CSRBOX
Doing Good in a Better Way

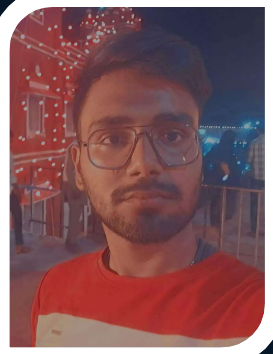
Presentation

HOUSE PRICE PREDICTION MODEL

CODER'S HOPE

IBM SkillsBuild for Adult Learners - Data Analytics

Team Members



Ankur

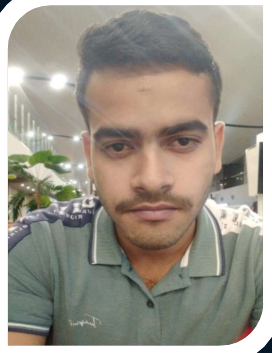
Himanshu

Bhuvan

Jannat

Harsh

Vivek



Anshul

Priya

CONTENT

INTRODUCTION

As India's real estate market experiences rapid growth, accurately predicting house prices becomes increasingly essential. Traditional methods often struggle to account for the complex interplay of factors influencing property values, such as location, infrastructure, and economic conditions. Machine learning offers a solution by analyzing large datasets to uncover patterns and correlations that traditional approaches might miss, leading to more precise and reliable price forecasts.

PROJECT SUMMARY

This project will use machine learning to develop a system for predicting house prices using regression models, Regression techniques can provide valuable insights into future price trends by establishing relationships between house prices and various influencing factors such as size, location, and amenities. Machine learning models can predict house prices in India, helping stakeholders make more informed decisions in a dynamic real estate market.

PURPOSE OF PROJECT

- **Accurate Price Forecasting**:
  Develop a machine learning model to predict house prices with high accuracy, aiding in better investment and purchasing decisions.
- **Market Insight:**
  Provide valuable insights into future price trends and market dynamics by analyzing historical data and identifying key influencing factors.
- **Informed Decision-Making:**
  Support stakeholders, including buyers, investors, and real estate professionals, in making well-informed decisions based on predictive data.
- **Enhanced Valuation Process:**
  Streamline and improve the property valuation process by offering a reliable tool that reduces uncertainty and enhances strategic planning.

MODULES USED

NumPy is a fundamental Python library offering a wide range of mathematical functions for data manipulation and analysis.

Python library for data manipulation and analysis, offering easy-to-use data structures and tools to efficiently work with structured data.

Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Scikit-learn is a widely-used library that provides a rich set of tools for classification, regression, clustering, and more for building and evaluating models.

SOURCE OF DATA USED

Data Source - Data is taken from Kaggle. There are several attributes in the data set on the basis of which we are predicting, analyzing, visualizing, and detection of diabetes.

LINK: **https://www.kaggle.com/datasets/anmolkumar/house-price-prediction challenge/data?select=train.csv**

	POSTED_BY	UNDER_CONSTRUCTION	RERA	BHK_NO.	BHK_OR_RK	SQUARE_FT	READY_TO_MOVE	RESALE	ADDRESS	LONGITUDE	LATITUDE	TARGET(PRICE_IN_LACS)
5639	Owner	0	0	2	BHK	650.026001	1	1	sector-121 Noida,Noida	28.602325	77.401994	25.0
12552	Dealer	0	0	3	BHK	1741.755690	1	1	Sector-29 Noida,Noida	28.568966	77.335388	150.0
15430	Owner	0	0	3	BHK	1340.183158	1	1	Kanke,Ranchi	23.355560	85.334720	60.0
9802	Owner	0	0	2	BHK	1020.176831	1	1	Sardar Patel Stadium, Motera,Gandhinagar	23.091949	72.596893	45.0
21896	Dealer	0	1	1	BHK	800.000000	1	1	Hinjewadi,Pune	18.585300	73.741300	39.4
10917	Owner	0	0	3	BHK	1548.369416	1	1	Rani Bazar,Bikaner	27.994242	73.326443	56.5
3383	Owner	0	0	2	BHK	950.045970	1	1	Ayyakoneru,Vizianagaram	18.116670	83.416670	31.0
5431	Dealer	1	1	2	BHK	910.104012	0	1	Tathawade,Pune	18.618200	73.744700	49.0
25807	Dealer	0	0	1	BHK	333.035315	1	1	Kundli,Sonipat	28.874383	77.113251	14.9
16329	Dealer	0	0	2	BHK	1100.110011	1	1	Aundh,Pune	18.556218	73.811303	100.0

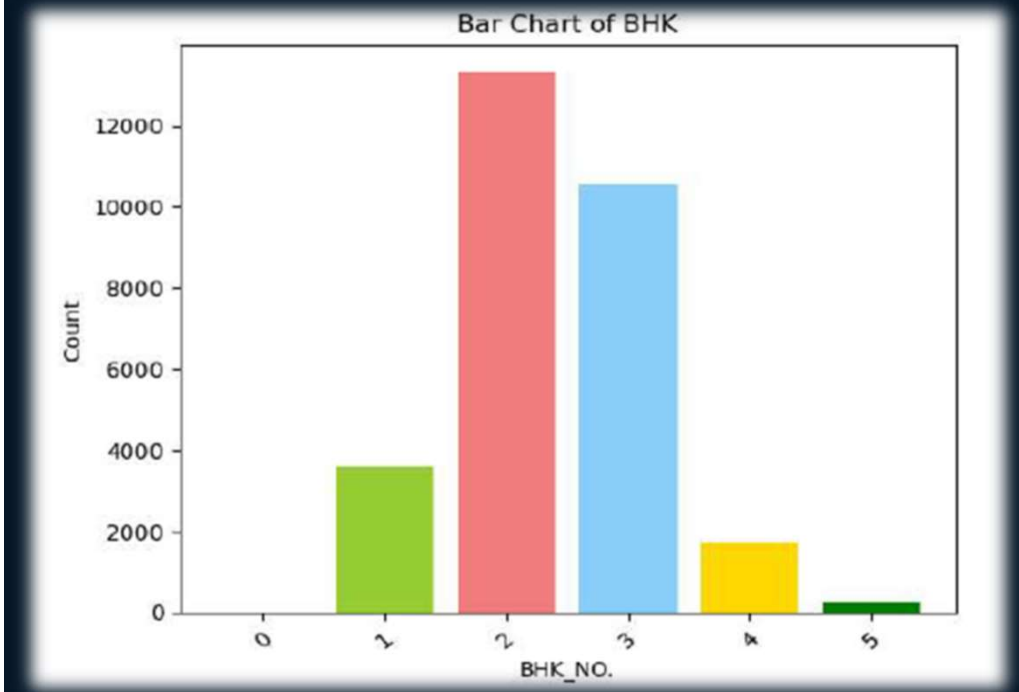
DATA CLEANI NG

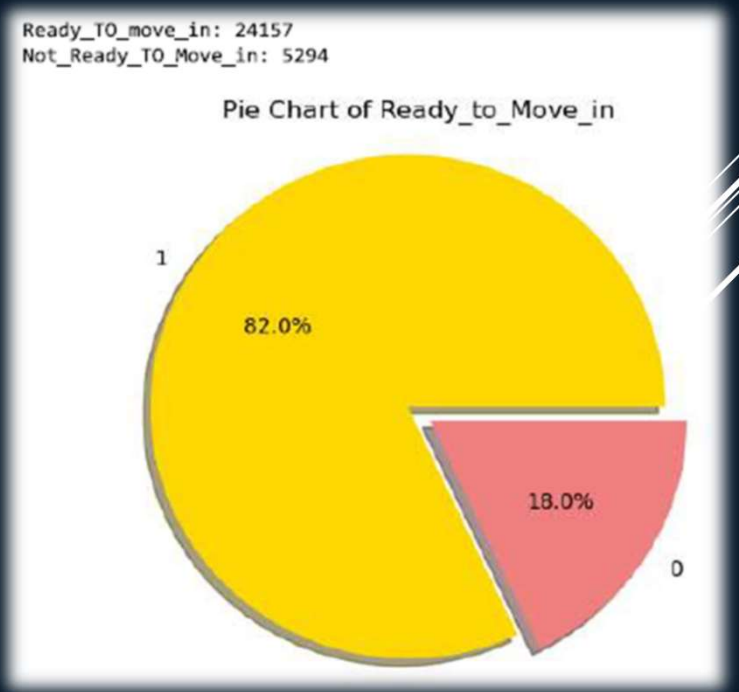
- the bhk or rk is not having much significant impact, the RK value has BHK_NO = 1.
- The low count of RK values in the BHK or RK column is the reason
- Creating a new column named 'CITY' by splitting the 'ADDRESS' column and extracting the second element (index 1) after splitting by ','
- Splitting Posted_By column into 3 different Posted_By_Dealer, Posted_By_Owner, Posted_By_Builder

```
# Display randomly any number of records of data
df.sample(10)
```

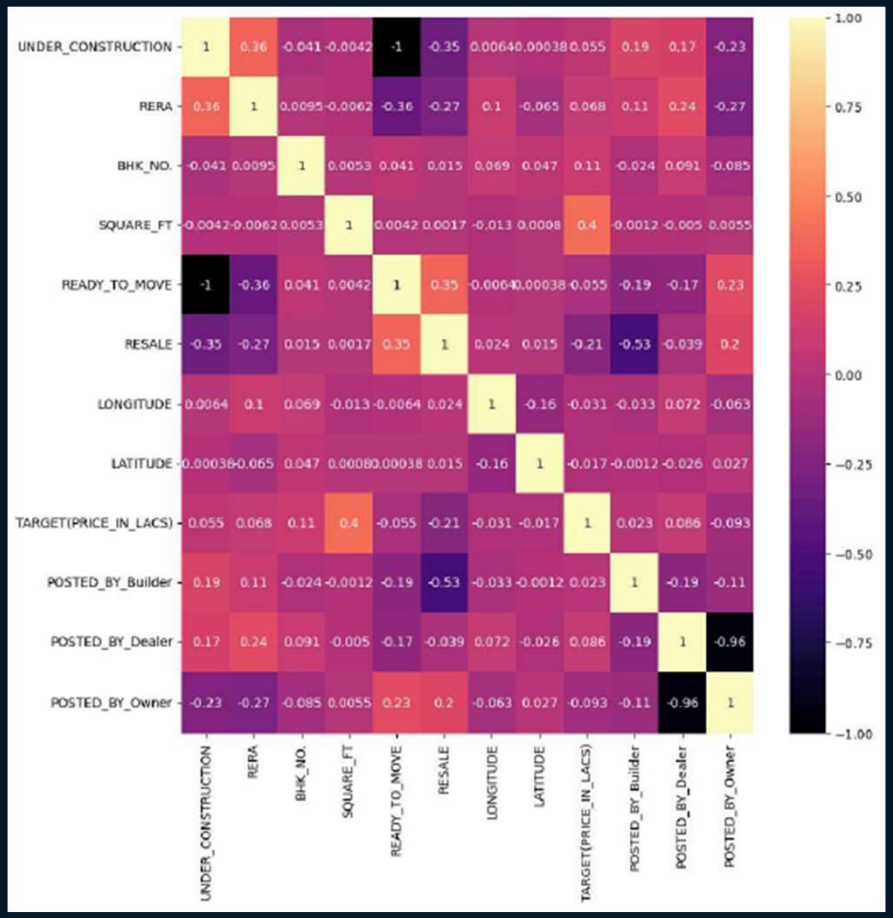
	UNDER_CONSTRUCTION	RERA	BHK_NO.	SQUARE_FT	READY_TO_MOVE	RESALE	LONGITUDE	LATITUDE	TARGET(PRICE_IN_LACS)	POSTED_BY_Builder	POSTED_BY_Dealer	POSTED_BY_Owner	CITY
28164	0	1	2	719.978746	1	1	18.651598	73.735448	54.2	0	1	0	Pune
22043	0	0	2	1090.083270	1	1	13.066394	77.571988	72.0	0	0	1	Bangalore
22094	1	1	2	407.725322	0	0	24.690200	78.416000	24.7	1	0	0	Lalitpur
16437	1	1	3	1405.032571	0	1	19.223942	73.139554	110.0	0	1	0	Mumbai
25056	1	1	1	635.055038	0	1	24.690280	78.418890	60.0	0	1	0	Lalitpur
21344	0	0	3	1600.000000	1	1	22.802995	86.167625	60.0	0	0	1	Jamshedpur
13703	0	0	2	1280.000000	1	1	29.892311	78.000723	40.0	0	0	1	Haridwar
11100	0	0	3	1253.213368	1	1	22.459169	88.364505	39.0	0	1	0	Kolkata
4472	0	0	2	867.183934	1	1	18.914541	72.818211	190.0	0	1	0	Mumbai
16738	0	1	1	538.116592	1	1	18.693610	74.138060	18.0	0	0	1	Pune

DATA VISUALISATION

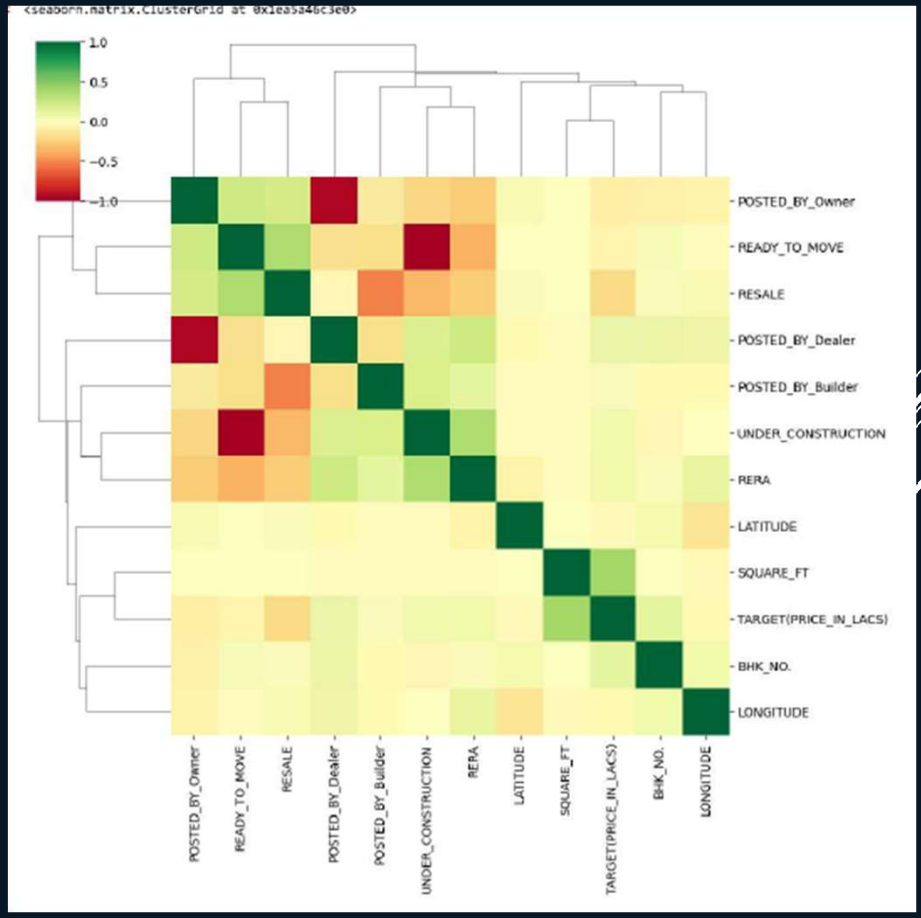




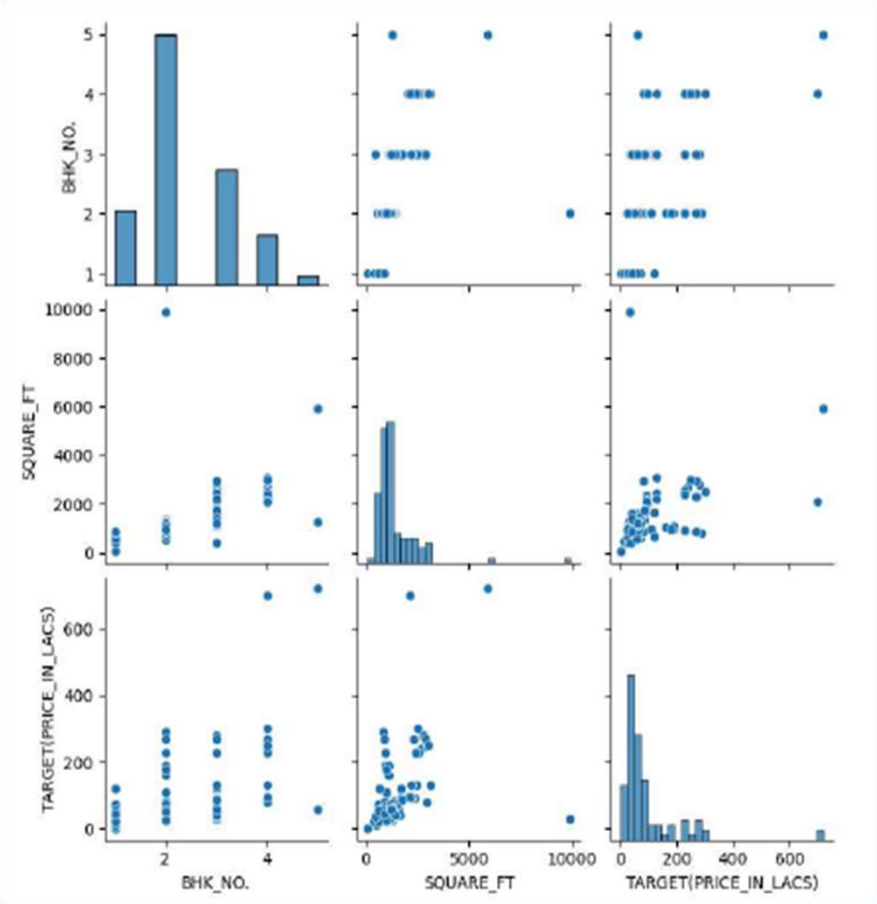
HEATMAP



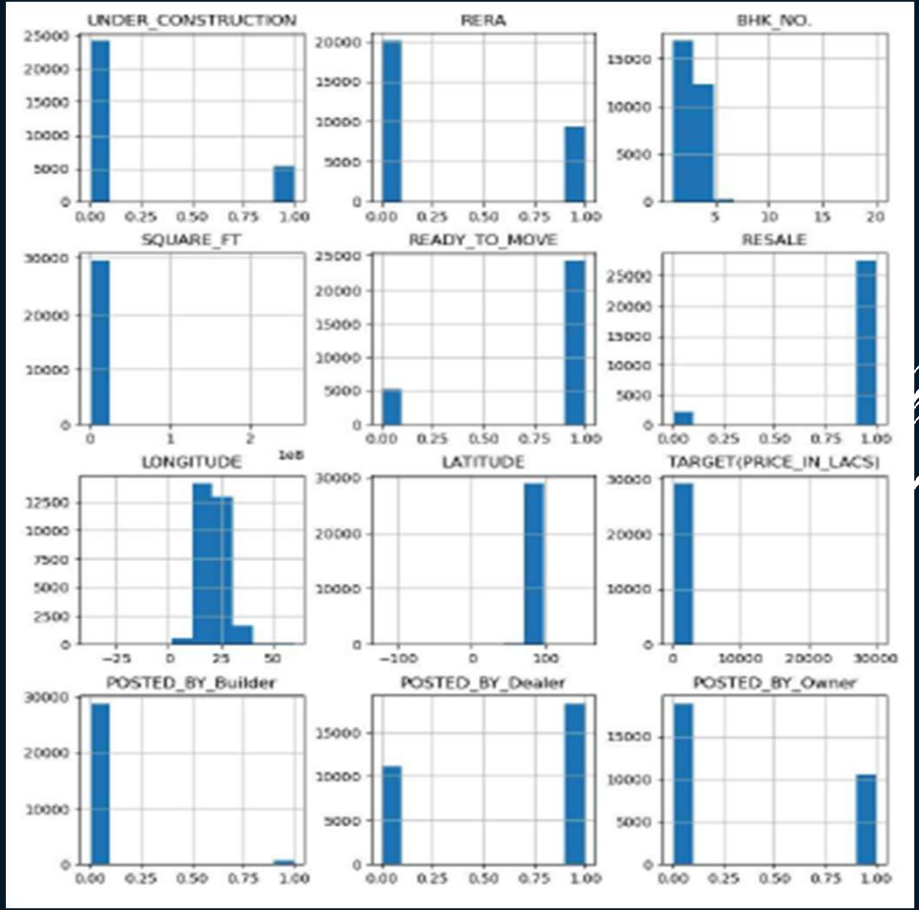
CLUSTERMAP



PAIRPLOT



HISTOGRAM



MSE AND R SQUARED VALUES

Linear Regression

```
from sklearn.metrics import mean_squared_error, r2_score

# Linear Regression
mse_lr = mean_squared_error(y_test,lr_pred)
r2_lr = r2_score(y_test, lr_pred)

print("Linear Regression:\n")

adjr2 = 1-(1-r2_lr)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_lr)
print( ' R2 :' , adjr2)
```

```
Linear Regression:

 MSE : 362080.326728981
 R2 : 0.09124695515664938
```

GradientBoosting Regressor

```
# GradientBoosting Regressor
mse_gb = mean_squared_error(y_test, gbm_pred)
r2_gb = r2_score(y_test, gbm_pred)

print("Gradient Boosting Regressor:\n")

adjr2 = 1-(1-r2_gb)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_gb)
print( ' R2 :' , adjr2)
```

```
Gradient Boosting Regressor:

 MSE : 16093.35777680913
 R2 : 0.9596087199391655
```

MSE AND R SQUARED VALUES

Random Forest Regressor

```
# Random Forest Regressor
mse_rf = mean_squared_error(y_test, rf_pred)
r2_rf = r2_score(y_test, rf_pred)

print("Random Forest Regressor:\n")

adjr2 = 1-(1-r2_rf)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_rf)
print( ' R2 :' , adjr2)
```

```
Random Forest Regressor:

 MSE : 14837.316994735642
 R2 : 0.9627611444176463
```

Extreme GradientBoosting Regressor

```
# Extreme Gradient Boosting Regressor
mse_xgb = mean_squared_error(y_test, xgbr_pred)
r2_xgb = r2_score(y_test, xgbr_pred)

print("Extreme Gradient Boosting Regressor:\n")

adjr2 = 1-(1-r2_xgb)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_xgb)
print( ' R2 :' , adjr2)
```

```
Extreme Gradient Boosting Regressor:

 MSE : 68627.52541170032
 R2 : 0.827757908745404
```

MSE AND R SQUARED VALUES

Lasso Regression

```
# Lasso Regression
mse_lasso = mean_squared_error(y_test, lasso_pred)
r2_lasso = r2_score(y_test, lasso_pred)

print("Lasso Regression:\n")

adjr2 = 1-(1-r2_lasso)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_lasso)
print( ' R2 :' , adjr2)
```

```
Lasso Regression:

 MSE : 362101.32746942656
 R2 : 0.09119424727551073
```

Ridge Regression

```
# Ridge Regression
mse_ridge = mean_squared_error(y_test, ridge_pred)
r2_ridge = r2_score(y_test, ridge_pred)

print("Ridge Regression:\n")

adjr2 = 1-(1-r2_ridge)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_ridge)
print( ' R2 :' , adjr2)
```

```
Ridge Regression:

 MSE : 362085.04072784627
 R2 : 0.09123512391229283
```

MSE AND R SQUARED VALUES

Elastic Net Regression

```
# Elastic Net Regression
mse_en = mean_squared_error(y_test, en_pred)
r2_en = r2_score(y_test, en_pred)

print("Elastic Net Regression:\n")

adjr2 = 1-(1-r2_en)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_en)
print( ' R2 :' , adjr2)
```

```
Elastic Net Regression:

 MSE : 371051.2427040931
 R2 : 0.06873165508203605
```

LightGBM Regression

```
# LightGBM Regressor
mse_lg = mean_squared_error(y_test, lg_pred)
r2_lg = r2_score(y_test, lg_pred)

print("LightGBM Regressor:\n")

adjr2 = 1-(1-r2_lg)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_lg)
print( ' R2 :' , adjr2)
```

```
LightGBM Regressor:

 MSE : 68627.52541170032
 R2 : 0.827757908745404
```

MSE AND R SQUARED VALUES

ADABoost Regression

```
# ADABoost Regressor
mse_ada = mean_squared_error(y_test, ada_pred)
r2_ada = r2_score(y_test, ada_pred)

print("ADABoost Regressor:\n")

adjr2 = 1-(1-r2_ada)*(29000-1)/(29000-10-1)
print( ' MSE :' , mse_ada)
print( ' R2 :' , adjr2)
```

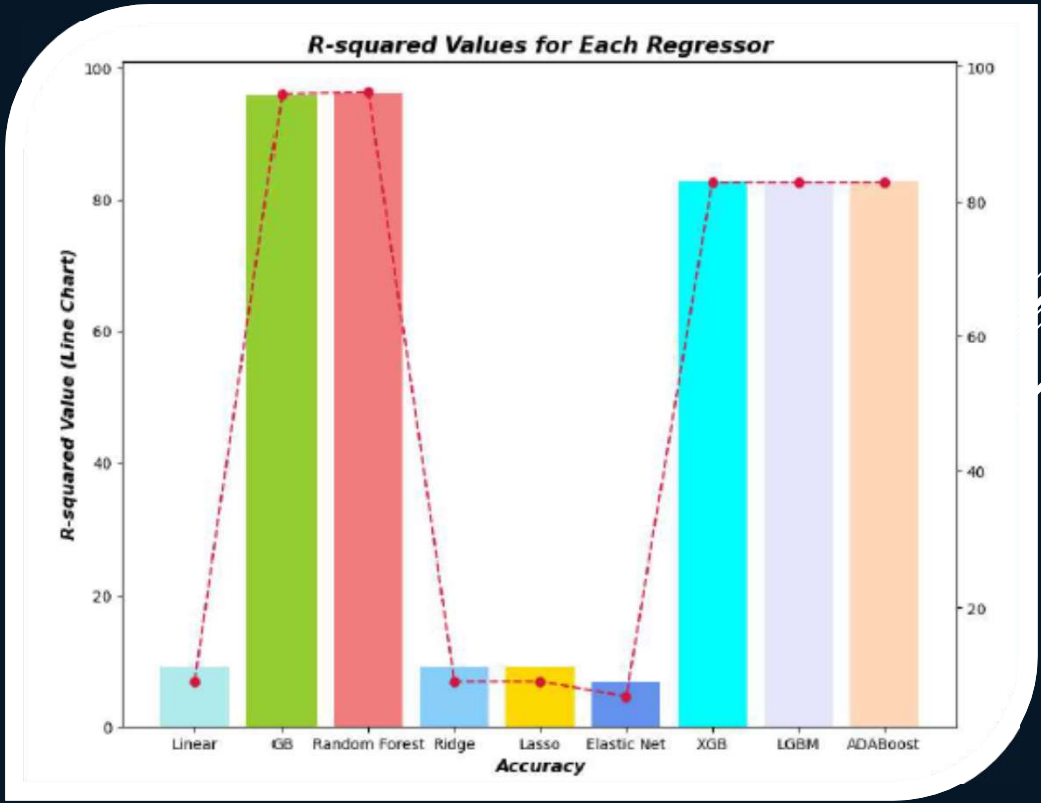
```
ADABoost Regressor:

 MSE : 68627.52541170032
 R2 : 0.827757908745404
```

Accuracy Conclusion and Visualisation

From the Graph We Can conclude Some points :

- GradientBoosting and Random Forest are best suited models with 95.96% and 96.27%.
- XGB, LightGBM and ADABoost are also good with 82.77% accuracy.
- Linear, Ridge, Lasso and Rlastic Net are not suited as they have very low Accuracy Score.



CSRBOX®
Doing Good in a Better Way

THANK YOU