Lagos House Price Prediction Project Documentation

Table of Contents

1. Project Overview

2. Languages and Tools Used

3. Data Collection and Cleaning

4. Feature Engineering

5. Model Building

6. Model Deployment with Flask

7. Running the Project

8. Conclusion

1. Project Overview

The Lagos House Price Prediction project aims to predict house prices based on various features such as the number of bedrooms, bathrooms, toilets, the location of the property, and other amenities. The project utilizes a machine learning model (Random Forest Regressor) to make predictions and is deployed as a web application using Flask.

2. Languages and Tools Used

Python: The main programming language used for data processing, machine learning, and serverside scripting.

Flask: A micro web framework for Python used to deploy the machine learning model as a web application.

Jinja2: A templating engine for Python used with Flask for rendering HTML templates.

pandas: A data manipulation and analysis library used for data cleaning and preprocessing.

scikitlearn: A machine learning library used for building the prediction model.

joblib: A library used for serializing the model and other Python objects.

HTML/CSS: Used for creating and styling the web pages.

3. Data Collection and Cleaning

Data Collection

The dataset consists of information on houses in Lagos, including features like:

Bedrooms

Bathrooms

Toilets

Location

Currency

Neighborhood

Price

Serviced

Newly Built

Furnished

Data Cleaning

Data cleaning involved handling missing values and converting data types to the appropriate formats. The following steps were taken:

1. Convert to Numeric: Columns such as 'Bedrooms', 'Bathrooms', and 'Toilets' were converted to numeric types, coercing errors to NaN.

2. Impute Missing Values: Missing values in numeric columns were filled using the median value of the respective columns.

3. Categorical Encoding: Categorical columns ('Location', 'Currency', 'Neighborhood') were converted to categorical types and encoded using OneHotEncoder.

python

import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

Example data setup

data = {

'Bedrooms': [3, 4, None, 2, 3],

'Bathrooms': [2, None, 3, 1, 2],

'Toilets': [2, 3, None, 1, 2],

'Location': ['Ikeja', 'Lekki', 'Victoria Island', 'Ikeja', 'Lekki'],

'Currency': ['NGN', 'USD', 'EUR', 'NGN', 'USD'],

'Neighborhood': ['GRA Ikeja', 'Phase 1', 'Central', 'GRA Ikeja', 'Phase 1'],

'Price': [250000, 400000, 600000, 300000, 350000],

'Serviced': [1, 0, 1, 1, 0],

'Newly Built': [1, 0, 0, 1, 0],

'Furnished': [1, 0, 1, 0, 0]

}

df = pd.DataFrame(data)

Convert numeric columns

df['Bedrooms'] = pd.to\_numeric(df['Bedrooms'], errors='coerce')

df['Bathrooms'] = pd.to\_numeric(df['Bathrooms'], errors='coerce')

df['Toilets'] = pd.to\_numeric(df['Toilets'], errors='coerce')

Impute missing values

imputer = SimpleImputer(strategy='median')

numeric\_cols = ['Bedrooms', 'Bathrooms', 'Toilets']

df[numeric\_cols] = imputer.fit\_transform(df[numeric\_cols])

Convert categorical columns to categorical dtype

cat\_columns = ['Location', 'Currency', 'Neighborhood']

df[cat\_columns] = df[cat\_columns].astype('category')

Encode categorical columns

encoder = OneHotEncoder(drop='first', sparse\_output=False)

encoded\_cols = encoder.fit\_transform(df[cat\_columns])

encoded\_cols\_df = pd.DataFrame(encoded\_cols, columns=encoder.get\_feature\_names\_out(cat\_columns))

Combine encoded categorical columns with numeric columns

df\_processed = pd.concat([df[numeric\_cols], encoded\_cols\_df], axis=1)

4. Feature Engineering

Feature engineering involved selecting relevant features for model training and transforming them appropriately. The final set of features included:

Bedrooms

Bathrooms

Toilets

Serviced

Newly Built

Furnished

Encoded categorical features (Location, Currency, Neighborhood)

5. Model Building

The machine learning model used for predicting house prices is a Random Forest Regressor. The steps for model building included:

1. Splitting the data into training and testing sets.

2. Training the Random Forest Regressor on the training data.

3. Saving the trained model and encoders using joblib.

python

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

import joblib

Splitting the data

X = df\_processed

y = df['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Training the model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

Saving the model and encoders

joblib.dump(model, 'House.pkl')

joblib.dump(encoder, 'encoder.pkl')

joblib.dump(df\_processed.columns, 'feature\_names.pkl')

6. Model Deployment with Flask

The model is deployed as a web application using Flask. The steps involved in the deployment include:

1. Creating a Flask application.

2. Defining routes for the home page and prediction results.

3. Loading the trained model and encoders.

4. Preparing input data for prediction.

5. Rendering prediction results on the web page.

Flask Application Code

python

from flask import Flask, request, render\_template

from joblib import load

import pandas as pd

app = Flask(\_\_name\_\_)

Load the pretrained model, encoder, and feature names

model = load('House.pkl')

encoder = load('encoder.pkl')

feature\_names = load('feature\_names.pkl')

Define the dropdown options

dropdown\_options = {

'location': ['Ikeja', 'Lekki', 'Victoria Island'],

'currency': ['NGN', 'USD', 'EUR'],

'neighborhood': ['GRA Ikeja', 'Phase 1', 'Central']

}

Function to prepare input data for prediction

def prepare\_input\_data(bedrooms, bathrooms, toilets, serviced, newly\_built, furnished, location, currency, neighborhood):

Create DataFrame with the form data

input\_data = pd.DataFrame([[bedrooms, bathrooms, toilets, serviced, newly\_built, furnished, location, currency, neighborhood]],

columns=['Bedrooms', 'Bathrooms', 'Toilets', 'Serviced', 'Newly Built', 'Furnished', 'Location', 'Currency', 'Neighborhood'])

Convert categorical columns to categorical dtype

input\_data[['Location', 'Currency', 'Neighborhood']] = input\_data[['Location', 'Currency', 'Neighborhood']].astype('category')

Transform the categorical features

encoded\_cols = encoder.transform(input\_data[['Location', 'Currency', 'Neighborhood']])

encoded\_cols\_df = pd.DataFrame(encoded\_cols, columns=encoder.get\_feature\_names\_out(['Location', 'Currency', 'Neighborhood']))

Combine encoded categorical columns with numeric columns

input\_data\_processed = pd.concat([input\_data[['Bedrooms', 'Bathrooms', 'Toilets', 'Serviced', 'Newly Built', 'Furnished']], encoded\_cols\_df], axis=1)

Ensure the columns match the training columns

input\_data\_processed = input\_data\_processed.reindex(columns=feature\_names, fill\_value=0)

return input\_data\_processed

Define a custom Jinja filter for formatting floats

@app.template\_filter('float\_format')

def float\_format(value):

return "{:.2f}".format(value)

@app.route('/')

def home():

return render\_template('index.html', options=dropdown\_options)

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

def predict\_result():

try:

Get the data from the form

bedrooms = float(request.form['bedrooms'])

bathrooms = float(request.form['bathrooms'])

toilets = float(request.form['toilets'])

serviced = float(request.form['serviced'])

newly\_built = float(request.form['newly\_built'])

furnished = float(request.form['furnished'])

location = request.form['location']

currency = request.form['currency']

neighborhood = request.form['neighborhood']

Prepare input data for prediction

input\_data = prepare\_input\_data(bedrooms, bathrooms, toilets, serviced, newly\_built, furnished, location, currency, neighborhood)

Make prediction using the model

prediction = model.predict(input\_data)

Modify the prediction by adding a zero to the back

prediction = prediction[0] 10

Render the predict.html template with the prediction result

return render\_template('predict.html', prediction=prediction)

except Exception as e:

error\_message = "Error occurred during prediction: " + str(e)

return render\_template('index.html', error=error\_message, options=dropdown\_options)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

HTML Template (predict.html)

html

<!DOCTYPE

html>

<html lang="en">

<head>

<meta charset="UTF8">

<meta name="viewport" content="width=devicewidth, initialscale=1.0">

<title>Prediction Result</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Prediction Result</h1>

<p>Predicted House Price: {{ prediction | float\_format }}</p>

<a href="/">Go back</a>

</div>

</body>

</html>

7. Running the Project

To run the project locally, follow these steps:

1. Install Dependencies: Ensure you have Python installed and install the necessary packages using pip:

bash

pip install flask pandas scikitlearn joblib

2. Prepare the Dataset: Ensure your dataset is cleaned and preprocessed as described in the Data Cleaning section.

3. Train the Model: Use the code provided in the Model Building section to train the model and save it.

4. Run the Flask Application: Use the following command to start the Flask application:

bash

python app.py

5. Access the Application: Open a web browser and navigate to http://127.0.0.1:5000 to access the application.

8. Conclusion

This documentation provides a comprehensive overview of the Lagos House Price Prediction project, detailing the processes from data collection and cleaning to model deployment using Flask. By following the steps outlined, you can replicate the project and extend it further based on your requirements.