**Source code** **: Forecasting housing price accurately using smart regression technique in data science**

import pandas as pd import numpy as np import seaborn as sns import matplotlib.pyplot as plt from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score import xgboost as xgb

import gradio as gr

# 1. Data Loading & Preprocessing (Using California Housing Dataset)

from sklearn.datasets import fetch\_california\_housing # Import California housing dataset

# Load California housing dataset california = fetch\_california\_housing()

data = pd.DataFrame(california.data, columns=california.feature\_names) data['PRICE'] = california.target # Target variable is 'PRICE'

# 2. EDA (Exploratory Data Analysis)

sns.set(style="whitegrid") plt.figure(figsize=(10, 6))

sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1) plt.title('Correlation Matrix of Features') plt.show()

# Pairplot to visualize relationships sns.pairplot(data, diag\_kind='kde') plt.show()

# 3. Data Preprocessing

# Splitting the dataset into features (X) and target variable (y)

X = data.drop('PRICE', axis=1) y = data['PRICE']

# Splitting the dataset into training and testing sets

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

# 4. Modeling: Linear Regression, Random Forest Regressor, and XGBoost Regressor

# Linear Regression Model lr\_model = LinearRegression()

lr\_model.fit(X\_train, y\_train)

# Random Forest Regressor Model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) rf\_model.fit(X\_train, y\_train)

# XGBoost Regressor Model

xgb\_model = xgb.XGBRegressor(n\_estimators=100, learning\_rate=0.05, random\_state=42) xgb\_model.fit(X\_train, y\_train)

# 5. Model Evaluation (Mean Absolute Error, Mean Squared Error, and R^2 score) def evaluate\_model(model, X\_test, y\_test): y\_pred = model.predict(X\_test) mae = mean\_absolute\_error(y\_test, y\_pred) mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred) return mae, mse, r2

# Evaluating all models models = {

"Linear Regression": lr\_model,

"Random Forest": rf\_model,

"XGBoost": xgb\_model

}

for name, model in models.items(): mae, mse, r2 = evaluate\_model(model, X\_test, y\_test) print(f"Model: {name}") print(f"Mean Absolute Error: {mae:.2f}") print(f"Mean Squared Error: {mse:.2f}") print(f"R^2 Score: {r2:.2f}\n")

# 6. Visualization of Model Performance

# Comparing predicted vs actual for XGBoost (best model) y\_pred = xgb\_model.predict(X\_test)

plt.figure(figsize=(10, 6)) plt.scatter(y\_test, y\_pred) plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2) plt.xlabel('Actual Prices') plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Housing Prices (XGBoost)') plt.show()

# 7. Gradio UI for Model Deployment

def predict\_price(features): # Convert features to a DataFrame

features = np.array(features).reshape(1, -1)

# Predict using XGBoost model (best performing model) predicted\_price = xgb\_model.predict(features) return predicted\_price[0]

# Create Gradio Interface for prediction

# The number of features in California housing dataset is different from Boston

# We need to update the number of sliders based on the new dataset # Changed gr.inputs.Slider to gr.Slider and gr.outputs.Textbox to gr.Textbox inputs = [gr.Slider(minimum=data[col].min(), maximum=data[col].max(),

value=data[col].mean(), label=col) for col in california.feature\_names]

output = gr.Textbox(label="Predicted Housing Price")

gr.Interface(fn=predict\_price, inputs=inputs, outputs=output, live=True).launch()