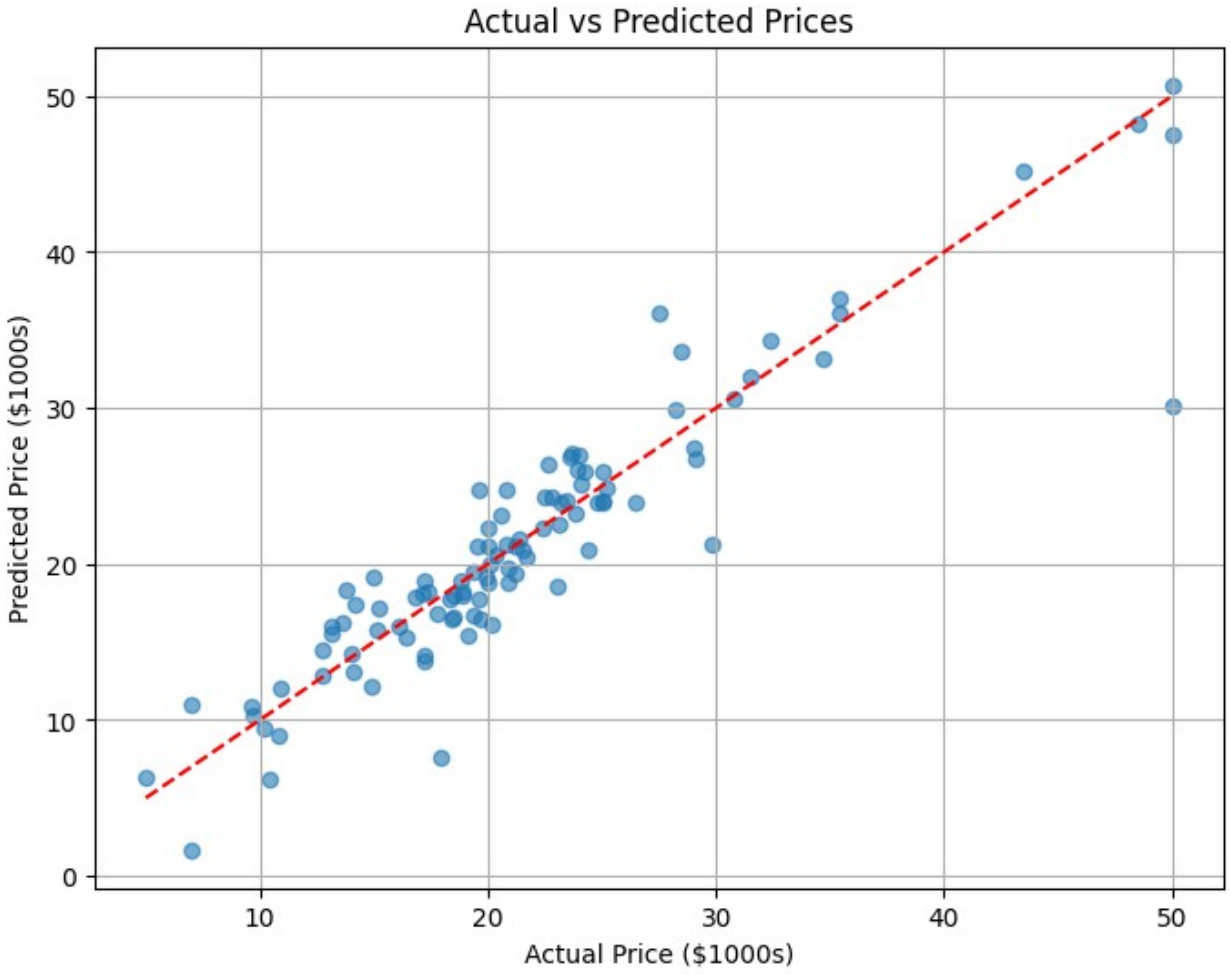
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Load Boston Housing dataset
def load_boston_data():
    data_url = "http://lib.stat.cmu.edu/datasets/boston"
    raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22,
header=None)
    data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
    target = raw_df.values[1::2, 2]
    feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
'AGE',
                     'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
    df = pd.DataFrame(data, columns=feature_names)
    df['PRICE'] = target  # MEDV (target) in $1000s
    return df

# Prepare data
boston_df = load_boston_data()
X = boston_df.drop('PRICE', axis=1).values
y = boston_df['PRICE'].values

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build model
model = Sequential([
    Dense(64, activation='relu', input_shape=(13,)),
    Dense(32, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

# Train model
model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=0)

# Predict on test set
```

```
y_pred = model.predict(X_test).flatten()

# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
print(f"\nMean Absolute Error: ${mae * 1000:.2f}")

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--')
plt.xlabel('Actual Price ($1000s)')
plt.ylabel('Predicted Price ($1000s)')
plt.title('Actual vs Predicted Prices')
plt.grid(True)
plt.show()

# Test cases with ACTUAL prices (for comparison)
test_cases = [
    {
        "features": [0.02731, 0.0, 7.07, 0, 0.469, 6.421, 78.9,
4.9671, 2, 242, 17.8, 396.90, 9.14],
        "actual_price": 21.6
    },
    {
        "features": [0.01, 95, 1.2, 1, 0.3, 8.5, 15, 7.0, 1, 180, 12,
400, 2.5],
        "actual_price": 50.0
    },
    {
        "features": [5.0, 0, 25, 0, 0.9, 4.0, 90, 2.0, 24, 666, 20,
300, 30],
        "actual_price": 10.0
    }
]

# Generate predictions and compare
results = []
for case in test_cases:
    scaled_features =
scaler.transform(np.array(case["features"]).reshape(1, -1))
    predicted_price = model.predict(scaled_features)[0][0]

    results.append({
        "Actual Price ($1000s)": case["actual_price"],
        "Predicted Price ($1000s)": round(predicted_price, 2)
    })

# Create comparison table
results_df = pd.DataFrame(results)
```

```
print("\nTest Case Comparison Table:")
print(results_df.to_string(index=False))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step

Mean Absolute Error: $2150.29
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

Test Case Comparison Table:
 Actual Price ($1000s)  Predicted Price ($1000s)
                  21.6                 23.280001
                  50.0                 61.619999
                  10.0                 10.200000
```