5ntiyayph

May 5, 2025

1 Boston Housing Price Prediction with Keras

1.0.1 Step 1: Import Required Libraries

[1]:
```
# Data processing and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Sklearn tools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# TensorFlow Keras
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
```

1.0.2 Step 2: Load and View the Dataset

[2]:
```
df = pd.read_csv("datasets/boston_housing.csv")
df.head()
```

[2]:
```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
```

```
4  396.90   5.33  36.2
```

1.0.3 Step 3: Split Features and Target

[3]:
```
X = df.drop(['MEDV'], axis=1)
y = df['MEDV']
```

1.0.4 Step 4: Scale the Features

[4]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1.0.5 Step 5: Split into Training and Testing Sets

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

1.0.6 Step 6: Build the Neural Network Model

[6]:
```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)
])
```

1.0.7 Step 7: Compile the Model

[7]:
```
model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)
```

1.0.8 Step 8: Train the Model

[8]:
```
history = model.fit(
    X_train, y_train,
    epochs=50,
    validation_split=0.1,
    verbose=1
)
```

```
Epoch 1/50
12/12            1s 14ms/step -
loss: 581.5626 - mae: 22.2346 - val_loss: 552.5659 - val_mae: 21.9875
Epoch 2/50
12/12            0s 3ms/step - loss:
573.7921 - mae: 22.1359 - val_loss: 512.7275 - val_mae: 21.1069
Epoch 3/50
12/12            0s 4ms/step - loss:
500.3716 - mae: 20.4333 - val_loss: 465.1572 - val_mae: 20.0022
Epoch 4/50
12/12            0s 3ms/step - loss:
481.3070 - mae: 19.9221 - val_loss: 403.9552 - val_mae: 18.5018
Epoch 5/50
12/12            0s 3ms/step - loss:
411.9416 - mae: 18.0575 - val_loss: 329.0402 - val_mae: 16.5079
Epoch 6/50
12/12            0s 4ms/step - loss:
305.0258 - mae: 14.9884 - val_loss: 245.1579 - val_mae: 13.9974
Epoch 7/50
12/12            0s 4ms/step - loss:
213.0809 - mae: 12.1972 - val_loss: 166.3586 - val_mae: 11.0665
Epoch 8/50
12/12            0s 3ms/step - loss:
131.4219 - mae: 9.2341 - val_loss: 107.1288 - val_mae: 8.2478
Epoch 9/50
12/12            0s 4ms/step - loss:
104.0689 - mae: 8.0161 - val_loss: 73.5620 - val_mae: 6.1589
Epoch 10/50
12/12            0s 4ms/step - loss:
74.0127 - mae: 6.9324 - val_loss: 57.3545 - val_mae: 5.1876
Epoch 11/50
12/12            0s 3ms/step - loss:
60.5930 - mae: 6.2900 - val_loss: 48.4163 - val_mae: 4.6282
Epoch 12/50
12/12            0s 3ms/step - loss:
43.9501 - mae: 5.3162 - val_loss: 43.9488 - val_mae: 4.2788
Epoch 13/50
12/12            0s 3ms/step - loss:
32.3117 - mae: 4.5005 - val_loss: 41.8687 - val_mae: 4.1557
Epoch 14/50
12/12            0s 5ms/step - loss:
31.3455 - mae: 4.3628 - val_loss: 40.8295 - val_mae: 4.1188
Epoch 15/50
12/12            0s 5ms/step - loss:
27.0926 - mae: 3.9964 - val_loss: 40.1871 - val_mae: 4.1267
Epoch 16/50
12/12            0s 4ms/step - loss:
24.4301 - mae: 3.6270 - val_loss: 39.1458 - val_mae: 4.1493
```

```
Epoch 17/50
12/12            0s 4ms/step - loss:
23.1145 - mae: 3.7278 - val_loss: 38.6148 - val_mae: 4.1377
Epoch 18/50
12/12            0s 3ms/step - loss:
25.4708 - mae: 3.7476 - val_loss: 38.0797 - val_mae: 4.1542
Epoch 19/50
12/12            0s 3ms/step - loss:
25.0564 - mae: 3.6167 - val_loss: 37.5187 - val_mae: 4.1233
Epoch 20/50
12/12            0s 4ms/step - loss:
21.3877 - mae: 3.4222 - val_loss: 37.5324 - val_mae: 4.0762
Epoch 21/50
12/12            0s 3ms/step - loss:
22.5247 - mae: 3.4277 - val_loss: 36.4104 - val_mae: 3.9675
Epoch 22/50
12/12            0s 3ms/step - loss:
19.5505 - mae: 3.2973 - val_loss: 36.0621 - val_mae: 3.9226
Epoch 23/50
12/12            0s 4ms/step - loss:
20.2581 - mae: 3.4173 - val_loss: 35.2850 - val_mae: 3.8730
Epoch 24/50
12/12            0s 4ms/step - loss:
21.1851 - mae: 3.3776 - val_loss: 34.7377 - val_mae: 3.8389
Epoch 25/50
12/12            0s 4ms/step - loss:
18.5439 - mae: 3.1878 - val_loss: 34.5024 - val_mae: 3.8283
Epoch 26/50
12/12            0s 3ms/step - loss:
21.4263 - mae: 3.1442 - val_loss: 34.4287 - val_mae: 3.7952
Epoch 27/50
12/12            0s 3ms/step - loss:
17.6864 - mae: 3.0310 - val_loss: 34.1468 - val_mae: 3.7514
Epoch 28/50
12/12            0s 4ms/step - loss:
15.6420 - mae: 2.9012 - val_loss: 33.6757 - val_mae: 3.7262
Epoch 29/50
12/12            0s 3ms/step - loss:
17.0148 - mae: 3.0519 - val_loss: 32.6068 - val_mae: 3.6431
Epoch 30/50
12/12            0s 6ms/step - loss:
15.1782 - mae: 2.8446 - val_loss: 32.1881 - val_mae: 3.5988
Epoch 31/50
12/12            0s 5ms/step - loss:
17.2223 - mae: 3.0058 - val_loss: 31.0931 - val_mae: 3.5511
Epoch 32/50
12/12            0s 5ms/step - loss:
14.0008 - mae: 2.8352 - val_loss: 31.5389 - val_mae: 3.5611
```

```
Epoch 33/50
12/12            0s 9ms/step - loss:
16.4224 - mae: 2.9407 - val_loss: 30.9154 - val_mae: 3.5132
Epoch 34/50
12/12            0s 5ms/step - loss:
17.1202 - mae: 2.8992 - val_loss: 30.8427 - val_mae: 3.5197
Epoch 35/50
12/12            0s 5ms/step - loss:
14.4388 - mae: 2.7627 - val_loss: 30.0603 - val_mae: 3.5058
Epoch 36/50
12/12            0s 5ms/step - loss:
13.1486 - mae: 2.6630 - val_loss: 29.8477 - val_mae: 3.4819
Epoch 37/50
12/12            0s 4ms/step - loss:
12.8587 - mae: 2.5735 - val_loss: 29.6458 - val_mae: 3.4632
Epoch 38/50
12/12            0s 5ms/step - loss:
11.5893 - mae: 2.4936 - val_loss: 28.6923 - val_mae: 3.3993
Epoch 39/50
12/12            0s 4ms/step - loss:
13.5457 - mae: 2.6462 - val_loss: 27.9086 - val_mae: 3.3470
Epoch 40/50
12/12            0s 4ms/step - loss:
14.3147 - mae: 2.8455 - val_loss: 28.4011 - val_mae: 3.4203
Epoch 41/50
12/12            0s 5ms/step - loss:
14.7176 - mae: 2.6634 - val_loss: 27.8473 - val_mae: 3.3859
Epoch 42/50
12/12            0s 5ms/step - loss:
15.1011 - mae: 2.6999 - val_loss: 27.5924 - val_mae: 3.3982
Epoch 43/50
12/12            0s 5ms/step - loss:
12.6673 - mae: 2.6357 - val_loss: 27.6282 - val_mae: 3.4089
Epoch 44/50
12/12            0s 4ms/step - loss:
12.2337 - mae: 2.6344 - val_loss: 26.2196 - val_mae: 3.3371
Epoch 45/50
12/12            0s 5ms/step - loss:
11.1612 - mae: 2.4315 - val_loss: 26.5611 - val_mae: 3.3573
Epoch 46/50
12/12            0s 4ms/step - loss:
11.5243 - mae: 2.4337 - val_loss: 26.4900 - val_mae: 3.3527
Epoch 47/50
12/12            0s 4ms/step - loss:
11.9757 - mae: 2.5080 - val_loss: 25.4709 - val_mae: 3.3489
Epoch 48/50
12/12            0s 4ms/step - loss:
11.9704 - mae: 2.5175 - val_loss: 25.3799 - val_mae: 3.3569
```

```
Epoch 49/50
12/12              0s 5ms/step - loss:
11.5124 - mae: 2.4678 - val_loss: 25.4830 - val_mae: 3.3310
Epoch 50/50
12/12              0s 5ms/step - loss:
13.4999 - mae: 2.5512 - val_loss: 26.2756 - val_mae: 3.3284
```

1.0.9 Step 9: Predict on Test Set

[9]:
```
y_pred = model.predict(X_test).flatten()
```

```
4/4              0s 11ms/step
```

1.0.10 Step 10: Evaluate Model Performance

[10]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MAE  : {mae:.2f}")
print(f"MSE  : {mse:.2f}")
print(f"R2   : {r2:.2f}")
```

```
MAE  : 2.57
MSE  : 16.16
R2   : 0.78
```

1.0.11 Step 11: Show Predictions vs Actual

[11]:
```
print("Predicted Price vs Actual Price (First 5 Samples)")
for i in range(5):
    print(f"Predicted: {y_pred[i]:.2f} || Actual: {y_test.iloc[i]:.2f}")
```

```
Predicted Price vs Actual Price (First 5 Samples)
Predicted: 27.95 || Actual: 23.60
Predicted: 33.53 || Actual: 32.40
Predicted: 20.76 || Actual: 13.60
Predicted: 26.99 || Actual: 22.80
Predicted: 16.32 || Actual: 16.10
```

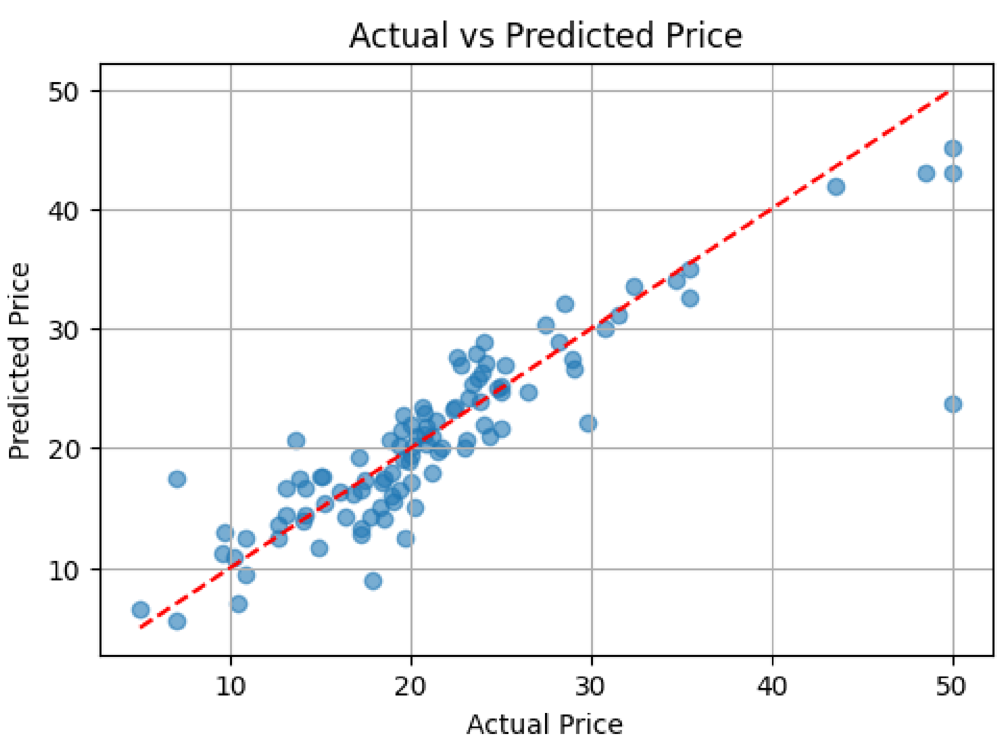
1.0.12 Step 12: Predict from User Input

[12]:
```
def predict_from_user_input():
    print("-- Enter feature values to predict the house price --")

    feature_names = X.columns
    user_input = {}
```

```
    for feature in feature_names:
        value = float(input(f"{feature}: "))
        user_input[feature] = value

    user_df = pd.DataFrame([user_input])
    user_scaled = scaler.transform(user_df)

    pred = model.predict(user_scaled)
    print(f"\nPredicted Price: ${pred[0][0]*1000:.2f}")

# Uncomment to use
# predict_from_user_input()
```

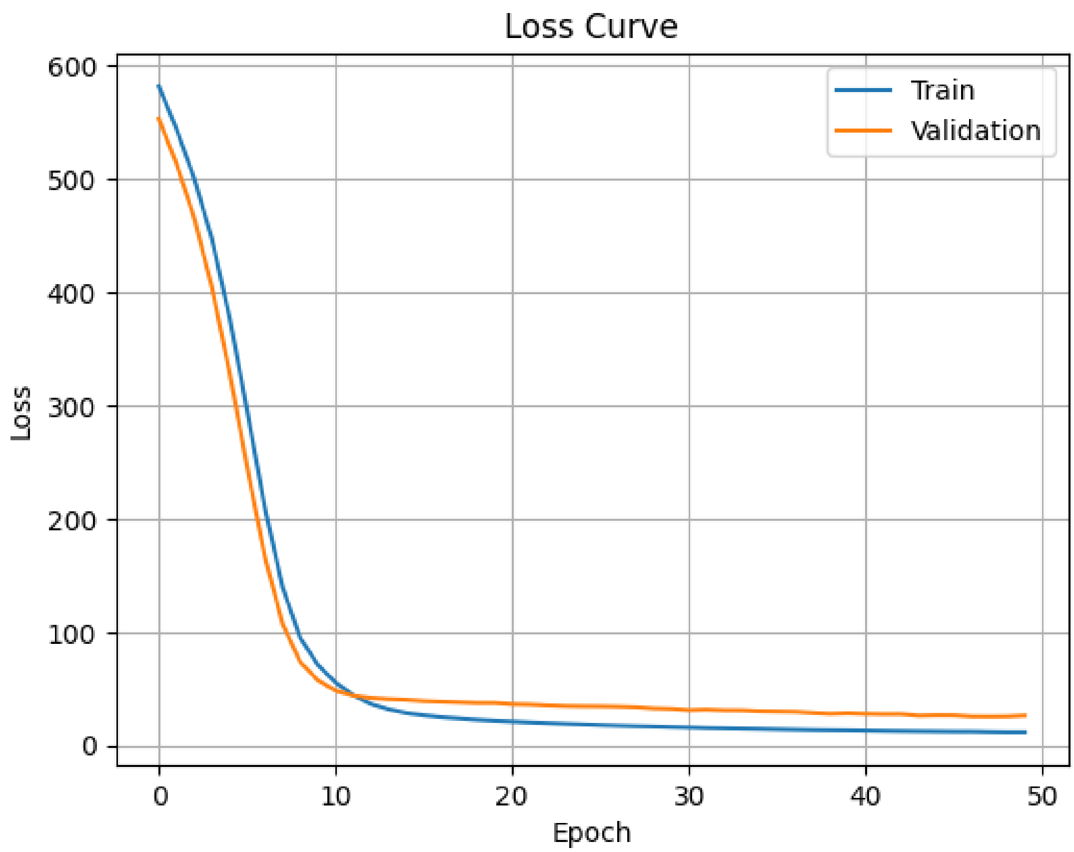
1.0.13 Step 13: Plot Actual vs Predicted Prices

[13]:
```
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Price")
plt.grid(True)
plt.show()
```



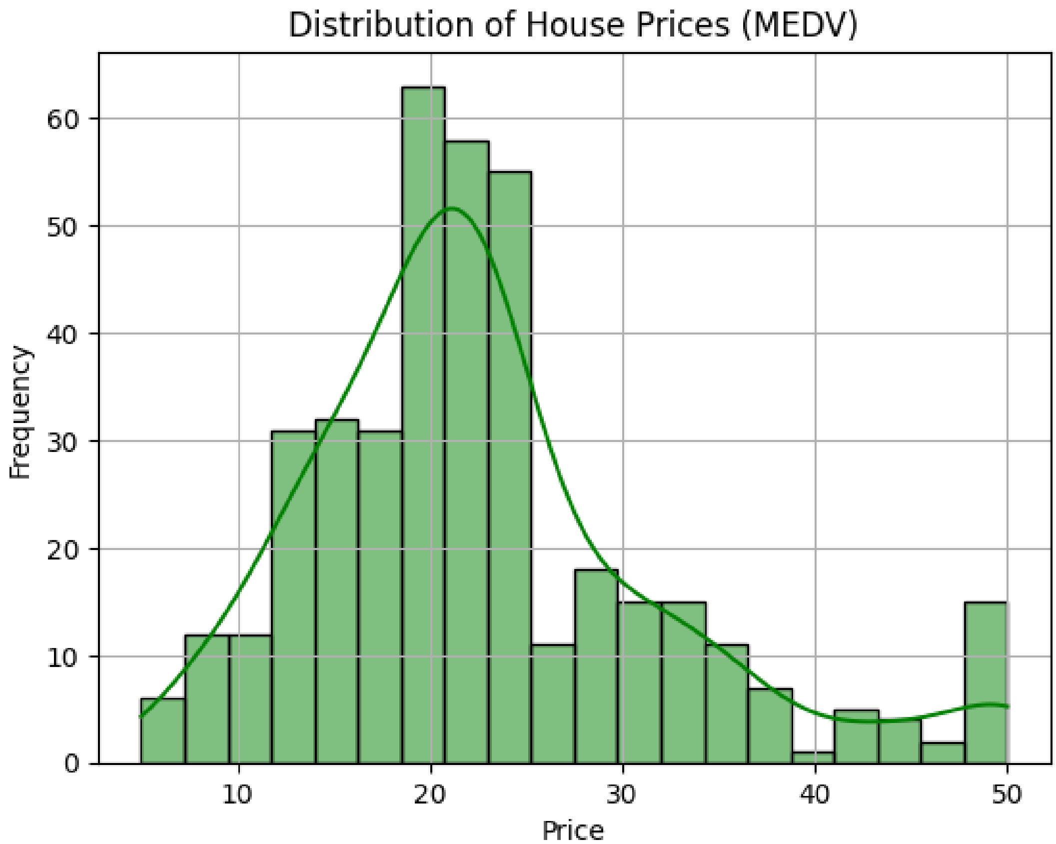
1.0.14 Step 14: Plot Loss Curve

[14]:
```
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title("Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```



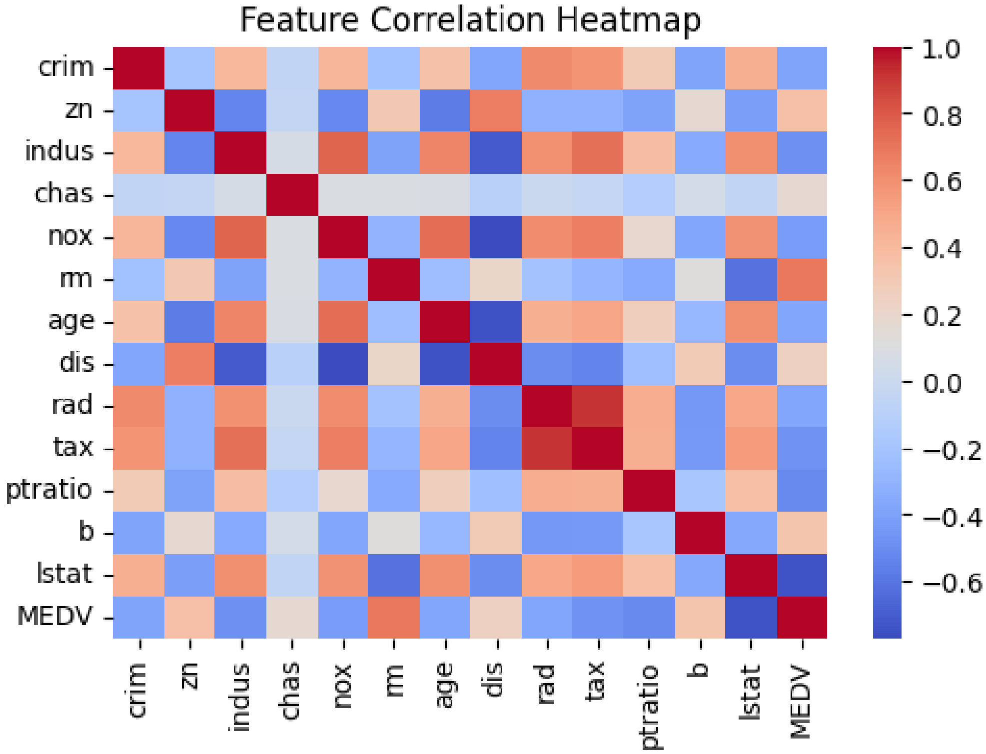
1.0.15 Step 15: Distribution of House Prices

[15]:
```
sns.histplot(y_train, bins=20, kde=True, color='green')
plt.title("Distribution of House Prices (MEDV)")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



1.0.16 Step 16: Correlation Heatmap of Features

[17]:
```
plt.figure(figsize=(6,4))
sns.heatmap(df.corr(), annot=False, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
```


Feature Correlation Heatmap
crim
zn
indus
chas
nox
rm
age
dis
rad
tax
ptratio
b
lstat
MEDV
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
[ ]:
```