Mumbai House Price Prediction Model

Aarushi Ashish Gupta

Manipal Academy of Higher Education

Objective

The primary objective of this analysis is to develop a model that can accurately and efficiently predict the house prices in Mumbai, India. The study was focused on analyzing the data and finding the model that best predicts the house prices based on location, property type, size, age, and status. The testing on different models ensures that stakeholders obtain reliable results that can aid them in making informed decisions and mitigate risk while investing in real estate. This can also help analyze market trends and real estate agents to offer precise pricing information and superior customer service.

Data Description

This house price dataset for Mumbai contains information on the sale prices of residential properties in Mumbai along with the location, size, and age of the properties.

```
df = pd.read_csv("Mumbai House Prices.csv")
df.head()
```

	bhk	type	locality	area	price	price_unit	region	status	age
0	3	Apartment	Lak And Hanware The Residency Tower	685	2.50	Cr	Andheri West	Ready to move	New
1	2	Apartment	Radheya Sai Enclave Building No 2	640	52.51	L	Naigaon East	Under Construction	New
2	2	Apartment	Romell Serene	610	1.73	Cr	Borivali West	Under Construction	New
3	2	Apartment	Soundlines Codename Urban Rainforest	876	59.98	L	Panvel	Under Construction	New
4	2	Apartment	Origin Oriana	659	94.11	L	Mira Road East	Under Construction	New

- There are 76038 rows and 9 columns in this dataset
- The dataset was cleaned and the prices were changes to lakhs based on the price_unit column and a new column price_lakhs was created
- The locality column was dropped from the dataset
- Columns

1. **Bhk:** Number of bedrooms in the house
   Unique values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
2. **Type:** Type of apartment (e.g., Apartment, Studio Apartment, Villa)
   Unique values: Apartment, Studio Apartment, Villa, Independent House, Penthouse
   Apartment: 74,854 entries
   Studio Apartment: 882 entries
   Villa: 226 entries
   Independent House: 73 entries
   Penthouse: 3 entries
3. **Area:** Area size of the property in square feet
   New: 38,072 entries
   Resale: 23,357 entries
   Unknown: 14,609 entries
4. **Region:** Geographic region in Mumbai where the property is located
   Total unique regions: 228

Top regions: Thane West (14,868 entries), Mira Road East (9,902 entries), Dombivali (3,041 entries), Kandivali East (2,568 entries), Kharghar (2,362 entries), and 223 other regions

Dimensionality Reduction was performed for this feature and the regions with less than 20 entries were renamed as ‘other’

5. **Status:** Status of the property (Ready to move, Under Construction)
   Ready to move: 44,982 entries
   Under Construction: 31,056 entries
6. **Age:** Age category of the property (New, Resale, Unknown)
   New: 38,072 entries
   Resale: 23,357 entries
   Unknown: 14,609 entries
7. **Price_lakhs:** Price of the property in lakhs (1 lakh = 100,000 INR)

- numerical_features = df1[['bhk', 'area', 'price_lakhs']]
- categorical_features = df1[['type', 'region', 'status', 'age']]
- Summary Statistics of numerical features

	bhk	area	price_lakhs
count	76038.000000	76038.000000	76038.000000
mean	2.015111	1024.536850	168.417795
std	0.922754	670.276165	217.665511
min	1.000000	127.000000	4.490000
25%	1.000000	640.000000	64.000000
50%	2.000000	872.000000	110.000000
75%	3.000000	1179.000000	194.000000
max	10.000000	16000.000000	6000.000000

Data Cleaning and Feature Engineering

1. Checking for null/missing values

```
df1.isnull().sum()
#there is no missing data in the dataset
```

```
bhk            0
type           0
area           0
region         0
status         0
age            0
price_lakhs    0
dtype: int64
```

There were no null/missing values found in the dataset

2. Converting the price and price_unit to price_in_lakhs

The price column originally contained values in both lakhs and crores, so a conversion was necessary to ensure uniformity and consistency across the dataset. This adjustment ensures that all price values are expressed in a standardized unit.

```
#Conversion of price to lakhs
#df['price'].where(condition, other)
df['price_lakhs'] = df['price'].where(df['price_unit'] == 'L', df['price'] * 100)
df1 = df.drop(['price', 'price_unit'], axis=1)
df1.head()
```

	bhk	type	locality	area	region	status	age	price_lakhs
0	3	Apartment	Lak And Hanware The Residency Tower	685	Andheri West	Ready to move	New	250.00
1	2	Apartment	Radheya Sai Enclave Building No 2	640	Naigaon East	Under Construction	New	52.51
2	2	Apartment	Romell Serene	610	Borivali West	Under Construction	New	173.00
3	2	Apartment	Soundlines Codename Urban Rainforest	876	Panvel	Under Construction	New	59.98
4	2	Apartment	Origin Oriana	659	Mira Road East	Under Construction	New	94.11

3. Reducing the number of unique values in regions

   The regions that occurred less than 20 times in the dataset were changed to 'others'

```
region
Thane West          14868
Mira Road East       9902
Dombivali            3041
Kandivali East       2568
Kharghar             2362
                     ...
Police Colony           1
GTB Nagar               1
Bandra                  1
Sector 14 Vashi         1
Goregaon                1
Name: count, Length: 228, dtype: int64
```

Before

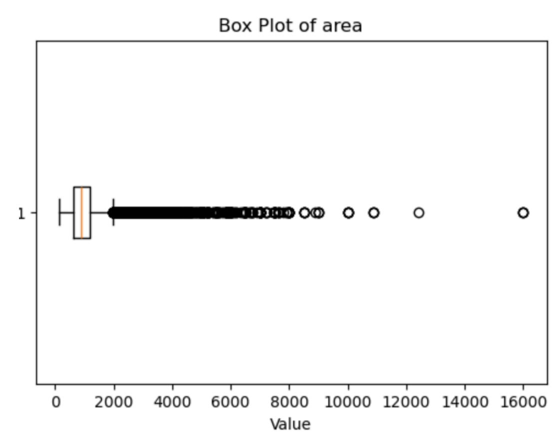
```
region
Thane West        14868
Mira Road East     9902
Dombivali          3041
Kandivali East     2568
Kharghar           2362
                   ...
Ambarnath            26
Umroli               25
Juinagar             24
Tardeo               23
Dombivali East       21
Name: count, Length: 104, dtype: int64
```

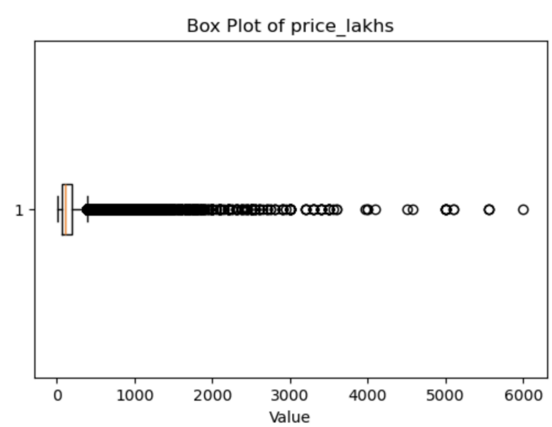
After

4. Outlier detection and removal from the dataset

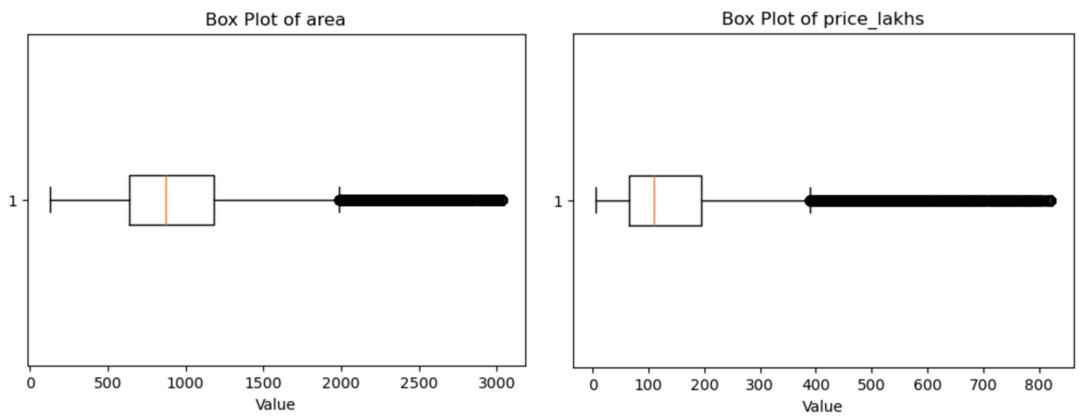
   Boxplot was used to detect outliers in the dataset

   Output



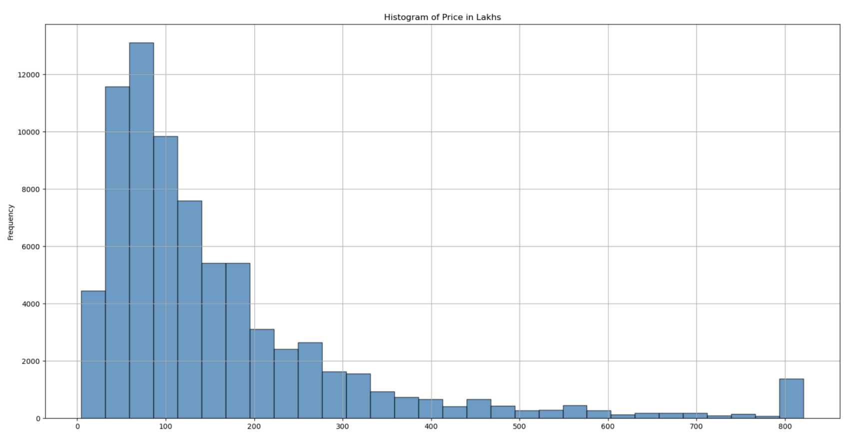


After Using z-score method to remove outliers

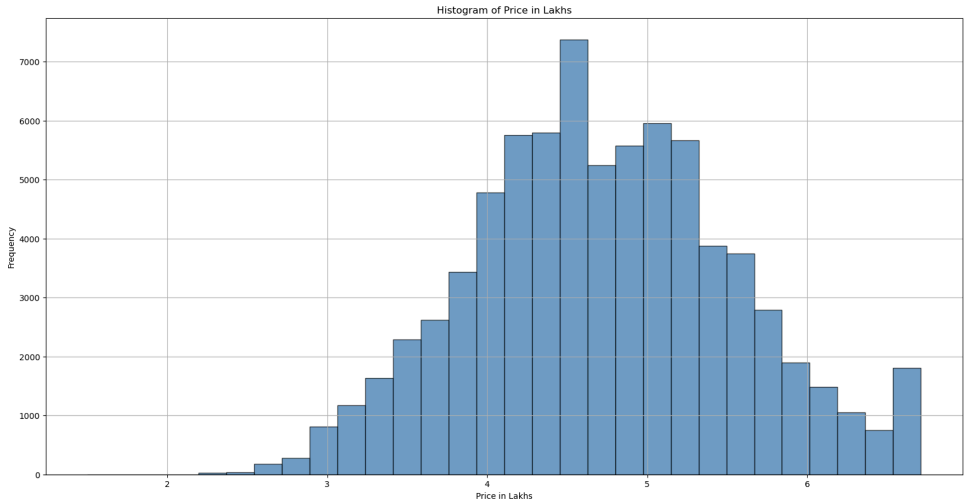


5. Log Transformation of Target Variable

A histogram was used to check whether the target variable is skewed or not



According to the above histogram the target variable was skewed so log transformation was applied on it



'Price in Lakhs' after log transform

6. Scaling of numerical features and Encoding of categorical features

Numerical features were scaled and categorical features were encoded using a preprocessor, optimizing the data preparation process for enhanced efficiency and consistency.

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bhk', 'area']),
        ('cat', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1), ['age', 'status']),
        ('ohe', OneHotEncoder(handle_unknown='ignore'), ['region', 'type'])
    ],
    remainder='passthrough'
)
```

Model 1: Support Vector Machines

Linear SVM is a variant of the SVM algorithm that uses a linear kernel to find the optimal hyperplane for classification or regression tasks. It works by maximizing the margin between classes in a linearly separable dataset or by fitting a linear function to the data for regression tasks. I chose Linear SVM for its simplicity and effectiveness in handling linearly separable datasets.

Model Configuration: The SVR (Support Vector Regression) model was selected with a linear kernel and regularization parameter C=10.

Pipeline Construction: A sklearn Pipeline was constructed to streamline the preprocessing steps and the SVM model application.

Training: The SVM model was trained on the training dataset (X_train and y_train) using the constructed pipeline.

Mean Absolute Error (MAE): The performance of the SVM model was evaluated using Mean Absolute Error (MAE), which measures the average absolute difference between the predicted and actual house prices.

Result:

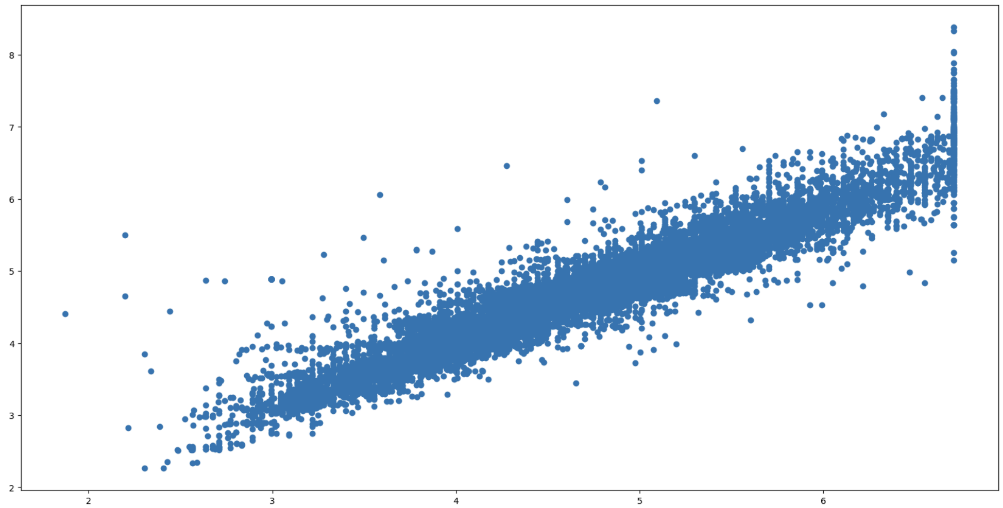
- Mean Absolute Error (Linear Support Vector Machine): 35.366730599529255

  (after reversing the log transformation)

Scatter plot between predicted y and y_train for linear svr

```
plt.scatter(y_test, y_pred_svr)
```

```
<matplotlib.collections.PathCollection at 0x175051f90>
```



Model 2: Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mean prediction of the individual trees for regression tasks. It leverages the strength of multiple decision trees to improve accuracy and generalizability. I chose Random Forest for its robustness to handle complex datasets, ability to capture non-linear relationships, and capability to handle both numerical and categorical features.

Model Configuration: A RandomForestRegressor model was chosen with 100 estimators and a random state of 10 for reproducibility.

Pipeline Construction: A sklearn Pipeline was built to streamline the preprocessing steps and the Random Forest model application.

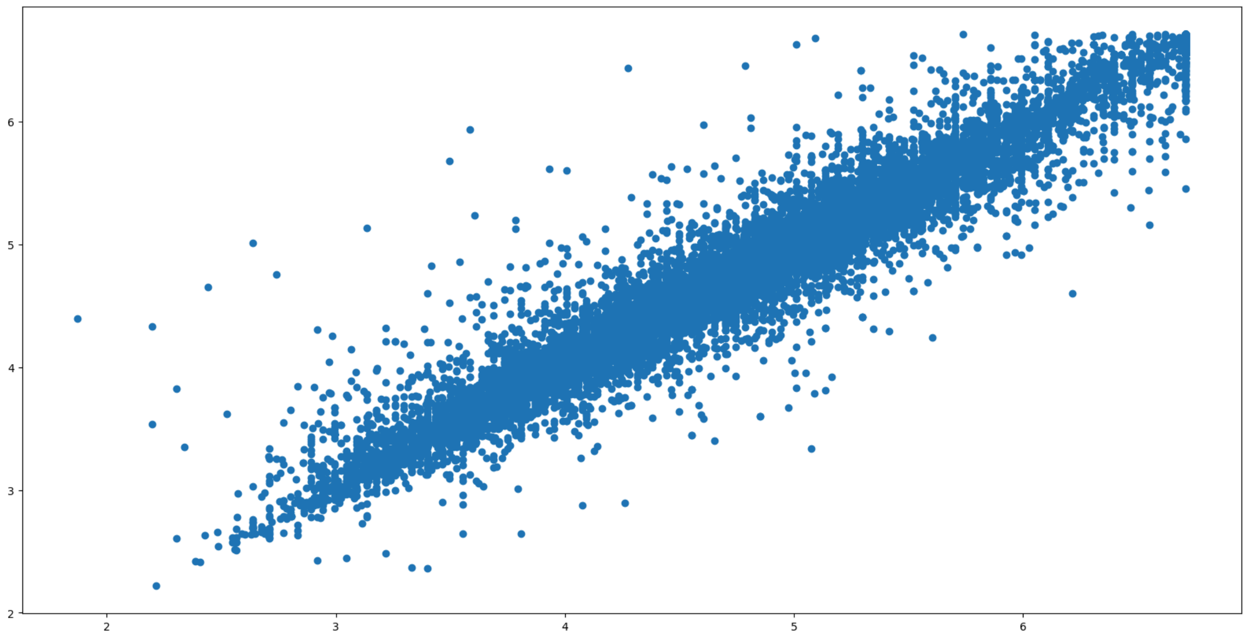
Training: The Random Forest model was trained on the training dataset (X_train and y_train) using the constructed pipeline.

Mean Absolute Error (MAE): The performance of the Random Forest model was evaluated using Mean Absolute Error (MAE), which measures the average absolute difference between the predicted and actual house prices.

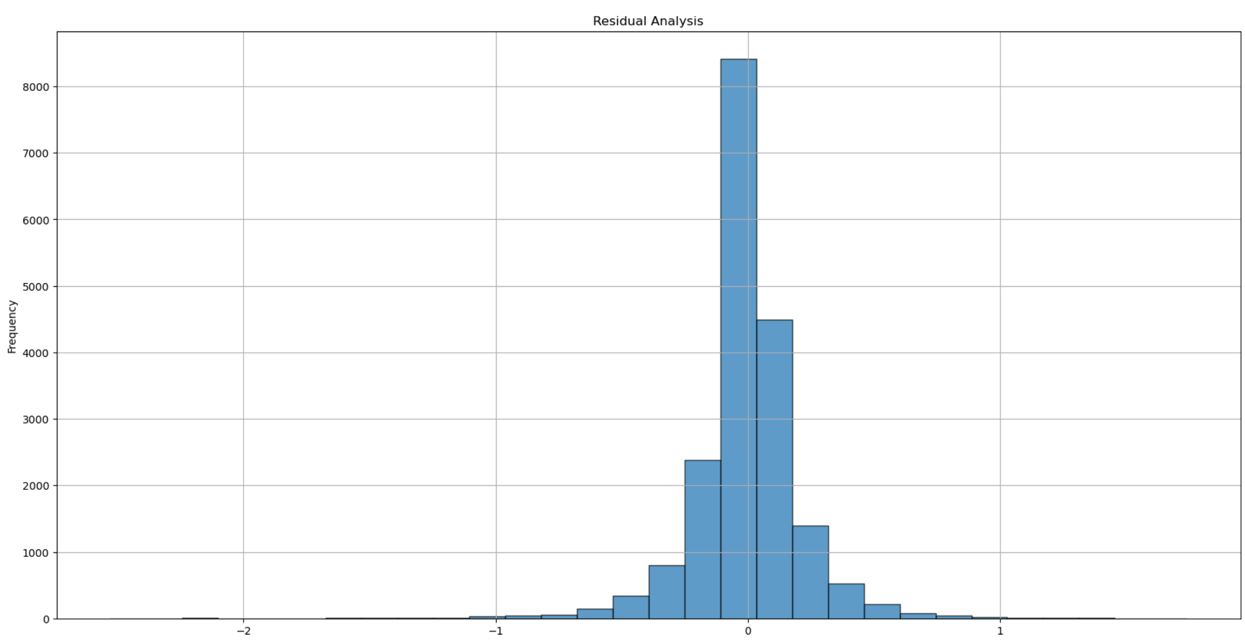
Result: Mean Absolute Error (Random Forest Regression): **21.527088769859404**

(after reversing the log transformation)

Scatter plot between predicted y and y_train for Random Forest



Residual Analysis for Random Forest using Histogram



Model 3: XGBoost

Model Configuration: XGBoost model was configured with specific parameters to optimize regression performance.

params = {"objective": "reg:squarederror", "tree_method": "hist"}

Training: The XGBoost model was trained on the training dataset (X_train_xg and y_train_xg) using the specified parameters.

model = xgb.train(params=params, dtrain=dtrain, num_boost_round=260, evals=evals, verbose_eval=10)

There was no need to scale or encode the categorical variables as xgboost takes care of it automatically. The verbose_eval=10 allowed the train-rmse and validation-rmse to be displayed after every ten boosting sessions. These errors were monitored for different num_boost_round values and it was found that after 260 rounds, validation-rmse started increasing so the num_boost_round was set to 260.

dtrain = xgb.DMatrix(X_train_xg, y_train_xg, enable_categorical=True)

dtest = xgb.DMatrix(X_test_xg, y_test_xg, enable_categorical=True)

The enable_categorical were set to True and the categorical variables were converted from object data type to category data type to ensure that there were no errors.

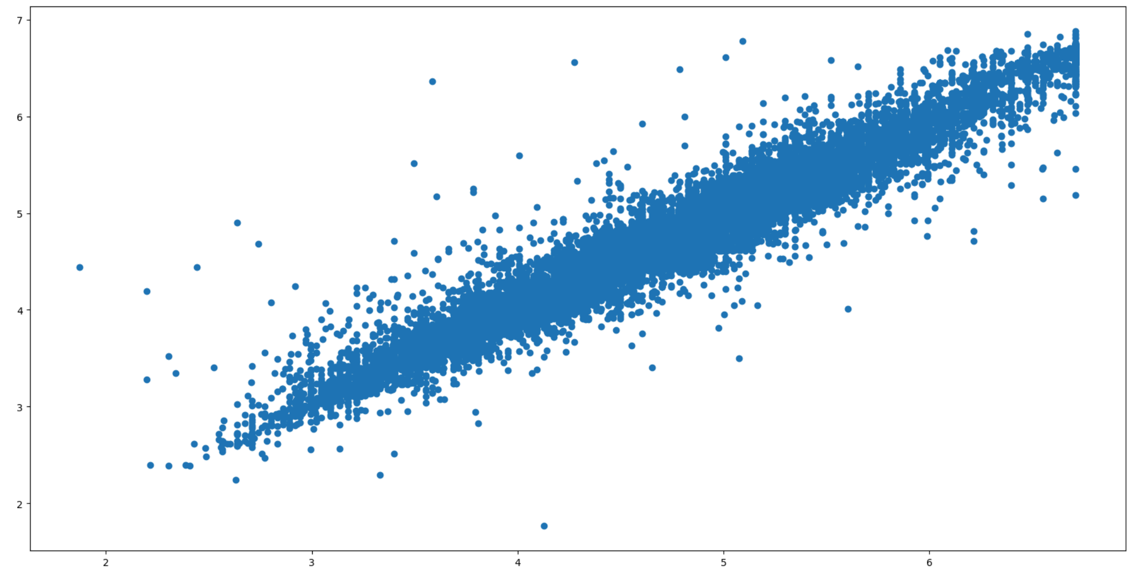
Prediction: Predictions were made on the test dataset (X_test_xg) using the trained XGBoost model.

Mean Absolute Error (MAE): The performance of the XGBoost model was evaluated using Mean Absolute Error (MAE), which measures the average absolute difference between the predicted and actual house prices.

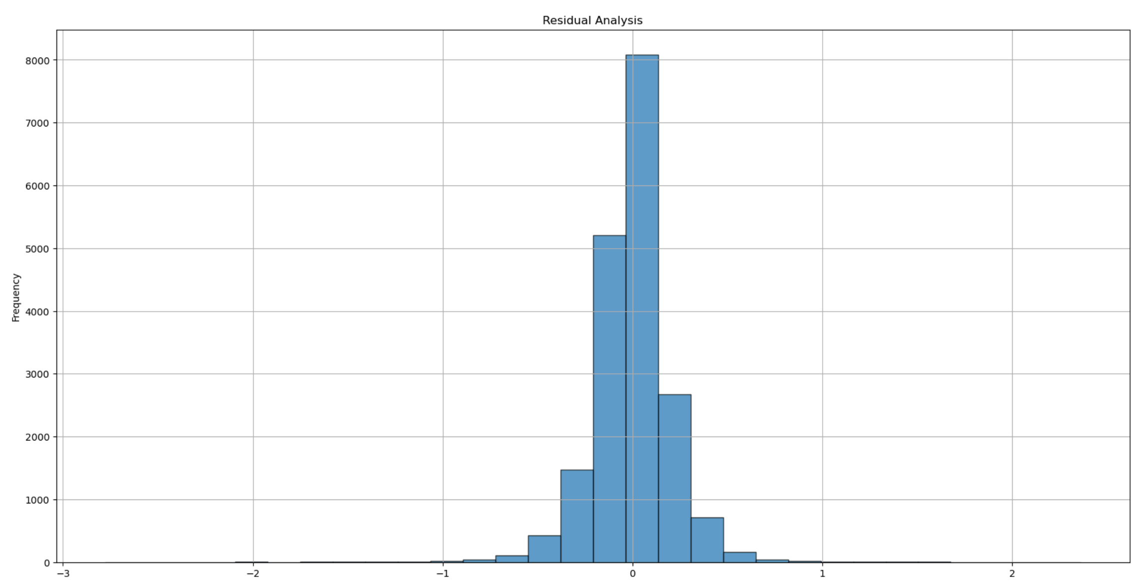
Result: Mean Absolute Error (XgBoost): **23.088281467824025** (after reversing the log transformation)

```
[0]	train-rmse:0.60283	validation-rmse:0.61016
[10]	train-rmse:0.21760	validation-rmse:0.22280
[20]	train-rmse:0.20739	validation-rmse:0.21654
[30]	train-rmse:0.20190	validation-rmse:0.21445
[40]	train-rmse:0.19659	validation-rmse:0.21219
[50]	train-rmse:0.19309	validation-rmse:0.21096
[60]	train-rmse:0.18971	validation-rmse:0.20975
[70]	train-rmse:0.18662	validation-rmse:0.20874
[80]	train-rmse:0.18403	validation-rmse:0.20805
[90]	train-rmse:0.18206	validation-rmse:0.20772
[100]	train-rmse:0.18051	validation-rmse:0.20728
[110]	train-rmse:0.17870	validation-rmse:0.20699
[120]	train-rmse:0.17716	validation-rmse:0.20683
[130]	train-rmse:0.17556	validation-rmse:0.20633
[140]	train-rmse:0.17401	validation-rmse:0.20618
[150]	train-rmse:0.17259	validation-rmse:0.20584
[160]	train-rmse:0.17122	validation-rmse:0.20559
[170]	train-rmse:0.17020	validation-rmse:0.20552
[180]	train-rmse:0.16922	validation-rmse:0.20554
[190]	train-rmse:0.16793	validation-rmse:0.20531
[200]	train-rmse:0.16681	validation-rmse:0.20516
[210]	train-rmse:0.16576	validation-rmse:0.20510
[220]	train-rmse:0.16480	validation-rmse:0.20513
[230]	train-rmse:0.16409	validation-rmse:0.20523
[240]	train-rmse:0.16310	validation-rmse:0.20520
[250]	train-rmse:0.16232	validation-rmse:0.20500
[259]	train-rmse:0.16173	validation-rmse:0.20496
Mean Absolute Error (XgBoost): 23.088281467824025
```

Scatter plot between predicted y and y_train for XGBoost



Residual Analysis for Random Forest using Histogram



By: Aarushi Ashish Gupta

Final Evaluation

Out of the three models, Linear SVM gave the highest mean absolute error of 35.366730599529255. However, the mae of Random Forest and XGBoost was in close proximity to each other at 21.527088769859404 for Random Forest and 23.088281467824025 for XGBoost. In both the models, the scatter plot is concentrated near the diagonal indicating that the model generally fits the data well. However, outliers in this plot indicate areas where the model may not perform as accurately, potentially due to unusual or unexpected data patterns. The histograms also indicate that the residuals are centered around zero, indicating that, on average, the model predictions are unbiased.

Another error metric Mean Absolute Percentage Error was used to evaluate the results from XGBoost and Random Forest since the two were similar

```python
# Calculate Mean Absolute Percentage Error (MAPE)
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# For Random Forest model
mape_rf = mean_absolute_percentage_error(y_test_original_rf, y_pred_original_rf)
print("Mean Absolute Percentage Error (Random Forest):", mape_rf)

# For XGBoost model
mape_xg = mean_absolute_percentage_error(y_test_original_xg, y_pred_original_xg)
print("Mean Absolute Percentage Error (XGBoost):", mape_xg)
```

```
Mean Absolute Percentage Error (Random Forest): 13.911179014566638
Mean Absolute Percentage Error (XGBoost): 14.673851754150297
```

Again the errors were in close proximity for the two models.

After evaluating all three models based on their mean absolute error, scatter plot analysis, residual plots, and MAPE, I concluded that Random Forest outperformed the others. Despite comparable error rates between Random Forest and XGBoost, Random Forest stood out due to its ease of interpretation and computational efficiency, especially on smaller datasets or those with fewer features. Moreover, its robustness in handling outliers further supported my decision to adopt it as the preferred model for this analysis.

Predictions

After training the model on the entire dataset, I proceeded to evaluate its efficiency by feeding it input arrays to assess its predictive performance.

```
#Making Predictions

input = [3,'Apartment',970,'Powai','Ready to move','Resale']
a = pd.DataFrame([input], columns=X.columns)
np.exp(pipeline.predict(a)[0])
```

```
262.13225862707884
```

```
input = [2,'Apartment',781,'Kandivali East','Under Construction','New']
a = pd.DataFrame([input], columns=X.columns)
np.exp(pipeline.predict(a)[0])
```

```
217.5030798046985
```

```
input = [3,'Apartment',1112,'Mulund West','Under Construction','New']
a = pd.DataFrame([input], columns=X.columns)
np.exp(pipeline.predict(a)[0])
```

```
258.69663403164446
```

The results were cross-checked against data from an online real estate platform, revealing predictions that closely aligned with actual market values. This validation shows that the model is reliable in estimating house prices based on real-world data.

Future Steps for Model Enhancement

I plan to conduct a thorough hyperparameter optimization using GridSearchCV for the random forest model. This includes fine-tuning parameters such as the number of estimators to further enhance predictive accuracy. Feature Engineering to make the dataset cleaner and provide valuable insights to improve model prediction accuracy could also be done.

Thank You

By: Aarushi Ashish Gupta