```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.metrics import confusion_matrix, classification_report
```

```
data = pd.read_csv('/content/Housing.csv')
print(data)
```

```
        price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0    13300000  7420         4          2        3      yes        no       no
1    12250000  8960         4          4        4      yes        no       no
2    12250000  9960         3          2        2      yes        no      yes
3    12215000  7500         4          2        2      yes        no      yes
4    11410000  7420         4          1        2      yes       yes      yes
..        ...   ...       ...        ...      ...      ...       ...      ...
540   1820000  3000         2          1        1      yes        no      yes
541   1767150  2400         3          1        1       no        no       no
542   1750000  3620         2          1        1      yes        no       no
543   1750000  2910         3          1        1       no        no       no
544   1750000  3850         3          1        2      yes        no       no

    hotwaterheating airconditioning  parking prefarea furnishingstatus
0                no             yes        2      yes        furnished
1                no             yes        3       no        furnished
2                no              no        2      yes   semi-furnished
3                no             yes        3      yes        furnished
4                no             yes        2       no        furnished
..              ...             ...      ...      ...              ...
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished

[545 rows x 13 columns]
```

```
# Assume 'price' is the target variable
X = data.drop('price', axis=1)
y = data['price']
```

```
# Identify categorical features and perform one-hot encoding
categorical_features = X.select_dtypes(include=['object']).columns
X = pd.get_dummies(X, columns=categorical_features, drop_first=True)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train
```

	area	bedrooms	bathrooms	stories	parking	mainroad_yes	guestroom_yes	basement_yes	hotwaterheating_yes	airconditioning_ye
46	6000	3	2	4	1	1	0	0	0	
93	7200	3	2	1	3	1	0	1	0	
335	3816	2	1	1	2	1	0	1	0	

```
y_train
```

```
    46     1
    93     1
    335    0
    412    0
    471    0
          ..
    71     1
    106    1
    270    0
    435    0
    102    1
    Name: price, Length: 436, dtype: int64
```

436 rows x 13 columns

```
# Standardize the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled
```

```
    array([[ 0.38416819,  0.05527092,  1.53917323, ..., -0.55262032,
            -0.870669  , -0.67690027],
           [ 0.9291807 ,  0.05527092,  1.53917323, ..., -0.55262032,
             1.14854209, -0.67690027],
           [-0.60775457, -1.28351359, -0.5579503 , ..., -0.55262032,
            -0.870669  , -0.67690027],
           ...,
           [-0.29709744,  0.05527092,  1.53917323, ..., -0.55262032,
            -0.870669  , -0.67690027],
           [-0.5060189 , -1.28351359, -0.5579503 , ..., -0.55262032,
            -0.870669  ,  1.47732249],
           [ 0.15707965,  0.05527092,  1.53917323, ..., -0.55262032,
             1.14854209, -0.67690027]])
```

```
X_test_scaled
```

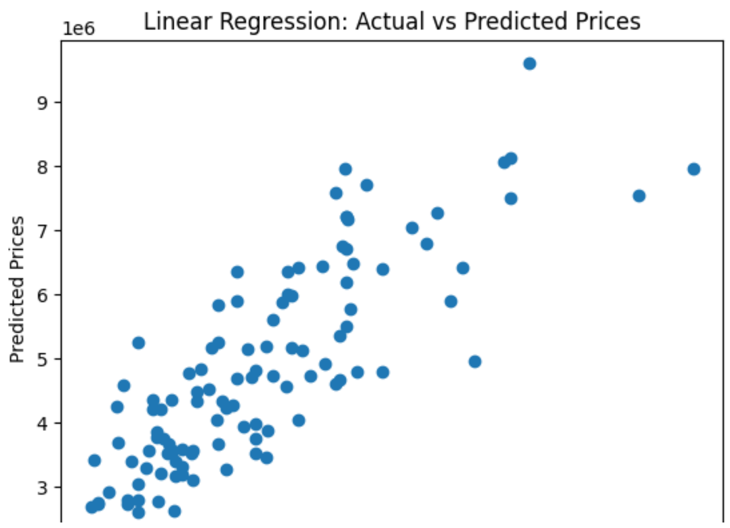
```
    array([[ 0.33875048,  1.39405543,  1.53917323, ..., -0.55262032,
            -0.870669  ,  1.47732249],
           [ 0.61125674,  0.05527092,  1.53917323, ...,  1.80956067,
            -0.870669  , -0.67690027],
           [-0.5060189 , -1.28351359, -0.5579503 , ..., -0.55262032,
             1.14854209, -0.67690027],
           ...,
           [ 0.38416819,  1.39405543,  1.53917323, ..., -0.55262032,
             1.14854209, -0.67690027],
           [ 0.38416819,  0.05527092,  1.53917323, ..., -0.55262032,
             1.14854209, -0.67690027],
           [ 0.4295859 ,  0.05527092,  1.53917323, ...,  1.80956067,
            -0.870669  , -0.67690027]])
```

```
# 1. Linear Regression model (as a baseline)
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
```

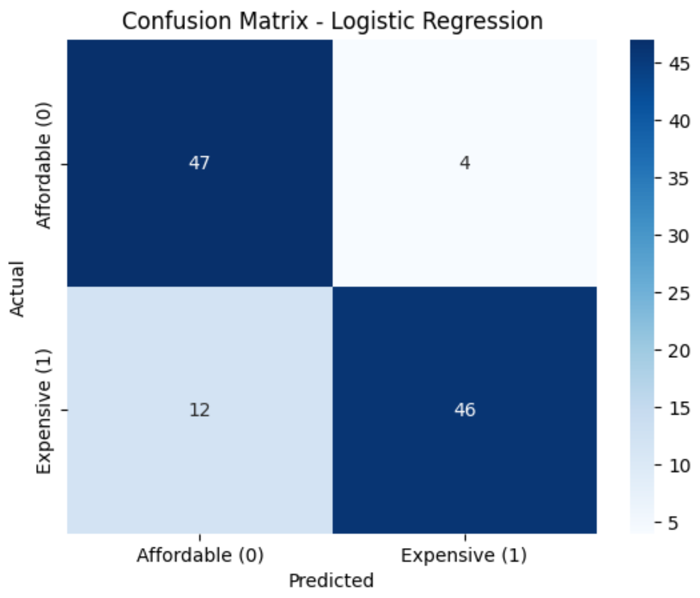
```
    ▾ LinearRegression
    LinearRegression()
```

```
linear_predictions = linear_model.predict(X_test_scaled)

# Create a scatter plot
plt.scatter(y_test, linear_predictions)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Linear Regression: Actual vs Predicted Prices')
plt.show()
```



```
# Visualize the Confusion Matrix using a heatmap
sns.heatmap(conf_matrix_logistic, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Affordable (0)', 'Expensive (1)'],
            yticklabels=['Affordable (0)', 'Expensive (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



```
# Make predictions on the test set
linear_predictions = linear_model.predict(X_test_scaled)
```

```
# Evaluate the performance
linear_mse = mean_squared_error(y_test, linear_predictions)
print(f'Linear Regression MSE: {linear_mse}')
```

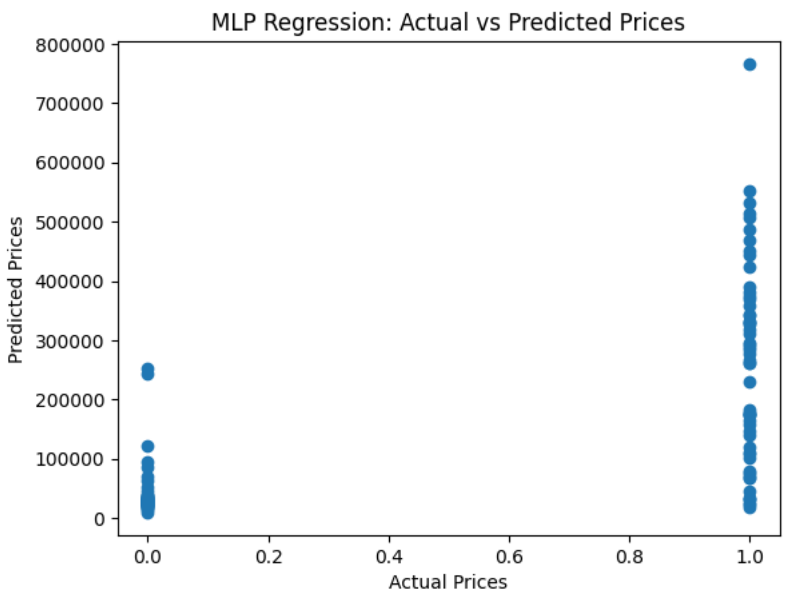
```
    Linear Regression MSE: 1754318687330.6677
```

```
# 2. MLP (Multi-Layer Perceptron) using scikit-learn
mlp_model = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42)
mlp_model.fit(X_train_scaled, y_train)
```

```
    /usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron
```

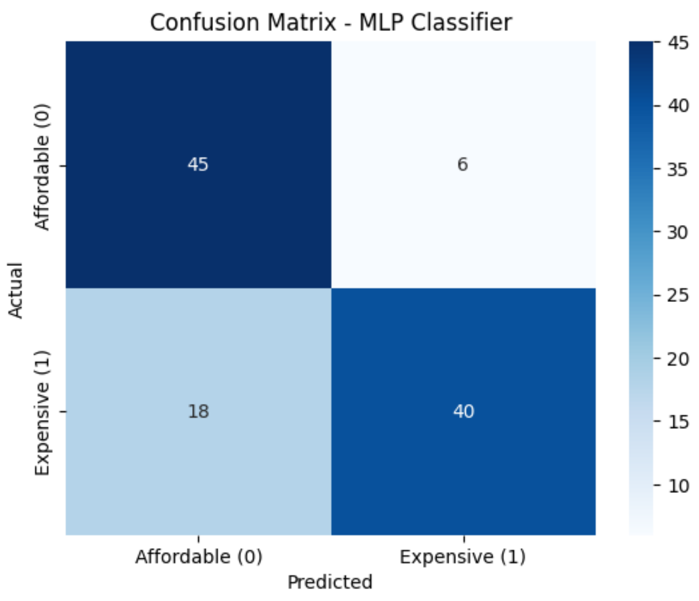
```
mlp_predictions = mlp_model.predict(X_test_scaled)

# Create a scatter plot
plt.scatter(y_test, mlp_predictions)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('MLP Regression: Actual vs Predicted Prices')
plt.show()
```



```
# Make predictions on the test set
mlp_predictions = mlp_classifier.predict(X_test_scaled)


# Visualize the Confusion Matrix using a heatmap
sns.heatmap(conf_matrix_mlp, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Affordable (0)', 'Expensive (1)'],
            yticklabels=['Affordable (0)', 'Expensive (1)'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - MLP Classifier')
plt.show()
```



```
# Make predictions on the test set
mlp_predictions = mlp_model.predict(X_test_scaled)
```

```
# Evaluate the performance
mlp_mse = mean_squared_error(y_test, mlp_predictions)
print(f'MLP MSE: {mlp_mse}')
```

```
    MLP MSE: 28027817222821.31
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
```

```
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=2)
```

```
    Epoch 1/50
    11/11 - 1s - loss: 0.2572 - val_loss: 0.2093 - 1s/epoch - 112ms/step
    Epoch 2/50
    11/11 - 0s - loss: 0.1370 - val_loss: 0.1859 - 77ms/epoch - 7ms/step
    Epoch 3/50
    11/11 - 0s - loss: 0.1181 - val_loss: 0.1810 - 72ms/epoch - 7ms/step
    Epoch 4/50
    11/11 - 0s - loss: 0.1037 - val_loss: 0.1737 - 86ms/epoch - 8ms/step
    Epoch 5/50
    11/11 - 0s - loss: 0.0942 - val_loss: 0.1706 - 90ms/epoch - 8ms/step
    Epoch 6/50
    11/11 - 0s - loss: 0.0867 - val_loss: 0.1769 - 88ms/epoch - 8ms/step
    Epoch 7/50
    11/11 - 0s - loss: 0.0825 - val_loss: 0.1724 - 82ms/epoch - 7ms/step
    Epoch 8/50
    11/11 - 0s - loss: 0.0744 - val_loss: 0.1718 - 74ms/epoch - 7ms/step
    Epoch 9/50
    11/11 - 0s - loss: 0.0699 - val_loss: 0.1754 - 92ms/epoch - 8ms/step
    Epoch 10/50
    11/11 - 0s - loss: 0.0629 - val_loss: 0.1785 - 85ms/epoch - 8ms/step
    Epoch 11/50
    11/11 - 0s - loss: 0.0584 - val_loss: 0.1762 - 68ms/epoch - 6ms/step
    Epoch 12/50
    11/11 - 0s - loss: 0.0537 - val_loss: 0.1745 - 91ms/epoch - 8ms/step
    Epoch 13/50
    11/11 - 0s - loss: 0.0498 - val_loss: 0.1901 - 76ms/epoch - 7ms/step
    Epoch 14/50
    11/11 - 0s - loss: 0.0450 - val_loss: 0.1849 - 72ms/epoch - 7ms/step
    Epoch 15/50
    11/11 - 0s - loss: 0.0411 - val_loss: 0.1771 - 93ms/epoch - 8ms/step
    Epoch 16/50
    11/11 - 0s - loss: 0.0396 - val_loss: 0.1886 - 79ms/epoch - 7ms/step
    Epoch 17/50
    11/11 - 0s - loss: 0.0372 - val_loss: 0.1912 - 66ms/epoch - 6ms/step
    Epoch 18/50
    11/11 - 0s - loss: 0.0323 - val_loss: 0.1916 - 73ms/epoch - 7ms/step
    Epoch 19/50
    11/11 - 0s - loss: 0.0309 - val_loss: 0.1843 - 80ms/epoch - 7ms/step
    Epoch 20/50
    11/11 - 0s - loss: 0.0274 - val_loss: 0.1962 - 75ms/epoch - 7ms/step
    Epoch 21/50
    11/11 - 0s - loss: 0.0265 - val_loss: 0.1914 - 77ms/epoch - 7ms/step
    Epoch 22/50
    11/11 - 0s - loss: 0.0232 - val_loss: 0.1868 - 80ms/epoch - 7ms/step
    Epoch 23/50
    11/11 - 0s - loss: 0.0214 - val_loss: 0.1959 - 69ms/epoch - 6ms/step
    Epoch 24/50
    11/11 - 0s - loss: 0.0199 - val_loss: 0.1960 - 79ms/epoch - 7ms/step
    Epoch 25/50
    11/11 - 0s - loss: 0.0191 - val_loss: 0.2019 - 69ms/epoch - 6ms/step
    Epoch 26/50
```

```
11/11 - 0s - loss: 0.0175 - val_loss: 0.1948 - 67ms/epoch - 6ms/step
Epoch 27/50
11/11 - 0s - loss: 0.0173 - val_loss: 0.2104 - 82ms/epoch - 7ms/step
Epoch 28/50
11/11 - 0s - loss: 0.0155 - val_loss: 0.1994 - 78ms/epoch - 7ms/step
Epoch 29/50
11/11 - 0s - loss: 0.0157 - val_loss: 0.2097 - 95ms/epoch - 9ms/step
```

```
# Evaluate the model on the test set
nn_predictions = model.predict(X_test_scaled)
nn_mse = mean_squared_error(y_test, nn_predictions)
print(f'Neural Network MSE: {nn_mse}')
```

```
4/4 [==============================] - 0s 6ms/step
Neural Network MSE: 0.1815404556697086
```