House Price Prediction - Project Documentation

This document provides a theoretical explanation of building a House Price Prediction Web Application using Machine Learning and Flask. The project involves data preparation, model training, backend development, and frontend integration.

---

1. Data Preparation & Preprocessing

1.1.1 Understanding the Dataset

The dataset consists of various features that influence house prices, such as:

- **bedrooms**: Number of bedrooms
- **bathrooms**: Number of bathrooms
- **floors**: Number of floors in the house
- **yr_built**: The year the house was built
- **sqft_living**: The square footage of the living area
- **price**: The actual price of the house (target variable)

1.1.2 Loading the Dataset

We use pandas to load and analyze the dataset.

import pandas as pd

df = pd.read_csv('house_data.csv')

1.1.3 Select Required Columns

We extract the relevant columns needed for training the model.

columns = ['bedrooms', 'bathrooms', 'floors', 'yr_built', 'sqft_living', 'price']

df = df[columns]

Features (X): bedrooms, bathrooms, floors, yr_built, sqft_living

Target (y): price (house price)

1.2. Data Splitting

We split the dataset into training and testing sets using train_test_split from scikit-learn.

from sklearn.model_selection import train_test_split

X = df.iloc[:, 0:5]

y = df.iloc[:, 5]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

75% training data
25% testing data

random_state=42 ensures reproducibility.

1.3 Model Training

We train two models:

Linear Regression

Random Forest Regressor

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

 # Initialize models

lr = LinearRegression()

rf = RandomForestRegressor(random_state=42)

 # Train models

lr.fit(X_train, y_train)

rf.fit(X_train, y_train)

1.4.Model Evaluation

We compare model performance using the **R^2 score** on the test set.

lr_score = lr.score(X_test, y_test)

rf_score = rf.score(X_test, y_test)

print("Linear Regression R²:", lr_score)

print("Random Forest R²:", rf_score)

Linear Regression R^2: 0.5517193889089222

Random Forest R^2: 0.5343406907435422

Flask Framework

Flask is a lightweight Python web framework used to build APIs and web applications. The script creates a simple API that:

- Serves an HTML page (index.html).
- Accepts JSON data via a **POST request** for price prediction.
- Returns the predicted house price as a JSON response.

```
from flask import Flask, request, jsonify, render_template

import pickle

import numpy as np
```

Loading the Pre-Trained Model

The **Pickle** module is used to load the trained model (best_model.pkl) at runtime. This allows the API to use the model without retraining it each time.

```
with open('best_model.pkl', 'rb') as file:
   model = pickle.load(file)
```

pickle.load(file): Deserializes (loads) the trained model.

The model is stored in the model variable for future predictions.

Creating API Endpoints

The API defines two key routes:

1. Home Route (/)

Returns the **index.html** page where users can manually enter house details.

```
@app.route('/')
def home():
```

return render_template('index.html')

2. Prediction Route (/predict)

Handles **POST requests** for house price predictions.

@app.route('/predict', methods=['POST'])

- Receives JSON data from the request body.
- Validates and extracts the required house features.
- Passes the features to the model for prediction.
- Returns the prediction in **JSON format**.

2.4 Handling JSON Data

The API expects **house feature values** in JSON format. Example input:

json
CopyEdit

```
{
  "bedrooms": 3,
  "bathrooms": 2,
  "floors": 1,
  "yr_built": 2005,
  "sqft_living": 1800
}
```

To extract this data, we use:

```
data = request.get_json()
if not data:
  return jsonify({'error': 'No data provided'}), 400
```

- request.get_json(): Retrieves **JSON** data from the request.
- If no data is provided, returns a 400 Bad Request response.

2.5 Data Validation & Extraction

Each feature is extracted and converted into a **float**:

```
bedrooms = float(data.get('bedrooms', 0))
bathrooms = float(data.get('bathrooms', 0))
floors = float(data.get('floors', 0))
yr_built = float(data.get('yr_built', 0))
sqft_living = float(data.get('sqft_living', 0))
```

get(key, default_value): Retrieves a value from JSON. If missing, defaults to 0.

- float(): Ensures the values are in numeric format.

2.6 Preparing Data for Prediction

Machine learning models expect **structured numerical inputs**. We reshape the extracted data into a **2D NumPy array**:

features = np.array([bedrooms, bathrooms, floors, yr_built, sqft_living]).reshape(1, -1)

- **np.array([...]): Creates a NumPy array from input values.**
- **.reshape(1, -1): Converts it into a** 2D array **(required by scikit-learn models).**

2.7 Making Predictions

Once the data is formatted, we pass it to the loaded model:

prediction = model.predict(features)[0]

- model.predict(features): Returns an array of predictions.
- [0]: Extracts the first (and only) predicted value.

The result is **formatted** to two decimal places before sending the response:

return jsonify({'prediction': f"{prediction:,.2f}"})

- f"{prediction:,.2f}": Formats the price **with commas** (e.g., **₹1,200,000.00**).

2.8 Error Handling

The API includes a **try-except** block to catch any errors:

except Exception as e:
return jsonify({'error': str(e)}), 500

- If an error occurs, a **500 Internal Server Error** response is sent.
- The error message is included in the response.

2.9 Running the Flask App

The script runs the Flask server on **port 5500**:

```
if __name__ == '__main__':
    app.run(debug=True, port=5500)
```

- **debug=True: Enables** debug **mode** for easier error tracking.
- port=5500: Runs the app on **port 5500** instead of the default (5000).

Frontend (Interactive Chat-based UI)

The frontend is built using **HTML, CSS, and JavaScript** to provide a chatbot-like experience for collecting user input.

Key Components

Chat Flow Logic

- The chatbot asks five sequential questions about the house:
    1. Number of bedrooms
    2. Number of bathrooms
    3. Number of floors
    4. Year built
    5. Square footage of the living area
- The user provides responses, which are collected in an array (responses).

User Input Handling

```
chatForm.addEventListener('submit', function(event) {
    event.preventDefault();
    const message = userInput.value.trim();
    if (message) {
        addChatBubble(message, 'user');
        responses.push(message);
        userInput.value = '';
        currentQuestion++;
        askNextQuestion();
    }
});
```

- When the user submits an input, it gets added to the chat window, and the next question is triggered.

Prediction Request

```
fetch('/predict', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
```

```
  body: JSON.stringify({
    bedrooms: responses[0],
    bathrooms: responses[1],
    floors: responses[2],
    yr_built: responses[3],
    sqft_living: responses[4]
  })
})
```

- After collecting all five responses, a request is sent to the /predict endpoint to get the estimated house price.

Displaying Results

- The bot responds with either the predicted house price or an error message.