Name : Patel Megh B **Er No.**: 22162121011 **Sem** : 7 **Subject** : CGC

Mini Project : House Price Prediction Model

Problem Statement : Build a Machine Learning model that predicts the price of a house based on its features using regression algorithms.

Tasks Performed :

☐ Load and clean the **"House Price India.csv"** dataset.

☐ Perform **basic analysis and visualization** (e.g., condition of house vs. average price).

☐ Train and compare three regression models:

- Decision Tree Regressor (with hyperparameter tuning using GridSearchCV)
- Linear Regression
- Random Forest Regressor (with hyperparameter tuning using GridSearchCV)

☐ Evaluate models using:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**

☐ Save the **best model** as model.pkl using joblib for future use (e.g., in a web app like Streamlit or Flask).

Results and Interpretation:

After training and evaluating the three models on the test set, we compare their performance based on **MSE** and **MAE**:

- **Decision Tree Regressor**
  - MSE: 67739876625.71646
  - MAE: 161090.87323793155
- **Linear Regression**
  - MSE: 65514176740.267784
  - MAE: 165026.2962930765
- **Random Forest Regressor**
  - MSE: 61319014579.76153
  - MAE: 158602.34822515058

Code :

App.py:

```
import streamlit as st
import numpy as np
import joblib

model = joblib.load('model1.pkl')

st.title("House Price Prediction")

st.divider()

st.write("Enter the details of the house to predict its price:")

st.divider()

bedrooms = st.number_input("Number of Bedrooms", min_value=1, value=1)
bathrooms = st.number_input("Number of Bathrooms", min_value=0, value=1)
livingarea = st.number_input("Square Footage of Living Area", min_value=0,
value=2000)
condition = st.number_input("Condition of the House", min_value=0, value=3)
noofschools = st.number_input("Number of Schools Nearby", min_value=0,
value=0)

st.divider()

X = [[bedrooms, bathrooms, livingarea, condition, noofschools]]

predictbutton = st.button("Predict Price")

if predictbutton:
    st.balloons()

    X_array = np.array(X)
    prediction = model.predict(X_array)

    st.write(f"🏠 The predicted price of the house
is:  {prediction[0]:,.2f}")

else:
    st.write("Please click the **Predict Price** button after entering
details.")



# Command to run : streamlit run g:/housePricePrediction/app1.py
```

Main.py:

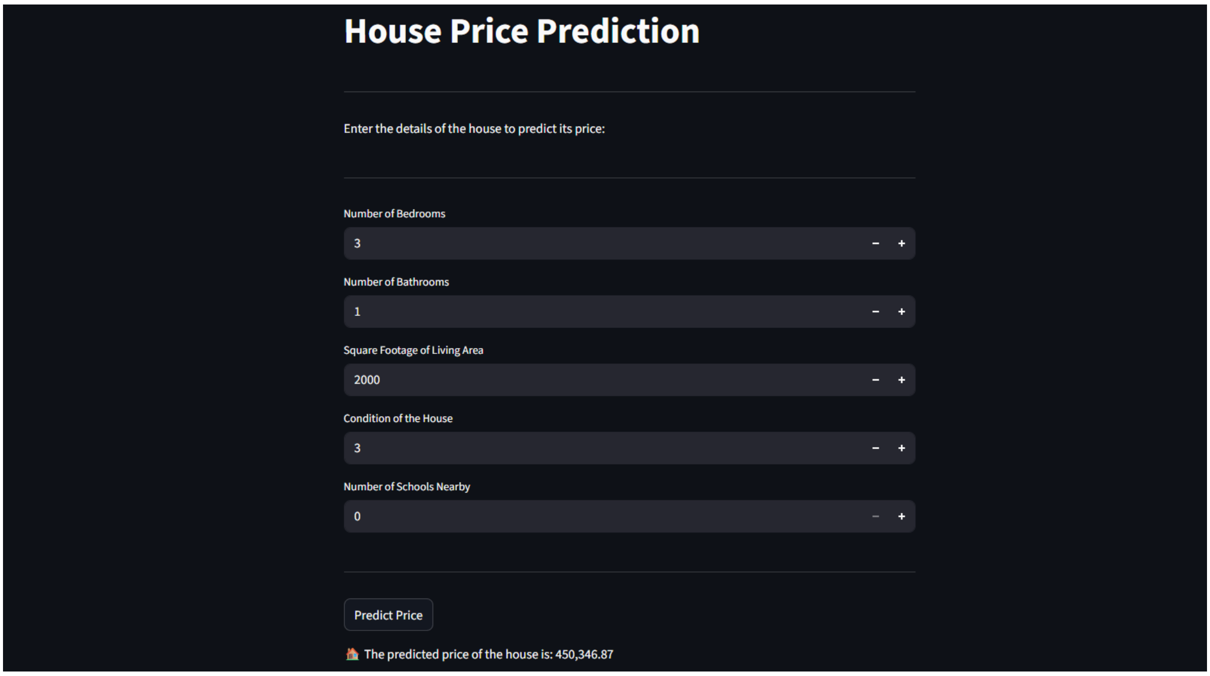
```python
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error

import joblib

data = pd.read_csv('House Price India.csv')

# Show basic info about columns, data types, and non-null counts
print("\n--- Data Info ---")
print(data.info())

#Basic Statistics of Target Variable (Price)
# Describe the "Price" column: count, mean, std, min, quartiles, max
stats = data["Price"].describe().reset_index()

# Round the statistics values to 2 decimal places for better readability
stats["Price"] = stats["Price"].round(2)

print("\n--- Price Statistics ---")
print(stats)

#Check Missing and Duplicate Values

# Total number of missing values in the entire DataFrame
total_missing = data.isna().sum().sum()
print(f"\nTotal missing values in dataset: {total_missing}")

# Total number of fully duplicated rows
total_duplicates = data.duplicated().sum()
print(f"Total duplicated rows in dataset: {total_duplicates}")

#Handle Missing and Duplicate Values

# Drop rows with any missing value
data.dropna(inplace=True)

# Drop exactly duplicated rows
data.drop_duplicates(inplace=True)

print("\n--- Data after cleaning (head) ---")
```

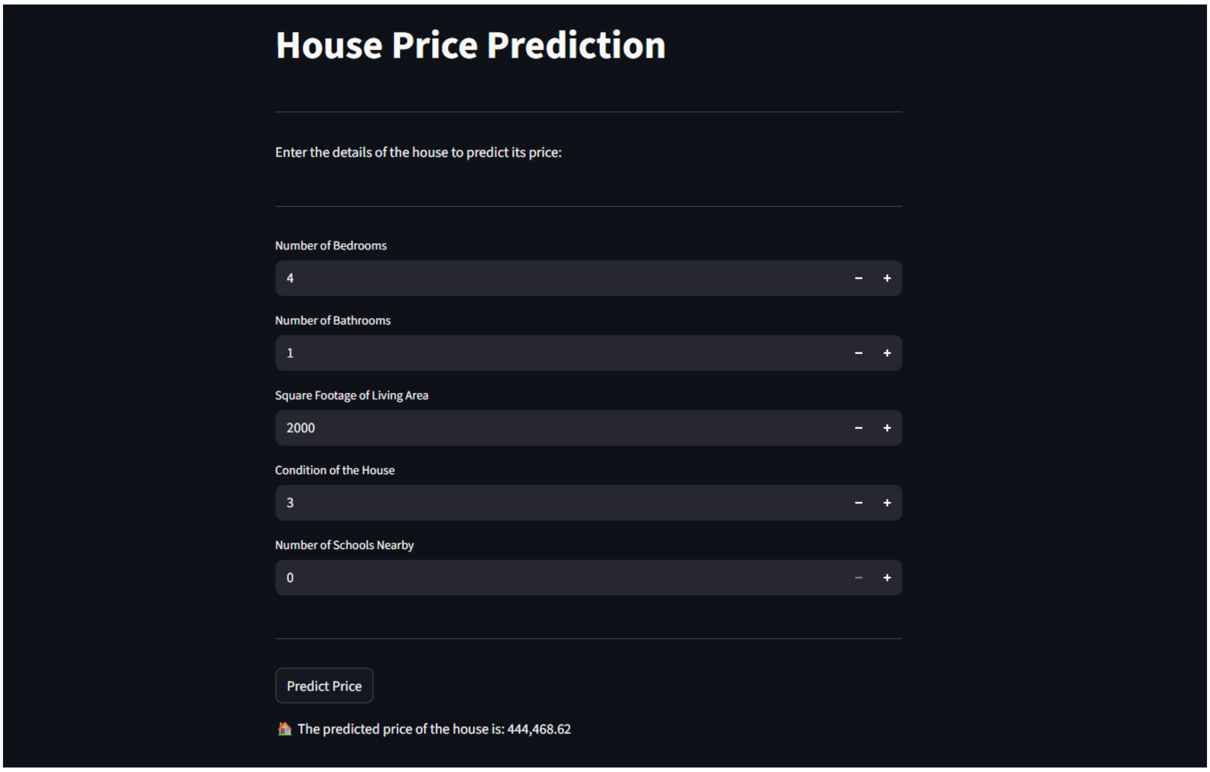
```
print(data.head())

data.groupby("condition of the
house")["Price"].mean().sort_values(ascending=True).plot(kind='bar')
plt.title("Condition of the House vs Average Price")
plt.ylabel("Average Price")
plt.xlabel("Condition of the House")
plt.show()

X = data[['number of bedrooms','number of bathrooms','living area','condition
of the house','Number of schools nearby']]

X

y = data["Price"]

y

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

print("\nTraining set size:", X_train.shape, y_train.shape)
print("Test set size:", X_test.shape, y_test.shape)

param_grid_tree = {
    "criterion": ["squared_error", "friedman_mse", "absolute_error"],
    "splitter": ["best", "random"],
    "max_depth": [None, 10, 20, 30, 40, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}


# Base Decision Tree model
tree_model = DecisionTreeRegressor(random_state=42)

# GridSearchCV tries all combinations of parameters and picks the best
grid_tree = GridSearchCV(
    estimator=tree_model,
    param_grid=param_grid_tree,
    scoring="neg_mean_squared_error",  # we want to minimize
MSE                         # use all CPU cores
)

# Fit the model on training data
grid_tree.fit(X_train, y_train)

print("\n--- Best Parameters for Decision Tree ---")
```

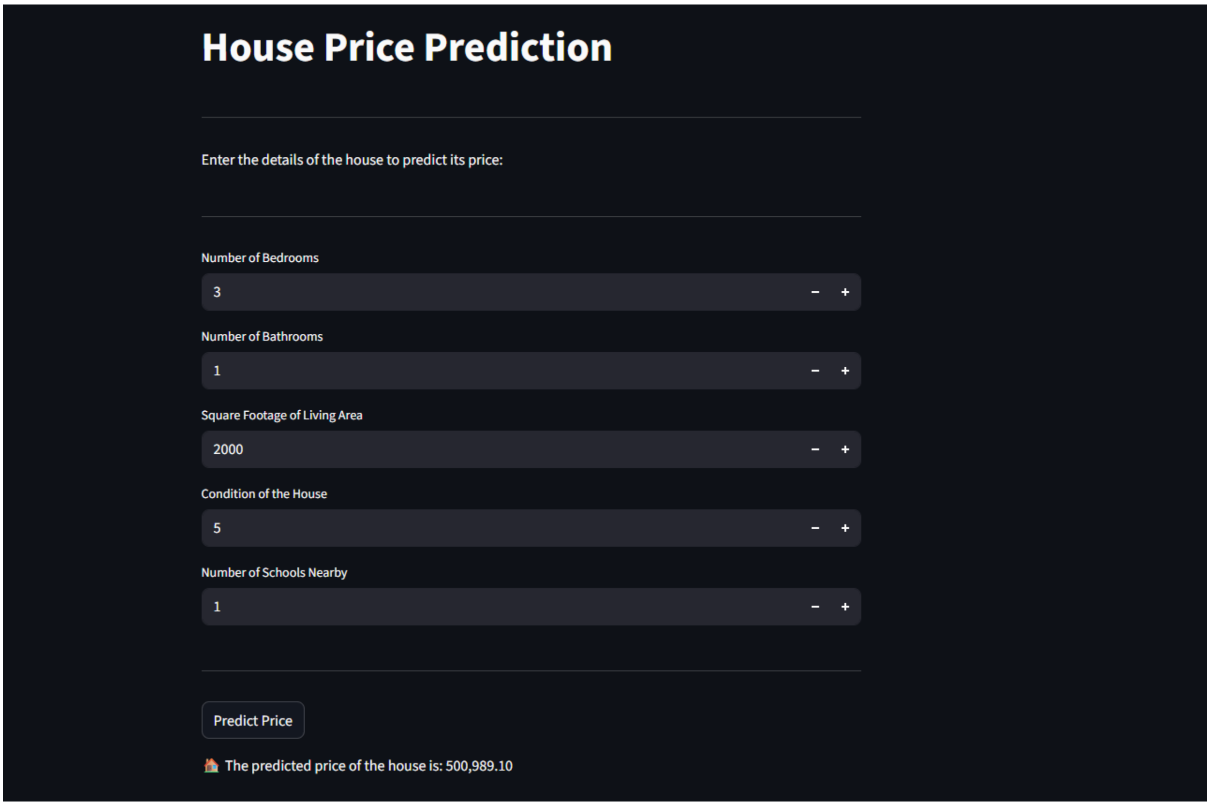
```python
print(grid_tree.best_params_)

# Predictions on test set using best Decision Tree model
tree_preds = grid_tree.predict(X_test)

# Evaluate Decision Tree
tree_mse = mean_squared_error(y_test, tree_preds)
tree_mae = mean_absolute_error(y_test, tree_preds)

print("\nDecision Tree Performance:")
print("MSE:", tree_mse)
print("MAE:", tree_mae)

# Initialize Linear Regression model
lr = LinearRegression()

# Train on training data
lr.fit(X_train, y_train)

# Predict on test data
lr_preds = lr.predict(X_test)

# Evaluate Linear Regression
lr_mse = mean_squared_error(y_test, lr_preds)
lr_mae = mean_absolute_error(y_test, lr_preds)

print("\nLinear Regression Performance:")
print("MSE:", lr_mse)
print("MAE:", lr_mae)

# Base Random Forest model
rf_model = RandomForestRegressor(random_state=42)

# Parameter grid for Random Forest
param_grid_rf = {
    "max_depth": [5, 10, 15],
    "n_estimators": [2, 3, 4, 5, 6, 7, 8, 9, 10]
}


# GridSearchCV for Random Forest
grid_rf = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid_rf,
    scoring="neg_mean_squared_error"
)

# Fit Random Forest with GridSearch on training data
```

```
grid_rf.fit(X_train, y_train)

print("\n--- Best Parameters for Random Forest ---")
print(grid_rf.best_params_)

# Predictions with best Random Forest model
rf_preds = grid_rf.predict(X_test)

# Evaluate Random Forest
rf_mse = mean_squared_error(y_test, rf_preds)
rf_mae = mean_absolute_error(y_test, rf_preds)

print("\nRandom Forest Performance:")
print("MSE:", rf_mse)
print("MAE:", rf_mae)

#Save best model as pkl file
joblib.dump(grid_rf, 'model1.pkl')
```

Output :




House Price Prediction
Enter the details of the house to predict its price:
Number of Bedrooms
4
Number of Bathrooms
1
Square Footage of Living Area
2000
Condition of the House
3
Number of Schools Nearby
0
Predict Price
The predicted price of the house is: 444,468.62


House Price Prediction
Enter the details of the house to predict its price:
Number of Bedrooms
3
Number of Bathrooms
1
Square Footage of Living Area
2000
Condition of the House
5
Number of Schools Nearby
1
Predict Price
The predicted price of the house is: 500,989.10