```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Load your dataset (replace 'your_data.csv' with your actual dataset)
# Make sure your dataset includes features and target variable (house prices)
data = pd.read_csv('Housing.csv')

# Assuming 'date' is the name of the column with date values
data['date'] = pd.to_datetime(data['date'])

# Extract features from the date column
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day

# Drop the original date column
data = data.drop(['date'], axis=1)

# Separate features and target variable
X = data.drop('price', axis=1)
y = data['price']

# Identify categorical columns
categorical_features = X.select_dtypes(include=['object']).columns
```

```
# Create a preprocessor using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), X.select_dtypes(include=['number']).columns),
        ('cat', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)

# Transform the data
X_scaled = preprocessor.fit_transform(X)

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

# Build the MLP model
model = Sequential()

# Add input layer and first hidden layer
model.add(Dense(units=64, activation='relu', input_dim=X_scaled.shape[1]))

# Add additional hidden layers
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))

# Add output layer (1 unit for regression, no activation function)
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
```

```
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)

plt.figure(figsize=(5,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error Loss')
plt.legend()
plt.show()

# Evaluate the model on the test set
loss = model.evaluate(X_test, y_test)
print(f'Mean Squared Error on Test Set: {loss}')

# Make predictions
predictions = model.predict(X_test)

# Plot actual vs predicted values with the best-fit regression line
plt.figure(figsize=(5,5))
plt.scatter(y_test, predictions, label='Actual vs Predicted')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')

# Fit a linear regression line
regression_line = np.polyfit(y_test, predictions.flatten(), 1)
plt.plot(y_test, np.polyval(regression_line, y_test), color='red', label='Regression Line')

plt.title('Actual Prices vs Predicted Prices with Regression Line')
```

```python
plt.legend()

plt.show()
```