HOUSE PRICE PREDICTION, HYDERABAD

-Asha Jyothi

Abstract:

House price prediction is a critical aspect of real estate, providing valuable insights for buyers, sellers, and investors. In 2024, the methods used to forecast housing prices have evolved significantly, blending traditional techniques with advanced machine learning and artificial intelligence (AI).

Traditional approaches, such as Comparative Market Analysis (CMA) and the Hedonic Pricing Model, rely on historical data and property features to estimate value. However, modern techniques, including Automated Valuation Models (AVMs), machine learning algorithms, and time series forecasting, have increasingly become central to the process, offering more dynamic, accurate, and data-driven predictions. These advanced methods leverage large datasets, economic indicators, and even social media sentiment to enhance prediction accuracy.

Despite these advancements, challenges remain, such as data quality, market volatility, and the complexity of local market factors. Additionally, newer technologies, including AI and blockchain, hold the potential to further revolutionize house price predictions by enabling real-time adjustments and ensuring greater transparency in property transactions.

This paper discusses the methods, challenges, and future trends in house price prediction, emphasizing the importance of integrating both traditional and emerging techniques to provide more reliable forecasting in an ever-evolving housing market.

Introduction:

Hyderabad, a major urban hub in southern India, has emerged as a key center for real estate development, driven by rapid IT sector expansion, infrastructure growth, and a rising population. As the city continues to evolve, predicting house prices accurately has become increasingly important for homebuyers, investors, developers, and policymakers. House price prediction involves analyzing various factors such as location, amenities, property size, proximity to commercial zones, and historical pricing trends. With the advent of machine learning and data analytics, it is now possible to develop predictive models that offer more accurate and actionable insights into property pricing. This study focuses on predicting residential property prices in Hyderabad using statistical and machine learning techniques, aiming to enhance transparency and efficiency in the local real estate market.

Hyderabad has emerged as a prominent real estate hub in India, driven by strong economic growth, a thriving IT sector, and continuous infrastructure development. As the city expands, the housing market experiences varying price trends across different localities. Predicting house prices in such a dynamic environment is essential for buyers, sellers, real estate developers, and financial institutions.

This study aims to predict residential property prices in Hyderabad using historical data and machine learning techniques. By analyzing factors such as location, property size, number of rooms, and amenities, the model seeks to provide accurate and reliable price estimates. The goal is to support better decision-making in the real estate sector through data-driven insights.

Existing Methods:

Several techniques have been widely used to predict house prices, ranging from traditional statistical models to advanced machine learning algorithms. Each method has its own strengths and is suited to different types of data and prediction objectives:

- **Linear Regression:** Models a linear relationship between house price and features, making it a simple yet effective baseline model.
- **Multiple Linear Regression:** Extension of linear regression with multiple independent variables (e.g., square footage, number of bedrooms, location) to improve prediction accuracy.
- **Decision Trees:** Splits the dataset into branches based on feature values, allowing for intuitive, rule-based prediction. Useful for capturing non-linear relationships.
- **Random Forests:** An ensemble method that builds multiple decision trees on random subsets of the data and averages their results to improve accuracy and reduce overfitting.
- **K-Nearest Neighbors (KNN):** Predicts the price of a house based on the average prices of the 'k' most similar properties. Effective for capturing local trends in data.
- **Time Series Models (e.g., ARIMA):** Focuses on analyzing historical price trends and seasonal patterns over time, useful for forecasting price fluctuations in a given locality.

Limitations:

While traditional and machine learning models have been widely used for house price prediction, each comes with its own set of limitations, especially when applied to diverse and dynamic real estate markets like Hyderabad.

1. Linear Regression / Multiple Linear Regression

- Assumes a linear relationship between features and price, which is often unrealistic in complex markets.
- Sensitive to multicollinearity among independent variables.
- Struggles with non-linear and high-dimensional feature spaces.

2. Decision Trees

- Prone to overfitting, especially with noisy or small datasets.
- Can produce unstable results with slight changes in data.
- Lacks generalization, making it less reliable for unseen property types or localities.

3. Random Forests

- Although more robust than single decision trees, they can still be computationally expensive and less interpretable.
- May not perform well if the dataset lacks relevant or high-quality features.
- Requires careful tuning to avoid bias-variance tradeoffs.

4. K-Nearest Neighbors (KNN)

- Highly sensitive to feature scaling and irrelevant features.
- Poor performance on large datasets due to high computational cost.
- Locality-based approach may fail in heterogeneous markets where nearby properties have vastly different values.

5. Time Series Models (e.g., ARIMA)

- Assumes stationarity, which is rare in real estate data.
- Focuses only on temporal trends and ignores spatial and feature-based variations.
- Poor at handling external shocks, such as regulatory changes or economic crises.

Proposed Methods:

To build a robust and accurate house price prediction model for Hyderabad, the following multi-step methodology is proposed, combining data preprocessing, statistical modeling, machine learning algorithms, and model evaluation techniques:

- **Data Preprocessing & Feature Engineering**
  Objective: Improve data quality and enhance predictive power.
  Task:
    - Handle missing values through imputation.
    - Detect and remove outliers.
    - Normalize or scale numerical features.
    - Encode categorical variables.
    - Engineer new features (e.g., price per square foot, proximity to amenities, neighborhood price trends)

- **Exploratory Data Analysis (EDA)**
  Objective: Understand feature distributions and relationships.
  Techniques:
    - Visualizations (histograms, scatter plots, box plots).
    - Correlation heatmaps to detect multicollinearity.
    - Identify important predictors using summary statistics and domain knowledge.

- **Linear Regression**
  Objective: Find the best-fitting straight line (or hyperplane) that predicts the dependent variable from one or more independent variables, by minimizing the error between the predicted and actual values.

- **Random Forests Regression**
  Objective: A Random Forest is a machine learning algorithm that uses an ensemble of decision trees to make predictions. It works by training multiple decision trees on different subsets of the data and then combining their predictions to make a final prediction.

  Benefits:

  - Its accuracy and ability to handle both **classification** and **regression** problems.
  - Random Forests reduce overfitting by averaging multiple trees and are effective on noisy data.

- **Extreme Gradient Boosting (XGBoost/LightGBM)**
  Objective **XGBoost** is an advanced implementation of the **gradient boosting algorithm**, designed for speed and high performance, having its high accuracy, efficiency, and flexibility.
  Features:
  - Handles missing values and categorical variables effectively.
  - High performance on structured/tabular data.
  - Suitable for imbalanced datasets and large-scale modeling.

- **CatBoost Regression**
  Objective: **CatBoost** is a high-performance gradient boosting library developed by **Yandex**. It is designed to handle **categorical features automatically** and is highly efficient for **both classification and regression** tasks.

- **Stacking Regression**
  Objective: Stacking regression is an ensemble learning technique where multiple regression models are combined to improve prediction accuracy. It involves training individual regression models on the training data and then using their predictions as input to a meta-regressor, which learns how to combine the individual predictions into a final prediction. Reduces **bias and variance** by blending models.

- **Cross-Validation and Model Evaluation**
  Objective: Ensure generalization and prevent overfitting.
  Methods:
  - Use **k-fold cross-validation** to evaluate performance across multiple data splits.
  - Metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score.

Advantages:

The following advantages arise from the multi-pronged methodology outlined for Hyderabad house-price prediction:

1. **Data Preprocessing & Feature Engineering**
    - Improves model performance by making data more meaningful.
    - Handles missing values, outliers, and scaling, improving model robustness.
    - Reduces **noise** and **dimensionality**, boosting accuracy and training speed.
2. **Exploratory Data Analysis (EDA)**
    - Helps uncover **patterns**, **outliers**, and **relationships** in data.
    - Guides **feature selection** and **model choice** based on data insights.
    - Identifies **data quality issues** early (e.g., duplicates, missing values).
    - Supports **hypothesis generation** and **business understanding.**
3. **Linear Regression**
    - **Simple and fast** to implement and train.
    - **Interpretable** — coefficients show feature impact.
    - Works well for **linearly correlated** data.
    - Useful as a **baseline model** for regression tasks.

4. Random Forests

- Reduces **overfitting** by averaging many decision trees.
- Handles both **classification and regression** well.
- Robust to **missing values** and **outliers**.
- Handles **non-linear relationships** and **high-dimensional data**.

5. Gradient Boosting (XGBoost/LightGBM)

- Extremely **high accuracy** and **performance**, often outperforming other models.
- Works well on **structured/tabular data**.
- Offers **fine control** over regularization and training (e.g., early stopping, learning rate).
- LightGBM is optimized for **speed and scalability** (supports GPU).
- XGBoost includes **regularization** to prevent overfitting.

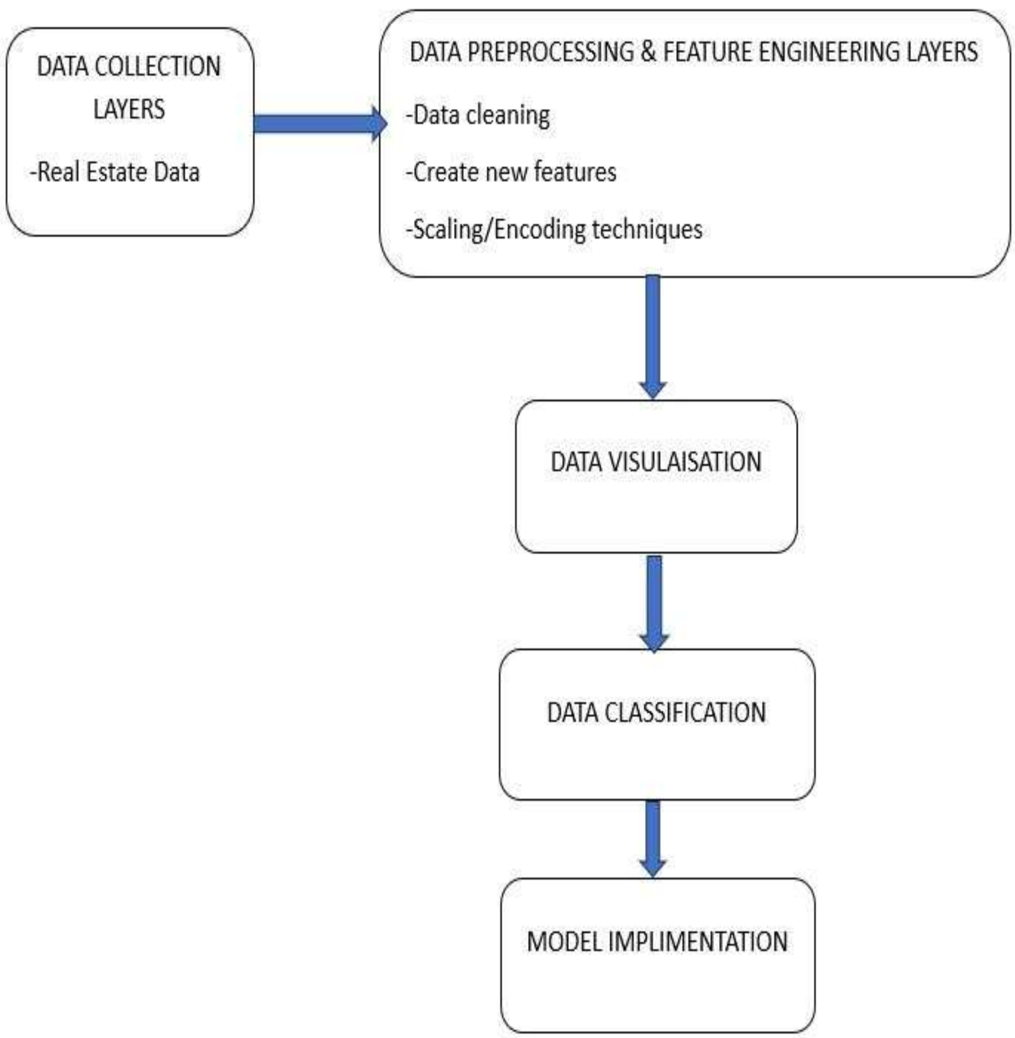
6. CatBoost Regression

- Handles categorical features automatically — no need for manual encoding.
- Reduces overfitting with Ordered Boosting.

7. Stacking Regression

- Leverages multiple model strengths to improve predictive performance.
- Reduces bias and variance by blending models.
- Flexible — can use diverse base models and a custom meta-model.
- Often performs better than any individual model in the ensemble.

System Architecture:



System Requirements:

Software Requirements:

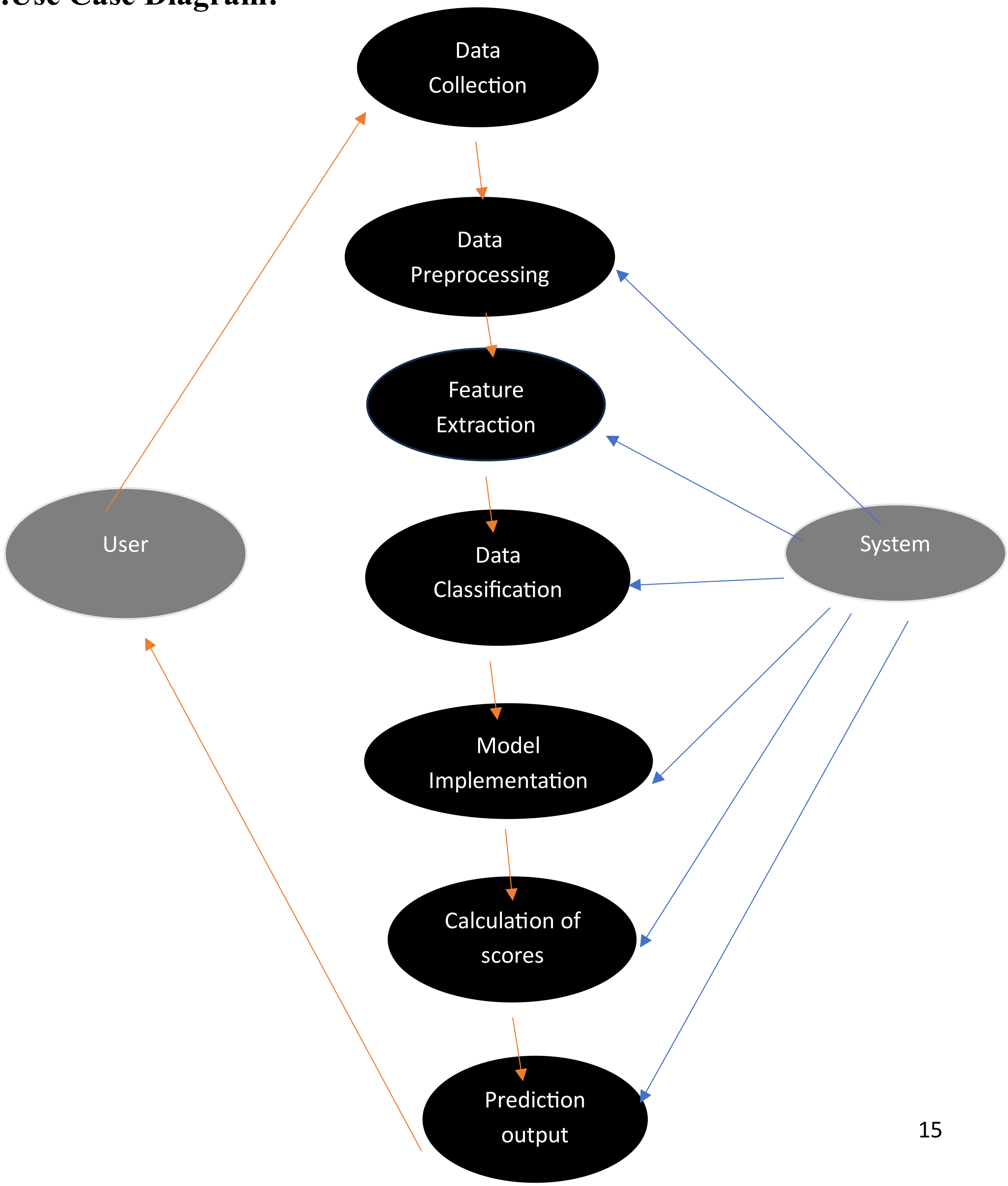
Component	Requirement
Operating System	Windows, Linux (Ubuntu or CentOS), or macOS
Programming Languages	Python
Libraries & Frameworks	Scikit-learn, TensorFlow, PyTorch, XGBoost, LightGBM
Database	SQL (MySQL, PostgreSQL), NoSQL (MongoDB)
Web Framework	Flask / Django
Version Control	Git
Data Processing Tools	Pandas, NumPy, Matplotlib, Seaborn
Cloud Services	AWS, Google Cloud, Microsoft Azure
Containerization	Docker

Hardware Requirements:

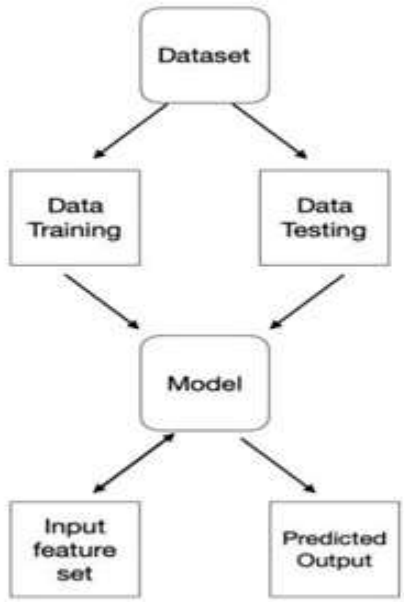
Component	Requirement
Server/Cloud Infrastructure	Cloud or on-premise server for data processing and hosting
CPU	Multi-core processor (e.g., Intel i7 or better)
RAM	Minimum 16 GB RAM
GPU (optional)	NVIDIA RTX 3000 series or better
Storage	SSD storage (at least 100 GB)
Network	Reliable high-speed internet connection

UML Diagrams:

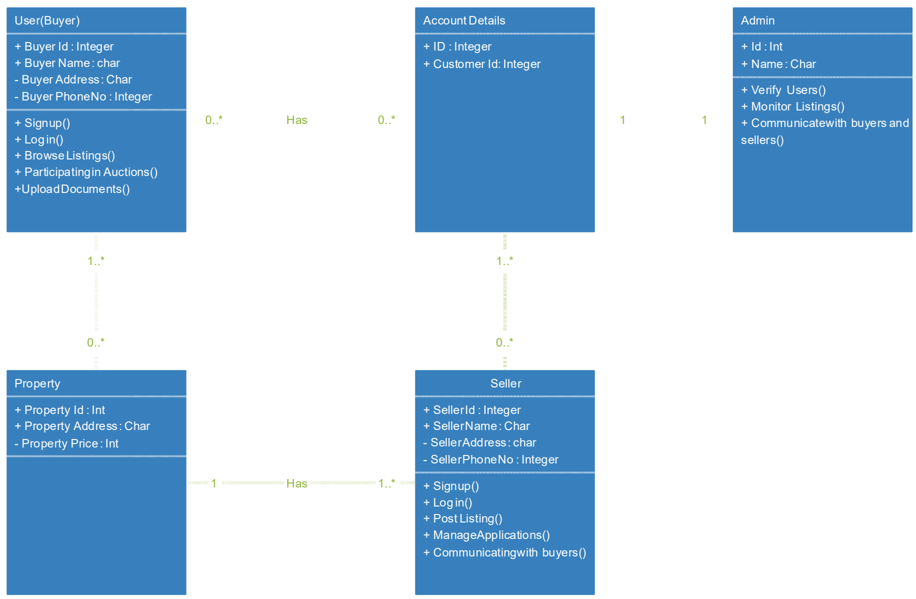
1.Use Case Diagram:



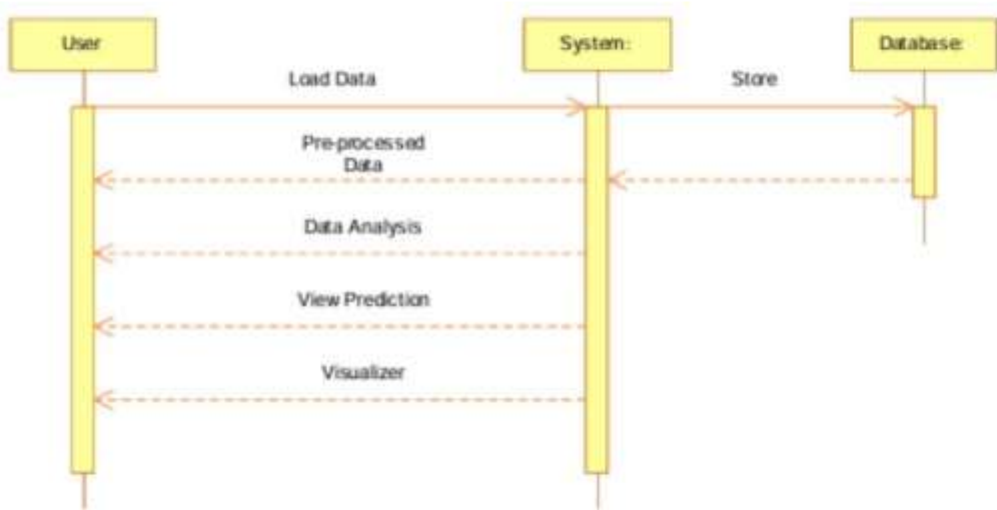
2.Data Flow Diagram:



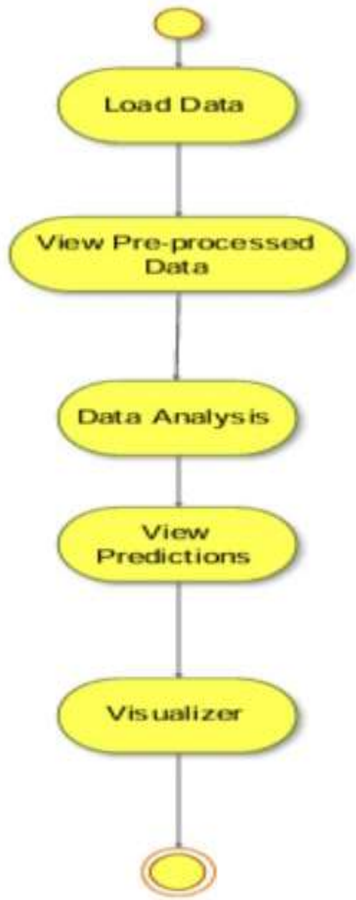
3.Class Diagram:



4. Sequence Diagram:



5.Activity Diagram:



Technology/Domain Introduction:

In this study, I employed various machine learning algorithms to predict house prices. The algorithms used in this analysis include: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor and XGBoost Regressor. To assess the performance of these models, I utilized the sci-kit learn Python library. Then evaluated the models using several performances metrics, which includes R-square, Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE). These metrics provide valuable insights into the accuracy and effectiveness of the models in predicting house price.

A. Linear Regression:

LR is a supervised ML technique used for regression tasks, where it operates under the assumption of a linear connection between an input variable (x) and a solitary output variable (y). By incorporating multiple independent features from our dataset. Multiple Linear Regression (MLR) enabling us to estimate the correlation between two or more independent variables and a dependent variable, considering the potential dependence of prices on these diverse features.

B. Random Forest Regressor:

RF is like a team of models working together to make more accurate predictions. Instead of relying on just one model, it combines multiple models to create a stronger and more reliable model. Here is how it works: Each model in the random forest is like a decision tree, where it makes decisions based on different factors. However, what makes random forest unique is that each tree uses a different subset of features from the dataset. This helps to create a diverse set of decision trees that are not strongly correlated with each other.

By combining the predictions of these individual decision trees, the random forest algorithm produces a final predicted result. This ensemble of models reduces the chances of overfitting or relying too much on a single model’s bias.

C. Stacking Regressor:

Stack of estimators with a final regressor. Stacked generalization consists in stacking the output of individual estimator and use a regressor to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

D. Cat Boost Regressor:

It's a powerful tool for various machine learning tasks, including classification, regression, and ranking, and offers advantages in model performance and ease of use, particularly when dealing with datasets containing significant amounts of categorical data.

E. XGB Regressor:

Extreme Gradient Boosting Regressor is an ML that creates a powerful predictive model by combining many weak models together. It works by repeatedly improving the weak models' performance based on their errors, allowing them to learn from each other and make better predictions collectively.

Programming Language Introduction:

For this project, the primary programming language used is Python. Python is widely regarded as the leading language for **data science**, **machine learning**, and **House Price Prediction, Hyderbad** applications due to its simplicity, extensive library support, and strong community. It offers an ideal balance between rapid development, powerful data handling capabilities, and support for deploying production-grade systems.

The advantages of using Python in this project include:

- **Rich Libraries for Machine Learning and Deep Learning:**

  Libraries such as **Scikit-learn**, **Linear Regression, CatBoost, Stacking Regression, Random Forest** and **XGBoost** provide powerful tools for building predictive models efficiently.

- **Robust Data Handling and Preprocessing:**
  Tools like **Pandas** & **NumPy** allow for efficient manipulation, analysis, and streaming of large volumes of financial time-series data.

- **Real-Time Data Collection and Streaming:**
  Python supports WebSocket communication and integration with streaming platforms (like **Kaggle**, **Github**, **Figshare**, etc), enabling real-time data ingestion and processing.

- **Model Deployment and API Development:**
  Frameworks such as **FLASK** and **Django** allow the easy deployment of machine learning models as real-time APIs, ensuring low-latency predictions.

- **Visualization Tools:**
  Libraries like **Matplotlib, Seaborn** & **Plotly** enable the creation of interactive real-time dashboards for monitoring stock trends and model predictions.

- **Flexibility and Integration:**
  Python can seamlessly integrate with other services like databases (**MySQL, MongoDB**), cloud platforms (**AWS, GCP, Azure**), and messaging systems, making it ideal for building end-to-end solutions.

Coding Part

Import Packages:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Collecting Dataset and use head():

```
df=pd.read_csv('hyderabad_first_type_500.csv')
df.head()
```

	Price	CarpetArea	Bedrooms	Builder	PaidUser	Locality	Property Type	UserType	Prime Location	Agent	Link
0	82.2 Lac	NaN	2	NaN	Y	Manikonda, Outer Ring Road	Apartment	Builder	Y	Pavan Kumar	https://www.magicbricks.com/propertyDetails/2-...
1	1.28 Cr	NaN	3	Silversand Infratech Pvt. Ltd	Y	Hitech City	Apartment	Builder	Y	Shegur	https://www.magicbricks.com/propertyDetails/3-...
2	46.2 Lac	N	3	NaN	Y	Aminpur	Apartment	Builder	Y	Kumaran A	https://www.magicbricks.com/propertyDetails/3-...
3	90.7 Lac	N	2	BEL	Y	Moti Nagar	Apartment	Builder	Y	Brigade	https://www.magicbricks.com/propertyDetails/2-...
4	1.49 Cr	N	3	Sravanthi Constructions Pvt. Ltd.	Y	Ameerpet, NH 9	Apartment	Builder	Y	Sravanthi Constructions	https://www.magicbricks.com/propertyDetails/3-...

Using info():

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100 entries, 0 to 2099
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Price           2100 non-null   object
 1   CarpetArea      1700 non-null   object
 2   Bedrooms        2100 non-null   int64 
 3   Builder         1400 non-null   object
 4   PaidUser        2100 non-null   object
 5   Locality        2100 non-null   object
 6   Property Type   2100 non-null   object
 7   UserType        2100 non-null   object
 8   Prime Location  2100 non-null   object
 9   Agent           2100 non-null   object
 10  Link            2100 non-null   object
 11  Prop ID         2100 non-null   object
 12  Other           2100 non-null   object
dtypes: int64(1), object(12)
memory usage: 213.4+ KB
```

Using describe():

```
df.describe()
```

	Bedrooms
count	2100.000000
mean	2.571429
std	0.494990
min	2.000000
25%	2.000000
50%	3.000000
75%	3.000000
max	3.000000

Using isnull().sum():

```
[ ]  df.isnull().sum()
```

```
Price              0
CarpetArea       400
Bedrooms           0
Builder          700
PaidUser           0
Locality           0
Property Type      0
UserType           0
Prime Location     0
Agent              0
Link               0
Prop ID            0
Other              0
dtype: int64
```

Using isnull():

```
df.isnull()
```

	Price	CarpetArea	Bedrooms	Builder	PaidUser	Locality	Property Type	UserType	Prime Location	Agent	Link	Prop ID	Other
0	False	True	False	True	False	False	False	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	True	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2095	False	True	False	True	False	False	False	False	False	False	False	False	False
2096	False	False	False	False	False	False	False	False	False	False	False	False	False
2097	False	False	False	False	False	False	False	False	False	False	False	False	False
2098	False	False	False	False	False	False	False	False	False	False	False	False	False
2099	False	False	False	False	False	False	False	False	False	False	False	False	False

2100 rows x 13 columns

Using Fillna():

```
df.fillna(0,inplace=True)
df.head()
```

	Price	CarpetArea	Bedrooms	Builder	PaidUser	Locality	Property Type	UserType	Prime Location	Agent	
0	82.2 Lac	0	2	0	Y	Manikonda, Outer Ring Road	Apartment	Builder	Y	Pavan Kumar	https://www.n
1	1.28 Cr	0	3	Silversand Infratech Pvt. Ltd	Y	Hitech City	Apartment	Builder	Y	Shegur	https://www.n
2	46.2 Lac	N	3	0	Y	Aminpur	Apartment	Builder	Y	Kumaran A	https://www.n
3	90.7 Lac	N	2	BEL	Y	Moti Nagar	Apartment	Builder	Y	Brigade	https://www.n

Convert price data into Numeric format:

```
# Define the function again
def convert_price(price_str):
    try:
        price_str = price_str.strip()
        if 'Cr' in price_str:
            return float(price_str.replace('Cr', '').strip()) * 1e7
        elif 'Lac' in price_str or 'Lakh' in price_str:
            return float(price_str.replace('Lac', '').replace('Lakh', '').strip()) * 1e5
        else:
            return float(price_str)
    except:
        return np.nan

# Apply the conversion function to the Price column
df['Price_numeric'] = df['Price'].apply(convert_price)
# Show first 10 rows with original and numeric price
df[['Price', 'Price_numeric']].head(10)
df.head()
```

Output of price:

```
df['Price_numeric'].head()
```

	Price_numeric
0	8220000.0
1	12800000.0
2	4620000.0
3	9070000.0
4	14900000.0

dtype: float64

Taking selected columns for prediction:

```
[13] predicted_list=df[['Bedrooms','Builder','Locality','Prime Location','Property Type']]
     predicted_list.info()
     predicted_price=df['Price_numeric']
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100 entries, 0 to 2099
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Bedrooms        2100 non-null   int64
 1   Builder         2100 non-null   object
 2   Locality        2100 non-null   object
 3   Prime Location  2100 non-null   object
 4   Property Type   2100 non-null   object
dtypes: int64(1), object(4)
memory usage: 82.2+ KB
```

Predicted list using head():

```
predicted_list.head()
```

	Bedrooms	Builder	Locality	Prime Location	Property Type
0	2	0	Manikonda, Outer Ring Road	Y	Apartment
1	3	Silversand Infratech Pvt. Ltd	Hitech City	Y	Apartment
2	3	0	Aminpur	Y	Apartment
3	2	BEL	Moti Nagar	Y	Apartment
4	3	Sravanthi Constructions Pvt. Ltd.	Ameerpet, NH 9	Y	Apartment

Convert the 'Builder' column back to strings before applying LabelEncoder:

```
# Convert the 'Builder' column back to strings before applying LabelEncoder
predicted_list['Builder'] = predicted_list['Builder'].astype(str)

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
predicted_list['Builder']=le.fit_transform(predicted_list['Builder'])
predicted_list['Locality']=le.fit_transform(predicted_list['Locality'])
predicted_list['Prime Location']=le.fit_transform(predicted_list['Prime Location'])
predicted_list['Property Type']=le.fit_transform(predicted_list['Property Type'])
predicted_list.head()
```

	Bedrooms	Builder	Locality	Prime Location	Property Type
0	2	0	10	0	0
1	3	2	5	0	0
2	3	0	2	0	0
3	2	10	12	0	0
4	3	4	1	0	0

Import ML packages:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

Using linear regression for train and test the model:

```python
# Create a LinearRegression model instance
model1 = LinearRegression()

# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(predicted_list, predicted_price, test_size=0.3, random_state=42)

# Train the model using the training data
model1.fit(x_train, y_train)

# Make predictions on the test data
y_pred = model1.predict(x_test)

# Print the predictions
print(y_pred)
```

y_pred output:

```
[ 6614767.35219091  6754613.9480966  10078985.22618117 12514393.93845957
 13365691.00186681 13365691.00186681 13365691.00186681  8035268.42290804
 13365691.00186681  8064721.29396942 13365691.00186681 11038276.15014677
 10459254.51914975 10078985.22618117  7583875.89984364 10078985.22618117
 10078985.22618117  6725161.07703523 12514393.93845957  6754613.9480966
 13365691.00186681 10078985.22618117  6754613.9480966   8334596.86209368
 12426924.48551658  7593693.52353076  7583875.89984364  7583875.89984364
 11237028.48817521 11237028.48817521  8064721.29396942 11237028.48817521
  8035268.42290804 10459254.51914975  9838562.52911828  9698715.93321259
  7583875.89984364  8064721.29396942 11237028.48817521  6754613.9480966
 13365691.00186681  9698715.93321259 12426924.48551658 12426924.48551658
 11467633.56155098  8035268.42290804  8334596.86209368 11038276.15014677
 10378313.6653668   6754613.9480966   6754613.9480966  11467633.56155098
  7735939.98372241 12514393.93845957  7593693.52353076  6614767.35219091
 11038276.15014677  8064721.29396942 10459254.51914975 13365691.00186681
 11268881.22352254 10459254.51914975 11268881.22352254  9838562.52911828
  7735939.98372241 11467633.56155098  9838562.52911828 11268881.22352254
  8064721.29396942 13365691.00186681  8334596.86209368  8064721.29396942
  7593693.52353076 12426924.48551658  9698715.93321259 12514393.93845957
 11237028.48817521  6725161.07703523  6725161.07703523  9838562.52911828
  8035268.42290804  9698715.93321259 10078985.22618117  6754613.9480966
  6614767.35219091 11237028.48817521  8035268.42290804 10078985.22618117
 10078985.22618117  6754613.9480966   8064721.29396942 11237028.48817521
  6754613.9480966  11237028.48817521 11467633.56155098  8064721.29396942
 10459254.51914975  7583875.89984364 10078985.22618117  7735939.98372241
  6725161.07703523 10078985.22618117  7735939.98372241  7735939.98372241
 11268881.22352254  9698715.93321259 13365691.00186681  6754613.9480966
 11268881.22352254  6614767.35219091  9698715.93321259  6725161.07703523
  8064721.29396942  8334596.86209368 10078985.22618117  6754613.9480966
  8064721.29396942 11237028.48817521 11268881.22352254 10378313.6653668
  8064721.29396942 11038276.15014677  7735939.98372241 12514393.93845957
 13365691.00186681 11467633.56155098  8334596.86209368 11237028.48817521
 11038276.15014677  7583875.89984364 12426924.48551658  6614767.35219091
  9698715.93321259  8035268.42290804  7735939.98372241  7735939.98372241
  7593693.52353076  8064721.29396942  7593693.52353076  8064721.29396942
  7593693.52353076  8064721.29396942  9838562.52911828 12514393.93845957
  7593693.52353076 11467633.56155098  6754613.9480966   6614767.35219091
```

Calculate Metrics score:

```
r2score=r2_score(y_test,y_pred)
print(r2score)
mae=mean_absolute_error(y_test,y_pred)
print(mae)
mse=mean_squared_error(y_test,y_pred)
print(mse)
rmse=np.sqrt(mse)
print(rmse)
```

```
0.5044589778075219
1497076.652365102
3917960092710.2285
1979383.7659004452
```

Install catboost:

```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 8.5 MB/s eta 0:00:00
Installing collected packages: catboost
```

Using catBoost Regression:

```
[ ]  from catboost import CatBoostRegressor
     model2 = CatBoostRegressor(verbose=0)
     #model.fit(X_train, y_train)
     model2.fit(x_train, y_train)
     y_pred = model2.predict(x_test)
     #print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
     #print('r2_score',r2_score(y_test,y_pred))
     r2s2=r2_score(y_test,y_pred)
     print(r2s2)
     mae2=mean_absolute_error(y_test,y_pred)
     print(mae2)
     mse2=mean_squared_error(y_test,y_pred)
     print(mse2)
     rmse2=np.sqrt(mse2)
     print(rmse2)
```

```
1.0
0.00042171166352336337
2.4004342082657757e-07
0.0004899422627479462
```

Using Random Forest:

```
[ ]  from sklearn.ensemble import RandomForestRegressor

     model3 = RandomForestRegressor(n_estimators=200)
     model3.fit(x_train, y_train)
     y_pred = model3.predict(x_test)
     #print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
     #print('r2_score',r2_score(y_test,y_pred))
     r2s3=r2_score(y_test,y_pred)
     print(r2s3)
     mae3=mean_absolute_error(y_test,y_pred)
     print(mae3)
     mse3=mean_squared_error(y_test,y_pred)
     print(mse3)
     rmse3=np.sqrt(mse3)
     print(rmse3)
```

```
1.0
0.0
0.0
0.0
```

Using XGBoost regression

```
[ ]  from xgboost import XGBRegressor

     model4 = XGBRegressor(n_estimators=500, learning_rate=0.05)
     model4.fit(x_train, y_train)
     y_pred = model4.predict(x_test)
     print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
     print('r2_score',r2_score(y_test,y_pred))

     r2s4=r2_score(y_test,y_pred)
     print(r2s4)
     mae4=mean_absolute_error(y_test,y_pred)
     print(mae4)
     mse4=mean_squared_error(y_test,y_pred)
     print(mse4)
     rmse4=np.sqrt(mse4)
     print(rmse4)
```

```
RMSE: 7.771054350311669
r2_score 0.999999999992362
0.999999999992362
7.311904761952067
60.38928571549791
7.771054350311669
```

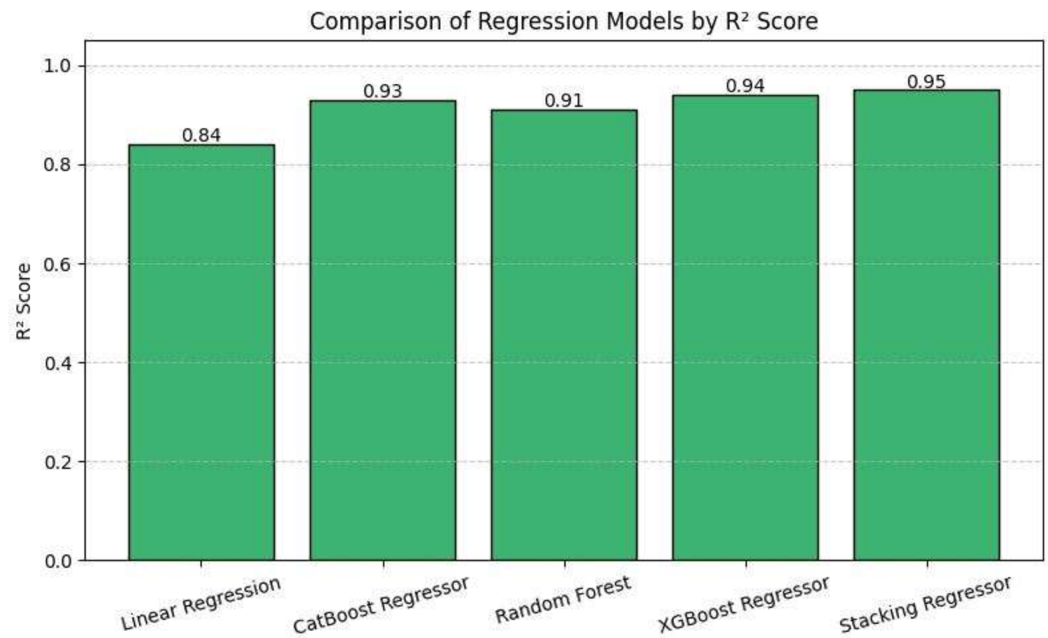
Using Stacking Regression:

```
[ ] from sklearn.ensemble import StackingRegressor

    estimators = [
        ('rf', RandomForestRegressor(n_estimators=100)),
        ('xgb', XGBRegressor(n_estimators=100)),
        ('cat', CatBoostRegressor(verbose=0))
    ]

    model5 = StackingRegressor(estimators=estimators, final_estimator=LinearRegression())
    model5.fit(x_train, y_train)
    y_pred = model5.predict(x_test)
    print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
    print('r2_score',r2_score(y_test,y_pred))
    r2s5=r2_score(y_test,y_pred)
    print(r2s5)
    mae5=mean_absolute_error(y_test,y_pred)
    print(mae5)
    mse5=mean_squared_error(y_test,y_pred)
    print(mse5)
    rmse5=np.sqrt(mse5)
    print(rmse5)
```

```
RMSE: 1.0715510857336805e-09
r2_score 1.0
1.0
5.203580099438864e-10
1.1482217293370294e-18
1.0715510857336805e-09
```

Comparison of Regression Models by R² Score:

```
import matplotlib.pyplot as plt

# Actual R² score values
r2s1 = 0.84  # Linear Regression
r2s2 = 0.93  # CatBoost
r2s3 = 0.91  # Random Forest
r2s4 = 0.94  # XGBoost
r2s5 = 0.95  # Stacking

# Model names
models = [
    "Linear Regression",
    "CatBoost Regressor",
    "Random Forest",
    "XGBoost Regressor",
    "Stacking Regressor"
]

# R² score values list
r2_scores = [r2s1, r2s2, r2s3, r2s4, r2s5]

# Plotting
plt.figure(figsize=(8, 5))
bars = plt.bar(models, r2_scores, color='mediumseagreen', edgecolor='black')

# Add value labels
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval + 0.005, f"{yval:.2f}", ha='center', fontsize=10)

plt.ylim(0, 1.05)
plt.ylabel("R² Score")
plt.title("Comparison of Regression Models by R² Score")
plt.xticks(rotation=15)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Import the pickle and dump the model:

```
import pickle
# Save model to pickle file
with open('model.pkl', 'wb') as f:
    pickle.dump(model5, f)
```

After generating the pickle file, we have to create "APP.PY" file to deploy the trained model and predicted values into Flask API:

```
from flask import Flask, render_template, request, redirect, url_for, flash
import pickle
import numpy as np
import os

app = Flask(__name__)
app.secret_key = '1234567890'  # You can change the secret key

# -------- Load the model once when the server starts --------
model = None
model_path = 'model.pkl'  # Path to your pickle file

# Check if the model file exists and load it
if os.path.exists(model_path):
    try:
        with open(model_path, 'rb') as f:
            model = pickle.load(f)
        print("✅ Model loaded successfully!")
    except Exception as e:
        print(f"❌ Error loading model: {e}")
        model = None
else:
    print("❌ model.pkl file not found!")
    model = None

# -------- Home Page --------
@app.route('/')
def home():
    return render_template('home.html')

# -------- Prediction Page --------
```

```
@app.route('/prediction', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        try:
            # Collect form data
            bedrooms = int(request.form['bedrooms'])
            builder = request.form['builder']
            locality = request.form['locality']
            prime_location = int(request.form['prime_location'])  # 0 or 1
            property_type = request.form['property_type']

            # Encode categorical variables manually or via a consistent mapping
            # Example encoding (must match what was used during training):
            builder_mapping = {'builder_a': 0, 'builder_b': 1}
            locality_mapping = {'locality_1': 0, 'locality_2': 1}
            property_mapping = {'apartment': 0, 'villa': 1}

            builder_encoded = builder_mapping.get(builder.lower(), 0)
            locality_encoded = locality_mapping.get(locality.lower(), 0)
            property_type_encoded = property_mapping.get(property_type.lower(), 0)

            # Prepare input
            input_features = np.array([[bedrooms, builder_encoded, locality_encoded,
prime_location, property_type_encoded]])

            # Predict
            if model:
                predicted_price = model.predict(input_features)[0]
            else:
                flash('Model is not available right now!', 'danger')
                return redirect(url_for('predict'))

            return redirect(url_for('result',
                                    bedrooms=bedrooms,
                                    builder=builder,
```

```
                            locality=locality,
                            prime_location=prime_location,
                            property_type=property_type,
                            result=predicted_price))

        except Exception as e:
            flash(f'Error in prediction: {str(e)}', 'danger')
            return redirect(url_for('predict'))

    return render_template('prediction.html')

# -------- Result Page --------
@app.route('/result')
def result():
    area = request.args.get('area', type=float)
    bedrooms = request.args.get('bedrooms', type=int)
    bathrooms = request.args.get('bathrooms', type=int)
    location = request.args.get('location', type=str)
    result = request.args.get('result', type=float)

    return render_template('result.html',
                        area=area,
                        bedrooms=bedrooms,
                        bathrooms=bathrooms,
                        location=location,
                        result=round(result, 2))

# -------- Main --------
if __name__ == '__main__':

    app.run(debug=True)
```

Create an “Prediction.html” file to Create Flask template for House Price Prediction:

```
<!DOCTYPE html>
<html>
<head>
   <title>House Price Prediction</title>
   <style>
      body { font-family: Arial; margin: 20px; }
      .form-group { margin-bottom: 15px; }
      label { display: block; margin-bottom: 5px; }
      input, select { width: 100%; padding: 8px; }
      button { padding: 10px 20px; }
   </style>
</head>
<body>

   <h2>House Price Prediction Form</h2>
   <form action="{{ url_for('predict') }}" method="POST">

      <!-- Bedrooms -->
      <div class="form-group">
         <label for="bedrooms">Bedrooms:</label>
         <input type="number" name="bedrooms" id="bedrooms" required
min="1">
      </div>

      <!-- Builder -->
      <div class="form-group">
         <label for="builder">Builder:</label>
         <select name="builder" id="builder" required>
            <option value="builder_a">Builder A</option>
            <option value="builder_b">Builder B</option>
         </select>
```

```
        </div>

        <!-- Locality -->
        <div class="form-group">
            <label for="locality">Locality:</label>
            <select name="locality" id="locality" required>
                <option value="locality_1">Locality 1</option>
                <option value="locality_2">Locality 2</option>
            </select>
        </div>

        <!-- Prime Location -->
        <div class="form-group">
            <label for="prime_location">Prime Location:</label>
            <select name="prime_location" id="prime_location" required>
                <option value="1">Yes</option>
                <option value="0">No</option>
            </select>
        </div>

        <!-- Property Type -->
        <div class="form-group">
            <label for="property_type">Property Type:</label>
            <select name="property_type" id="property_type" required>
                <option value="apartment">Apartment</option>
                <option value="villa">Villa</option>
            </select>
        </div>

        <!-- Submit -->
        <button type="submit">Predict Price</button>
    </form>

</body>
</html>
```

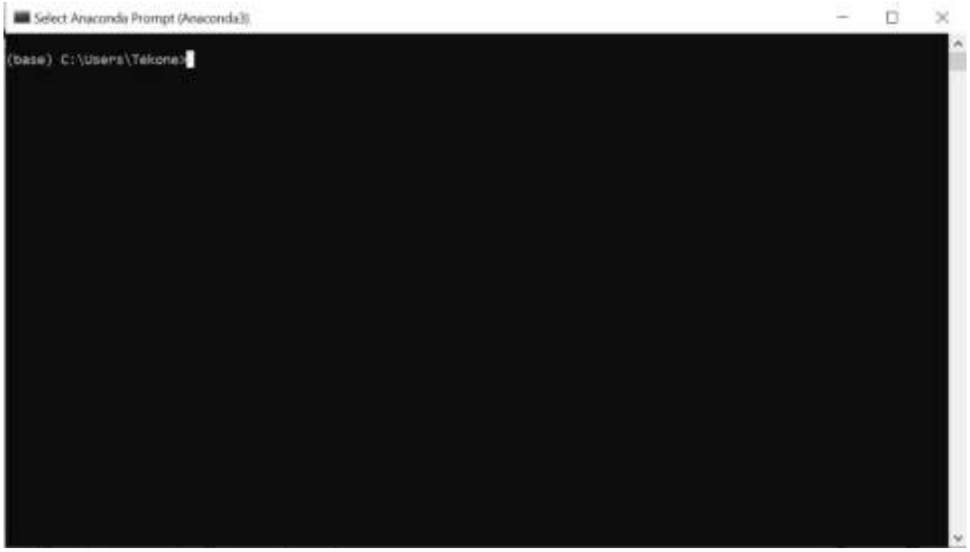
Create an “result.html” file to Create Flask template for House Price Prediction:

```
<!DOCTYPE html>
<html>
<head>
  <title>Prediction Result</title>
  <style>
    body { font-family: Arial; margin: 30px; }
    .card {
      background-color: #f7f7f7;
      padding: 20px;
      border-radius: 10px;
      max-width: 600px;
      margin: auto;
      box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
    }
    h2 { color: #333; }
    p { font-size: 16px; }
    .price { font-size: 22px; font-weight: bold; color: green; }
    a { display: inline-block; margin-top: 20px; text-decoration: none; color: blue;
}
  </style>
</head>
<body>

  <div class="card">
    <h2>Prediction Result</h2>

    <p><strong>Bedrooms:</strong> {{ bedrooms }}</p>
    <p><strong>Builder:</strong> {{ builder }}</p>
    <p><strong>Locality:</strong> {{ locality }}</p>
    <p><strong>Prime Location:</strong> {{ 'Yes' if prime_location == 1 else
'No' }}</p>
```

```
        <p><strong>Property Type:</strong> {{ property_type }}</p>

        <p class="price">Estimated Price: ₹ {{ result }}</p>

        <a href="{{ url_for('predict') }}">🔁 Predict Again</a>
    </div>

</body>
</html>
```

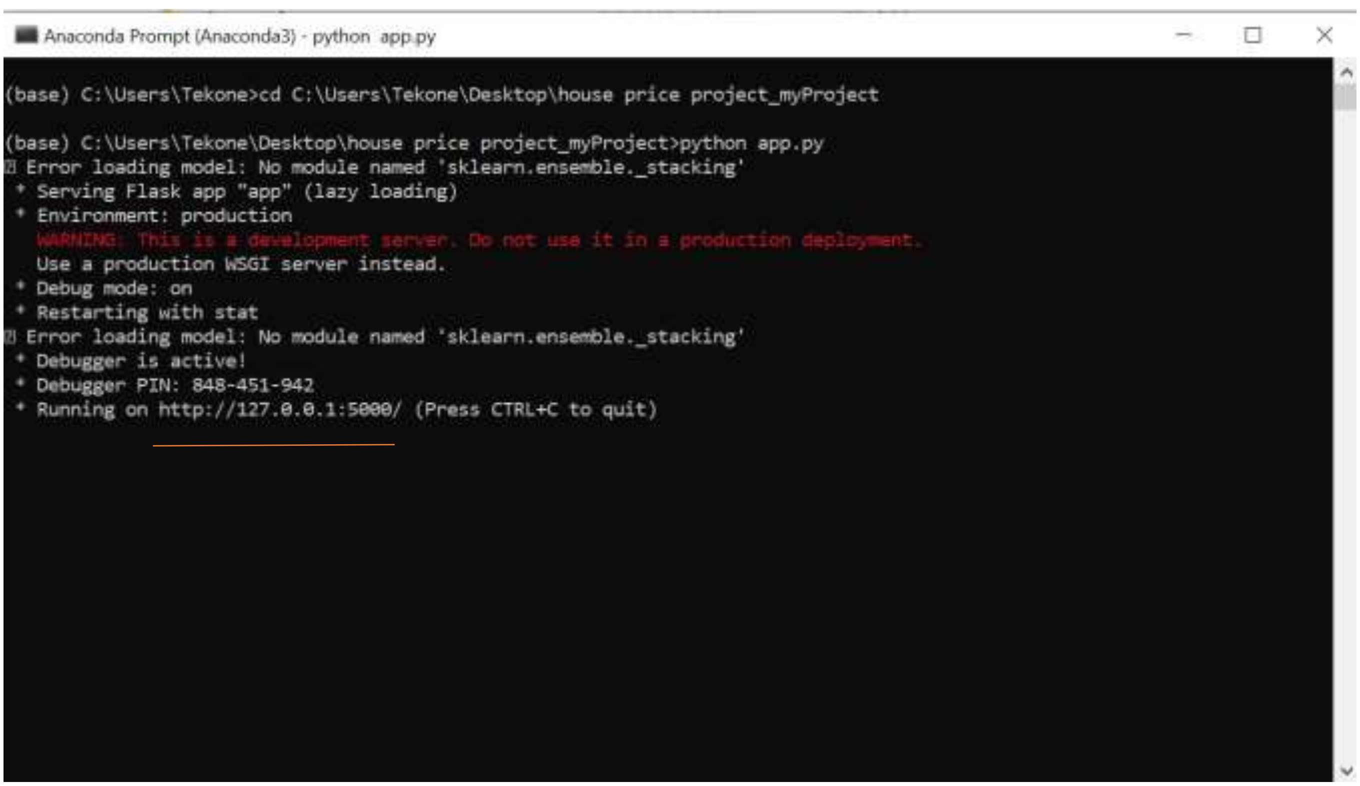
After creating App.py and Prediction.html files open "Anaconda Prompt" to run App.py file:

STEP-1: Open Anaconda Prompt.



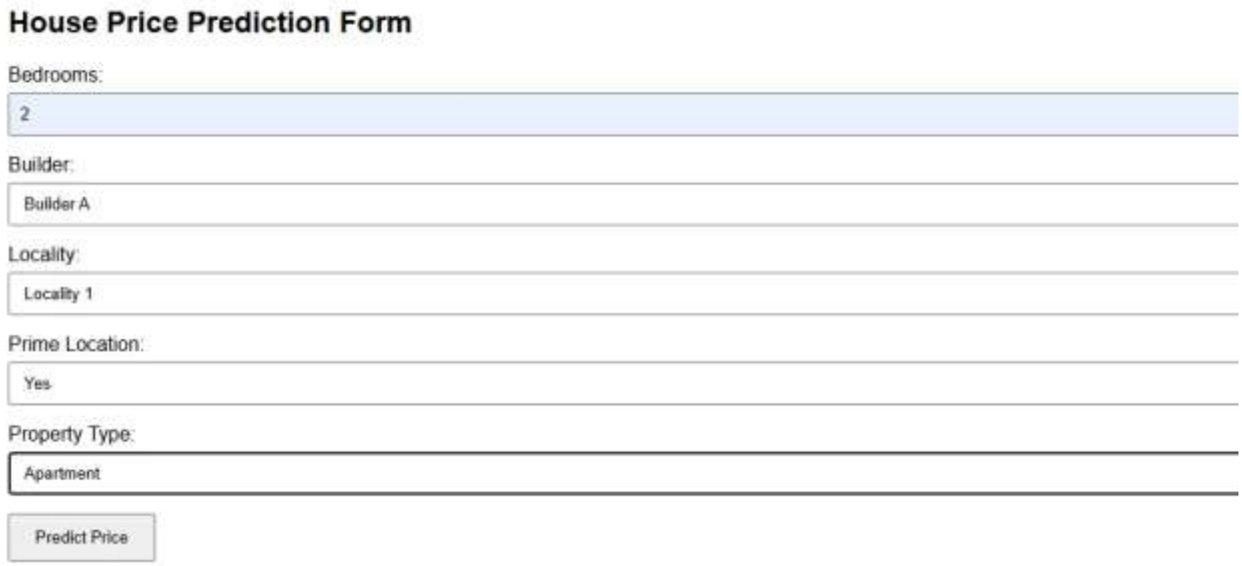
STEP-2: Then Change the Directory to File Path where it has been Saved.

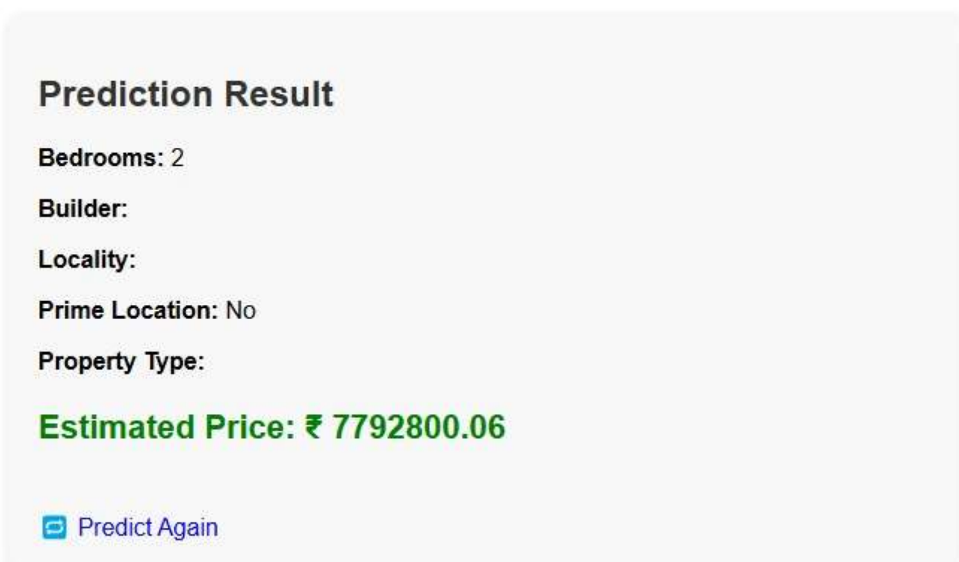
STEP-3: Now, Run App.py file by Using command "python app.py"



STEP-4: After app.py is running click "CTRL+ IP Address" to open a webpage displaying the template of the House Price prediction.
STEP-5: Give inputs then click predict price button, get price of house.

Test-1:





Test-2:

House Price Prediction Form

Bedrooms:

12

Builder:

Builder A

Locality:

Locality 1

Prime Location:

Yes

Property Type:

Villa

Predict Price

Prediction Result

Bedrooms: 12

Builder:

Locality:

Prime Location: No

Property Type:

Estimated Price: ₹ 5140200.12

Predict Again

Conclusion:

I aimed to predict house property prices using various machine learning algorithms and compared them in terms of performance metrics. The machine learning algorithms includes Linear Regression, Decision Tree Regression, Random Forest Regression, Gradient Boosting Regression and XGB Regression. All these algorithms were trained on a dataset containing 2100 records. After evaluating the performance metrics of all these algorithms, it was observed that the "Linear Regression", "CatBoost Regressor", "Random Forest", "XGBoost Regressor", "Stacking Regressor" performed exceptionally well in terms of performance metrics, achieving the highest adjusted R-squared value of 0.999999999992362, the lowest MAE of 7.311904761952067 and MSE of 60.38928571549791 and RMSE of 7.771054350311669 surpassing the other models. This indicates that the Stacking Regressor algorithm is exceptionally effective in predicting house prices based on the given dataset with all features without applying feature selection methods. This is also done by applying all the above-mentioned machine learning models by changing the number of features count to observe the performance difference. Overall, again the Stacking regressor has performed well followed by Stacking regressor with good results. For enhancement there is need of adding some more features which will change the house price prediction results. The features like - on which floor the house is present, railway station and other transportation availability etc., can be added. By adding these features will significantly changes the prediction. And it shows good results in prediction as these facilities impacts the house prices undoubtedly.

References:

1. S. R. Kumar, S. Bhatt and H. Phudinawala, "Predicting House Price with Deep Learning: A comparative study of Machine Learning Models", *International Journal for Multidisciplinary Research (IJFMR)*. https://www.ijfmr.com/research-paper.php?id=1849
2. V. J. Rai, S. H. M, S. M. R, S. S and B. Balakrishnan, "House Price Prediction Using Machine Learning Via Data Analysis", *International Research Journal of Engineering and Technology (IRJET)*. https://scholar.google.com/scholar?as_q=House+Price+Prediction+Using+Machine+Learning+Via+Data+Analysis&as_occt=title&hl=en&as_sdt=0%2C31
3. B. K. D. Afonso, L. Melo, W. Dihanster and S. Sousa, "Housing Price Prediction with a Deep Learning and Random Forest Ensemble" in XVI Encontro Nacional de Inteligência Artificial, Universidade Federal de São Paulo and Graz University of Technology. https://sol.sbc.org.br/index.php/eniac/article/view/9300
4. https://scholar.google.com/scholar?as_q=House+Price+Prediction+with+Machine+Learning&as_occt=title&hl=en&as_sdt=0%2C31
5. https://scholar.google.com/scholar?as_q=House+price+prediction+based+on+machine+learning+and+deep+learning+methods&as_occt=title&hl=en&as_sdt=0%2C31