Boston Housing Price Prediction

1. nhập thêm các thư viện cần thiết

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

- `pandas`: Để thao tác và phân tích dữ liệu.
- `numpy`: Đối với các phép toán số.
- `matplotlib`: Để trực quan hóa dữ liệu.
- `scikit-learn`:
  - `train_test_split`: Để chia dữ liệu thành các tập huấn luyện và kiểm tra.
  - `StandardScaler`: Để chuẩn hóa các tính năng.
  - `Regression models`: (Tuyến tính, Cây quyết định và Rừng ngẫu nhiên) và các chỉ số hiệu suất.
- `tensorflow`: Để tạo và huấn luyện mô hình Perceptron đa lớp (MLP).

2. Tải cơ sở dữ liệu nhà ở Bostont

```
data = pd.read_csv("BostonHousing.csv")

#chia dữ liệu thành các tính năng và mục tiêu
X = data.drop(columns=["medv"])
y = data["medv"]
print("dataset chạy thành công")

print(data)
```

```
dataset chạy thành công
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax
\
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296

1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
```

```
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242

3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222

4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222

..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...

501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273

502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273

503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273

504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273

505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273


     ptratio       b  lstat  medv
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       ...     ...    ...   ...
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

- tập dữ liệu được tải xuống từ tệp file .CSV,chia dữ liệu thành các tính năng (`X`) và biến mục tiêu (`y`).
- note: `BostonHousing.csv` lưu ý nhớ coi file path trước khi chạy
- `X`: Independent variables (features).
- `y`: Dependent variable (target house price).

3. Chia dữ liệu thành các tập huấn luyện và kiểm tra

- Chúng tôi chia tập dữ liệu thành tập con huấn luyện (80%) và tập con kiểm tra (20%).

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

X_train.shape,y_train.shape
```

```
((404, 13), (404,))

X_test.shape,y_test.shape

((102, 13), (102,))
```

4. Chia tỷ lệ các tính năng

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- `StandardScaler`: các tính năng của thang đo có giá trị trung bình là 0 và độ lệch chuẩn là 1, cải thiện hiệu suất của mô hình.

5. Xây dựng và huấn luyện mô hình Perceptron đa lớp (MLP)

```
mlp_model = Sequential([
    Dense(32, activation='relu', input_dim=X_train.shape[1]),
    Dense(16, activation='relu'),
    Dense(1, activation='linear')
])

c:\Users\nguye\OneDrive\Desktop\Practice-exercise-4-Multilayer-
Perceptron-Regression-Exercise-Predicting-House-Prices\myvenv\lib\
site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

- **Lớp đầu vào**: Khớp với số lượng đối tượng trong tập dữ liệu.
- **Lớp ẩn**:

```
    - Lớp 1: 32 nơ-ron, kích hoạt ReLU.

    - Lớp 2: 16 nơ-ron, kích hoạt ReLU.
```

- **Lớp đầu ra**: 1 nơ-ron, kích hoạt tuyến tính (đối với các tác vụ hồi quy).

```
mlp_model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mae'])
```

- **Optimizer**: Adam (adaptive learning rate optimization).
- **Loss Function**: Mean Squared Error (MSE) for regression.

- **Metric**: Mean Absolute Error (MAE) to monitor performance during training.

```
history = mlp_model.fit(X_train_scaled, y_train, epochs=100,
batch_size=32, validation_data=(X_test_scaled, y_test))

Epoch 1/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - loss: 651.2151 - mae:
23.5192 - val_loss: 535.2134 - val_mae: 21.4877
Epoch 2/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 607.9666 - mae:
22.7803 - val_loss: 509.0358 - val_mae: 20.9620
Epoch 3/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 560.2676 - mae:
21.8475 - val_loss: 483.5324 - val_mae: 20.4198
Epoch 4/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 565.5593 - mae:
22.0460 - val_loss: 453.8782 - val_mae: 19.7742
Epoch 5/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 491.6194 - mae:
20.5224 - val_loss: 420.1325 - val_mae: 19.0116
Epoch 6/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 446.8846 - mae:
19.4337 - val_loss: 379.8843 - val_mae: 18.0616
Epoch 7/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 436.1978 - mae:
19.0843 - val_loss: 333.0276 - val_mae: 16.8725
Epoch 8/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 360.4162 - mae:
17.4046 - val_loss: 281.3918 - val_mae: 15.4233
Epoch 9/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 312.6243 - mae:
16.0607 - val_loss: 227.7730 - val_mae: 13.6987
Epoch 10/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 232.1521 - mae:
13.5650 - val_loss: 178.1972 - val_mae: 11.8683
Epoch 11/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 181.2276 - mae:
11.6213 - val_loss: 136.3117 - val_mae: 10.1228
Epoch 12/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 144.8631 - mae: 9.7759
- val_loss: 104.2893 - val_mae: 8.4426
Epoch 13/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 102.5768 - mae: 8.1258
- val_loss: 81.7736 - val_mae: 7.0916
Epoch 14/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 87.3960 - mae: 7.2777
- val_loss: 65.6077 - val_mae: 6.0872
Epoch 15/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 76.8984 - mae: 6.5276
- val_loss: 53.6206 - val_mae: 5.3558
```

```
Epoch 16/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 58.3953 - mae: 5.6619
- val_loss: 45.0536 - val_mae: 4.7623
Epoch 17/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 52.6483 - mae: 5.2966
- val_loss: 39.6363 - val_mae: 4.3841
Epoch 18/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 34.8642 - mae: 4.5656
- val_loss: 36.1519 - val_mae: 4.1331
Epoch 19/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 33.2423 - mae: 4.3611
- val_loss: 33.9484 - val_mae: 3.9609
Epoch 20/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 28.3041 - mae: 3.9216
- val_loss: 32.1574 - val_mae: 3.8410
Epoch 21/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 30.3481 - mae: 4.0539
- val_loss: 30.8103 - val_mae: 3.7695
Epoch 22/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 28.5744 - mae: 3.9725
- val_loss: 29.8177 - val_mae: 3.7140
Epoch 23/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 26.9627 - mae: 3.9562
- val_loss: 29.0372 - val_mae: 3.6652
Epoch 24/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 27.7105 - mae: 3.9352
- val_loss: 28.3435 - val_mae: 3.6237
Epoch 25/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 32.6684 - mae: 4.0072
- val_loss: 27.5410 - val_mae: 3.5789
Epoch 26/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 25.5609 - mae: 3.6868
- val_loss: 26.8885 - val_mae: 3.5391
Epoch 27/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 24.1779 - mae: 3.7610
- val_loss: 26.2681 - val_mae: 3.4908
Epoch 28/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 25.3812 - mae: 3.7704
- val_loss: 25.6948 - val_mae: 3.4589
Epoch 29/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 21.4308 - mae: 3.4997
- val_loss: 25.0883 - val_mae: 3.4138
Epoch 30/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 26.3029 - mae: 3.6750
- val_loss: 24.5194 - val_mae: 3.3761
Epoch 31/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 21.4757 - mae: 3.5212
- val_loss: 24.1599 - val_mae: 3.3361
Epoch 32/100
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 26.1672 - mae: 3.7468
- val_loss: 23.6142 - val_mae: 3.3140
Epoch 33/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 21.0969 - mae: 3.5173
- val_loss: 23.1688 - val_mae: 3.2698
Epoch 34/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 20.7601 - mae: 3.3754
- val_loss: 22.8007 - val_mae: 3.2433
Epoch 35/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 22.2007 - mae: 3.4715
- val_loss: 22.3865 - val_mae: 3.2078
Epoch 36/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 19.7752 - mae: 3.3441
- val_loss: 21.9668 - val_mae: 3.1764
Epoch 37/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 17.9741 - mae: 3.2136
- val_loss: 21.6667 - val_mae: 3.1479
Epoch 38/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 21.2042 - mae: 3.4136
- val_loss: 21.3193 - val_mae: 3.1199
Epoch 39/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 16.3023 - mae: 3.1215
- val_loss: 20.9401 - val_mae: 3.0732
Epoch 40/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 18.7172 - mae: 3.1105
- val_loss: 20.6678 - val_mae: 3.0429
Epoch 41/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 16.4926 - mae: 3.0135
- val_loss: 20.4231 - val_mae: 3.0121
Epoch 42/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 19.2094 - mae: 3.1801
- val_loss: 20.0487 - val_mae: 2.9829
Epoch 43/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 20.4164 - mae: 3.3344
- val_loss: 19.7020 - val_mae: 2.9446
Epoch 44/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 18.6711 - mae: 3.1278
- val_loss: 19.4437 - val_mae: 2.9130
Epoch 45/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 16.4402 - mae: 3.0116
- val_loss: 19.1865 - val_mae: 2.8958
Epoch 46/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 18.2101 - mae: 3.0512
- val_loss: 18.9128 - val_mae: 2.8550
Epoch 47/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 16.4251 - mae: 2.9980
- val_loss: 18.6696 - val_mae: 2.8234
Epoch 48/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 17.8169 - mae: 3.1168
```

```
- val_loss: 18.4680 - val_mae: 2.8014
Epoch 49/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 19.2464 - mae: 3.1376
- val_loss: 18.2621 - val_mae: 2.7784
Epoch 50/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 17.3785 - mae: 3.0490
- val_loss: 18.0665 - val_mae: 2.7552
Epoch 51/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 16.3323 - mae: 2.9896
- val_loss: 17.8422 - val_mae: 2.7346
Epoch 52/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 14.8022 - mae: 2.9186
- val_loss: 17.6690 - val_mae: 2.7086
Epoch 53/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 14.9452 - mae: 2.8516
- val_loss: 17.4803 - val_mae: 2.7042
Epoch 54/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 15.7706 - mae: 2.9153
- val_loss: 17.2767 - val_mae: 2.6756
Epoch 55/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 14.9909 - mae: 2.8723
- val_loss: 17.1296 - val_mae: 2.6656
Epoch 56/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 16.5936 - mae: 2.9703
- val_loss: 16.9557 - val_mae: 2.6352
Epoch 57/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 14.0020 - mae: 2.6841
- val_loss: 16.8044 - val_mae: 2.6241
Epoch 58/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 18.5100 - mae: 2.9640
- val_loss: 16.6040 - val_mae: 2.6061
Epoch 59/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 16.4949 - mae: 2.9622
- val_loss: 16.5175 - val_mae: 2.5917
Epoch 60/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 15.9313 - mae: 2.8706
- val_loss: 16.3750 - val_mae: 2.5792
Epoch 61/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 13.5293 - mae: 2.6272
- val_loss: 16.2649 - val_mae: 2.5599
Epoch 62/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 14.8530 - mae: 2.7938
- val_loss: 16.0541 - val_mae: 2.5532
Epoch 63/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 12.5100 - mae: 2.5669
- val_loss: 15.9834 - val_mae: 2.5385
Epoch 64/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 16.6211 - mae: 2.9603
- val_loss: 15.9013 - val_mae: 2.5328
```

```
Epoch 65/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 14.0919 - mae: 2.6546
- val_loss: 15.7207 - val_mae: 2.5289
Epoch 66/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 15.1709 - mae: 2.7596
- val_loss: 15.6565 - val_mae: 2.5170
Epoch 67/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 15.2452 - mae: 2.7207
- val_loss: 15.4887 - val_mae: 2.5062
Epoch 68/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 13.8933 - mae: 2.6306
- val_loss: 15.4209 - val_mae: 2.4930
Epoch 69/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 14.8634 - mae: 2.7196
- val_loss: 15.2740 - val_mae: 2.5148
Epoch 70/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 14.6009 - mae: 2.6027
- val_loss: 15.2146 - val_mae: 2.5012
Epoch 71/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 13.7478 - mae: 2.6479
- val_loss: 15.0372 - val_mae: 2.4952
Epoch 72/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 11.2899 - mae: 2.5167
- val_loss: 14.8941 - val_mae: 2.4837
Epoch 73/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 14.0380 - mae: 2.6000
- val_loss: 14.7652 - val_mae: 2.4781
Epoch 74/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 15.3237 - mae: 2.6942
- val_loss: 14.7095 - val_mae: 2.4846
Epoch 75/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 11.8025 - mae: 2.5840
- val_loss: 14.6373 - val_mae: 2.4777
Epoch 76/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 9.6906 - mae: 2.3246 -
val_loss: 14.5681 - val_mae: 2.4644
Epoch 77/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 10.3089 - mae: 2.3516
- val_loss: 14.4873 - val_mae: 2.4867
Epoch 78/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 14.6295 - mae: 2.6818
- val_loss: 14.3288 - val_mae: 2.4749
Epoch 79/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 13.0592 - mae: 2.6161
- val_loss: 14.2852 - val_mae: 2.4636
Epoch 80/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 10.7335 - mae: 2.2819
- val_loss: 14.1507 - val_mae: 2.4565
Epoch 81/100
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 11.1453 - mae: 2.3993
- val_loss: 14.1260 - val_mae: 2.4742
Epoch 82/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 12.4765 - mae: 2.5344
- val_loss: 14.0074 - val_mae: 2.4477
Epoch 83/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 11.3507 - mae: 2.3774
- val_loss: 13.9082 - val_mae: 2.4577
Epoch 84/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.6632 - mae: 2.4349
- val_loss: 13.8713 - val_mae: 2.4568
Epoch 85/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.2815 - mae: 2.3327
- val_loss: 13.8235 - val_mae: 2.4556
Epoch 86/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 13.4177 - mae: 2.4761
- val_loss: 13.7198 - val_mae: 2.4528
Epoch 87/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 12.5036 - mae: 2.4759
- val_loss: 13.7004 - val_mae: 2.4485
Epoch 88/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 9.7793 - mae: 2.2868 -
val_loss: 13.6443 - val_mae: 2.4620
Epoch 89/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 10.7813 - mae: 2.3745
- val_loss: 13.5436 - val_mae: 2.4391
Epoch 90/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 10.5225 - mae: 2.3319
- val_loss: 13.4543 - val_mae: 2.4346
Epoch 91/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 11.9096 - mae: 2.4444
- val_loss: 13.3750 - val_mae: 2.4237
Epoch 92/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.8327 - mae: 2.4046
- val_loss: 13.3345 - val_mae: 2.4255
Epoch 93/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 11.5533 - mae: 2.4038
- val_loss: 13.2479 - val_mae: 2.4157
Epoch 94/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 11.8179 - mae: 2.4225
- val_loss: 13.2092 - val_mae: 2.4099
Epoch 95/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 11.3988 - mae: 2.3925
- val_loss: 13.2314 - val_mae: 2.4286
Epoch 96/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.3585 - mae: 2.3280
- val_loss: 13.1357 - val_mae: 2.4151
Epoch 97/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.1213 - mae: 2.3683
```

```
- val_loss: 13.0416 - val_mae: 2.4035
Epoch 98/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 10.8135 - mae: 2.3816
- val_loss: 12.9739 - val_mae: 2.3962
Epoch 99/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 10.3414 - mae: 2.2812
- val_loss: 12.9556 - val_mae: 2.4086
Epoch 100/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 12.9422 - mae: 2.4091
- val_loss: 12.8965 - val_mae: 2.4065
```

- **Epochs**: Số lần đi qua dữ liệu huấn luyện.
- **Batch Size**: Số lượng mẫu được xử lý trước khi cập nhật mô hình.
- **Validation Data**: Theo dõi hiệu suất trên tập kiểm tra trong quá trình đào tạo.

6. Đánh giá mô hình MLP

- Chúng tôi đánh giá MLP đã được đào tạo trên tập kiểm tra và trực quan hóa các dự đoán.

```
test_loss, test_mae = mlp_model.evaluate(X_test_scaled, y_test)
print(f'Test Loss (MSE): {test_loss}')
print(f'Test Mean Absolute Error (MAE): {test_mae}')

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 9.6963 - mae: 2.3081
Test Loss (MSE): 12.896533012390137
Test Mean Absolute Error (MAE): 2.406492233276367
```

- Đánh giá mô hình MLP bằng MSE và MAE trên dữ liệu thử nghiệm

```
y_pred_mlp = mlp_model.predict(X_test_scaled) #prediction

WARNING:tensorflow:5 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x000001F90D349240> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/stepWARNING:tensorflow:6 out of the
last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x000001F90D349240> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors
with different shapes, (3) passing Python objects instead of tensors.
```

```
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

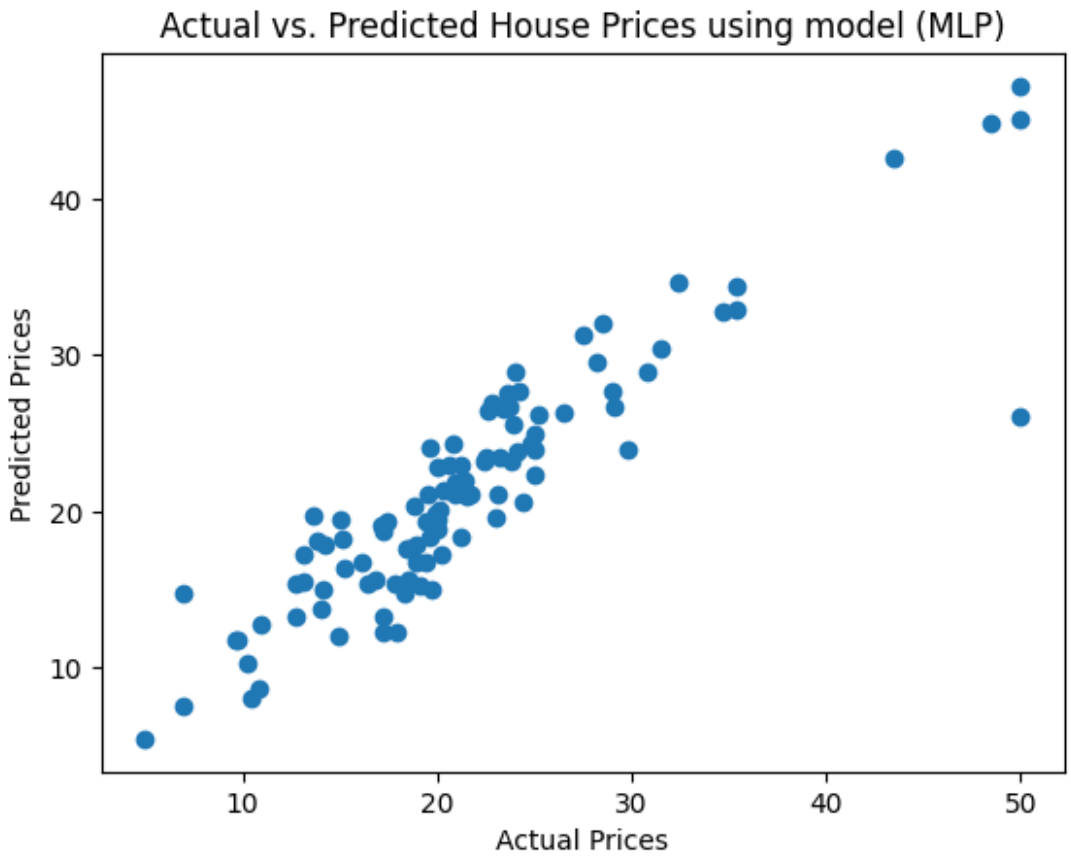
- **in kết quả**
- In ra MSE và $R2$ thể hiện mức độ tốt của mô hình MLP

```
y_pred_mlp = y_pred_mlp.flatten()#maˆng 1D

#tính toán các chiˆ sôˊ
mse_mlp = mean_squared_error(y_test, y_pred_mlp)
r2_mlp = r2_score(y_test, y_pred_mlp)

#in ra kêˊt quaˆ
print("MLP Model Evaluation:")
print(f"Mean Squared Error (MSE): {mse_mlp:.2f}")
print(f"R² Score: {r2_mlp:.2f}")
```

```
MLP Model Evaluation:
Mean Squared Error (MSE): 12.90
R² Score: 0.82
```

- `y_pred_mlp`: Dự đoán của mô hình MLP

```
plt.scatter(y_test, y_pred_mlp)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs. Predicted House Prices using model (MLP)')
plt.show()
```



- Vẽ biểu đồ phân tán để so sánh giá nhà thực tế `(Actual Prices)` và dự đoán `(Predicted Prices)`.

7. So sánh với các mô hình hồi quy khác

- Chúng tôi so sánh mô hình MLP với hồi quy tuyến tính(`Linear Regression`), hồi quy cây quyết định(`Decision Tree`) và hồi quy rừng ngẫu nhiên(`Random Forest`).

7.1 Linear Regression

```
#Linear Regression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)
print("predicted value:")
print(y_pred_lr)
```

```
predicted value:
[28.99672362 36.02556534 14.81694405 25.03197915 18.76987992
23.25442929
 17.66253818 14.34119    23.01320703 20.63245597 24.90850512
18.63883645
 -6.08842184 21.75834668 19.23922576 26.19319733 20.64773313
5.79472718
```

```
 40.50033966 17.61289074 27.24909479 30.06625441 11.34179277
24.16077616
 17.86058499 15.83609765 22.78148106 14.57704449 22.43626052
19.19631835
 22.43383455 25.21979081 25.93909562 17.70162434 16.76911711
16.95125411
 31.23340153 20.13246729 23.76579011 24.6322925  13.94204955
32.25576301
 42.67251161 17.32745046 27.27618614 16.99310991 14.07009109
25.90341861
 20.29485982 29.95339638 21.28860173 34.34451856 16.04739105
26.22562412
 39.53939798 22.57950697 18.84531367 32.72531661 25.0673037
12.88628956
 22.68221908 30.48287757 31.52626806 15.90148607 20.22094826
16.71089812
 20.52384893 25.96356264 30.61607978 11.59783023 20.51232627
27.48111878
 11.01962332 15.68096344 23.79316251  6.19929359 21.6039073
41.41377225
 18.76548695  8.87931901 20.83076916 13.25620627 20.73963699
9.36482222
 23.22444271 31.9155003  19.10228271 25.51579303 29.04256769
20.14358566
 25.5859787   5.70159447 20.09474756 14.95069156 12.50395648
20.72635294
 24.73957161 -0.164237   13.68486682 16.18359697 22.27621999
24.47902364]
```

7.2 Decision Tree

```
#Decision Tree Regression
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)
print("predicted value:")
print(y_pred_dt)
```

```
predicted value:
[28.1 33.1 17.3 22.  23.2 18.5 16.6 16.6 22.7 22.  20.5 27.1  8.4 21.4
 18.5 23.9 18.8 10.5 46.  13.  23.1 24.4 13.6 22.  14.5 11.7 21.  13.5
 19.4 20.7 18.8 23.1 10.4 16.2 13.3 13.1 33.4 18.5 20.4 24.8 19.8 28.4
 46.  19.3 22.  13.  14.9 24.1 17.7 32.  21.7 36.1 16.7 28.4 43.1 18.5
 15.2 22.8 22.  22.5 24.5 33.  29.4 19.3 26.6 14.4 13.  22.9 22.8 14.1
 21.8 28.7  8.3 18.6 21.5 10.5 19.8 50.  13.3  8.1 21.2 16.3 19.4 10.5
 14.5 29.9 14.8 23.1 22.9 18.  23.3  8.8 19.2 17.6 16.2 19.3 50.  16.3
 11.7 16.3 19.  26.4]
```

7.3 Random Forest Regression

```
#Random Forest Regression
rf = RandomForestRegressor(random_state=42, n_estimators=100)
rf.fit(X_train_scaled, y_train)
y_pred_rf = rf.predict(X_test_scaled)
print("predicted value:")
print(y_pred_rf)
```

```
predicted value:
[22.839 30.689 16.317 23.51  16.819 21.425 19.358 15.62  21.091 21.073
 20.028 19.298  8.611 21.456 19.378 25.453 19.187  8.538 46.132 14.536
 24.728 23.996 14.509 23.847 14.363 14.796 21.121 13.663 19.535 21.29
 19.45  23.392 29.3   20.338 14.596 15.594 33.835 19.129 20.915 24.376
 19.286 29.61  46.108 19.428 22.653 13.676 15.037 24.321 18.689 28.821
 21.107 33.823 16.502 25.763 44.922 21.994 15.416 32.032 22.596 20.296
 25.597 33.928 28.134 18.551 26.745 17.568 13.992 23.195 29.022 15.663
 21.064 27.426 10.06  21.569 21.956  7.084 19.905 46.154 11.274 12.981
 21.288 12.501 19.579  9.392 20.76  27.283 15.383 23.398 23.628 17.617
 21.681  8.019 19.616 18.714 22.592 19.786 41.733 12.726 12.632 13.066
 20.603 23.902]
```

- `Mean Squared Error`(MSE): Đo chênh lệch bình phương trung bình giữa giá trị thực tế và giá trị dự đoán.
- `R-squared`: Cho biết mô hình giải thích phương sai của biến mục tiêu tốt như thế nào.
- So sánh MLP với các mô hình truyền thống (Hồi quy tuyến tính`(Linear Regression)`, Cây quyết định`(Decision Tree)`, Rừng ngẫu nhiên`(Random Forest)`)

```
models = {
    "MLP": y_pred_mlp.flatten(),
    "Linear Regression": y_pred_lr,
    "Decision Tree": y_pred_dt,
    "Random Forest": y_pred_rf
}

for model_name, y_pred in models.items():
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{model_name} -> MSE: {mse:.2f}, r2-squared: {r2:.2f}")
```

```
MLP -> MSE: 12.90, r2-squared: 0.82
Linear Regression -> MSE: 24.29, r2-squared: 0.67
Decision Tree -> MSE: 10.42, r2-squared: 0.86
Random Forest -> MSE: 7.91, r2-squared: 0.89
```

source code github: https://github.com/HUyEsona/Practice-exercise-4-Multilayer-Perceptron-Regression-Exercise-Predicting-House-Prices.git