```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from google.colab import files
import io
```

```
# Step 1: Upload Dataset
uploaded = files.upload()
file_name = list(uploaded.keys())[0]
df = pd.read_csv(io.BytesIO(uploaded[file_name]))

# Display first few rows
print(df.head())
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Boston.csv to Boston.csv
   Unnamed: 0     crim    zn  indus  chas    nox     rm   age     dis  rad  \
0           1  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1
1           2  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2
2           3  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2
3           4  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3
4           5  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3

   tax  ptratio   black  lstat  medv
0  296     15.3  396.90   4.98  24.0
1  242     17.8  396.90   9.14  21.6
2  242     17.8  392.83   4.03  34.7
```

```
# Step 2: Data Preprocessing
# Separate features and target variable (assuming 'medv' is the target column)
X = df.drop(columns=['medv'])
y = df['medv']

# Split dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize data (important for DNNs)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Step 3: Build the Deep Neural Network (DNN)
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),  # Input Layer
    Dense(32, activation='relu'),  # Hidden Layer
    Dense(1)  # Output Layer (Linear Activation)
])

# Compile model (Mean Squared Error is good for regression)
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse', metrics=['mae'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Step 4: Train the Model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2, verbose=1)
```

```
Epoch 1/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 377.1385 - mae: 16.0154 - val_loss: 36.6708 - val_mae: 4.3078
Epoch 2/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 29.1718 - mae: 3.7453 - val_loss: 24.5379 - val_mae: 3.2361
Epoch 3/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 18.6144 - mae: 3.0768 - val_loss: 20.5844 - val_mae: 3.0258
Epoch 4/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 14.4939 - mae: 2.7486 - val_loss: 17.9542 - val_mae: 2.9053
```

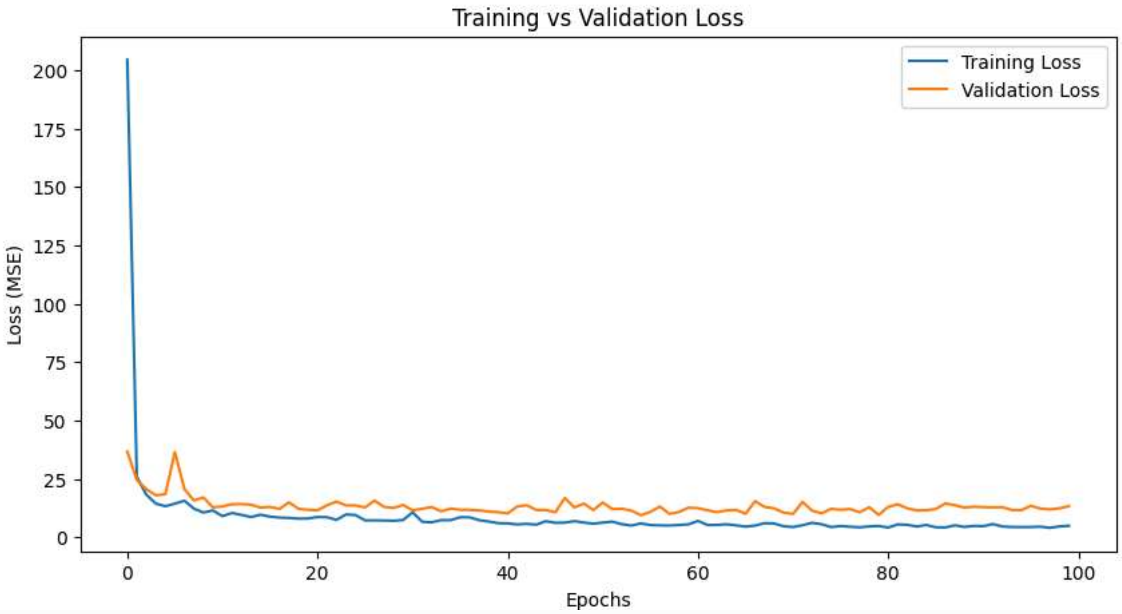
```
Epoch 5/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 10.3349 - mae: 2.4667 - val_loss: 18.4358 - val_mae: 2.8554
Epoch 6/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 13.5411 - mae: 2.7098 - val_loss: 36.5118 - val_mae: 4.8896
Epoch 7/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.5023 - mae: 3.3168 - val_loss: 20.6692 - val_mae: 3.2011
Epoch 8/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 13.2875 - mae: 2.6805 - val_loss: 15.7324 - val_mae: 2.6973
Epoch 9/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.9623 - mae: 2.3011 - val_loss: 17.0449 - val_mae: 2.9953
Epoch 10/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 14.2324 - mae: 2.6401 - val_loss: 12.8198 - val_mae: 2.4961
Epoch 11/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 9.7990 - mae: 2.3959 - val_loss: 13.1965 - val_mae: 2.6431
Epoch 12/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 11.2616 - mae: 2.5019 - val_loss: 14.1391 - val_mae: 2.7003
Epoch 13/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.2004 - mae: 2.2464 - val_loss: 14.2178 - val_mae: 2.7588
Epoch 14/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.1269 - mae: 2.0064 - val_loss: 13.9802 - val_mae: 2.4873
Epoch 15/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 10.4420 - mae: 2.4146 - val_loss: 12.6913 - val_mae: 2.5738
Epoch 16/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 9.2549 - mae: 2.2729 - val_loss: 12.9785 - val_mae: 2.4575
Epoch 17/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 6.8022 - mae: 1.9667 - val_loss: 12.1422 - val_mae: 2.4750
Epoch 18/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 8.2326 - mae: 2.1455 - val_loss: 14.9554 - val_mae: 2.7512
Epoch 19/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 8.4942 - mae: 2.0474 - val_loss: 12.1552 - val_mae: 2.5075
Epoch 20/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 8.1659 - mae: 2.2574 - val_loss: 11.7733 - val_mae: 2.3965
Epoch 21/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 8.5520 - mae: 2.2016 - val_loss: 11.5364 - val_mae: 2.4401
Epoch 22/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 8.2382 - mae: 2.1006 - val_loss: 13.6963 - val_mae: 2.5731
Epoch 23/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 6.7328 - mae: 1.9022 - val_loss: 15.2535 - val_mae: 2.8218
Epoch 24/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.4213 - mae: 2.0161 - val_loss: 13.5929 - val_mae: 2.5597
Epoch 25/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.4943 - mae: 2.3258 - val_loss: 13.6395 - val_mae: 2.5682
Epoch 26/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 5.6861 - mae: 1.8445 - val_loss: 12.7524 - val_mae: 2.6567
Epoch 27/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.4869 - mae: 2.1499 - val_loss: 15.6871 - val_mae: 2.8807
Epoch 28/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.1814 - mae: 2.1523 - val_loss: 12.9767 - val_mae: 2.5830
Epoch 29/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 7.3871 - mae: 1.9687 - val_loss: 12.5529 - val_mae: 2.5842
```

```
# Step 5: Evaluate Model Performance
test_loss, test_mae = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {test_loss:.2f}")
print(f"Test MAE: {test_mae:.2f}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 10.0008 - mae: 2.3930
Test Loss (MSE): 11.84
Test MAE: 2.51
```

```
# Step 6: Plot Training History
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.title('Training vs Validation Loss')
plt.show()
```




```
1 # Step 7: Make Predictions
2 y_pred = model.predict(X_test).flatten()  # Convert predictions to 1D array
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

```
1 import seaborn as sns
2 # Step 8: Regression Plot (Actual vs Predicted)
3 plt.figure(figsize=(8,6))
4 sns.regplot(x=y_test, y=y_pred, scatter_kws={"color": "blue"}, line_kws={"color": "red"})
5 plt.xlabel("Actual House Prices")
6 plt.ylabel("Predicted House Prices")
7 plt.title("Regression Plot: Actual vs Predicted Prices")
8 plt.show()
```