🔧 Problem Statement

Title: Boston Housing Price Prediction using Linear Regression via Deep Neural Network

Objective:

To predict median house prices in Boston suburbs using the Boston Housing dataset by implementing **linear regression** using a **deep neural network (DNN)** approach with TensorFlow/Keras.

📊 Dataset Description:

The **Boston Housing Dataset** includes 506 samples and 13 numerical/categorical features such as:

- CRIM: Crime rate per capita
- RM: Average number of rooms per dwelling
- LSTAT: % lower status of the population
- PTRATIO: Pupil-teacher ratio, etc.

Target Variable: MEDV (Median value of owner-occupied homes in $1000s)

🧠 Methodology:

Although this is a **linear regression** problem, we implement it using a **deep neural network** configured to learn a linear function:

Steps:

1. **Import Libraries** (TensorFlow, Keras, NumPy, Pandas, etc.)
2. **Load the Dataset** (from sklearn.datasets or keras.datasets)
3. **Preprocess the Data**
    - Normalize features
    - Train/test split
4. **Define the Neural Network Architecture**
    - Input layer matching number of features
    - Only 1 dense layer with **no activation** (linear regression)
    - Loss function: **Mean Squared Error**

- Optimizer: **SGD** or **Adam**

5. **Train the Model**
6. **Evaluate the Model** on test set
7. **Visualize** the predicted vs actual house prices

✅ Expected Outcome

A trained DNN model that approximates linear regression to predict house prices, evaluated using Mean Absolute Error (MAE) or Mean Squared Error (MSE).

📌 Goal of the Practical

You are using the **Boston Housing dataset** to predict house prices (regression problem) using a **neural network model built with TensorFlow/Keras**.

🔍 Step-by-step Explanation

🔹 1. Importing Required Libraries

```
import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import numpy as np

import matplotlib.pyplot as plt
```

✅ **Explanation**:

- tensorflow: Used to build and train the neural network.
- sklearn: For splitting data and normalization.
- matplotlib.pyplot: To visualize predictions.
- numpy: Numerical operations.

🔹 2. Load the Boston Housing Dataset

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.boston_housing.load_data()
```

✅ **Explanation**:

- Loads the dataset from Keras.
- x_train, x_test: Feature values (13 input variables like crime rate, room count, etc.).
- y_train, y_test: Target variable (house price in $1000s).

🔹 3. Normalize the Features

```
scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)

x_test_scaled = scaler.transform(x_test)
```

✅ **Explanation**:

- StandardScaler: Scales features to have **mean = 0 and std = 1**.
- Normalizing helps the model learn faster and improves accuracy.

🔹 4. Build the DNN Model (Linear Regression)

```
model = tf.keras.models.Sequential([

    tf.keras.layers.Input(shape=(13,), name='input-layer'),

    tf.keras.layers.Dense(100, name='hidden-layer-2'),

    tf.keras.layers.BatchNormalization(name='hidden-layer-3'),

    tf.keras.layers.Dense(50, name='hidden-layer-4'),

    tf.keras.layers.Dense(1, name='output-layer')

])
```

✅ **Explanation**:

- Sequential: Stacks layers in order.
- Input(shape=(13,)): Specifies that input has 13 features.
- Dense(100): Fully connected hidden layer with 100 neurons.
- BatchNormalization: Stabilizes and speeds up training.

- Dense(50): Another hidden layer with 50 neurons.
- Dense(1): Output layer with **1 neuron** (for predicting a single number = house price).

💡 Even though it's called a DNN, this is used for **regression**, not classification.

🔹 5. Compile the Model

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

✅ **Explanation**:

- optimizer='adam': Adaptive optimizer (good default choice).
- loss='mse': Mean Squared Error (used for regression).
- metrics=['mae']: Mean Absolute Error shown during training.

🔹 6. Train the Model

model.fit(x_train_scaled, y_train, epochs=100, validation_split=0.2)

✅ **Explanation**:

- Trains the model for 100 epochs.
- Uses 20% of training data for validation.
- Adjusts model weights to minimize error.

🔹 7. Evaluate the Model

loss, mae = model.evaluate(x_test_scaled, y_test)

print(f"Test MAE: {mae}")

✅ **Explanation**:

- Evaluates model performance on **unseen test data**.
- MAE (Mean Absolute Error) tells us how far predictions are from actual values on average.

🔹 8. Make and Plot Predictions

y_pred = model.predict(x_test_scaled)

```
plt.scatter(y_test, y_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--r')

plt.show()
```

✅ **Explanation**:

- model.predict(): Makes predictions on test data.
- scatter plot: Compares actual vs predicted prices.
- Red line: Ideal line where prediction = actual. Closer points are to the line → better model.

✅ Summary for Viva or Written:

- You used a DNN to perform linear regression.
- The dataset had 13 features; the model had input → hidden layers → output.
- Used MSE loss function and Adam optimizer.
- Normalized data before training.
- Evaluated with MAE and visualized predictions.

🔍 What the columns mean:

Layer Type	Output Shape	Description
InputLayer	(None, 13)	Accepts input of 13 features
Dense (100)	(None, 100)	100 neurons, connected to all 13 inputs
BatchNormalization	(None, 100)	Normalizes 100 outputs
Dense (50)	(None, 50)	50 neurons
Dense (1)	(None, 1)	Final output: 1 value (price prediction)