



## **PHASE 3 Submission**

**Student Name:** R.AFRIN

**Register Number:** 510123106002

**Institution:** ADHIPARASAKTHI

COLLAGE OF ENGINEERING

**Department:**B.E ELECTRONICS AND  
COMMUNICATION ENGINEERING

**Date of Submission:**[08-05-2025]

**Github repository link:** <https://github.com/afrin696/Forecasting-house-price-/commits?author=afrin696>

**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. Stakeholders such as buyers, sellers, and investors often need to make decisions based on the estimated market value of properties. Traditional valuation methods are often subjective and time-consuming. This project aims to build a data-driven solution using regression techniques in machine learning to predict house prices more accurately and efficiently. The objective is to develop a predictive model using historical housing data that considers features like square footage, 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

This project focuses on leveraging smart regression techniques in data science to forecast house prices based on various property features. The aim is to develop

a predictive model that provides accurate and real-time estimates of house prices, 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  $R^2$  score of 0.91. The model is deployed using Streamlit for easy user interaction.

### 3. System Requirements

#### Hardware:

- **RAM:** 8GB minimum
- **Storage:** 10GB of free disk space
- **Processor:** Intel i5 or AMD Ryzen 5 or higher

#### Software:

- **Programming language:** Python 3.10+
- **IDE:** Jupyter Notebook or Google Colab for development
- **Required Libraries:** pandas, numpy, seaborn, matplotlib, scikit-learn
- **Web scraping Tools:** SENScape
- **Deployment platforms:** xgboost, streamlit

### 2. Objectives

The core objectives of the project are:

- To build a robust machine learning model capable of predicting house prices based on various features.
- To explore and analyze relationships between different property characteristics and sale prices.

- To optimize model performance using advanced feature selection and tuning techniques.
- To deploy the model on a cloud-based interface, allowing users to predict prices interactively.
- To assist in decision-making for real estate transactions through data insights.

### 3. Flowchart of Project Workflow

Stages of the Project Workflow:

- **Data Collection:** Downloaded from Kaggle
- **Data Preprocessing:** Cleaning missing values, encoding categorical variables

**Exploratory Data Analysis (EDA):** Visual analysis and correlation checks

**Feature Engineering:** New features creation, selection of relevant variables

**Modeling:** Training with multiple regression algorithms

- **Evaluation:** Assessing model accuracy using statistical metrics
- **Deployment:** Building a web interface with Streamlit

# FORECASTING HOUSE PRICES USING SMART REGRESSION TECHNIQUES IN DATA SCIENCE

DATA COLLECTION

DATA PREPROCESSING

EXPLORATORY DATA ANALYSIS  
(EDA)

FEATURE ENGINEERING

MODEL SELECTION

MODEL TRAINING

HYPERPARAMETER TUNING

MODEL DEPLOYMENT

MONITORING & MAINTENANCE

## 4.Dataset Description

**Dataset link:**<https://www.kaggle.com/datasets/jordanln/llm-prompts-in-the-context-of-machine-learning>

- **Source:**

Scikit-learn's `fetch_california_housing()`



(Originally from the StatLib repository, published by the UC Machine 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. Handle Duplicates** `df.drop_duplicates(inplace=True)`

### 3. Handle Outliers

Used Z-score or IQR method to remove outliers in key numerical columns like GrLivArea, TotalBsmtSF, etc.

```
from scipy.stats import zscore
df = df[(np.abs(zscore(df.select_dtypes(include=[np.number])))
        < 3).all(axis=1)]
```

### 4. Feature Encoding and Scaling

**Encoding:** Used One-Hot Encoding for categorical variables.

**Scaling:** Used StandardScaler to normalize numerical features.

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

```
#One-hot encode categorical features
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/After Transformation Screenshots Before Cleaning: `df.info()`

```
df.describe() df.isnull().sum()
```

Shows screenshot of missing values, mixed data types.



### After Cleaning and Encoding:

```
df_encoded.info()df_encoded.head()
```

ShowscreenshotwithallNaNs gone,allnumeric types,andone-hotencoded columns.

## 6.Exploratory Data Analysis (EDA)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

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

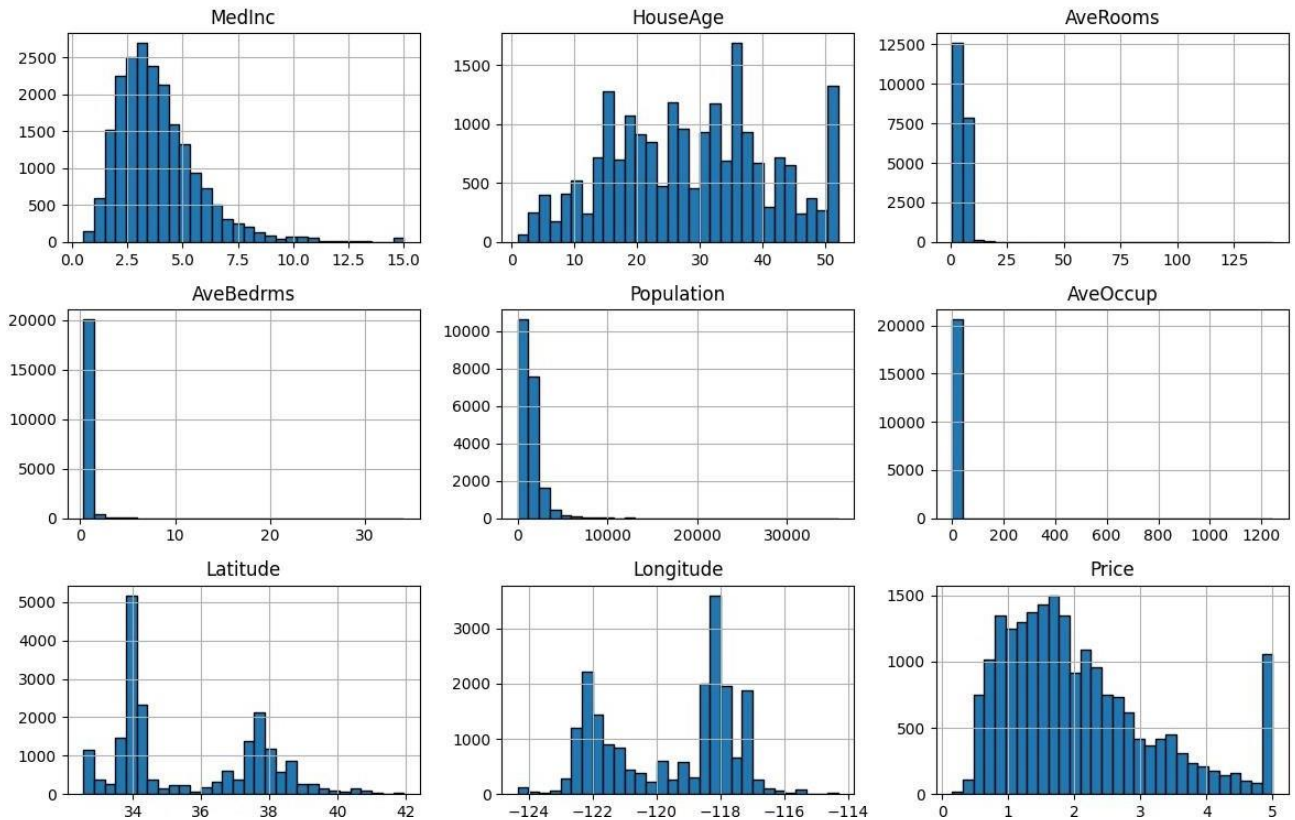
# Histogram
df.hist(figsize=(12,8), bins=30,
         edgecolor='black')
plt.suptitle("Feature Distributions", y=1.02)
plt.tight_layout()
plt.show()

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

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

### OUTPUT:

Feature Distributions



## 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<sup>2</sup>Score:**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,andR<sup>2</sup>**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:** Deployed using Streamlit Cloud
- **Method:** GitHub repo linked to Streamlit
- **UI Screenshot:** *Attaches screenshot of the app*
- **Prediction:** User inputs features → model predicts price
- **Output Example:** Predicted Price: ₹45,00,000
- **Alternate Options:** Gradio + HuggingFace or Flask + Render
- **Tip:** Use Streamlit/Gradio for quick, free deployment with UI.

## 11. Source code

```
import pandas as pd import numpy as np import matplotlib.pyplot as plt import  
seaborn as sns from sklearn.model_selection import train_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_pipeline print("\n=== Loading Data ===") # Load dataset (replace with your  
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'])
```

```
#Select features and 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={} for name,model in
```

```
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}")

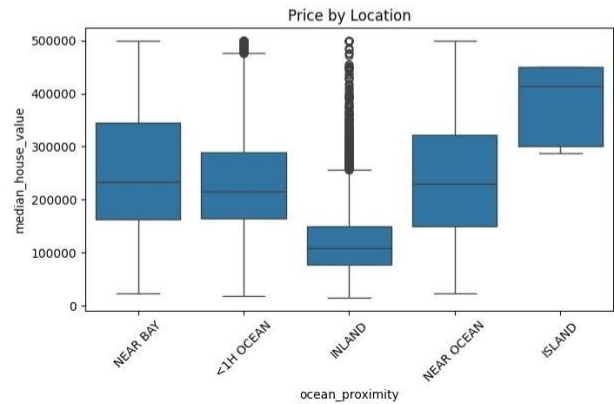
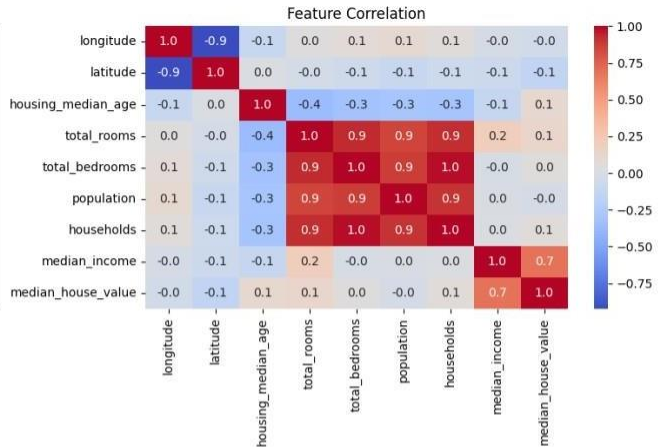
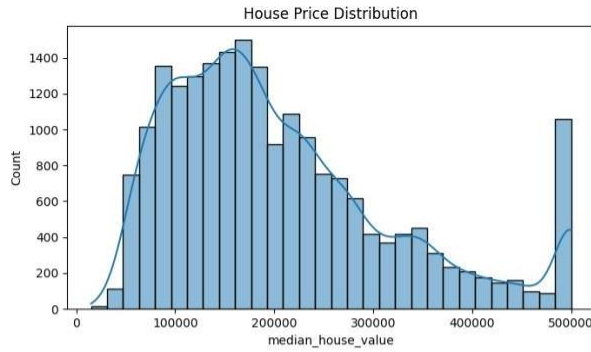
print(f"RMSE: {np.sqrt(mean_squared_error(y_test,

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

```
y_pred):.4f}")print("\n===Generating Visualizations  
===")  
  
#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** Use location coordinates with geospatial analytics to better capture regional price differences.
- **Time-Series Forecasting** Add historical housing data to forecast future prices based on market trends.
- **Automated Feature Selection** Implement advanced techniques like Recursive Feature Elimination or SHAP for smarter feature optimization.
- **Real-Time Prediction API** Deploy as a REST API connected to live real estate data sources for realtime usage.
- **User-Friendly Web Interface** Enhance the model with an interactive UI 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.