
| X\_test | Features for testing |
| y\_train | Prices for training |
| y\_test | Prices for testing |

### ****In 1 line:****

We separated **inputs** and **outputs**, and **divided data** into **training** and **testing** parts.

### ⭐ Important point:

* **X** = independent features (input).
* **y** = dependent variable (target/price).
* **Training data** = to build (learn) the model.
* **Testing data** = to check if model is predicting correctly.

3). **lr = LinearRegression() # Create linear regression model object "lr"**

**lr.fit(X\_train, y\_train) # Train linear regression model using "X\_train" and "y\_train"**

**y\_pred = lr.predict(X\_test) # Make prediction on test case (X\_train); predicated value stored in variable (y\_pred)**

**# Evaluation**

**print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))**

**print("R-squared (R²):", r2\_score(y\_test, y\_pred)):**

Let's go **very simple and easy** again 🔥:

### ****What are we doing overall?****

✅ We are **creating**, **training**, **predicting**, and **checking** the performance of a **Linear Regression Model**.

### ****Line-by-Line Explanation:****

### lr = LinearRegression()

* **Create** a **Linear Regression model object** named lr.
* (This is like creating an empty machine that will learn how to predict.)

**➔ In simple words:**  
✅ Created an empty **Linear Regression model**.

### lr.fit(X\_train, y\_train)

* **Train** the model using:
  + **X\_train** → input features.
  + **y\_train** → real house prices.
* **fit()** means "learn from training data".

**➔ In simple words:**  
✅ Taught the model how **input features** relate to **house prices**.

### y\_pred = lr.predict(X\_test)

* **Predict** the house prices for the **testing data**.
* **X\_test** = inputs (features) of testing data.
* **y\_pred** = predicted house prices.

**➔ In simple words:**  
✅ Asked the model to **guess prices** based on new unseen data.

### ****Now Evaluation (Checking how good the model is):****

### print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

* **Mean Squared Error (MSE)** = how much **error** the model made (small MSE = better model).
* It measures the **average of square of errors**.

**➔ In simple words:**  
✅ Shows **how wrong** the model's predictions are.

### print("R-squared (R²):", r2\_score(y\_test, y\_pred))

* **R² (R-squared Score)** = measures **how well** the model fits the data.
* Value range:
  + **Close to 1** → very good model.
  + **Close to 0** → bad model.

**➔ In simple words:**  
✅ Shows **how much of the data** the model can **explain** correctly.

### ****Summary in One Shot:****

| **Step** | **What Happened** |
| --- | --- |
| Create model | lr = LinearRegression() |
| Train model | lr.fit(X\_train, y\_train) |
| Predict on test data | y\_pred = lr.predict(X\_test) |
| Evaluate model | print MSE and R² |

### ⭐ In Super Easy Language:

1. Create the model (empty machine).
2. Teach it using old data (training).
3. Test it on new data (testing).
4. Check how smart it became (evaluation).

4). **plt.figure(figsize=(6,6))**

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

**plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red')**

**plt.xlabel('Actual Prices')**

**plt.ylabel('Predicted Prices')**

**plt.title('Actual vs Predicted Prices')**

**plt.grid(True)**

**plt.show():**

Alright! 😄  
Let's again go **very simple and easy**:

## ****What this code is doing:****

✅ We are **drawing a graph** to **compare**:

* **Actual prices** (real house prices)  
  **with**
* **Predicted prices** (prices guessed by our model).

## ****Line-by-Line Explanation:****

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

* Set the **size of the graph** to 6x6 inches.
* (Square shape.)

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

* **Plot blue dots** where:
  + **x-axis** = real house prices (y\_test).
  + **y-axis** = predicted house prices (y\_pred).

🔵 Each blue dot = one house.

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

* **Draw a red straight line** (perfect prediction line).
* This line shows **"Perfect Predictions"** where:
  + **Actual = Predicted**.

❤️ If a blue dot is **close to red line**, prediction is **good**.

### plt.xlabel('Actual Prices')

* Label x-axis as **"Actual Prices"**.

### plt.ylabel('Predicted Prices')

* Label y-axis as **"Predicted Prices"**.

### plt.title('Actual vs Predicted Prices')

* Set the title of graph as **"Actual vs Predicted Prices"**.

### plt.grid(True)

* Add a grid to the graph (makes it easier to see).

### plt.show()

* **Display** the graph.

## ****Summary:****

| **Step** | **What Happened** |
| --- | --- |
| Create graph size | 6x6 inches |
| Plot blue dots | Actual vs Predicted prices |
| Draw red line | Perfect prediction line |
| Add labels, title, grid | Make graph understandable |
| Show the graph | Display it! |

## ****In Super Simple Language:****

✅ **We made a graph** to **see if our model's guesses (predictions) are close to the real house prices.**  
✅ **If the blue dots are near the red line → model is accurate.**

**Questions for VIVA:**Here are some **basic questions** that can be asked in a viva based on your Linear Regression model using the Boston Housing dataset:

### ****1. What is Linear Regression?****

**Answer:**  
Linear Regression is a statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (features). In simple linear regression, it assumes a linear relationship, meaning that the change in the dependent variable is proportional to the change in the independent variable(s).

### ****2. What is the Boston Housing Dataset used for?****

**Answer:**  
The Boston Housing Dataset is commonly used for predicting house prices based on various features of the houses, such as crime rate, average number of rooms, distance to employment centers, and more. It contains 506 samples and 14 feature variables, with the goal being to predict the median value of homes (**MEDV**).

### ****3. What is the purpose of splitting the dataset into training and testing sets?****

**Answer:**  
Splitting the dataset into training and testing sets allows us to train the model on one subset (training set) and evaluate its performance on a different subset (testing set). This helps to ensure that the model is able to generalize well to unseen data and not simply memorize the training data (overfitting).

### ****4. What does the**** train\_test\_split() ****function do?****

**Answer:**  
The train\_test\_split() function from the sklearn.model\_selection module is used to randomly split the dataset into training and testing subsets. By default, it splits the data into 80% for training and 20% for testing. This is important for evaluating the model’s performance on unseen data.

### ****5. What is the purpose of the**** fit() ****method in Linear Regression?****

**Answer:**  
The fit() method is used to train the linear regression model by finding the best-fitting line (or hyperplane in higher dimensions) that minimizes the error between the predicted and actual values of the target variable. It adjusts the model parameters (coefficients) to best match the training data.

### ****6. What does**** y\_pred = lr.predict(X\_test) ****do?****

**Answer:**  
The predict() method is used to make predictions on new, unseen data (in this case, the test set). It takes the test features (X\_test) as input and outputs the predicted target values (y\_pred), which are the predicted house prices for the test data.

### ****7. What are Mean Squared Error (MSE) and R-squared (R²)?****

**Answer:**

* **Mean Squared Error (MSE)** is a metric that measures the average squared difference between the predicted and actual values. It penalizes larger errors more than smaller ones.
* **R-squared (R²)** is a statistical measure that represents the proportion of the variance in the target variable that is explained by the independent variables. An **R² value close to 1** indicates that the model explains most of the variance, while an **R² value close to 0** suggests the model does not explain much of the variance.

### ****8. Why do we use**** drop() ****to remove the 'medv' column?****

**Answer:**  
The **'medv'** column is the target variable (the house price we are trying to predict), so it should not be used as an independent feature in the model. Therefore, we drop it from the features (X) and keep it as the dependent variable (y).

### ****9. What does the correlation heatmap show in your code?****

**Answer:**  
The correlation heatmap is used to visualize the relationships between different features in the dataset. The heatmap shows how strongly each feature is correlated with the others. Strong positive or negative correlations might suggest which features are important or redundant. This helps in understanding the dataset and selecting relevant features for modeling.

### ****10. What is the significance of the scatter plot in the evaluation part?****

**Answer:**  
The scatter plot compares the actual values (y\_test) with the predicted values (y\_pred). If the model is perfect, all points will lie along the red diagonal line (where actual = predicted). The closer the points are to this line, the better the model's predictions.

### ****11. What would happen if you used more or fewer features in the model?****

**Answer:**

* Using **more features** might increase the model's complexity and the risk of **overfitting**, especially if irrelevant or redundant features are included.
* Using **fewer features** could make the model simpler, but it might also **underfit** if important features are excluded, leading to poor predictions.

### ****12. How would you handle missing data in the dataset?****

**Answer:**  
To handle missing data, you could:

* **Remove** rows with missing values if they are not significant.
* **Impute** missing values using the mean, median, or mode of the feature.
* Use more advanced techniques such as predicting missing values using other features (e.g., using models like KNN or regression).

### ****13. What is the role of the**** LinearRegression() ****model in your code?****

**Answer:**  
The LinearRegression() model is an instance of the linear regression algorithm in the sklearn library. It is used to fit the model to the training data (X\_train and y\_train), and once the model is trained, it is used to make predictions on the test data (X\_test).

### ****14. How can you improve the performance of this model?****

**Answer:**  
Some ways to improve the model’s performance include:

* **Feature engineering**: Creating new features or transforming existing ones to make them more informative.
* **Regularization**: Using techniques like **Ridge** or **Lasso regression** to prevent overfitting.
* **Cross-validation**: Using k-fold cross-validation to assess model performance on different subsets of the data.
* **Data scaling**: Scaling the features (using standardization or normalization) to make the model more robust.