# House price prediction model with user inputs using Tensorflow :-

Here's a simple example CSV content for **house_prices.csv** that you can save and use with the code:

```
size_sqft,bedrooms,bathrooms,age_years,price
1500,3,2,10,300000
2000,4,3,5,450000
850,2,1,20,150000
1200,3,2,15,250000
1800,4,3,7,400000
950,2,1,30,130000
2200,5,4,2,500000
1100,3,2,18,240000
1750,4,3,12,380000
1400,3,2,8,320000
```

---

### How to save it:

1. Open a text editor (Notepad, VSCode, etc.).
2. Copy-paste the above data.
3. Save as `house_prices.csv` in the same folder where you keep your Python script.

Here's the **house price prediction model with user inputs** — so after training, it will ask you to enter house features, then predict the price.

---

## Full code (train + predict with user inputs):

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Load dataset
df = pd.read_csv('house_prices.csv')  # Your CSV file path

# Features and target
X = df[['size_sqft', 'bedrooms', 'bathrooms', 'age_years']].values
y = df['price'].values
```

```
# Train/test split
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 = tf.keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)  # regression output
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train model
model.fit(X_train, y_train, epochs=50, validation_split=0.2)

# Evaluate
loss, mae = model.evaluate(X_test, y_test)
print(f"Test MAE: {mae}")

# User input prediction loop
def predict_price():
    print("\nEnter house details to predict price:")
    try:
        size_sqft = float(input("Size (sqft): "))
        bedrooms = int(input("Number of bedrooms: "))
        bathrooms = int(input("Number of bathrooms: "))
        age_years = int(input("Age of house (years): "))
    except ValueError:
        print("Invalid input. Please enter numeric values.")
        return

    user_data = np.array([[size_sqft, bedrooms, bathrooms, age_years]])
    user_data_scaled = scaler.transform(user_data)
    prediction = model.predict(user_data_scaled)
    print(f"Predicted house price: ${prediction[0][0]:,.2f}")

# Run prediction interactively
while True:
    predict_price()
    cont = input("Predict another? (y/n): ").strip().lower()
    if cont != 'y':
        break
print("Goodbye!")
```

---

## How to use:

- Make sure `house_prices.csv` is in the folder.

- Run this script.
- It trains the model, then repeatedly asks you for house info.
- It predicts and prints the price.
- Type 'n' when you want to quit.

# Here's a **line-by-line explanation** of your house price prediction code using TensorFlow:

---

## □ 1. Import necessary libraries

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
```

- `pandas`: For reading and handling CSV data.
- `tensorflow`: Deep learning library used to build the model.
- `tensorflow.keras.layers`: Prebuilt layers like `Dense`, used in the model.
- `train_test_split`: Splits your data into training and testing sets.
- `StandardScaler`: Normalizes feature values to improve model performance.
- `numpy`: For handling arrays and numerical operations.

---

## □ 2. Load the dataset

```
df = pd.read_csv('house_prices.csv')  # Your CSV file path
```

- Loads your CSV file into a DataFrame `df` with rows and columns representing data.

---

## □ 3. Split features and target

```
X = df[['size_sqft', 'bedrooms', 'bathrooms', 'age_years']].values
y = df['price'].values
```

- `X`: Features (inputs) used to predict price.

- `y`: Target variable (house price).

---

## □ 4. Split into training and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- 80% data for training, 20% for testing.
- `random_state=42` ensures reproducibility (same split every time).

---

## □ 5. Scale features

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Normalizes input data to have zero mean and unit variance.
- Makes training more stable and faster.

---

## □ 6. Build the neural network model

```
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)  # regression output
])
```

- `Sequential`: Builds a model layer by layer.
- `Dense(64, activation='relu')`: Fully connected layer with 64 neurons and ReLU activation.
- `input_shape=(X_train.shape[1],)`: First layer input size matches the number of features.
- Final `Dense(1)`: Output layer with 1 neuron (regression value = predicted price).

---

## □ 7. Compile the model

```
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

- `optimizer='adam'`: Efficient optimizer for gradient descent.

- `loss='mse'`: Mean Squared Error (good for regression).
- `metrics=['mae']`: Mean Absolute Error for evaluation.

---

## □ 8. Train the model

```
model.fit(X_train, y_train, epochs=50, validation_split=0.2)
```

- Trains model for 50 epochs.
- Uses 80% of training data to train, 20% to validate during training.
- `fit()` updates model weights to minimize loss.

---

## □ 9. Evaluate model on test data

```
loss, mae = model.evaluate(X_test, y_test)
print(f"Test MAE: {mae}")
```

- Tests model on unseen data (`X_test`, `y_test`) to evaluate accuracy.
- Prints Mean Absolute Error on test set.

---

## □ 10. Define function to predict from user input

```
def predict_price():
    print("\nEnter house details to predict price:")
    try:
        size_sqft = float(input("Size (sqft): "))
        bedrooms = int(input("Number of bedrooms: "))
        bathrooms = int(input("Number of bathrooms: "))
        age_years = int(input("Age of house (years): "))
    except ValueError:
        print("Invalid input. Please enter numeric values.")
        return
```

- Asks user to enter house features via command line.
- Converts user inputs into correct numeric types.
- Handles input errors with `try/except`.

---

## □ 11. Predict and display result

```
    user_data = np.array([[size_sqft, bedrooms, bathrooms, age_years]])
    user_data_scaled = scaler.transform(user_data)
```

```
    prediction = model.predict(user_data_scaled)
    print(f"Predicted house price: ${prediction[0][0]:,.2f}")
```

- Formats user input into array shape the model expects.
- Scales input using the same scaler used for training.
- Predicts price and formats the result nicely with dollar sign and commas.

---

## □ 12. Run prediction loop

```
while True:
    predict_price()
    cont = input("Predict another? (y/n): ").strip().lower()
    if cont != 'y':
        break
print("Goodbye!")
```

- Repeatedly calls `predict_price()` to allow multiple predictions.
- Asks user if they want to continue.
- Exits loop when the user types anything other than `y`.

---

## □ Summary

This script:

- Trains a model to predict house prices using basic features.
- Evaluates the model.
- Lets the user enter new data to make predictions.

---

Let me know if you'd like:

- A GUI version of the input form
- Saving/loading the model for later use
- Visualizing training loss/accuracy

Here's how to **save and load** your trained house price prediction model for future use — so you don't need to retrain every time.

---

## 🔹 Modify your script to save the model and scaler

### 📌 Add these after training is complete:

```
# Save model
model.save('house_price_model')

# Save the scaler
import joblib
joblib.dump(scaler, 'scaler.save')
```

---

## 🔹 To load the model and scaler later:

### 📌 Create a new Python file like `predict_house_price.py`:

```
import tensorflow as tf
import numpy as np
import joblib

# Load model and scaler
model = tf.keras.models.load_model('house_price_model')
scaler = joblib.load('scaler.save')

# Function to predict from user input
def predict_price():
    print("\nEnter house details to predict price:")
    try:
        size_sqft = float(input("Size (sqft): "))
        bedrooms = int(input("Number of bedrooms: "))
        bathrooms = int(input("Number of bathrooms: "))
        age_years = int(input("Age of house (years): "))
    except ValueError:
        print("Invalid input. Please enter numeric values.")
        return
```

```
    user_data = np.array([[size_sqft, bedrooms, bathrooms, age_years]])
    user_data_scaled = scaler.transform(user_data)
    prediction = model.predict(user_data_scaled)
    print(f"Predicted house price: ${prediction[0][0]:,.2f}")

# Run prediction loop
while True:
    predict_price()
    cont = input("Predict another? (y/n): ").strip().lower()
    if cont != 'y':
        break
print("Goodbye!")
```

---

## Install dependencies if needed

```
pip install joblib tensorflow numpy
```

---

## Summary

| What | File created |
|---|---|
| Model | `house_price_model/` (folder) |
| Scaler | `scaler.save` |
| Load + Predict | `predict_house_price.py` |

Your **house price prediction project** is a **regression problem**, and it uses a type of neural network called a:

---

## Feedforward Neural Network (FNN)

Also known as:

- **Multilayer Perceptron (MLP)**
- **Fully Connected Neural Network**

---

### □ Why?

Because:

- Data flows **in one direction** — from input to output.
- You use `Dense` (fully connected) layers.
- There's **no recurrence** (like in RNNs) or spatial pattern processing (like in CNNs).
- The output is **a single continuous number** (house price) — so it's a **regression output**, not classification.

---

### □ Structure in your model:

```
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu'),  # Hidden Layer 1
    layers.Dense(64, activation='relu'),  # Hidden Layer 2
    layers.Dense(1)  # Output layer (for regression)
])
```

This is a **basic FNN/MLP** with:

- Input layer (based on 4 features)
- 2 hidden layers (with ReLU)
- 1 output neuron (for price)

---

### □ Not used in this project:

| Network Type | Used For |
|---|---|
| CNN | Images / spatial data |
| RNN / LSTM | Sequences / time series / NLP |
| GANs | Generative tasks |
| Transformers | NLP / vision / multimodal tasks |

---

### □ Conclusion:

Your project is a **Feedforward Neural Network (FNN)** used for a **regression** task.