House Price Prediction using Deep Learning (PyTorch)

```
Project Overview
This project implements a deep neural network using PyTorch to predict house prices based on various features such
as area, number of bedrooms, bathrooms, stories, parking, and more.

📊 Steps Covered:

Data Loading & Exploration (EDA) - Understanding the dataset, handling missing values, and visualizing
relationships.
Data Preprocessing & Cleaning - Encoding categorical features, normalizing numerical data, and splitting the
dataset.
Building a Deep Learning Model - Implementing a multi-layer perceptron (MLP) using PyTorch.
Training & Optimization - Using Mean Squared Error loss and Adam optimizer.
Model Evaluation - Calculating RMSE and R² Score for performance assessment.
Prediction & Visualization - Comparing actual vs. predicted prices with scatter plots.
```

In [ ]:

```

```

In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:

```python
df = pd.read_csv('/kaggle/input/housing-price-prediction/Housing.csv')
df
```

In [ ]:

```python
df.info()
```

In [ ]:

```python
df.describe()
```

In [ ]:
```
df.nunique()
```

In [ ]:
```
bedrooms_count = df["bedrooms"].value_counts()

plt.figure(figsize=(8, 5))
ax = sns.barplot(x=bedrooms_count.index, y=bedrooms_count.values, palette="viridis")

# Adding labels on top of the bars
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=10, color='black', weight='bold')

plt.title('Count of Bedrooms', fontsize=12)
plt.xlabel('Bedrooms', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

In [ ]:
```
bathrooms_count = df["bathrooms"].value_counts()

plt.figure(figsize=(8, 5))
ax = sns.barplot(x=bathrooms_count.index, y=bathrooms_count.values, palette="viridis")

# Adding labels on top of the bars
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=10, color='black', weight='bold')

plt.title('Count of bathrooms', fontsize=12)
plt.xlabel('bathrooms', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

In [ ]:
```
stories_count = df["stories"].value_counts()

plt.figure(figsize=(8, 5))
ax = sns.barplot(x=stories_count.index, y=stories_count.values, palette="viridis")

# Adding labels on top of the bars
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=10, color='black', weight='bold')

plt.title('Count of stories', fontsize=12)
plt.xlabel('stories', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

In [ ]:
```
bathrooms_count = df["bathrooms"].value_counts()

plt.figure(figsize=(8, 5))
ax = sns.barplot(x=bathrooms_count.index, y=bathrooms_count.values, palette="viridis")

# Adding labels on top of the bars
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=10, color='black', weight='bold')

plt.title('Count of bathrooms', fontsize=12)
plt.xlabel('bathrooms', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

In [ ]:
```
furnishingstatus_count = df["furnishingstatus"].value_counts()

plt.figure(figsize=(8, 5))
ax = sns.barplot(x=furnishingstatus_count.index, y=furnishingstatus_count.values, palette="viridis")

# Adding labels on top of the bars
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=10, color='black', weight='bold')

plt.title('Count of furnishingstatus', fontsize=12)
plt.xlabel('furnishingstatus', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

In [ ]:
```
df.isnull().sum()
```

In [ ]:
```
df.duplicated().sum()
```

In [ ]:
```
# Visualizing price distribution
sns.histplot(df["price"], bins=30, kde=True)
plt.title("House Price Distribution")
plt.show()
```

In [ ]:
```
df.head()
```

Data Preprocessing

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
def preprocess_data(df):
    binary_cols = ["mainroad", "guestroom", "basement", "hotwaterheating", "airconditioning", "prefarea"]
    df[binary_cols] = df[binary_cols].applymap(lambda x: 1 if x == "yes" else 0)

    encoder = LabelEncoder()
    df["furnishingstatus"] = encoder.fit_transform(df["furnishingstatus"])

    return df

df = preprocess_data(df)
```

In [ ]:
```
df.head()
```

In [ ]:
```
df["furnishingstatus"].value_counts()
```

In [ ]:
```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", linewidths=0.5, cbar=True)
plt.show()
```

In [ ]:
```
X = df.drop(columns=["price"])
y = df["price"]
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = X.copy()
X_scaled[["area", "bedrooms", "bathrooms", "stories", "parking"]] = scaler.fit_transform(
    X_scaled[["area", "bedrooms", "bathrooms", "stories", "parking"]]
)
```

In [ ]:
```
X_scaled
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Model Building

In [ ]:
```
import torch
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
```

In [ ]:
```
import torch.nn as nn
import torch.optim as optim
```

In [ ]:
```
class house_price_prediction(nn.Module):
    def __init__(self, input_features = 12 , hidden1 = 64, hidden2 = 32, hidden3 = 16, output=1):
        super().__init__()
        self.fc1 = nn.Linear(input_features, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, hidden3)
        self.out = nn.Linear(hidden3, output)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.out(x)
        return x
```

In [ ]:
```
model = house_price_prediction()
```

In [ ]:
```
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

In [ ]:
```
epochs = 1000
final_loss = []

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    y_pred = model(X_train_tensor)
    loss = loss_function(y_pred, y_train_tensor)
    loss.backward()
    optimizer.step()
    final_loss.append(loss.item())

    if (epoch+1) % 50 == 0:
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item()}")
```

In [ ]:
```
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test_tensor)
    test_loss = loss_function(y_pred_test, y_test_tensor)
    print(f"Test Loss: {test_loss.item()}")

# Plot Training Loss
plt.plot(final_loss)
plt.title("Training Loss Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

In [ ]:

```
model.eval()
with torch.no_grad():
    y_pred_test = model(X_test_tensor)
# Convert predictions to numpy
y_pred_test = y_pred_test.numpy()
y_test = y_test_tensor.numpy()
```

In [ ]:

```
from sklearn.metrics import mean_squared_error, r2_score
rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
r2 = r2_score(y_test, y_pred_test)
print(f'RMSE: {rmse:.2f}')
print(f'R² Score: {r2:.4f}')

# Scatter Plot - Actual vs. Predicted
plt.scatter(y_test, y_pred_test, alpha=0.7)
plt.xlabel('Actual House Prices')
plt.ylabel('Predicted House Prices')
plt.title('Actual vs. Predicted Prices')
plt.show()
```

In [ ]:

In [ ]:

In [ ]: