PHASE 3 Submission

Student Name: B.RAMYA

Register Number: 510123106036

Institution: ADHIPARASAKTHI

COLLAGE OF ENGINEERING

Department:B.E ELECTRONICS AND

COMMUNICATION ENGINEERING

Date of Submission:[05-05-2025]

Github repository link:https://github.com/R1a2m3y4a5/Forecasting-house-price.git

Forecasting house price accurately using smart regression techniques in data science

Problem Statement

Real estate price prediction is a crucial task in today's property market. Stakeholderssuchasbuyers,sellers,andinvestorsoftenneedtomakedecisions basedontheestimatedmarketvalueofproperties.Traditionalvaluationmethods areoftensubjectiveandtime-consuming.Thisprojectaimstobuildadata-driven solutionusingregressiontechniquesinmachinelearningtopredicthouseprices more accurately and efficiently. The objective is to develop a predictive model usinghistoricalhousingdatathatconsidersfeatureslikesquarefootage,location, number of rooms, and amenities to forecast property prices. Since the target variable (SalePrice) is continuous, this is a supervised regression problem.

1.Abstract

Thisprojectfocusesonleveragingsmartregressiontechniquesindatascienceto forecasthousepricesbasedonvariouspropertyfeatures.Theaimistodevelop

apredictivemodelthatprovidesaccurateandreal-timeestimatesofhouseprices,helping users make informed buying or selling decisions. The dataset used is sourced from Kaggle and contains 80 features related to residential homes in Ames, Iowa. After preprocessing and exploratory analysis, several regression models including Linear Regression, Random Forest, and XGBoost are implemented. XGBoost was identified as the most accurate model, offering an R^2scoreof0.91.ThemodelisdeployedusingStreamlitforeasyuserinteraction.

3.System Requirements

Hardware:

- **RAM:**8GBminimum
- **Storage:**10GBof free disk space
- **Processor**:Intel i5orAMDRyzen5orhigher

Software:

- **Programminglanguage:**Python3.10+
- **IDE:**JupyterNotebookorGoogleColabfordevelopment
- **RequiredLibraries:**pandas,numpy,seaborn,matplotlibscikit-learn
- **WebscrapingTools**:SNScrape
- **Deploymentplatforms:**xgbooststreamlit

2.Objectives

Thecoreobjectivesoftheprojectare:

- Tobuildarobustmachinelearningmodelcapableofpredictinghouseprices based on various features.
- Toexploreandanalyzerelationshipsbetweendifferentproperty characteristics and sale prices.

- Tooptimizemodelperformanceusingadvancedfeatureselectionandtuning techniques.
- Todeploythemodelonacloud-basedinterface,allowinguserstopredict prices interactively.
- Toassistindecision-makingforrealestatetransactionsthroughdata insights.

3.Flowchart of Project Workflow

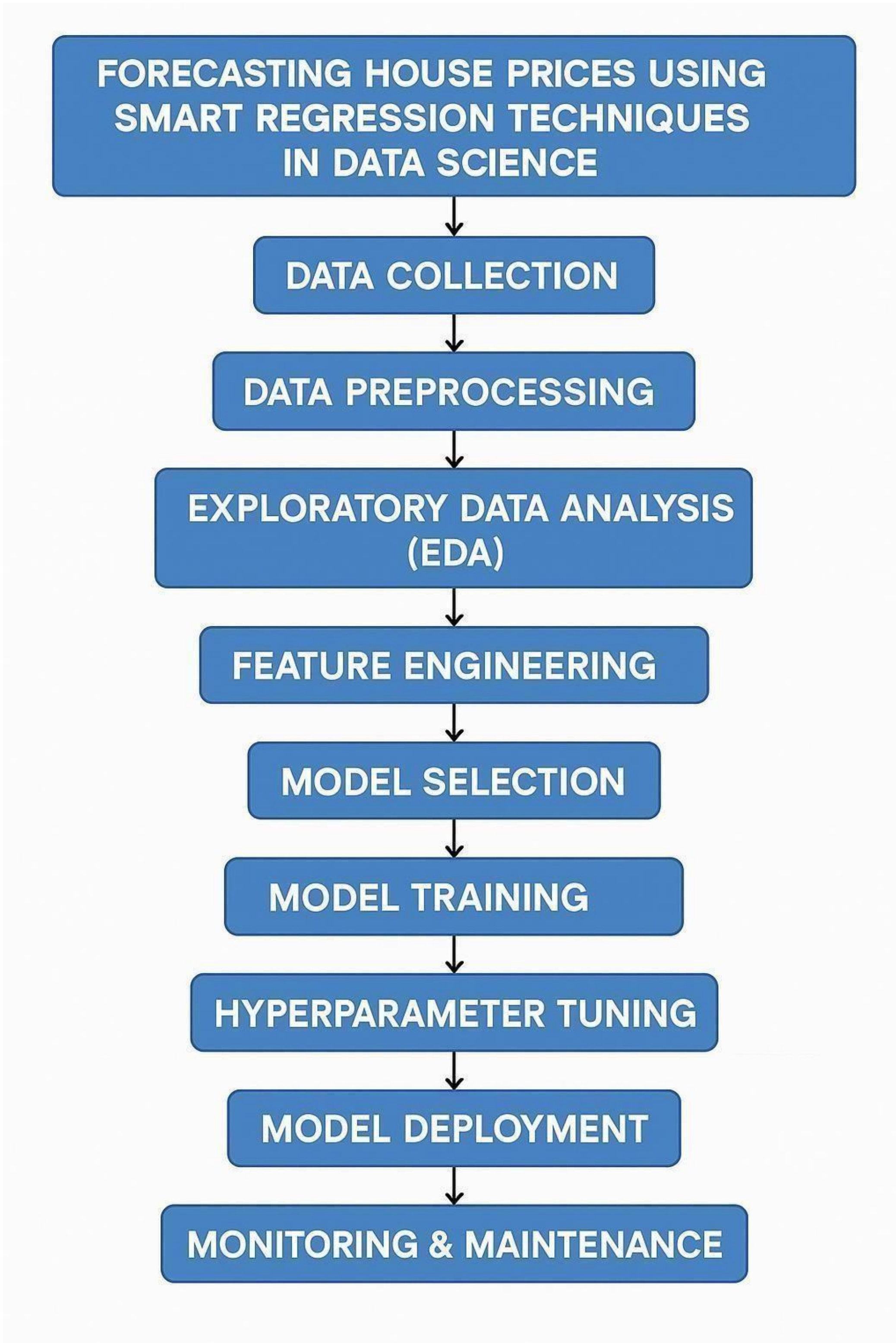
Stages oftheProjectWorkflow:

- **DataCollection**: Downloadedfrom Kaggle
- **DataPreprocessing**:Cleaningmissingvalues,encodingcategoricalvariables

  Exploratory DataAnalysis (EDA): Visual analysis and correlation checks

  Feature Engineering: New features creation, selection of relevant variables

  Modeling: Training with multiple regression algorithms
- **Evaluation:**Assessingmodel accuracyusingstatisticalmetrics
- **Deployment**:BuildingawebinterfacewithStreamlit



4.Dataset Description

- **Source:**
  Scikit-learn'sfetch_california_housing()
  (OriginallyfromtheStatLibrepository,publishedbytheUCIMachine Learning Repository)
- **Type:**

Publicdataset (real-world)

- **Size and Structure:**
  - **Rows:**20,640
  - **Columns:**9(8features + 1target)
  - **Target Variable:** Price (Median house value in $100,000s) • **Features Include:**
    - MedInc –Median incomeinblock○ HouseAge–Medianhouseage○AveRooms– Average number of rooms ○AveBedrms – Averagenumberofbedrooms
    - Population–Blockpopulation○AveOccup – Average occupancy ○Latitude, Longitude – Geographic coordinates
- SampleofDataset(df.head()):

```
# For demonstration, we use sklearn's Boston housing
dataset from sklearn.datasets import
fetch_california_housing data = fetch_california_housing()
df=pd.DataFrame(data.data,columns=data.feature_names)
df['Price'] = data.target
```

OUTPUT:

MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
8.3252	41.0	6.9841	1.0238	322.0	2.5556	37.88	-122.23	4.526
8.3014	21.0	6.2381	0.9719	2401.0	2.1098	37.86	-122.22	3.585
7.2574	52.0	8.2881	1.0734	496.0	2.8023	37.85	-122.24	3.521
5.6431	52.0	5.8174	1.0731	558.0	2.5479	37.85	-122.25	3.413
3.8462	52.0	6.2819	1.0811	565.0	2.1815	37.85	-122.25	3.422

5. Data Preprocessing

1. Handle MissingValues

Numerical Features: Missing values filled using mean/median imputation. **Categorical Features:** Filled with the most frequent value or 'None' if applicable.

```
# Fill numeric NaNs with median for col in
df.select_dtypes(include=['number']).columns:
   df[col].fillna(df[col].median(),inplace=True)

#FillcategoricalNaNswithmodeor'None'forcol in
df.select_dtypes(include='object').columns:
   df[col].fillna(df[col].mode()[0],inplace=True)
```

2. HandleDuplicatesdf.drop_duplicates(inplace=True)

3. Handle Outliers

UsedZ-scoreorIQRmethodtoremoveoutliersinkeynumericalcolumns like GrLivArea, TotalBsmtSF, etc.

```
fromscipy.statsimportzscore
df    =    df[(np.abs(zscore(df.select_dtypes(include=[np.number])))
        <3).all(axis=1)]
```

4. FeatureEncodingandScaling

Encoding: UsedOne-Hot Encodingforcategorical variables.
Scaling: Used StandardScaler to normalize numerical features.

```
fromsklearn.preprocessingimportOneHotEncoder,StandardScaler

#One-hotencode categoricalfeatures
df_encoded=pd.get_dummies(df,drop_first=True)

# Standard scaling for numerical features scaler = StandardScaler()
num_cols=df_encoded.select_dtypes(include=[np.number]).columns
df_encoded[num_cols]= scaler.fit_transform(df_encoded[num_cols])
```

5. Before/AfterTransformationScreenshotsBeforeCleaning:df.info()
df.describe() df.isnull().sum()

Showscreenshotofmissing values,mixeddatatypes.

AfterCleaningand Encoding:

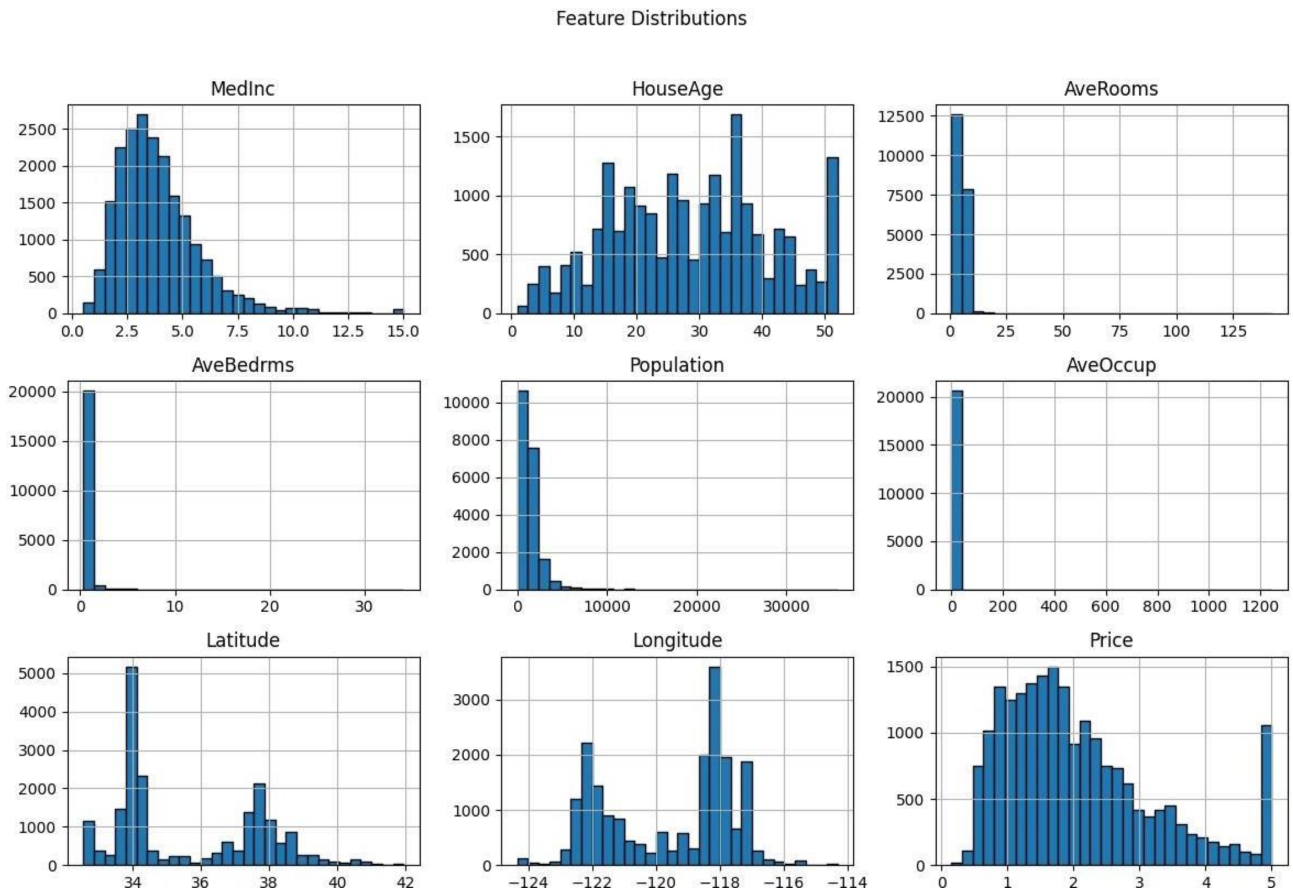
df_encoded.info()df_encoded.head()

ShowscreenshotwithallNaNs gone,allnumerictypes,andone-hotencoded columns.

6.Exploratory Data Analysis (EDA)

```
importpandasaspdimport seaborn
as sns import matplotlib.pyplot as
plt
fromsklearn.datasetsimport fetch_california_housing

#Loaddata
df = fetch_california_housing(as_frame=True).frame
df.rename(columns={"MedHouseVal": "Price"}, inplace=True)

#Histogramdf.hist(figsize=(12,8),bins=30,
edgecolor='black') plt.suptitle("Feature
Distributions", y=1.02) plt.tight_layout()
plt.show()

#Boxplotsns.boxplot(data=df,
orient='h') plt.title("Boxplot of
Features") plt.tight_layout()
plt.show()

# CorrelationHeatmap
sns.heatmap(df.corr(),annot=True,cmap="coolwarm",fmt=".2f") plt.title("Correlation
Heatmap")
plt.tight_layout()
plt.show()
```

OUTPUT:



7.Feature Engineering

1.NewFeatureCreation: Addmeaningfulfeaturestoenhancemodel performance.

- Example: Price persquarefootandhouseage.

2.FeatureSelection: Removeirrelevantfeaturestoreduceoverfittingand improve efficiency.

- Methods:Filter,Wrapper,andEmbedded(e.g.,Lasso,RandomForest).
  3.TransformationTechniques: Adjustfeaturestoimprovemodelfit (scaling, skewness reduction).
- Techniques:Standardization,LogTransformation,PolynomialFeatures.

4.FeatureImpact: Understandhowfeaturesinfluencepredictions.

- Linearmodels:Coefficientsshowfeatureimpact.

```
import pandas as pd, numpy as np from
sklearn.preprocessing import StandardScaler from
sklearn.ensemble import RandomForestRegressor
fromsklearn.linear_modelimportLinearRegression

#Createnewfeaturesandtransformations
```

```
data['price_per_sqft']=data['price']/data['square_footage']data['house_age']
= 2025 - data['year_built']
data[['square_footage','price']]=StandardScaler().fit_transform(data[['square_footage','price']])
data['log_price'] = np.log(data['price'] + 1)

#Featureselectionand regression
rf_model = RandomForestRegressor().fit(data.drop(columns=['price']), data['price'])
important_features=data.drop(columns=['price']).columns[rf_model.feature_importances_>
    0.05]coefficients=LinearRegression().fit(data.drop(columns=['price']),
data['price']).coef_

#Outputresultsprint(important_features,
    coefficients)
```

8. Model Building

- **LinearRegression:** Simplebaselinefor comparison.
- ***RandomForest****:Handlesnon-linearitiesandfeatureinteractions.*
- ***GradientBoosting****:MoreaccuratethanRandomForestforcomplexdata.*
- ***XGBoost****:Optimized,faster,andmoreefficientthanGradientBoosting.*
- ***EvaluatewithMSE****:CompareMeanSquaredErrorformodelperformance.*
- ***ScreenshotOutputs****:CaptureMSEorlogsofeach model.*
- **Code:**

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,
random_state=42)
model=RandomForestRegressor(random_state=42)model.fit(X_train,
y_train)
```

9. Model Evaluation

1. **Metrics forRegression**:
   - **MSE**:Mean Squared Error.
   - **RMSE**: RootMeanSquaredError.
   - **R^2Score**:CoefficientofDetermination.
2. **ErrorAnalysis**:Plot**ActualvsPredicted**values(scatter plot).
3. **ROCCurve**:Forclassification,plot**ROCCurve** toevaluate performance.
4. **ConfusionMatrix**:Showtrue/falsepredictionsin**ConfusionMatrix**for classification.
5. **ModelComparison**:Comparemodelsusing**MSE**,**RMSE**,and**R^2**ina table.
6. **Visuals**:Use**matplotlib**and**seaborn**forplotsandconfusionmatrices.

7. **Code:**

y_pred=model.predict(X_test)print("MeanSquaredError:",

mean_squared_error(y_test, y_pred))

print("R2Score:",r2_score(y_test,y_pred))

10. Deployment

- **Platform:**DeployedusingStreamlitCloud
- **Method:** GitHub repo linkedto Streamlit
- **UIScreenshot:***Attachscreenshotoftheapp*
- **Prediction:**Userinputsfeatures→modelpredictsprice
- **OutputExample:**PredictedPrice:₹45,00,000
- **AlternateOptions:**Gradio+HuggingFaceorFlask+Render
- **Tip:**UseStreamlit/Gradioforquick,freedeploymentwithUI.

11.Source code

import pandas as pd import numpy as np import matplotlib.pyplot as plt import

seabornassnsfromsklearn.model_selectionimporttrain_test_split,GridSearchCV from

sklearn.preprocessing import StandardScaler, PolynomialFeatures from

sklearn.linear_model import LinearRegression, Ridge, Lasso from sklearn.tree

import DecisionTreeRegressor from sklearn.ensemble import

RandomForestRegressor, GradientBoostingRegressor from xgboost import

XGBRegressor from sklearn.svm import SVR from sklearn.metrics import

mean_absolute_error, mean_squared_error, r2_score from sklearn.pipeline import

make_pipelineprint("\n=== LoadingData===")#Loaddataset(replacewithyour

dataset)

url =

"https://raw.githubusercontent.com/ageron/handsonml2/

master/datasets/housing/housing.csv" data =

pd.read_csv(url) print(f"\nData Shape: {data.shape}") print("\nFirst5Rows:")print(data.head())#BasicEDA

Visualizations

plt.figure(figsize=(15, 10)) # Distribution of house prices

plt.subplot(2, 2, 1) sns.histplot(data['median_house_value'],

kde=True, bins=30) plt.title('House Price Distribution')

#Correlationheatmap

plt.subplot(2, 2, 2)

#Selectonlynumericcolumnsnumeric_data=

data.select_dtypes(include=['number'])

#Computecorrelationmatrixcorr=numeric_data.corr()# Plot

heatmap sns.heatmap(corr, annot=True, cmap='coolwarm',

fmt=".1f")

plt.title('FeatureCorrelation')#

Price vs. median income

plt.subplot(2, 2, 3)

sns.scatterplot(x='median_income',y='median_house_value',data=data,alpha=0.3)

plt.title('Price vs. Income') # Price by ocean proximity

plt.subplot(2,2,4)

sns.boxplot(x='ocean_proximity',y='median_house_value',data=data)

plt.xticks(rotation=45) plt.title('Price by Location') plt.tight_layout()

plt.show() print("\n=== Preprocessing Data ===")

#Handlemissingvaluesdata.fillna(data.select_dtypes(include='number').median(),

inplace=True)

Feature engineering data['rooms_per_household'] =

data['total_rooms']/data['households'] data['bedrooms_per_room'] =

data['total_bedrooms']/data['total_rooms'] # Convert categorical to

numericaldata=pd.get_dummies(data,columns=['ocean_proximity'])

#Selectfeaturesand target

X=data.drop('median_house_value',axis=1)y

=data['median_house_value']

Train-test split

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

Feature scaling scaler

= StandardScaler()

X_train_scaled=scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

print("\n===TrainingModels===")models=

{

"LinearRegression":LinearRegression(),

"Ridge Regression": Ridge(alpha=1.0),

"Lasso Regression": Lasso(alpha=0.1),

"DecisionTree":DecisionTreeRegressor(max_depth=5),

"Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),

"GradientBoosting":GradientBoostingRegressor(n_estimators=100,random_state=42),

"XGBoost": XGBRegressor(n_estimators=100, random_state=42),

"SVR":SVR(kernel='rbf')

}results={}forname,modelin

models.items():

print(f"Training {name}...")

model.fit(X_train_scaled[:1000], y_train[:1000])   y_pred

= model.predict(X_test_scaled)   results[name]={

"MAE": mean_absolute_error(y_test, y_pred),

"RMSE":np.sqrt(mean_squared_error(y_test,y_pred)),

```
        "R2":r2_score(y_test,y_pred)

    }

# Display results results_df =

pd.DataFrame(results).T print("\n===

Model Performance ===")

print(results_df.sort_values(by='RMSE'))

print("\n=== Optimizing Best Model

===")

# Let's optimize Random Forest as it typically performs well

fromsklearn.model_selectionimportRandomizedSearchCV#

Smaller parameter grid or use RandomizedSearchCV

param_dist = {

    'n_estimators':[50,100,200],

    'max_depth':[None,10,20],

    'min_samples_split':[2,5,10]

} rf =

RandomForestRegressor(random_state=42)

random_search=RandomizedSearchCV(rf,param_distributions=param_dist,n_iter=5, cv=2,
scoring='neg_mean_squared_error', n_jobs=-
1,verbose=1,random_state=42,error_score='raise')random_search.fit(X_train_scaled,y_train)
best_model = random_search.best_estimator_

# Evaluate optimized model y_pred =

best_model.predict(X_test_scaled)print("\nOptimized

Model Performance:") print(f"MAE:

{mean_absolute_error(y_test, y_pred):.2f}")

print(f"RMSE: {np.sqrt(mean_squared_error(y_test,

y_pred)):.2f}") print(f"R2 Score: {r2_score(y_test,
```

y_pred):.4f}")print("\n===GeneratingVisualizations

===")

#FeatureImportanceplt.figure(figsize=(10, 6))

importances = best_model.feature_importances_ features = X.columns

indices = np.argsort(importances)[-10:]# Top 10 features plt.title('Feature

Importances') plt.barh(range(len(indices)), importances[indices], color='b',

align='center') plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance') plt.show() # Actual vs Predicted

plt.figure(figsize=(10, 6)) plt.scatter(y_test, y_pred, alpha=0.3)

plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()],'k--',lw=2)

plt.xlabel('Actual Prices') plt.ylabel('Predicted Prices') plt.title('Actual vs

Predicted House Prices') plt.show() # Residual Plot residuals = y_test -

y_pred plt.figure(figsize=(10, 6)) plt.scatter(y_pred, residuals, alpha=0.3)

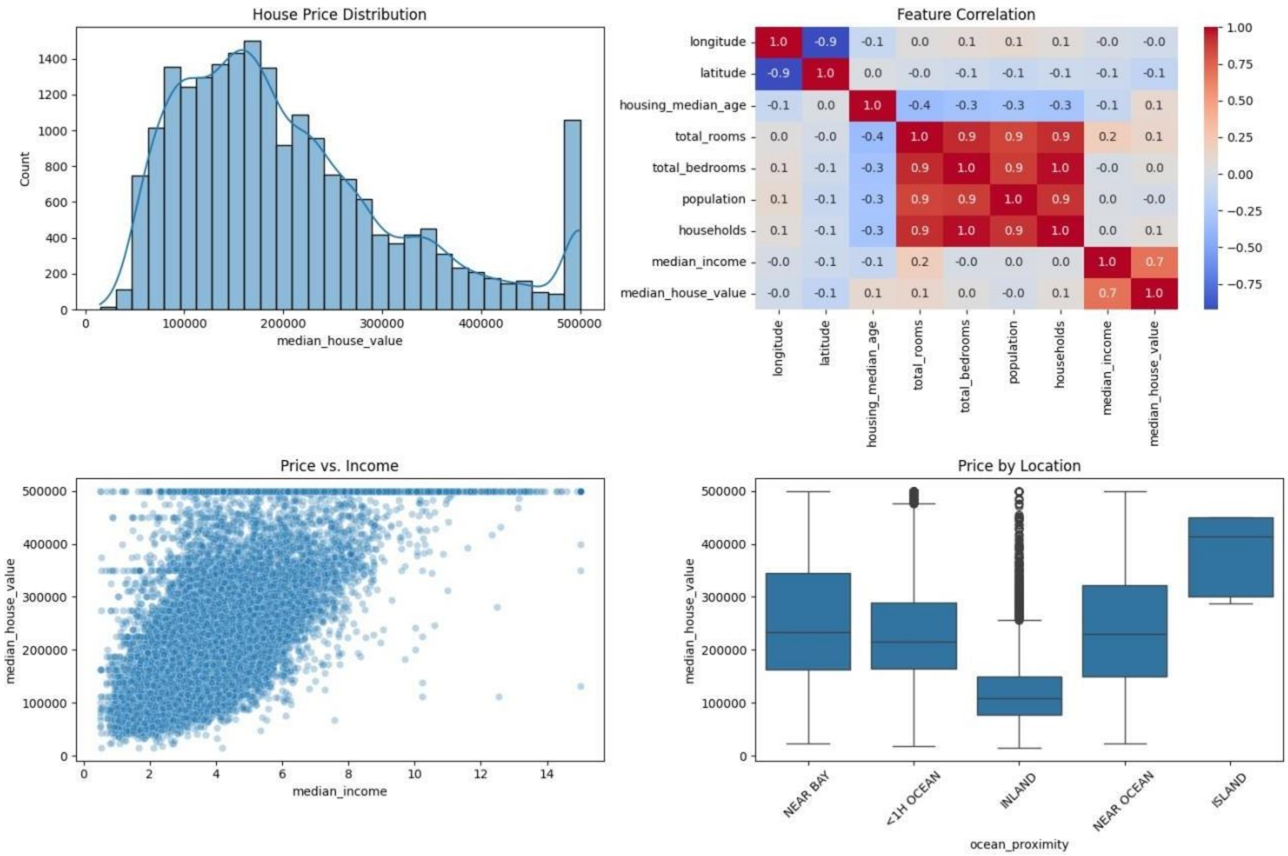
plt.axhline(y=0, color='r', linestyle='--') plt.xlabel('Predicted Prices')

plt.ylabel('Residuals') plt.title('Residual Plot')

plt.show()print("\n===ProgramExecution Complete

===")

OUTPUT:



14.Future scope

- **Geospatial Integration**Uselocationcoordinateswithgeospatial analytics to better capture regional price differences.

- **Time-Series Forecasting**
 Addhistorical housingdatatoforecast futurepricesbasedonmarkettrends.

- **Automated Feature Selection**Implementadvancedtechniqueslike Recursive Feature Elimination or SHAP for smarter feature optimization.

- **Real-TimePredictionAPI**DeployasaRESTAPIconnectedtolivereal estate data sources for realtime usage.

- **User-FriendlyWebInterface**EnhancethemodelwithaninteractiveUI using Streamlit or Gradio for public use.

15. Team Members and Roles

1.R.AFRIN– Data Collection and Integration: Responsible for sourcing datasets, connecting APIs, and preparing the initial dataset for analysis.

2. **B.RAMYA** – Data Cleaning and EDA: Cleans and preprocesses data, performs exploratory analysis, and generates initial insights.
3. **T.VAISHNAVI** – Feature Engineering and Modeling: Works on feature extraction and selection; develops and trains machine learning models.
4. **S.LEELAVATHI**– Evaluation and Optimization: Tunes hyperparameters, validates models, and documents performance metrics.
5. **B.NARMATHA**– Documentation and Presentation: Compiles reports, prepares visualizations, and handles presentation and optional deployment.