Title: House Price Building price Predictionnl

Introduction

The “House Price Prediction Web Application” is a Flask-based web application designed to predict house prices based on a dataset of house-related features. The application utilizes a Linear Regression model to make predictions and offers a user-friendly interface for users to input data and receive predictions and model evaluation metrics.

Exploratory Data Analysis (EDA):

```
Import pandas as pd

Import matplotlib.pyplot as plt

Import seaborn as sns


# Step 1: Load and Inspect Data

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


# Step 2: Data Summary

Print(data.head())  # Display the first few rows

Print(data.info())  # Check data types and non-null counts

Print(data.describe())  # Generate summary statistics for numerical features


# Step 3: Data Cleaning

# Handle missing values, outliers, and anomalies as needed


# Step 4: Data Visualization

# Numerical feature histograms

Numerical features = ['Square_Footage', 'Bedrooms', 'Bathrooms', 'Year_Built', 'Price']
```

```
For feature in numerical_features:
    Plt.figure(figsize=(6, 4))
    Sns.histplot(data[feature], kde=True)
    Plt.title(f'Distribution of {feature}')
    Plt.show()

# Step 5: Feature Analysis
# Calculate correlations between features and the target variable (Price)
Correlation_matrix = data.corr()
Plt.figure(figsize=(8, 6))
Sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
Plt.title('Correlation Matrix')
Plt.show()

# Step 6: Data Distribution
# Check the distribution of the target variable (Price)
Plt.figure(figsize=(6, 4))
Sns.histplot(data['Price'], kde=True)
Plt title('Distribution of Price')
Plt.show()

# Step 7: Handling Categorical Variables
# Visualize categorical variables, e.g., 'Location' and 'Zip_Code'
Categorical_features = ['Location', 'Zip_Code']
For feature in categorical_features:
    Plt.figure(figsize=(8, 6))
    Sns.countplot(data[feature])
    Plt.title(f'Count Plot of {feature}')
    Plt.xticks(rotation=90)
```

Plt.show()

Step 8: Multicollinearity

Check for multicollinearity among features, e.g., using a correlation matrix

Step 9: Feature Engineering

Create new features or transformations that improve predictive power

Step 10: Train-Test Split

Split the data into training and testing sets, e.g., using train_test_split

Step 11: Model Training

Choose an appropriate model and train it

Step 12: Model Evaluation

Assess model performance using appropriate metrics, e.g., RMSE

Continue with model development, deployment, and the Flask application as in the original code.

Components:

1. Identify and handle missing values, outliers, and anomalies in the data. You might need to apply data cleaning techniques such as imputation and removal of outliers.
2. Data Visualization:

Create various visualizations to explore relationships between features and the target variable (Price). You can use libraries like matplotlib and seaborn to create scatter plots, pair plots, and correlation matrices.

3. Feature Analysis:

Analyze the significance of each feature in relation to the target variable. You can calculate correlations and use feature importance techniques.

4. Data Distribution:

Check the distribution of the target variable (Price). A histogram or density plot can help you understand if the data is normally distributed.

5. Handling Categorical Variables:

If you have categorical variables (like “Location” and “Zip_Code”), explore their impact on house prices. You can create bar plots or box plots to visualize this.

6. Multicollinearity:

Detect and handle multicollinearity among features. This is important when using linear regression.

7. Feature Engineering:

Consider creating new features or transformations that could improve the model’s predictive power. For example, calculating the age of the house from a “Year_Built” feature.

8. Train-Test Split:

Split the data into training and testing sets using train_test_split. This will be essential for model evaluation.

9. Model Training:

After EDA, the code in your original script fits a Linear Regression model to the training data. However, you may also consider other models based on your EDA findings.

10. Model Evaluation:

The code calculates the Root Mean Squared Error (RMSE) as an evaluation metric. You can select additional metrics and assess how well the model performs, considering the insights gained during EDA.

The EDA process helps you understand the dataset, identify potential challenges, and make informed decisions regarding data preprocessing, feature selection, and model choice. Once you have a better grasp of the data, you can build a more effective house price prediction mode

Python

```
Import pandas as pd
Import matplotlib.pyplot as plt
Import seaborn as sns

# Step 1: Load and Inspect Data
Data = pd.read_csv('house_data.csv')

# Step 2: Data Summary
Print(data.head())  # Display the first few rows
Print(data.info())  # Check data types and non-null counts
Print(data.describe())  # Generate summary statistics for numerical features

# Step 3: Data Cleaning
# Handle missing values, outliers, and anomalies as needed

# Step 4: Data Visualization
# Numerical feature histograms
Numerical_features = ['Square_Footage', 'Bedrooms', 'Bathrooms', 'Year_Built', 'Price']
For feature in numerical_features:
    Plt.figure(figsize=(6, 4))
    Sns.histplot(data[feature], kde=True)
    Plt.title(f'Distribution of {feature}')
    Plt.show()

# Step 5: Feature Analysis
# Calculate correlations between features and the target variable (Price)
Correlation_matrix = data.corr()
Plt.figure(figsize=(8, 6))
```

```
Sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
Plt.title('Correlation Matrix')
Plt.show()

# Step 6: Data Distribution
# Check the distribution of the target variable (Price)
Plt.figure(figsize=(6, 4))
Sns.histplot(data['Price'], kde=True)
Plt.title('Distribution of Price')
Plt.show()

# Step 7: Handling Categorical Variables
# Visualize categorical variables, e.g., 'Location' and 'Zip_Code'
Categorical_features = ['Location', 'Zip_Code']
For feature in categorical_features:
    Plt.figure(figsize=(8, 6))
    Sns.countplot(data[feature])
    Plt.title(f'Count Plot of {feature}')
    Plt.xticks(rotation=90)
    Plt.show()

# Step 8: Multicollinearity
# Check for multicollinearity among features, e.g., using a correlation matrix

# Step 9: Feature Engineering
# Create new features or transformations that improve predictive power

# Step 10: Train-Test Split
# Split the data into training and testing sets, e.g., using train_test_split
```

Step 11: Model Training

Choose an appropriate model and train it

Step 12: Model Evaluation

Assess model performance using appropriate metrics, e.g., RMSE

Code :

index.html:-

```
<!DOCTYPE html>
<html>
<head>
   <title>House Price Prediction</title>
</head>
<body>
   <h1>House Price Prediction</h1>
   <form method=”POST” action=”/result”>
      <input type=”submit” value=”Predict”>
   </form>
</body>
</html>
```

App. Py:-

```
From flask import Flask, render_template, request, redirect, url_for
Import pandas as pd
From sklearn.model_selection import train_test_split
From sklearn.linear_model import LinearRegression
From sklearn.metrics import mean_squared_error
```

```
Import matplotlib.pyplot as plt
Import io
Import base64

App = Flask(__name__)

@app.route('/')
Def index():
   Return render_template('index.html')

@app.route('/result', methods=['GET', 'POST'])
Def result():
   If request.method == 'POST':
      Data = pd.read_csv('house_data.csv')
      Data = pd.get_dummies(data, columns=['Location', 'Zip_Code'])

      X = data.drop('Price', axis=1)
      Y = data['Price']

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

      Model = LinearRegression()
      Model.fit(X_train, y_train)

      Predictions = model.predict(X_test)

      Results = pd.DataFrame({'Actual Prices': y_test, 'Predicted Prices': predictions})

      Mse = mean_squared_error(y_test, predictions)
```

```
        Rmse = (mse) ** 0.5

        Plt.figure(figsize=(8, 6))
        Plt.scatter(y_test, predictions, alpha=0.5)
        Plt.title('Actual Prices vs. Predicted Prices')
        Plt.xlabel('Actual Prices')
        Plt.ylabel('Predicted Prices')
        Plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'k—', lw=2)

        Img = io.BytesIO()
        Plt.savefig(img, format='png')
        Img.seek(0)
        Plot_url = base64.b64encode(img.getvalue()).decode()

        Return render_template('result.html', mse=rmse, plot_url=plot_url)

    Return redirect(url_for('index'))

If __name__ == '__main__':
    App.run(debug=True)
```

Result.html:-

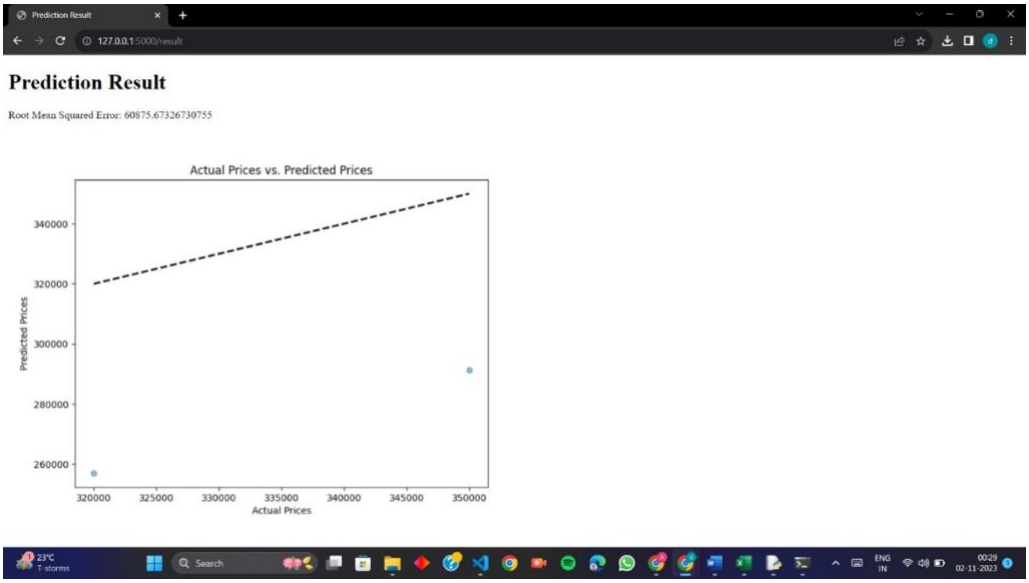
```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>
</head>
<body>
    <h1>Prediction Result</h1>
    <p>Root Mean Squared Error: {{ mse }}</p>
```

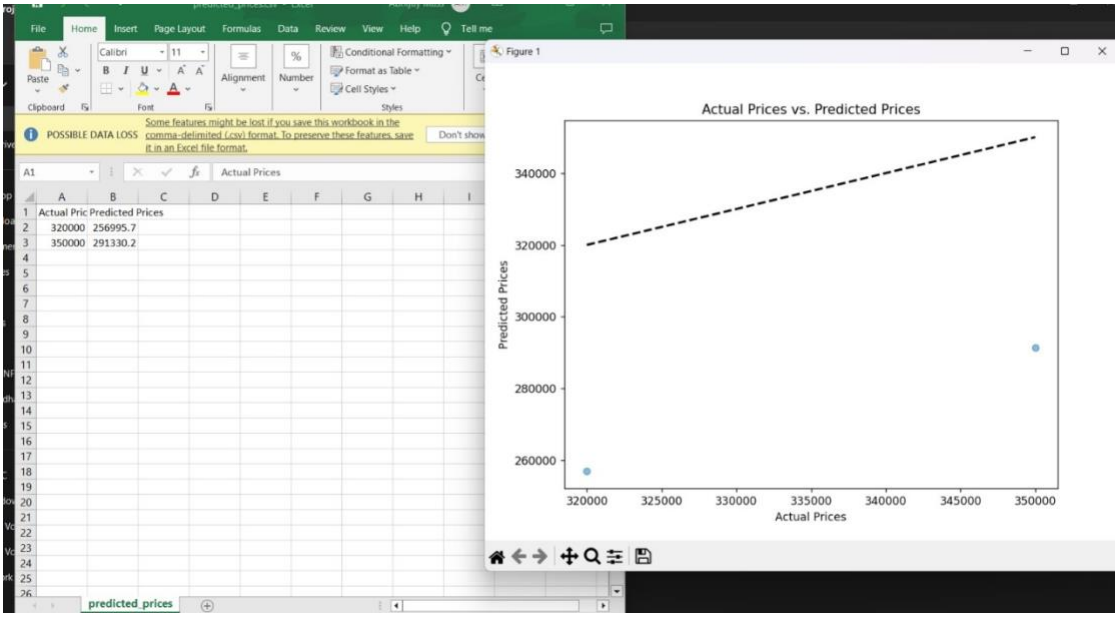
<img src=”{{ plot_url }}” alt=”Scatter Plot”>

</body>

</htm>

Result and output :-





Conclusion:

The “House Price Prediction Web Application” provides a user-friendly interface for predicting house prices. Users can input data, and the application employs a Linear Regression model to make predictions. It also calculates the RMSE as an evaluation metric and visualizes the predictions with a scatter plot. This application serves as a practical tool for individuals and professionals in the real estate industry to estimate house prices and make informed decisions. It can be further enhanced with additional features and deployed on a production server for wider accessibility.