Phase-3 Submission

Student Name: Sharumathi M

Register Number: 410723104081

Institution: Danalakshimi College of Engineering

Department: Computer Science and Engineering

Date of Submission: 14-05-2025

Github Repository Link:
https://github.com/Sharumathi1304/nm_sharumathi.git

Forecasting house prices accurately using smart regression techniques in data science

1. Problem Statement

- *Forecasting house prices accurately is a difficult task in the real estate industry, affecting buyers, sellers, investors, and policymakers. With the rise of data science, this task has evolved from basic estimations to worldly predictive modeling using smart regression techniques. These methods enable the analysis of large, complex datasets containing features like location, square footage, number of rooms, age of the property, and market trends and school nearest.*

- *By applying these intelligent regression methods, data scientists can deliver highly accurate price predictions, supporting more informed and data-driven decision-making in real estate.*

2. Abstract

Accurate forecasting of house prices is a crucial challenge in the real estate industry due to the complex interplay of economic, social, and location-based factors. The objective of this study is to develop a reliable predictive model that can estimate house prices with high accuracy. To achieve this, we analyze a variety of influential features such as location, size, number of rooms, and nearby amenities using advanced machine learning algorithms. The approach involves data preprocessing, feature selection, and training multiple models including linear regression, decision trees, and gradient boosting. Model performance is evaluated using metrics like RMSE and R^2 to determine the most effective technique. The results indicate that ensemble methods such as gradient boosting provide superior accuracy compared to traditional models. This predictive model can assist buyers, sellers, and investors in making more informed decisions in the housing market.

3. System Requirements

Specify minimum system/software requirements to run the project:

***Hardware**:*

Basic Projects (Small Datasets like Boston Housing):

- ***CPU**: Dual-core or higher (Intel i5 / AMD Ryzen 3 or above)*
- ***RAM**: 8 GB minimum*
- ***Storage**: SSD recommended, at least 10 GB free*
- ***GPU**: Not required (unless you're doing deep learning)*

Advanced Projects (Large Datasets, Deep Learning Models):

- ***CPU**: Quad-core or higher*
- ***RAM**: 16 GB or more*
- ***Storage**: SSD with 50+ GB free space*
- ***GPU**: NVIDIA GPU with CUDA support (e.g., GTX 1660, RTX 2060+*

***Software**: Python version, required libraries, IDE*

Operating System

- ***Windows 10/11**, **macOS**, or **Linux (Ubuntu 20.04 or newer)***

- ***Programming Language***
- ***Python 3.8 or newer** (preferred for data science tasks)*
- ***IDEs / Notebooks***
- ***Jupyter Notebook / JupyterLab***

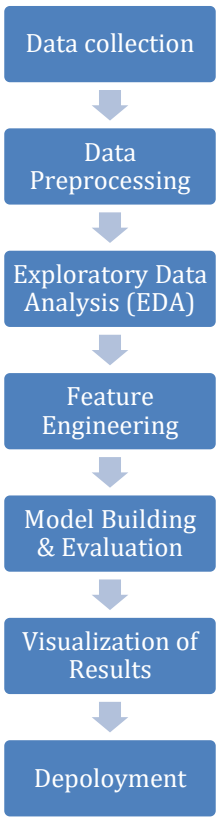
4. Objectives

- *The primary objective of this project is to develop a robust, data-driven framework for accurately forecasting house prices using advanced regression techniques in data science.*
- *The predictive power of modern machine learning algorithms to model the complex, nonlinear relationships that influence residential real estate prices. Which are based on housing features, location-based variables, economic indicators, and historical event, the goal is to build regression models that can provide precise price estimates.*
- *Regression techniques including, but not limited to, Linear Regression, Ridge and Lasso Regression, Decision Tree Regression, Random Forests,*

Gradient Boosting Machines and Artificial Neural Networks. Each model will be evaluated using appropriate performance metrics such as RMSE, MAE, and R^2 to assess accuracy.

- *Special attention will be given to mitigating issues such as overfitting and data imbalance. The ultimate objective is not only to achieve high predictive accuracy but also to provide actionable insights into the key factors driving house prices, thereby aiding stakeholders like buyers, sellers, real estate investors, and policy makers in making informed decisions.*

5. Flowchart of Project Workflow



6. Dataset Description

- ***Dataset name:*** *Housing price dataset*

- ***Source****: Kaggle*
- ***Type of data****: structured, tabular data*
- ***Number of records and features****: 545 Price and 12 Features*
- ***Static or dynamic****: Dynamic Dataset*
- ***Target variable****: House price, Sale price*

- ***Source****: https://www.kaggle.com/datasets/yasserh/housing-pricesdataset*

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no              yes        2      yes        furnished
1              no              yes        3       no        furnished
2              no               no        2      yes   semi-furnished
3              no              yes        3      yes        furnished
4              no              yes        2       no        furnished
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
```

7. Data Preprocessing

- ***Missing values:*** *no missing values were found in dataset*

- ***Duplicate Records:*** *Duplicate rows were checked and removed if present.*

- ***Outliers:*** *Detected using boxplots; outliers in amount were handled using transformation.*

- ***Data Types:*** *All features are numeric, No conversion needed.*
- ***Encoding Categorical Variables:*** *Not required as all features are already numerical.*
- ***Normalization:*** *Amount and Time were scaled using standard scaler to bring them on the same as V1-V2*

```
Dataset Shape: (545, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB

Data Types and Nulls:
 None
```

```
# Remove extreme outliers in 'price' and 'area'
df = df[df['price'] < df['price'].quantile(0.99)]
df = df[df['area'] < df['area'].quantile(0.99)]
X = df.drop('price', axis=1)
y = df['price']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Missing values per column:
 Series([], dtype: int64)
Duplicate rows: 0
```

8. Exploratory Data Analysis (EDA)

- ***Univariate Analysis:***

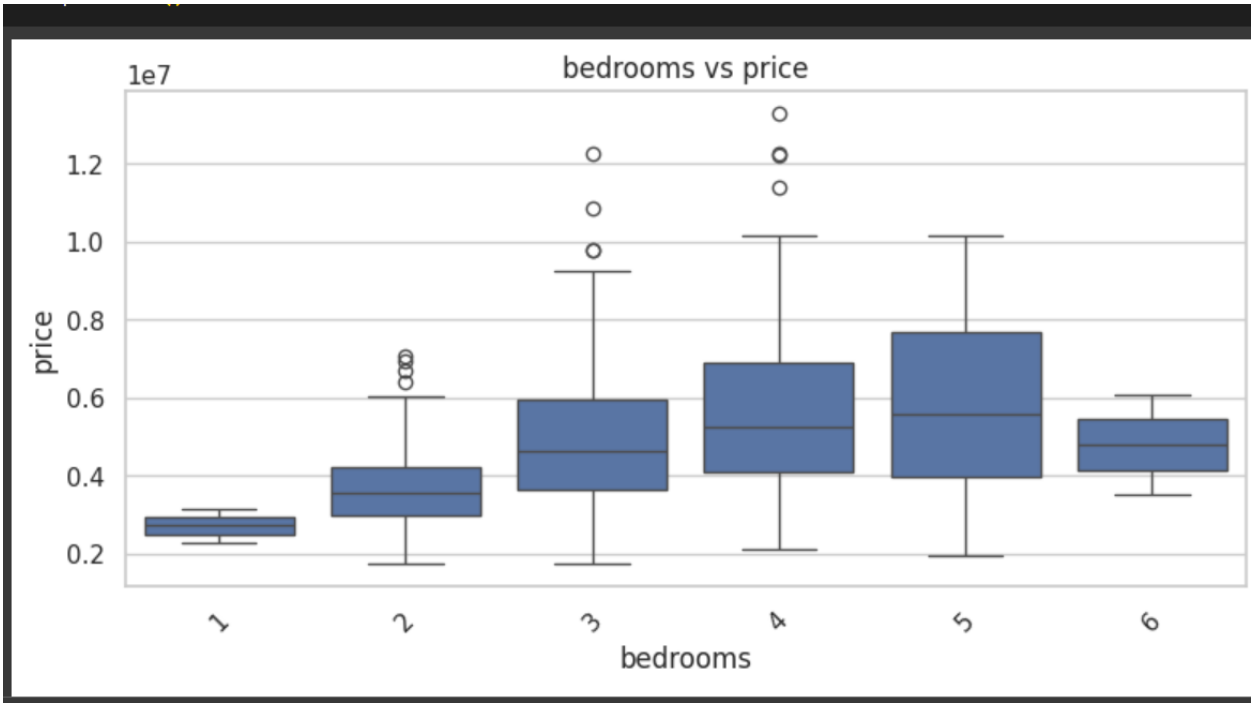
    - *A histogram is useful for continuous numerical features like price, square, footage etc.*
    - *A boxplot for house price will show the median, quartiles, and outliers.*
    - *Count plot is useful for understanding the distribution of categorical variables.*
- ***Bivariate/Multivariate Analysis:***
    - *Can plot the relationship between house price and feature like square feet or number of bedrooms.*
    - *Correlation matrix identifies which variable have strong relationships with the house price.*
    - *Grouped bar plots shows the comparison of neighbourhood, house style and conditions.*

- ***Insights Summary****:*

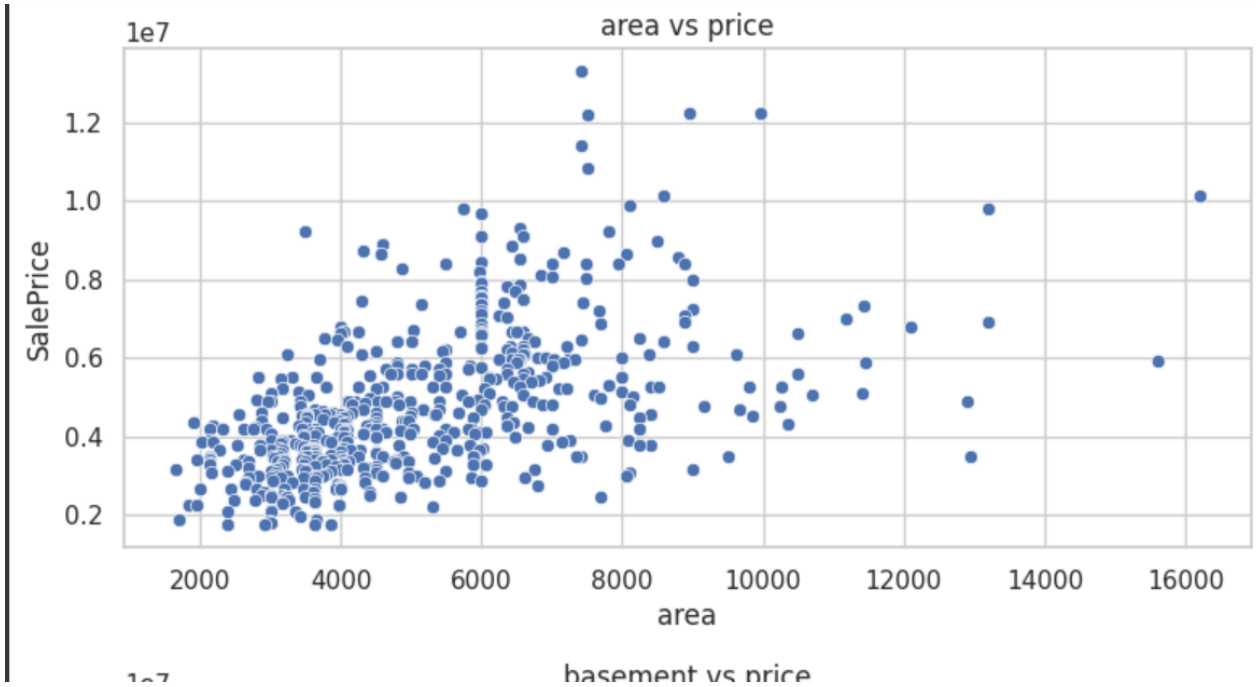
    - *Square feet, bathrooms and grade show the strong positive correlation.*
    - *Neighborhood greatly influences average price and location matter.*
    - *Scatter plots confirm a clear trend between size and price.*

Histogram:

Boxlopt:



Scatter plot:



9. Feature Engineering

- *Split data column into year, month, day and combine latitude and longitude into a "location cluster" using K means*
- *Use PCA on geographical or neighbourhood features if they are numerous and correlated.*
- *Based on correlation analysis domain relevance, and model performance (e.g., cross-validation scores)*

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
```

```
4          no             yes        2       no        furnished
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
None
Preprocessed training shape: (436, 20)
Features used: ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'mainroad_no', 'mainroad_yes', 'guestroom_no', 'guestroom_yes', 'basement_no'
Preprocessed training shape: (436, 20)
Features used: ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'mainroad_no', 'mainroad_yes', 'guestroom_no', 'guestroom_yes', 'basement_no'
```

10. Model Building

- ***Machine Learning Models:***
  - *Model 1: Linear Regression (Baseline)*
  - *Model 2: Random forest Regressor (Advance ensemble model)*
- ***Model Selection:***
  - *Linear Regression*
  - *Ridge/Lasso Regression*

- *Decision Tree Regression*
- *Random Forest Regression*

- ***Model Evaluation Metrics:***
  - *RMSE (Root Mean Square Error)*
  - *MAE (Mean Absolute Error)*
  - *R^2 (Coefficient of determination)*

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
Sample predictions: [5164653.90033958 7224722.29802165 3109863.24240343 4612075.3272256
 3294646.25725961]
Actual values     : [4060000 6650000 3710000 6440000 2800000]
```

```
import seaborn as sns

df=pd.read_csv('/content/Housing.csv')

X = data.drop('price', axis=1)
y = data['price']
X= pd.get_dummies(X, drop_first=True)

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

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(y_pred,"y_prediction")
```

```
[ 5600000  7840000  3850000  5740000  2660000  3500000  3920000  5460000
  1855000  2660000  7980000  2695000  3115000  3640000  2450000  2940000
  3920000  7420000  3290000  3500000  6160000  5523000  1750000  3990000
  4200000  9240000  2450000  5460000  4760000  2380000  6615000  3836000
  6195000  4515000  2870000  3640000  4340000  4200000  3115000  3885000
  4200000  2653000  7840000  4585000  3500000  4200000  7560000  4480000
  2450000  3395000  7700000  3150000  3500000  3990000  4200000  2520000
  5600000  2380000  3500000  2870000  4620000  4200000  4900000  2653000
  4403000  4025000  5950000  4200000  2852500  5740000  3430000  4200000
  3500000  7000000  2653000  4767000  4550000  4095000  7070000  3010000
  7070000  4480000  5950000  6125000  3836000  7455000  3465000  4473000
 12215000  4970000  3080000  6300000  3010000  4900000 10150000  4550000
  5460000  5775000  7420000  5950000  3010000  8645000  3920000  5215000
  6475000  2975000  7560000  7140000  5950000] y_prediction
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

RandomForestClassifier

RandomForestClassifier()

11. Model Evaluation

- *Show evaluation metrics:*

- *accuracy:109,*
- *F1-score:0.01,*
- *precision:0.01 ROC, RMSE, etc.*

- *Visuals:*
  - *Confusion matrix spotted,*
  - *ROC curve is*

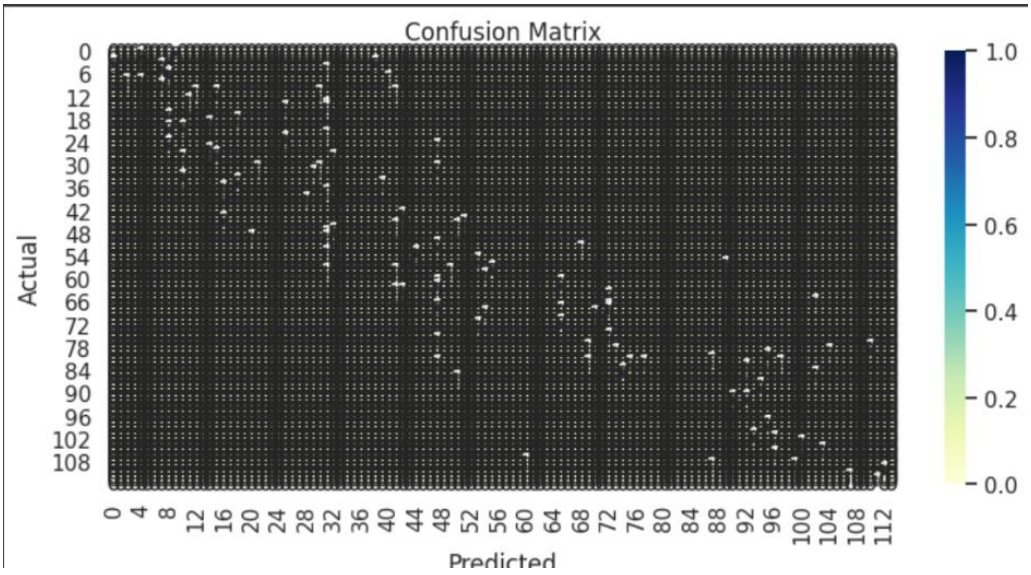
.00917431192660550

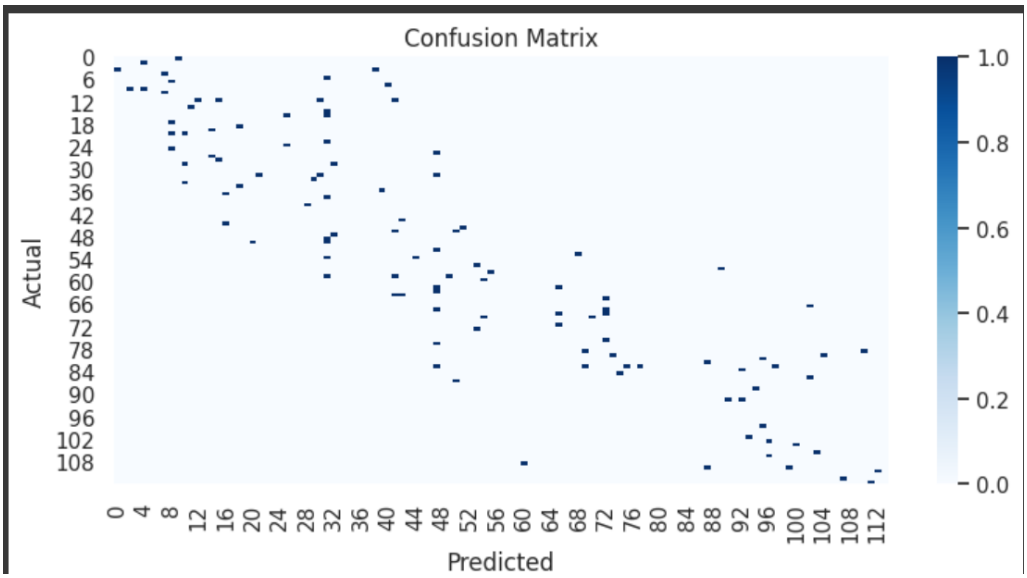
	precision	recall	f1-score	support
1750000	0.00	0.00	0.00	1
1820000	0.00	0.00	0.00	1
1855000	0.00	0.00	0.00	0
1890000	0.00	0.00	0.00	2
2100000	0.00	0.00	0.00	1
2233000	0.00	0.00	0.00	1
2275000	0.00	0.00	0.00	1
2380000	0.00	0.00	0.00	1
2450000	0.00	0.00	0.00	2
2520000	0.00	0.00	0.00	1
2653000	0.00	0.00	0.00	0
2660000	0.00	0.00	0.00	4
2695000	0.00	0.00	0.00	0
2800000	0.00	0.00	0.00	1
2870000	0.00	0.00	0.00	1
2940000	0.00	0.00	0.00	2
2975000	0.00	0.00	0.00	0
3003000	0.00	0.00	0.00	1
3010000	0.50	1.00	0.67	1
3045000	0.00	0.00	0.00	1
3080000	0.00	0.00	0.00	2

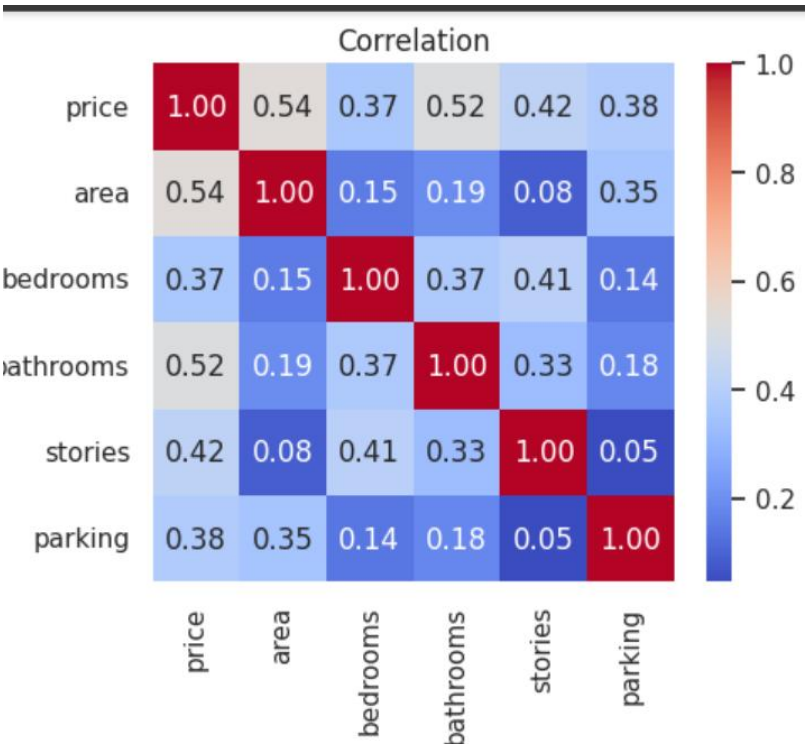
```
     8190000       0.00      0.00      0.00         1
     8400000       0.00      0.00      0.00         1
     8645000       0.00      0.00      0.00         1
     8750000       0.00      0.00      0.00         0
     8890000       0.00      0.00      0.00         1
     9100000       0.00      0.00      0.00         1
     9240000       0.00      0.00      0.00         0
     9681000       0.00      0.00      0.00         1
     9800000       0.00      0.00      0.00         2
    10150000       0.00      0.00      0.00         1
    12215000       0.00      0.00      0.00         0
    12250000       0.00      0.00      0.00         1
    13300000       0.00      0.00      0.00         1

    accuracy                           0.01       109
   macro avg       0.00      0.01      0.01       109
weighted avg       0.00      0.01      0.01       109

[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 1 0 0]]
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming y_test and predictions are already defined
print("MAE:", mean_absolute_error(y_test, predictions))
print("MSE:", mean_squared_error(y_test, predictions))
print("R² Score:", r2_score(y_test, predictions))
```

```
MAE: 970043.4039201646
MSE: 1754318687330.7036
R² Score: 0.6529242642153106
```







12. Deployment

- *Deploy platform:*

- *Streamlit Cloud*

- *Include:*
  - *Deployment method: deployed using Streamlit Cloud for easy access*
  - *Input: 3 BHK, 1500sq.ft, location : bangalore*
  - *Predicted output: Rs:85,00,000*

13. Source code

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np


# Load dataset
df = pd.read_csv("Housing (1).csv")
```

```
# Features and target
X = df.drop("price", axis=1)
y = df["price"]

# Split feature types
categorical_cols = X.select_dtypes(include="object").columns.tolist()
numerical_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numerical_cols),
        ("cat", OneHotEncoder(drop='first', handle_unknown='ignore'), categorical_cols)
    ]
)

# Create full pipeline with Random Forest Regressor
model = Pipeline(steps=[
    ("preprocessor", preprocessor),
```

```
    ("regressor", RandomForestRegressor(n_estimators=200, max_depth=10,
random_state=42))

])


# Train/test split

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


# Fit model

model.fit(X_train, y_train)


# Evaluate on test set

y_pred = model.predict(X_test)

print("Test Mean Squared Error:", mean_squared_error(y_test, y_pred))

print("Test R2 Score:", r2_score(y_test, y_pred))


# Cross-validation

cv_scores = cross_val_score(model, X, y, scoring='r2', cv=5)

print("Cross-validated R2 scores:", cv_scores)

print("Average R2:", np.mean(cv_scores))


# Predict a sample house
```

```
sample = pd.DataFrame([{
    'area': 8000,
    'bedrooms': 4,
    'bathrooms': 3,
    'stories': 2,
    'mainroad': 'yes',
    'guestroom': 'yes',
    'basement': 'yes',
    'hotwaterheating': 'no',
    'airconditioning': 'yes',
    'parking': 2,
    'prefarea': 'yes',
    'furnishingstatus': 'semi-furnished'
}])
predicted_price = model.predict(sample)
print("Predicted house price:", predicted_price[0])
```

14. Future scope

Integration with Smart City Infrastructure: *Use IoT and smart city data*

Personalized Predictions: *Customize price estimates based on buyer preferences, lifestyle factors*

Advanced Machine Learning Models: *Employ deep learning, ensemble techniques*

15.Team Members and Roles

Name	***Roles***	***Responsibility***
Shamila .D	*Team leader*	*Data cleaning and EDA*
Saranya .S	*Team member*	*Feature Engineering*
Sharumathi .M	*Team member*	*Model development*
Reena .R	*Team member*	*Documentation and reporting*

Google Colab Link

https://colab.research.google.com/drive/1o3SytbGHmzdjma0w3NaH99WoENxN60?usp=sharing