Here's your Full Pytorch Code for **Indian housing price prediction**, **to accept user input** after training — so users can predict prices for their own custom home features (like size, location, bedrooms, and bathrooms).

✅ Summary

Step	Description
Imports	Loads required ML and data tools
Simulated Data	Creates fake Indian housing dataset
Preprocessing	Encodes, scales, splits the data
Neural Network	Defines and trains a PyTorch model
Evaluation	Measures model accuracy on test data

Same example usign csv data file :-

Sample CSV Data: `housing_data.csv`

```
location,size,bedrooms,bathrooms,price
Mumbai,1200,2,2,8500000
Delhi,1500,3,2,9500000
Bangalore,1300,2,2,7800000
Chennai,1100,2,1,6200000
Kolkata,1000,2,1,5000000
Mumbai,2000,3,3,14000000
Delhi,1800,3,2,11500000
Bangalore,1600,3,2,9200000
Chennai,1700,3,2,8700000
Kolkata,1400,3,2,7300000
Mumbai,2500,4,3,17000000
Delhi,2400,4,3,16000000
Bangalore,2200,4,3,13500000
Chennai,2100,4,3,12800000
Kolkata,1900,4,3,10500000
Mumbai,800,1,1,4800000
Delhi,900,1,1,5200000
Bangalore,950,1,1,5100000
Chennai,1000,1,1,4700000
```

```
Kolkata,850,1,1,4300000
Mumbai,1450,2,2,9200000
Delhi,1350,2,2,8800000
Bangalore,1550,2,2,8900000
Chennai,1250,2,2,8100000
Kolkata,1150,2,2,7500000
```

How to Use `housing_data.csv`:

1. Open any text/code editor or Excel.
2. Copy-paste the content above.
3. Save the file as:
   `housing_data.csv`
4. Place it in the **same folder** as your `.py` file running the PyTorch model.

Full python code pytorch-housing-ai-model.py :-

```python
# Step 1: Imports
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 2: Load Housing Data from CSV
np.random.seed(42)
torch.manual_seed(42)

# 🔄 Load your CSV file here
df = pd.read_csv("housing_data.csv")  # Make sure this file exists in your working directory

# Store the original location list for user input
locations = df['location'].unique().tolist()

# Step 3: Preprocessing
df = pd.get_dummies(df, columns=['location'], drop_first=True)

X = df.drop('price', axis=1).values
y = df['price'].values.reshape(-1, 1)

scaler_X = StandardScaler()
scaler_y = StandardScaler()
```

```
X = scaler_X.fit_transform(X)
y = scaler_y.fit_transform(y)

# Convert to PyTorch tensors
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32)

# Step 4: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 5: Define the Neural Network
class HousingModel(nn.Module):
    def __init__(self, input_dim):
        super(HousingModel, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        return self.model(x)

# Step 6: Instantiate the model, loss function, optimizer
model = HousingModel(X.shape[1])
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Step 7: Training loop
epochs = 100
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")

# Step 8: Evaluate the model
model.eval()
with torch.no_grad():
```

```
    predictions = model(X_test)
    test_loss = criterion(predictions, y_test).item()

# Inverse scale predictions and actual values
preds_real = scaler_y.inverse_transform(predictions.numpy())
y_test_real = scaler_y.inverse_transform(y_test.numpy())

# Step 9: Show Results
print("\n📊 Sample Predictions:")
for i in range(5):
    print(f"Predicted: ₹{int(preds_real[i][0])} | Actual:
₹{int(y_test_real[i][0])}")

print(f"\n✅ Final Test Loss (MSE): {test_loss:.4f}")

# Step 10: Take User Input for Prediction
print("\n🏠 Enter details to predict housing price:")

# --- Collect user input ---
user_size = float(input("Enter house size (in sq ft): "))
user_bedrooms = int(input("Enter number of bedrooms: "))
user_bathrooms = int(input("Enter number of bathrooms: "))

print("Choose location from:", ", ".join(locations))
user_location = input("Enter location: ").strip()

if user_location not in locations:
    raise ValueError("❌ Invalid location selected.")

# --- Create input feature vector ---
input_data = {
    'size': user_size,
    'bedrooms': user_bedrooms,
    'bathrooms': user_bathrooms
}

# Add one-hot encoding for location
for col in df.columns:
    if col.startswith("location_"):
        location_name = col.split("_")[1]
        input_data[col] = 1 if user_location == location_name else 0

# If location was dropped first, ensure it is handled
if f"location_{user_location}" not in input_data and
f"location_{locations[0]}" not in df.columns:
    # First location is implicitly encoded, so handle it manually
    for loc in locations[1:]:
        input_data[f"location_{loc}"] = 0
```

```python
# Convert to DataFrame for processing
input_df = pd.DataFrame([input_data])
input_scaled = scaler_X.transform(input_df.values)

# Convert to tensor
input_tensor = torch.tensor(input_scaled, dtype=torch.float32)

# --- Make prediction ---
model.eval()
with torch.no_grad():
    prediction = model(input_tensor)
    price_pred = scaler_y.inverse_transform(prediction.numpy())[0][0]

print(f"\n💰 Predicted House Price: ₹{int(price_pred)}")
```

Output:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

✅ Final Test Loss (MSE): 0.0264

🏠 Enter details to predict housing price:
Enter house size (in sq ft): 400
Enter number of bedrooms: 2
Enter number of bathrooms: 2
Choose location from: Mumbai, Delhi, Bangalore, Chennai, Kolkata
Enter location: Mumbai

💰 Predicted House Price: ₹5972760
```

Sure! Here's a **step-by-step explanation** of your code that builds a **housing price prediction model using PyTorch**, using data loaded from a CSV file.

🔢 Step-by-Step Explanation

✅ Step 1: Import Required Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

- **torch**: Core PyTorch package for tensors and computation.
- **torch.nn**: Used to define the neural network layers.
- **torch.optim**: Optimizers like Adam or SGD for training.
- **pandas**: To load and handle tabular data from CSV.
- **numpy**: For numeric operations.
- **sklearn.model_selection.train_test_split**: To split data into training and testing sets.
- **sklearn.preprocessing.StandardScaler**: To scale (normalize) features.

✅ Step 2: Load Housing Data from CSV

```
np.random.seed(42)
torch.manual_seed(42)
df = pd.read_csv("housing_data.csv")
locations = df['location'].unique().tolist()
```

- Sets random seeds for reproducibility.
- Loads housing data from a CSV file.
- Extracts a list of all unique locations for later user input validation.

✅ Step 3: Preprocessing the Data

```
df = pd.get_dummies(df, columns=['location'], drop_first=True)
```

- Converts the **categorical `location` column** into multiple binary columns using one-hot encoding.
- `drop_first=True` avoids multicollinearity by dropping one location column (treated as base location).

```
X = df.drop('price', axis=1).values
y = df['price'].values.reshape(-1, 1)
```

- `X`: Features (size, bedrooms, bathrooms, and location columns).
- `y`: Target variable (price).

```
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X = scaler_X.fit_transform(X)
y = scaler_y.fit_transform(y)
```

- Standardizes both input features and target prices (mean = 0, std = 1) for better training performance.

```
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(y, dtype=torch.float32)
```

- Converts the scaled NumPy arrays to PyTorch tensors.

✅ Step 4: Split 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)
```

- 80% of data is used for training, 20% for testing.

✅ Step 5: Define the Neural Network

```
class HousingModel(nn.Module):
    def __init__(self, input_dim):
        ...
```

- Custom PyTorch model with:
    - Input layer → 128 neurons → ReLU
    - 128 → 64 neurons → ReLU
    - 64 → 1 output neuron (predicts price)
- `nn.Sequential` chains layers.

✅ Step 6: Initialize Model, Loss Function, and Optimizer

```
model = HousingModel(X.shape[1])
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

- **Model** takes number of input features (depends on how many one-hot encoded columns).
- **Loss Function**: Mean Squared Error for regression.
- **Optimizer**: Adam optimizer with a learning rate of 0.01.

✅ Step 7: Train the Model

```
for epoch in range(epochs):
    ...
```

- For 100 epochs:
  - Set model to training mode.
  - Clear gradients.
  - Forward pass (`model(X_train)`).
  - Compute loss.
  - Backpropagation (`loss.backward()`).
  - Update model weights (`optimizer.step()`).
  - Print loss every 10 epochs.

✅ Step 8: Evaluate the Model

```
model.eval()
with torch.no_grad():
    predictions = model(X_test)
    ...
```

- Turns off gradient tracking for evaluation.
- Predicts prices on the test set.
- Calculates MSE loss on actual vs predicted values.

```
preds_real = scaler_y.inverse_transform(predictions.numpy())
y_test_real = scaler_y.inverse_transform(y_test.numpy())
```

- Converts scaled predictions and targets back to real INR prices.

✅ Step 9: Display Sample Predictions

```
for i in range(5):
    print(f"Predicted: ₹{int(preds_real[i][0])} | Actual:
₹{int(y_test_real[i][0])}")
```

- Prints 5 sample predictions compared to actual values.

✅ Step 10: Take User Input and Predict Price

```
user_size = float(input(...))
user_bedrooms = int(input(...))
user_bathrooms = int(input(...))
user_location = input(...).strip()
```

- Collects input from the user for prediction.

```
for col in df.columns:
    if col.startswith("location_"):
        ...
```

- One-hot encodes the user-selected location to match model's input format.

```
input_df = pd.DataFrame([input_data])
input_scaled = scaler_X.transform(input_df.values)
input_tensor = torch.tensor(input_scaled, dtype=torch.float32)
```

- Formats and scales the input.

```
with torch.no_grad():
    prediction = model(input_tensor)
    price_pred = scaler_y.inverse_transform(prediction.numpy())[0][0]
```

- Predicts the price and converts it back from scaled form.

```
print(f"\n💰 Predicted House Price: ₹{int(price_pred)}")
```

- Prints the final predicted house price.

✅ Summary

Step	What it Does
1	Imports libraries
2	Loads housing data from CSV
3	Preprocesses data (scaling, one-hot encoding)
4	Splits data into train/test
5	Defines neural network model
6	Initializes model, loss, optimizer
7	Trains model using MSE loss
8	Evaluates on test set
9	Shows sample predictions
10	Takes user input and predicts housing price