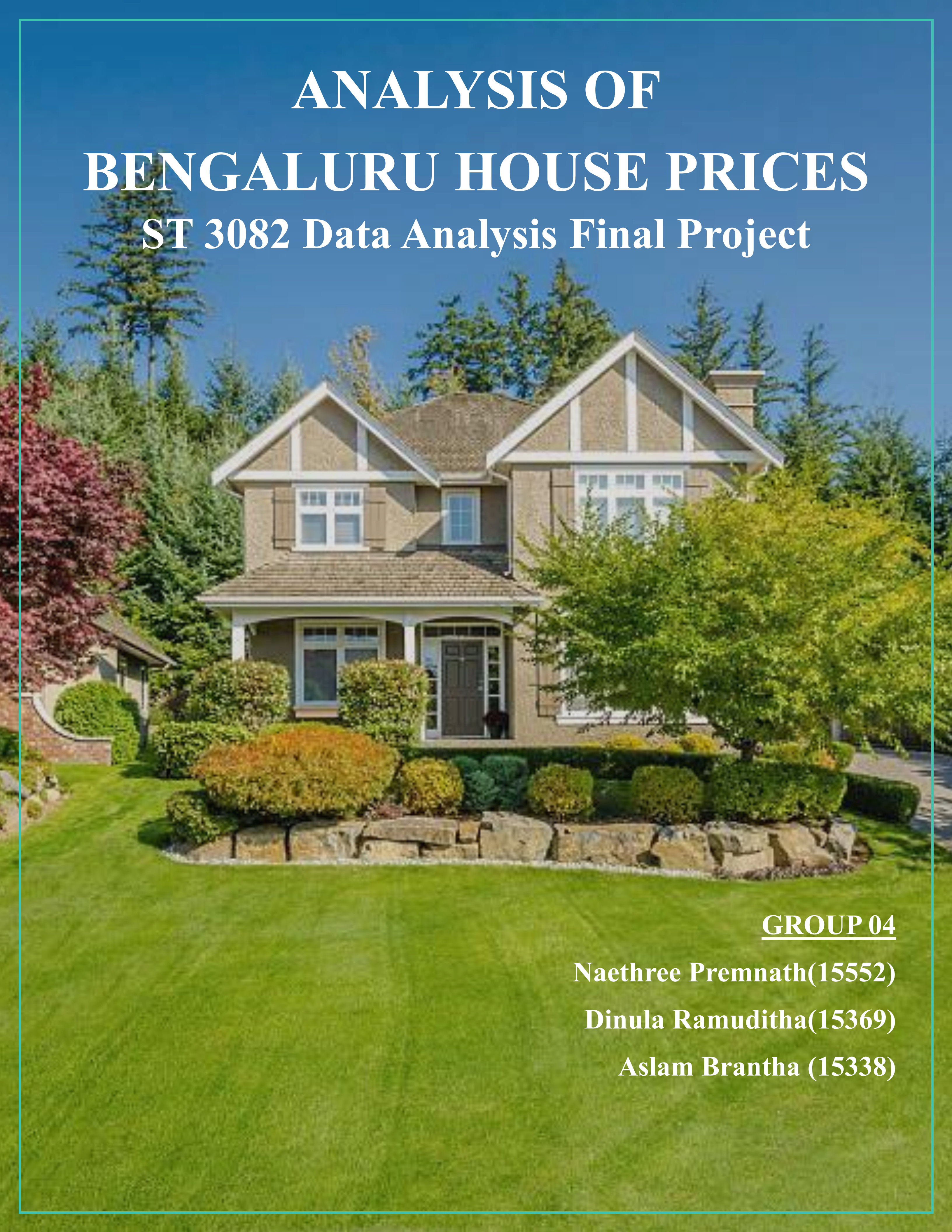
ANALYSIS OF BENGALURU HOUSE PRICES

ST 3082 Data Analysis Final Project



GROUP 04

Naethree Premnath(15552)

Dinula Ramuditha(15369)

Aslam Brantha (15338)

Abstract

This report aims to present the findings of the data analysis conducted on the Bengaluru House Prices dataset obtained from Kaggle. The primary objective of this analysis is to identify the factors that influence the house price and to predict the house price. A web application to predict house price and a dashboard where created at the end of the analysis. The insights derived from this analysis contribute to a deeper understanding of the factors influencing house price and facilitate the development of robust predictive models.

Table of Contents

List of Figures

List of Tables

1. Introduction

"Housing is a right, not a commodity"[1]

A house is one of the most basic necessities of life. As living standards improve, demand for housing increases rapidly. Although some people build houses for money and property, most people around the world buy a house as a shelter or to earn money. Every year, there is an increase in demand for housing, leading to an increase in housing prices. The issue arises when there are many variables such as location and the need for a building that may affect the value of the house; as a result, most stakeholders including buyers and developers, real estate developers, and the real estate industry would like to know the specific factors or factors that influence house prices to help investors make decisions and homeowners set house prices.

Understanding the factors influencing house prices greatly benefits home buyers by empowering them to make informed decisions about one of the most significant investments they'll ever make. With knowledge about what drives housing prices, buyers can assess the value of a property more accurately, ensuring they're not overpaying or underestimating its worth. Moreover, this understanding allows buyers to prioritize their preferences better, whether it's location, amenities, or property size, aligning their choices with their budget and long-term goals. Ultimately, being well-informed about house prices enables buyers to navigate the market confidently, securing a home that meets their needs and financial capabilities.

2. Description of the Question

This report aims to answer the following questions:

Q1) What are the factors that affect the price of a house?

Q2) How can the house prices be predicted based on the above factors?

3. Description of the Dataset

The Bengaluru House Price Prediction Dataset obtained from Kaggle contains 13320 observations with 9 variables. Description of each variable can be found in the table below:

Variable	Description	Type
Area_Type	Description of the area - “Built-Up Area”, “Carpet Area”, “Plot Area”, “Super built-up Area”	Categorical
Availability	When it can be possessed or when it is ready	Categorical
Location	Where it is located in Bengaluru	Categorical
Size	BHK(No.of Bedrooms)	Categorical
Society	To which society it belongs	Categorical
Total_sqft	Size of the property in square feets	Quantitative
Bath	Number of Bathrooms	Quantitative
Balcony	Number of Balconies	Quantitative
Price	Value of the property in lakhs (Indian Rupee)	Quantitative

Table 1

3.1. Data pre-processing

- 606 missing values were found in the dataset which were removed.
- 529 duplicate records were removed.
- ‘Society’,’Availability’,’Area_type’ variables were removed as they were not useful.
- ‘Total_sqft’ variables consisted of ranges which were converted to single numeric value which was the mean of the respective range. Furthermore, entries with units such as yards, grounds, sq_meters were converted to square feet.
- Trailing and leading white spaces were removed from the ‘Location’ variable. Moreover, all entries were converted to their lowercase forms. Also, since ‘Location’ contained 1244 unique locations, the locations with frequency<10 was combined to form ‘other’ category. Hence, the unique locations were reduced to 197. Finally, they were encoded using One Hot Encoding.

3.2. Feature Engineering

- A new variable 'Bedroom' was created by extracting only the numerical part of the 'size' variable. Then, the 'size' variable was removed.
- Feature scaling of all numeric variables to bring them to the same scale was performed before models were built.

3.3. Removal of outliers & unusual observations

- 'Price_per_sqft' was created and outliers were removed based on IQR. Then, 'price_per_sqft' variable was removed.
- 8 records where no.of bathrooms > bedrooms+2 were removed as they are assumed to be unusual based on the reference below.

  "General rule that realtors will follow, is the rule that there should be one less bathroom than there are bedrooms."[2]
- 523 records were removed by maintaining a minimum threshold of 300 square feet per bedroom as per the reference below.

  "Standard size of 2 BHK flat ranges between 650 - 800 square feet"[3]

4. Important results of the Descriptive Analysis

- The Price variable is skewed by Figure 1.
- Price and Total_sqft had a positive relationship among them by Figure 2.
- Price and Bedroom, Price and Bathroom, Price and Balcony seemed not have much relationships between them by Figures 3,4,5.

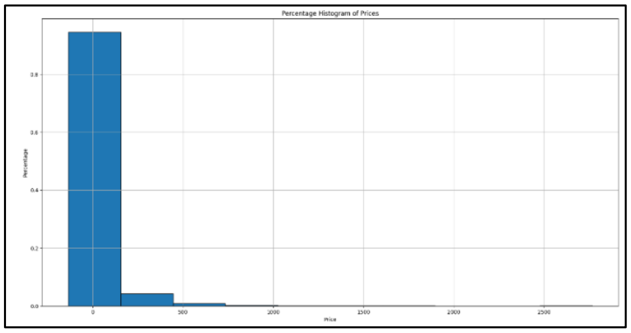


Figure 1

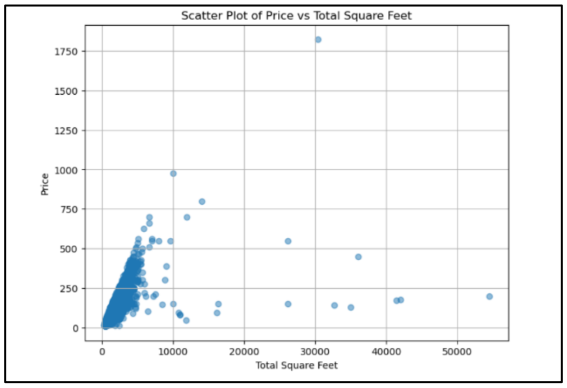


Figure 2

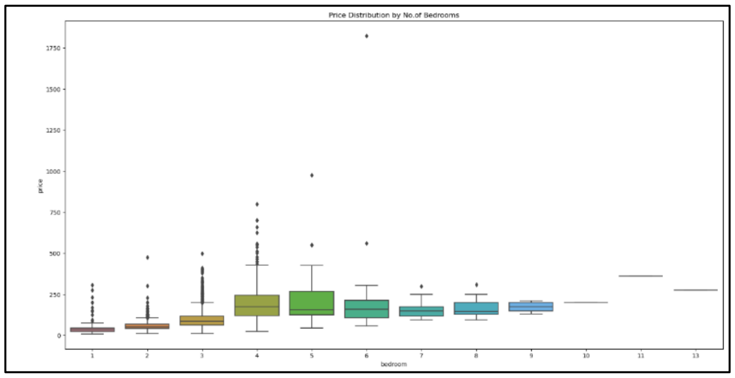


Figure 3

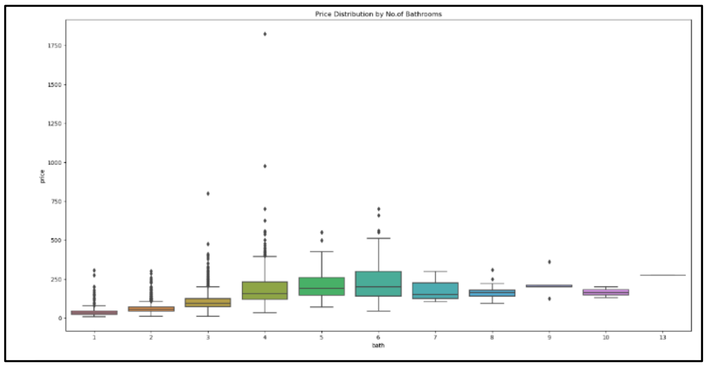


Figure 4

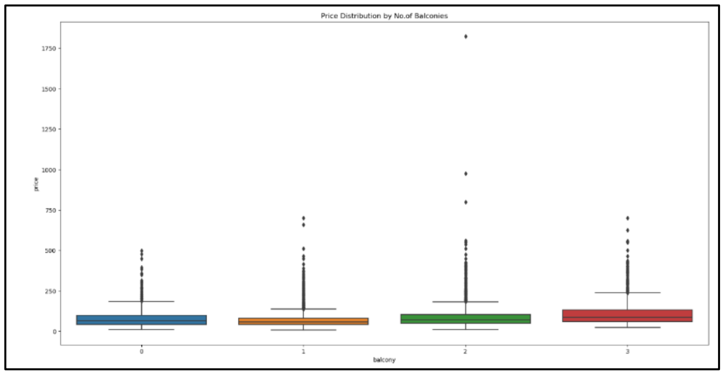


Figure 5

- ➢ There was high correlation between Bedroom and Bath variables by Figure 6.
- ➢ Evidence of Multicollinearity also can be seen by the Table 2.

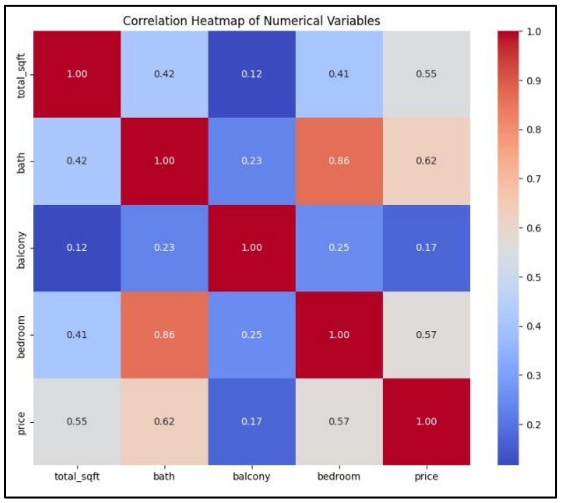


Figure 6

Variable	**VIF Values**
Total_sqft	3.152718
Bath	32.773512
Balcony	4.449669
Bed	32.086454

Table 2

5. Advanced Analysis

✓ **Dealing with outliers:**

The local outlier factor method has been used to deal with multivariate outliers. 847 outliers have been removed from the dataset and fitted the model. But for further analysis the models were fitted with outliers and without outliers as well.

✓ **Transformation on response variable Price:**

Price is skewed as seen earlier. To address this, a log transformation was applied to normalize the response variable, leading to improvements in the models.

- ❖ **Because multicollinearity appears among some variables, algorithms like Ridge, Lasso, Elastic Net and PLS regression were first considered. Since PLS Regression deals with multicollinearity while reducing the dimensions, it was first considered as the baseline model.**
- ❖ **PLS Regression**

Best parameter for PLS: {n_components=2}

This model performed better with outliers. So, the following results are with the outliers.

2 components explained significant variance and hence model with 2 components was fit.

	Train	**Test**
MSE	0.1230	0.2307
RMSE	0.3507	0.4803
MAE	0.2581	0.3334
R^2	0.6297	0.5051

Table 3

- ❖ **Ridge Regression**

Best parameter: {'alpha': 10}

Ridge Regression was chosen since it deals with multicollinearity and prevents overfitting. Outliers were not removed since the model performed better with the outliers.

	Train	**Test**
MSE	0.1229	0.2309
RMSE	0.3506	0.4805
MAE	0.2587	0.3342
R^2	0.6299	0.5046

Table 4

- ❖ **Lasso Regression**

Best parameter: {'alpha': 0.1}

Outliers were removed since the model performed better without the outliers.

	Train	**Test**
MSE	0.1819	0.3028
RMSE	0.4265	0.5502

MAE	0.3316	0.4054
R^2	0.4772	0.3504

Table 5

❖ Elastic Net

Best parameters: {'alpha': 0.1, 'l1_ratio': 0.1}

Elastic Net Regression was chosen since it is a compromise between Ridge and Lasso and we were interested to see how the model performed. Outliers were removed from this model since the model performed better after removing them. LOF(Local Outlier Factor) method was used to find the multivariate outliers and hence 847 outliers were removed before fitting the Elastic Net model.

	Train	**Test**
MSE	0.1684	0.2827
RMSE	0.4104	0.5317
MAE	0.3150	0.3884
R^2	0.5160	0.3934

Table 6

❖ Random Forest

Best parameters: {'max_depth': 5, 'n_estimators': 200}

This model also have a good test MSE with compares to other models. This model is performed better after removing the outliers.

	Train	**Test**
MSE	0.0784	0.1971
RMSE	0.2799	0.4440
MAE	0.2222	0.3163
R^2	0.7775	0.5770

Table 7

❖ XG Boost

Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}

	Train	**Test**
MSE	0.0493	0.1684

RMSE	0.221	0.4104
MAE	0.1684	0.2820
R^2	0.8515	0.6387

Table 8

6. Best Model

By considering the above Train Test MSE values, MAPE values and the R-squared values, it was concluded that the XG boost performs best in predicting the house prices. Table 9 shows part of the feature importance values obtained for the XGBoost model. Total_sqft and Location are the most important variables in predicting house price

Although, this model is considered as the best model, the overfitted nature of the model is not highly recommended. Since balcony, bedroom, bathroom doesn't play a huge role in prediction it was considered to remove them and fit the model. However, according to the reference ***"The main features a homebuyer considers are size of plot, number of bedrooms, number of bathrooms, number of floors, balcony facility and other facilities included gas and water facility etc"[4] ,*** it was decided not to remove the variables. Hence all the features were considered for the best model.

Feature	Feature Importance Value
Total_sqft	0.0482
btm layout	0.0278
electronic city phase	0.0236
begur	0.0186
chandapura	0.0176
sarjapur	0.0162
koramangala	0.0153
basavangudi	0.0141
raja rajeshwari	0.0138

Table 9

According to the Partial Dependence Plots given by Figure 7, it is observed that as total_sqft increases keeping other variables constant, the predicted log price of house increases, suggesting a positive partial dependency. On the contrary, bath, bedroom, balcony doesn't show much. However, as per the reference above, it was decided not to remove those variables.

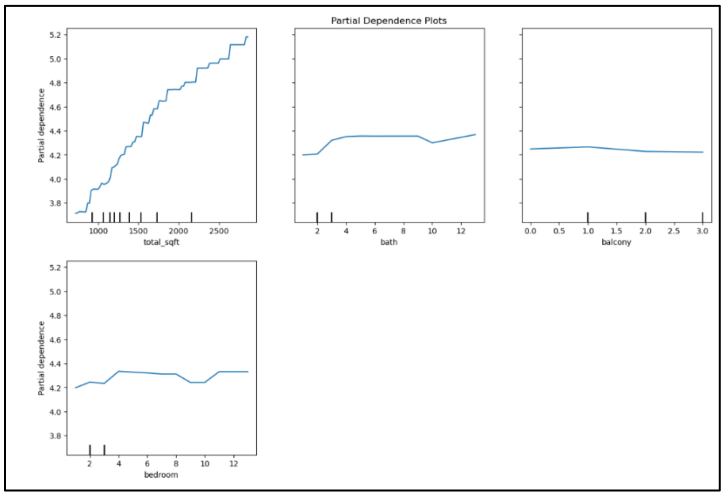


Figure 7

7. Issues encountered and proposed solutions

- Due to persistent multicollinearity it was not feasible to fit models like Multiple Linear Regression. If we attempted to remove variables such as Bedroom and Bathroom, it would result in information loss. Removing variables from a dataset which has only a few features isn't the best solution and hence models which can be affected by Multicollinearity were not considered.
- Univariate boxplots weren't considered to remove outliers since considering multivariate outliers is more appropriate. Hence LOF outlier removal method was used to remove 847 outliers. There could be other methods which could have resulted in better models, however since our final model XGBoost Model performs well without removing outliers(since it is robust to outliers) this issue doesn't affect the final chosen model.
- Due to the lack of information of the year of the data collection, the variable 'availability' couldn't be of use.
- 'Society' variable seemed to be unusual suggesting it may have been encoded. Since our objective revolved around building a product for the user, it doesn't make sense to keep this variable since information about the encoding was not given. Furthermore, considering that 5502 of them were missing values, it was decided to remove the variable.
- 'Area type' variable was removed as per our reference ***"Buyers must take heed that the price of an apartment is based on the carpet area and not the super built-up area or other types"[5]***, only 'carpet area' is considered by homebuyers.
- Overfitted nature can be observed with all of the models. Removing all variables and keeping just total_sqft would result in a non overfitted model. However removing bedroom, bath, balcony wasn't supported by our references as discussed above. Hence it was decided to keep the features just as they are. For future analysis, readers can try to find methods to deal with the overfitted nature.

8. Discussions and Conclusions

The results obtained by Advanced Analysis can be summarized as follows:

	Train MSE	**Test MSE**	**Train R^2**	**Test R^2**
Ridge	0.1229	0.2309	0.6299	0.5046
Lasso	0.1819	0.3028	0.4772	0.3504
Elastic Net	0.1684	0.2827	0.5160	0.3934
PLS	0.1231	0.2279	0.6297	0.5051

Random Forest	0.0784	0.1971	0.7775	0.5770
XGBoost	0.0493	0.1684	0.8515	0.6387

Table 10

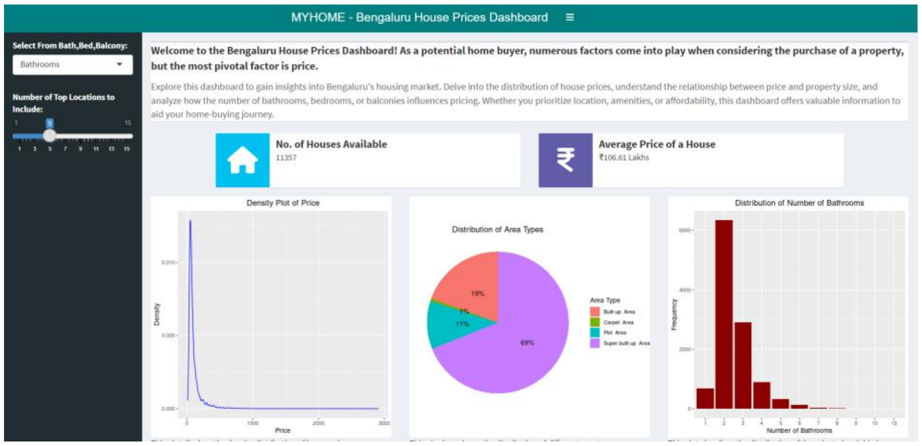
Hence, the XGBoost model was chosen as the best model with parameters *{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}*.

9. Product – MYHOME

Figure 8 shows the dashboard that was created using RShiny. Interactive features were added so that A user can visualize home prices in Bengaluru against factors affecting it.

Furthermore by Figure 9, the house Price prediction app can be seen in Which users are able to insert their Requirements, and within seconds the Home price is predicted.

A web application named MYHOME was created joining both in order to p provide a seamless experience for the user.


MYHOME - Bengaluru House Prices Dashboard

Select From Bath,Bed,Balcony: Bathrooms

Number of Top Locations to Include: 5

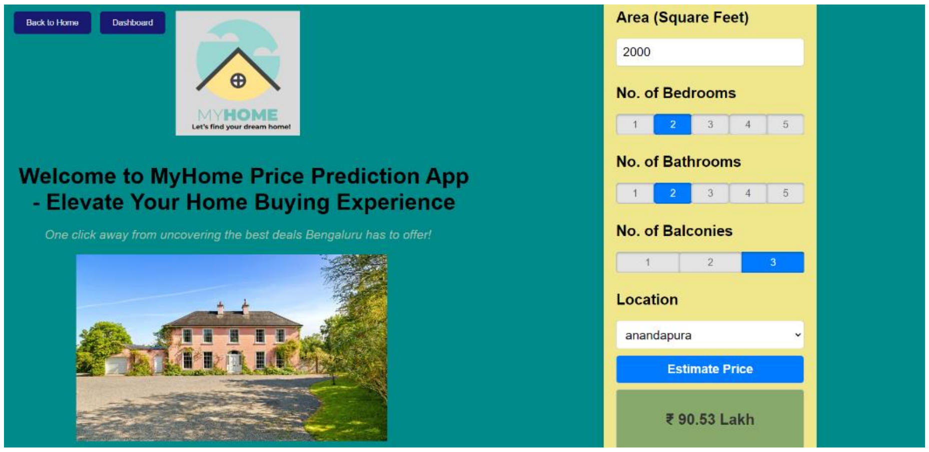
Welcome to the Bengaluru House Prices Dashboard! As a potential home buyer, numerous factors come into play when considering the purchase of a property, but the most pivotal factor is price.

Explore this dashboard to gain insights into Bengaluru's housing market. Delve into the distribution of house prices, understand the relationship between price and property size, and analyze how the number of bathrooms, bedrooms, or balconies influences pricing. Whether you prioritize location, amenities, or affordability, this dashboard offers valuable information to aid your home-buying journey.

No. of Houses Available: 11357

Average Price of a House: ₹106.61 Lakhs

Figure 8



Welcome to MyHome Price Prediction App - Elevate Your Home Buying Experience

One click away from uncovering the best deals Bengaluru has to offer!

Area (Square Feet): 2000

No. of Bedrooms: 2

No. of Bathrooms: 2

No. of Balconies: 3

Location: anandapura

₹ 90.53 Lakh

Figure 9

10. References

Dataset: https://www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data

[1] https://www.ohchr.org/en/special-procedures/sr-housing/human-right-adequate-housing

[2] https://www.modobath.com/inspiration/the-ideal-number-of-bathrooms-you-need-to-maximize-home-value/

[3] https://www.nobroker.in/forum/how-many-square-feet-are-needed-for-a-spacious-2bhk/

[4] http://ir.kdu.ac.lk/bitstream/handle/345/5247/61.pdf?sequence=1&isAllowed=y

[5] https://www.brigadegroup.com/blog/residential/all-you-need-to-know-about-carpet-area-built-up-area-and-super-built-up#:~:text=Buyers%20must%20take%20heed%20that,thickness%20of%20the%20inner%20walls.

[6]https://ijcrt.org/papers/IJCRT2305063.pdf

[7]https://ymerdigital.com/uploads/YMER211193.pdf

[8]https://www.kaggle.com/code/iamsouravbanerjee/there-s-no-place-like-home

[9]https://devmesh.intel.com/projects/bengaluru-house-prediction#about-section

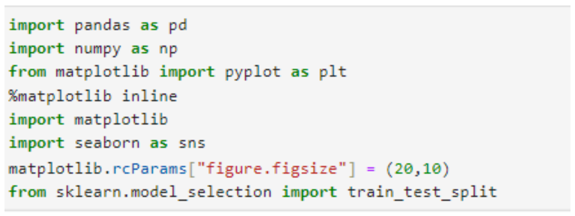
[10]https://www.researchgate.net/publication/354403038_Bangalore_House_Price_Prediction

[11]https://arxiv.org/ftp/arxiv/papers/1904/1904.05328.pdf

[12]https://www.news18.com/business/bengaluru-bangalore-real-estate-market-property-prices-8591495.html

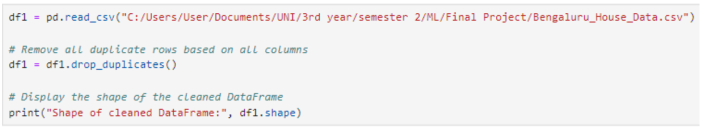
11.Appendix

1



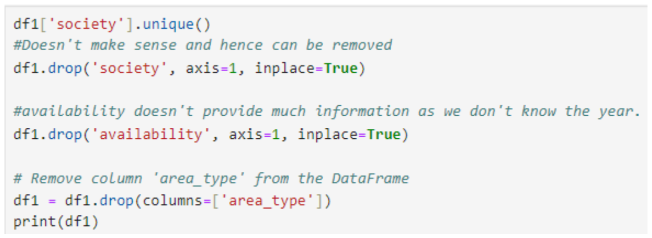
```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
import seaborn as sns
matplotlib.rcParams["figure.figsize"] = (20,10)
from sklearn.model_selection import train_test_split
```

2



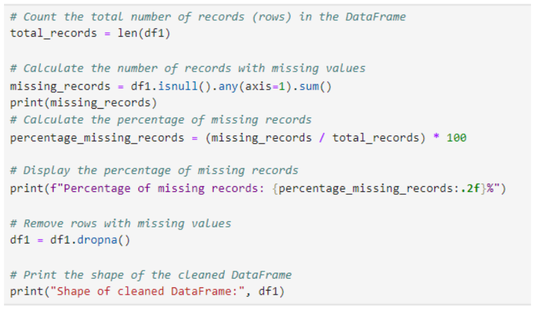
```python
df1 = pd.read_csv("C:/Users/User/Documents/UNI/3rd year/semester 2/ML/Final Project/Bengaluru_House_Data.csv")

# Remove all duplicate rows based on all columns
df1 = df1.drop_duplicates()

# Display the shape of the cleaned DataFrame
print("Shape of cleaned DataFrame:", df1.shape)
```

3



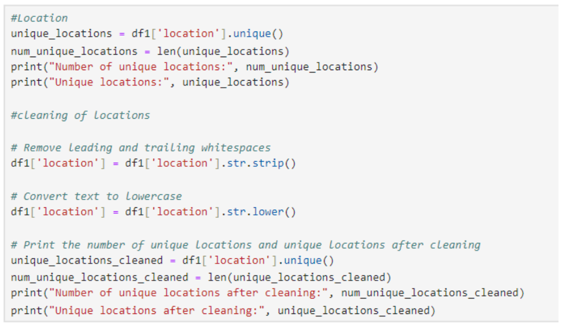
```python
df1['society'].unique()
#Doesn't make sense and hence can be removed
df1.drop('society', axis=1, inplace=True)

#availability doesn't provide much information as we don't know the year.
df1.drop('availability', axis=1, inplace=True)

# Remove column 'area_type' from the DataFrame
df1 = df1.drop(columns=['area_type'])
print(df1)
```

4



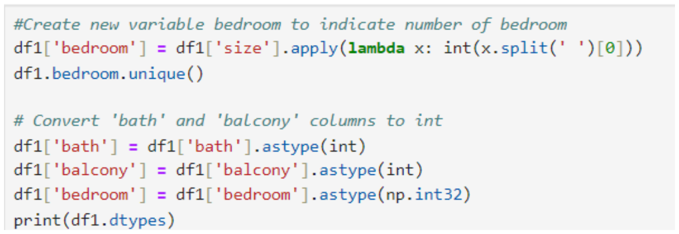
```python
# Count the total number of records (rows) in the DataFrame
total_records = len(df1)

# Calculate the number of records with missing values
missing_records = df1.isnull().any(axis=1).sum()
print(missing_records)
# Calculate the percentage of missing records
percentage_missing_records = (missing_records / total_records) * 100

# Display the percentage of missing records
print(f"Percentage of missing records: {percentage_missing_records:.2f}%")

# Remove rows with missing values
df1 = df1.dropna()

# Print the shape of the cleaned DataFrame
print("Shape of cleaned DataFrame:", df1)
```

5



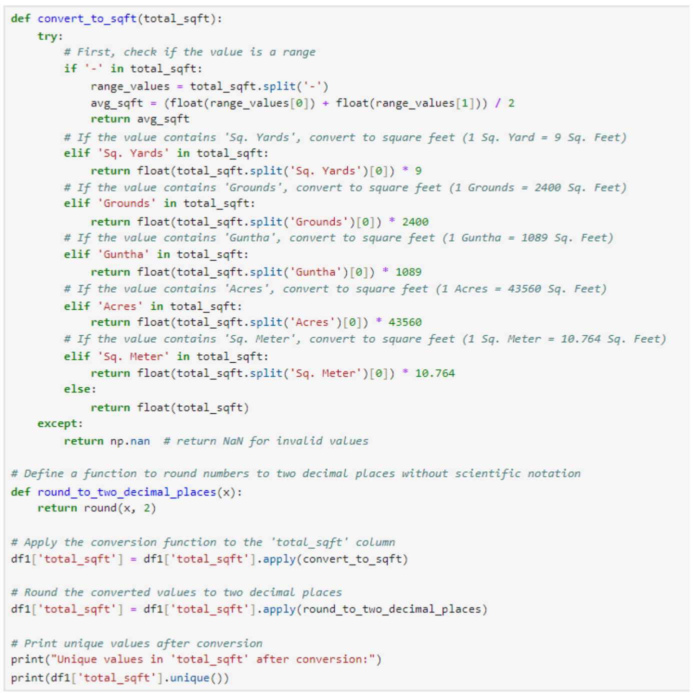
```python
#Location
unique_locations = df1['location'].unique()
num_unique_locations = len(unique_locations)
print("Number of unique locations:", num_unique_locations)
print("Unique locations:", unique_locations)

#cleaning of locations

# Remove leading and trailing whitespaces
df1['location'] = df1['location'].str.strip()

# Convert text to lowercase
df1['location'] = df1['location'].str.lower()

# Print the number of unique locations and unique locations after cleaning
unique_locations_cleaned = df1['location'].unique()
num_unique_locations_cleaned = len(unique_locations_cleaned)
print("Number of unique locations after cleaning:", num_unique_locations_cleaned)
print("Unique locations after cleaning:", unique_locations_cleaned)
```

6



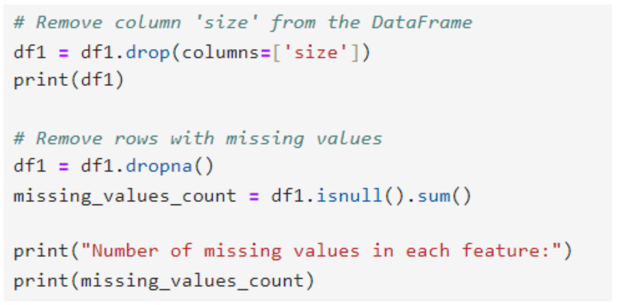
```python
#Create new variable bedroom to indicate number of bedroom
df1['bedroom'] = df1['size'].apply(lambda x: int(x.split(' ')[0]))
df1.bedroom.unique()

# Convert 'bath' and 'balcony' columns to int
df1['bath'] = df1['bath'].astype(int)
df1['balcony'] = df1['balcony'].astype(int)
df1['bedroom'] = df1['bedroom'].astype(np.int32)
print(df1.dtypes)
```

7



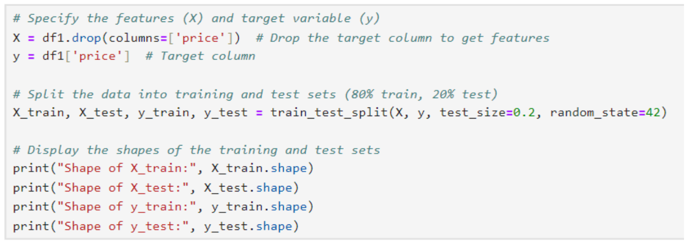
```python
def convert_to_sqft(total_sqft):
    try:
        # First, check if the value is a range
        if '-' in total_sqft:
            range_values = total_sqft.split('-')
            avg_sqft = (float(range_values[0]) + float(range_values[1])) / 2
            return avg_sqft
        # If the value contains 'Sq. Yards', convert to square feet (1 Sq. Yard = 9 Sq. Feet)
        elif 'Sq. Yards' in total_sqft:
            return float(total_sqft.split('Sq. Yards')[0]) * 9
        # If the value contains 'Grounds', convert to square feet (1 Grounds = 2400 Sq. Feet)
        elif 'Grounds' in total_sqft:
            return float(total_sqft.split('Grounds')[0]) * 2400
        # If the value contains 'Guntha', convert to square feet (1 Guntha = 1089 Sq. Feet)
        elif 'Guntha' in total_sqft:
            return float(total_sqft.split('Guntha')[0]) * 1089
        # If the value contains 'Acres', convert to square feet (1 Acres = 43560 Sq. Feet)
        elif 'Acres' in total_sqft:
            return float(total_sqft.split('Acres')[0]) * 43560
        # If the value contains 'Sq. Meter', convert to square feet (1 Sq. Meter = 10.764 Sq. Feet)
        elif 'Sq. Meter' in total_sqft:
            return float(total_sqft.split('Sq. Meter')[0]) * 10.764
        else:
            return float(total_sqft)
    except:
        return np.nan  # return NaN for invalid values

# Define a function to round numbers to two decimal places without scientific notation
def round_to_two_decimal_places(x):
    return round(x, 2)

# Apply the conversion function to the 'total_sqft' column
df1['total_sqft'] = df1['total_sqft'].apply(convert_to_sqft)

# Round the converted values to two decimal places
df1['total_sqft'] = df1['total_sqft'].apply(round_to_two_decimal_places)

# Print unique values after conversion
print("Unique values in 'total_sqft' after conversion:")
print(df1['total_sqft'].unique())
```

8



```python
# Remove column 'size' from the DataFrame
df1 = df1.drop(columns=['size'])
print(df1)

# Remove rows with missing values
df1 = df1.dropna()
missing_values_count = df1.isnull().sum()

print("Number of missing values in each feature:")
print(missing_values_count)
```

9



```python
# Specify the features (X) and target variable (y)
X = df1.drop(columns=['price'])  # Drop the target column to get features
y = df1['price']  # Target column

# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the shapes of the training and test sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

10

```
location_stats = X_train['location'].value_counts(ascending=False)
location_stats

len(location_stats[location_stats<=10])
location_stats_less_than_10 = location_stats[location_stats<=10]
print(location_stats_less_than_10)

X_train.location = X_train.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(X_train.location.unique())
```

11

```
# Find locations in test data that are not present in training data
test_locations_not_in_train = set(X_test['location']) - set(X_train['location'])

# Identify locations in test data that are in the 'other' category in training data
test_locations_to_other = set(X_test['location']) - set(location_stats.index)

# Update those locations in the test data to 'other'
X_test.loc[X_test['location'].isin(test_locations_not_in_train.union(test_locations_to_other)), 'location'] = 'other'

# Check the unique locations in the updated test data
print("Unique locations in updated test data:", len(X_test['location'].unique()))
```

12

```
# Concatenate training and test data to ensure consistent encoding
combined_data = pd.concat([X_train, X_test])

# Perform one-hot encoding on the combined 'location' column
combined_data_encoded = pd.get_dummies(combined_data, columns=['location'], drop_first=True)

# Remove 'location_' from column names
combined_data_encoded.columns = combined_data_encoded.columns.str.replace('location_', '')

# Split the combined data back into training and test sets
X_train_encoded = combined_data_encoded.iloc[:len(X_train)]
X_test_encoded = combined_data_encoded.iloc[len(X_train):]
```

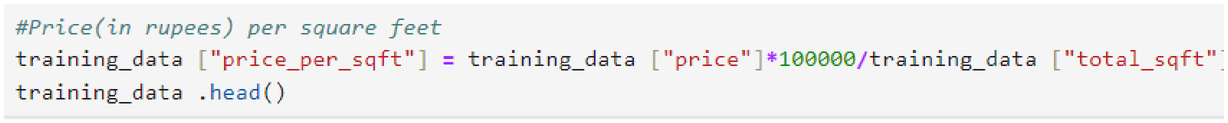
13

```
#It is unusual to have 2 more bathrooms than number of bedrooms in a home. So we are discarding that also.
#8 such records can be removed
# Concatenate training_X and training_y into a single DataFrame
training_data = pd.concat([X_train_encoded, y_train], axis=1)

training_data =training_data.drop(training_data[training_data.bath > training_data.bedroom + 2].index)
```

14

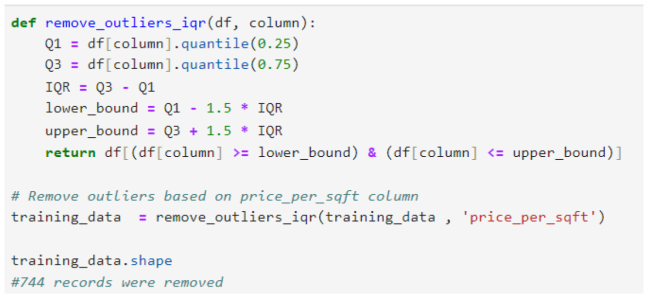
```
#normally if a square ft per bedroom is 300 (i.e. 2 bhk apartment is minimum 600 sqft.
#If you have for example 400 sqft apartment with 2 bhk than that seems suspicious and can be removed as an outlier.
#We will remove such outliers by keeping our minimum thresold per bhk to be 300 sqft
#523 records are removed
training_data = training_data [~(training_data .total_sqft/training_data .bedroom < 300)]
```

15



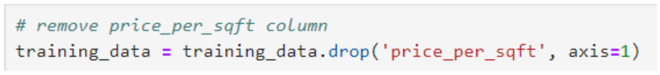
```python
#Price(in rupees) per square feet
training_data ["price_per_sqft"] = training_data ["price"]*100000/training_data ["total_sqft"]
training_data .head()
```

16



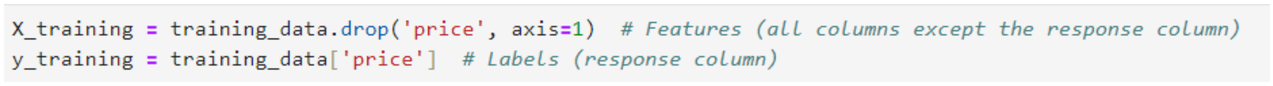
```python
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Remove outliers based on price_per_sqft column
training_data  = remove_outliers_iqr(training_data , 'price_per_sqft')

training_data.shape
#744 records were removed
```

17



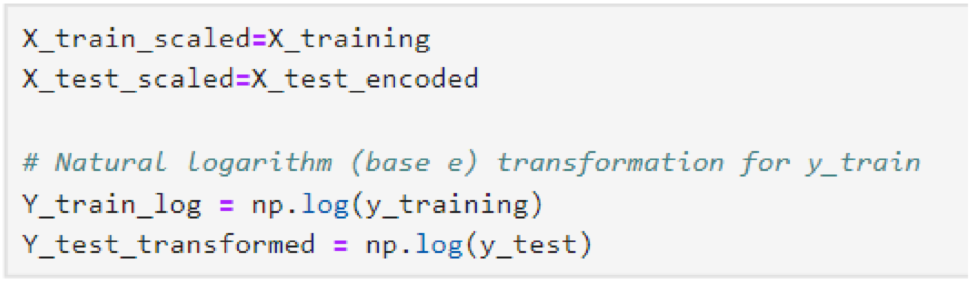
```python
# remove price_per_sqft column
training_data = training_data.drop('price_per_sqft', axis=1)
```

18



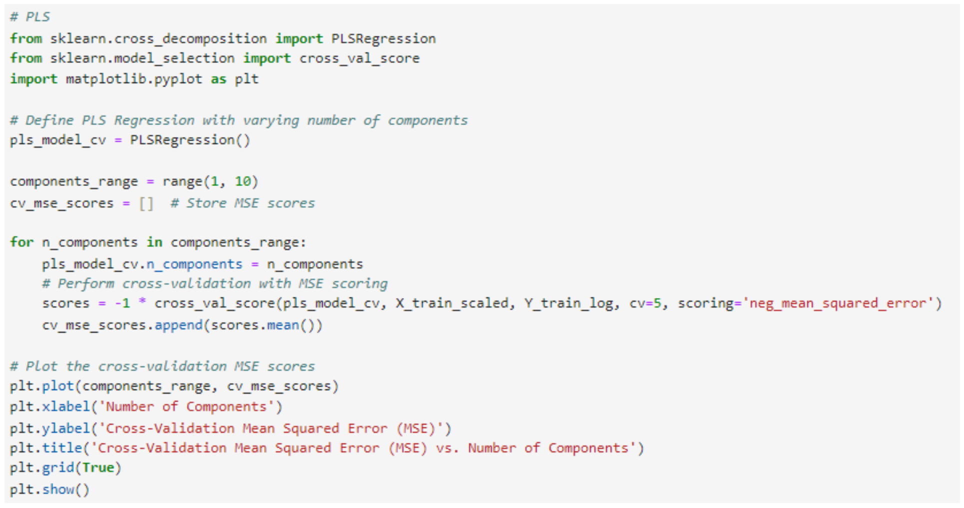
```python
X_training = training_data.drop('price', axis=1)  # Features (all columns except the response column)
y_training = training_data['price']  # Labels (response column)
```

19



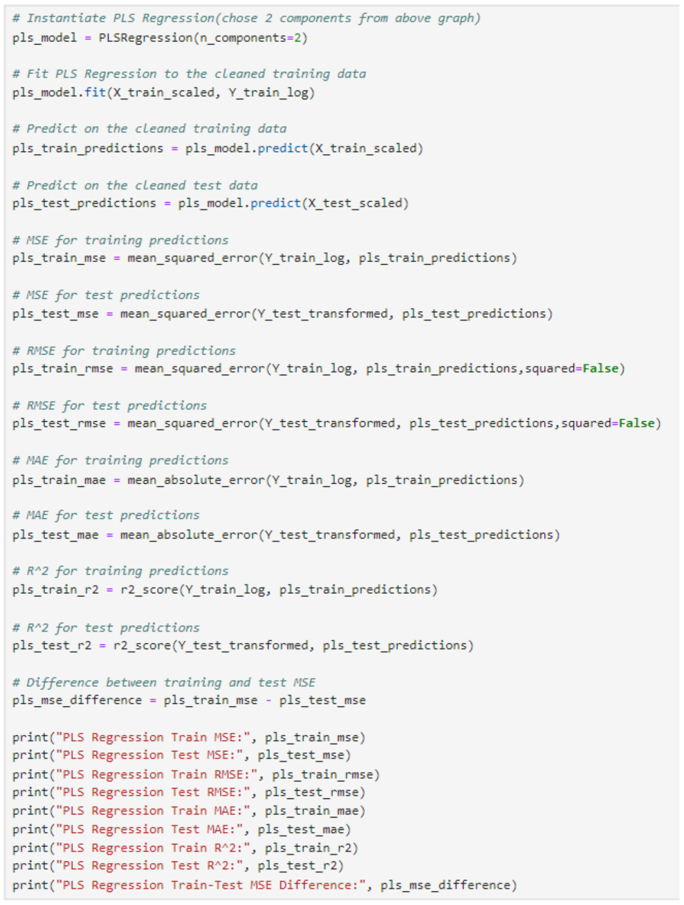
```python
X_train_scaled=X_training
X_test_scaled=X_test_encoded

# Natural logarithm (base e) transformation for y_train
Y_train_log = np.log(y_training)
Y_test_transformed = np.log(y_test)
```

20 PLS



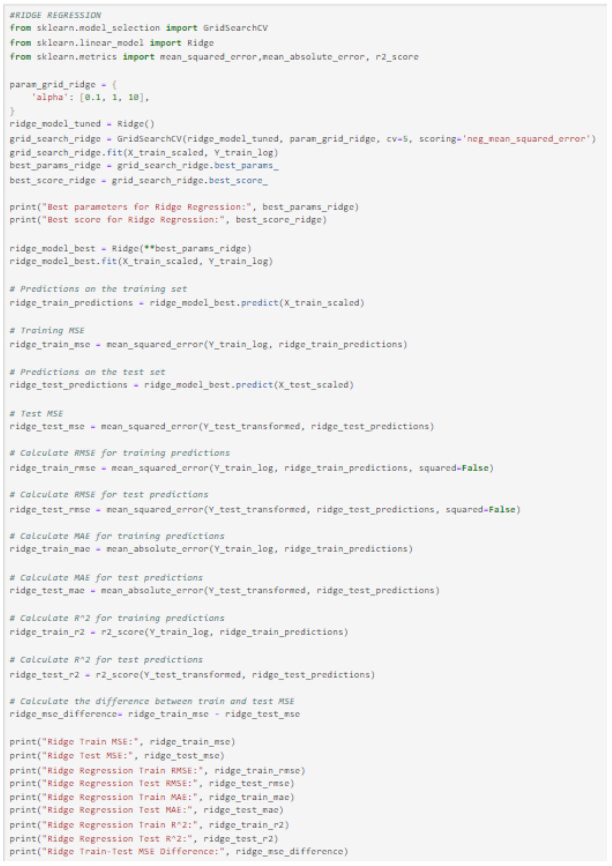
```python
# PLS
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt

# Define PLS Regression with varying number of components
pls_model_cv = PLSRegression()

components_range = range(1, 10)
cv_mse_scores = []  # Store MSE scores

for n_components in components_range:
    pls_model_cv.n_components = n_components
    # Perform cross-validation with MSE scoring
    scores = -1 * cross_val_score(pls_model_cv, X_train_scaled, Y_train_log, cv=5, scoring='neg_mean_squared_error')
    cv_mse_scores.append(scores.mean())

# Plot the cross-validation MSE scores
plt.plot(components_range, cv_mse_scores)
plt.xlabel('Number of Components')
plt.ylabel('Cross-Validation Mean Squared Error (MSE)')
plt.title('Cross-Validation Mean Squared Error (MSE) vs. Number of Components')
plt.grid(True)
plt.show()
```

21 PLS



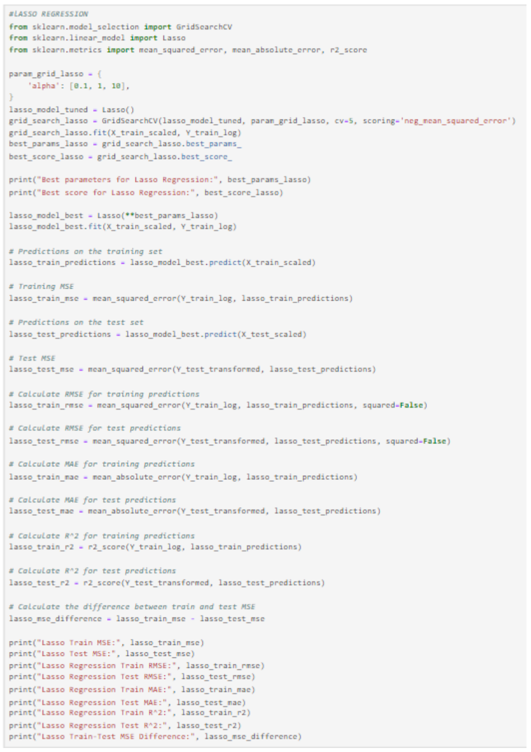
```
# Instantiate PLS Regression(chose 2 components from above graph)
pls_model = PLSRegression(n_components=2)

# Fit PLS Regression to the cleaned training data
pls_model.fit(X_train_scaled, Y_train_log)

# Predict on the cleaned training data
pls_train_predictions = pls_model.predict(X_train_scaled)

# Predict on the cleaned test data
pls_test_predictions = pls_model.predict(X_test_scaled)

# MSE for training predictions
pls_train_mse = mean_squared_error(Y_train_log, pls_train_predictions)

# MSE for test predictions
pls_test_mse = mean_squared_error(Y_test_transformed, pls_test_predictions)

# RMSE for training predictions
pls_train_rmse = mean_squared_error(Y_train_log, pls_train_predictions,squared=False)

# RMSE for test predictions
pls_test_rmse = mean_squared_error(Y_test_transformed, pls_test_predictions,squared=False)

# MAE for training predictions
pls_train_mae = mean_absolute_error(Y_train_log, pls_train_predictions)

# MAE for test predictions
pls_test_mae = mean_absolute_error(Y_test_transformed, pls_test_predictions)

# R^2 for training predictions
pls_train_r2 = r2_score(Y_train_log, pls_train_predictions)

# R^2 for test predictions
pls_test_r2 = r2_score(Y_test_transformed, pls_test_predictions)

# Difference between training and test MSE
pls_mse_difference = pls_train_mse - pls_test_mse

print("PLS Regression Train MSE:", pls_train_mse)
print("PLS Regression Test MSE:", pls_test_mse)
print("PLS Regression Train RMSE:", pls_train_rmse)
print("PLS Regression Test RMSE:", pls_test_rmse)
print("PLS Regression Train MAE:", pls_train_mae)
print("PLS Regression Test MAE:", pls_test_mae)
print("PLS Regression Train R^2:", pls_train_r2)
print("PLS Regression Test R^2:", pls_test_r2)
print("PLS Regression Train-Test MSE Difference:", pls_mse_difference)
```

22 RIDGE



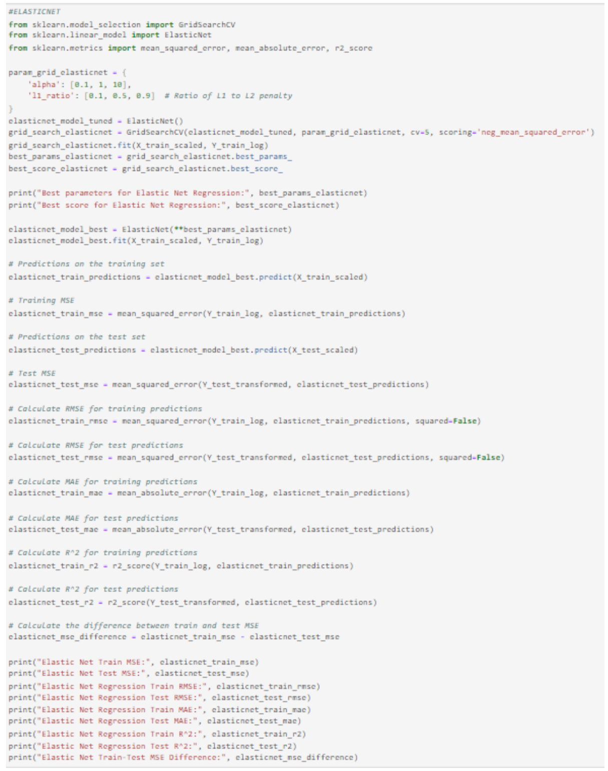
```
#RIDGE REGRESSION
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score

param_grid_ridge = {
    'alpha': [0.1, 1, 10],
}
ridge_model_tuned = Ridge()
grid_search_ridge = GridSearchCV(ridge_model_tuned, param_grid_ridge, cv=5, scoring='neg_mean_squared_error')
grid_search_ridge.fit(X_train_scaled, Y_train_log)
best_params_ridge = grid_search_ridge.best_params_
best_score_ridge = grid_search_ridge.best_score_

print("Best parameters for Ridge Regression:", best_params_ridge)
print("Best score for Ridge Regression:", best_score_ridge)

ridge_model_best = Ridge(**best_params_ridge)
ridge_model_best.fit(X_train_scaled, Y_train_log)

# Predictions on the training set
ridge_train_predictions = ridge_model_best.predict(X_train_scaled)

# Training MSE
ridge_train_mse = mean_squared_error(Y_train_log, ridge_train_predictions)

# Predictions on the test set
ridge_test_predictions = ridge_model_best.predict(X_test_scaled)

# Test MSE
ridge_test_mse = mean_squared_error(Y_test_transformed, ridge_test_predictions)

# Calculate RMSE for training predictions
ridge_train_rmse = mean_squared_error(Y_train_log, ridge_train_predictions, squared=False)

# Calculate RMSE for test predictions
ridge_test_rmse = mean_squared_error(Y_test_transformed, ridge_test_predictions, squared=False)

# Calculate MAE for training predictions
ridge_train_mae = mean_absolute_error(Y_train_log, ridge_train_predictions)

# Calculate MAE for test predictions
ridge_test_mae = mean_absolute_error(Y_test_transformed, ridge_test_predictions)

# Calculate R^2 for training predictions
ridge_train_r2 = r2_score(Y_train_log, ridge_train_predictions)

# Calculate R^2 for test predictions
ridge_test_r2 = r2_score(Y_test_transformed, ridge_test_predictions)

# Calculate the difference between train and test MSE
ridge_mse_difference= ridge_train_mse - ridge_test_mse

print("Ridge Train MSE:", ridge_train_mse)
print("Ridge Test MSE:", ridge_test_mse)
print("Ridge Regression Train RMSE:", ridge_train_rmse)
print("Ridge Regression Test RMSE:", ridge_test_rmse)
print("Ridge Regression Train MAE:", ridge_train_mae)
print("Ridge Regression Test MAE:", ridge_test_mae)
print("Ridge Regression Train R^2:", ridge_train_r2)
print("Ridge Regression Test R^2:", ridge_test_r2)
print("Ridge Train-Test MSE Difference:", ridge_mse_difference)
```

23 LASSO



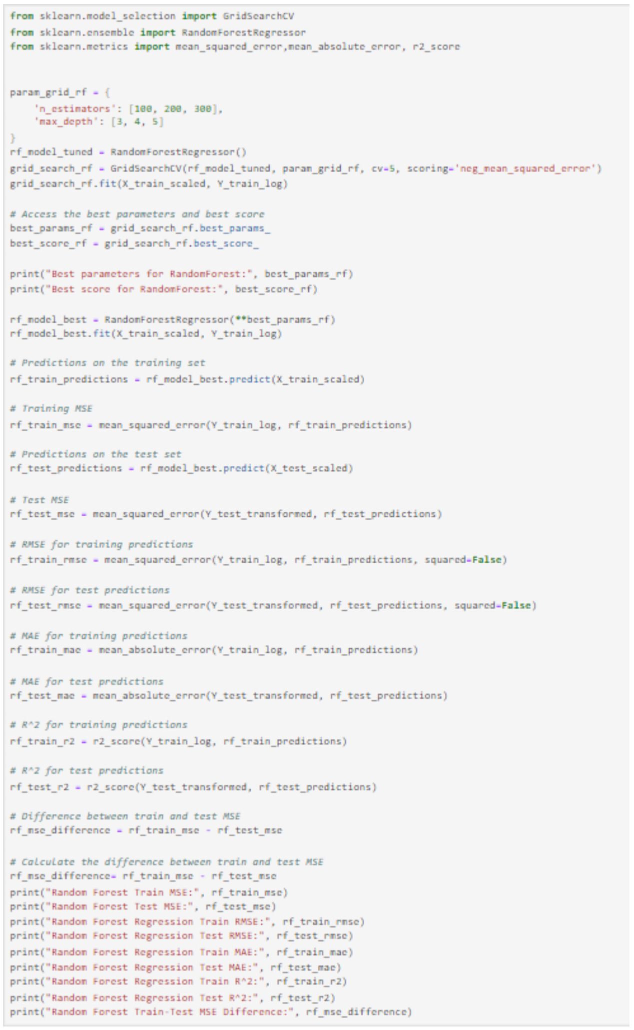
```
#LASSO REGRESSION
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_grid_lasso = {
    'alpha': [0.1, 1, 10],
}
lasso_model_tuned = Lasso()
grid_search_lasso = GridSearchCV(lasso_model_tuned, param_grid_lasso, cv=5, scoring='neg_mean_squared_error')
grid_search_lasso.fit(X_train_scaled, Y_train_log)
best_params_lasso = grid_search_lasso.best_params_
best_score_lasso = grid_search_lasso.best_score_

print("Best parameters for Lasso Regression:", best_params_lasso)
print("Best score for Lasso Regression:", best_score_lasso)

lasso_model_best = Lasso(**best_params_lasso)
lasso_model_best.fit(X_train_scaled, Y_train_log)

# Predictions on the training set
lasso_train_predictions = lasso_model_best.predict(X_train_scaled)

# Training MSE
lasso_train_mse = mean_squared_error(Y_train_log, lasso_train_predictions)

# Predictions on the test set
lasso_test_predictions = lasso_model_best.predict(X_test_scaled)

# Test MSE
lasso_test_mse = mean_squared_error(Y_test_transformed, lasso_test_predictions)

# Calculate RMSE for training predictions
lasso_train_rmse = mean_squared_error(Y_train_log, lasso_train_predictions, squared=False)

# Calculate RMSE for test predictions
lasso_test_rmse = mean_squared_error(Y_test_transformed, lasso_test_predictions, squared=False)

# Calculate MAE for training predictions
lasso_train_mae = mean_absolute_error(Y_train_log, lasso_train_predictions)

# Calculate MAE for test predictions
lasso_test_mae = mean_absolute_error(Y_test_transformed, lasso_test_predictions)

# Calculate R^2 for training predictions
lasso_train_r2 = r2_score(Y_train_log, lasso_train_predictions)

# Calculate R^2 for test predictions
lasso_test_r2 = r2_score(Y_test_transformed, lasso_test_predictions)

# Calculate the difference between train and test MSE
lasso_mse_difference = lasso_train_mse - lasso_test_mse

print("Lasso Train MSE:", lasso_train_mse)
print("Lasso Test MSE:", lasso_test_mse)
print("Lasso Regression Train RMSE:", lasso_train_rmse)
print("Lasso Regression Test RMSE:", lasso_test_rmse)
print("Lasso Regression Train MAE:", lasso_train_mae)
print("Lasso Regression Test MAE:", lasso_test_mae)
print("Lasso Regression Train R^2:", lasso_train_r2)
print("Lasso Regression Test R^2:", lasso_test_r2)
print("Lasso Train-Test MSE Difference:", lasso_mse_difference)
```

24 ELASTIC NET



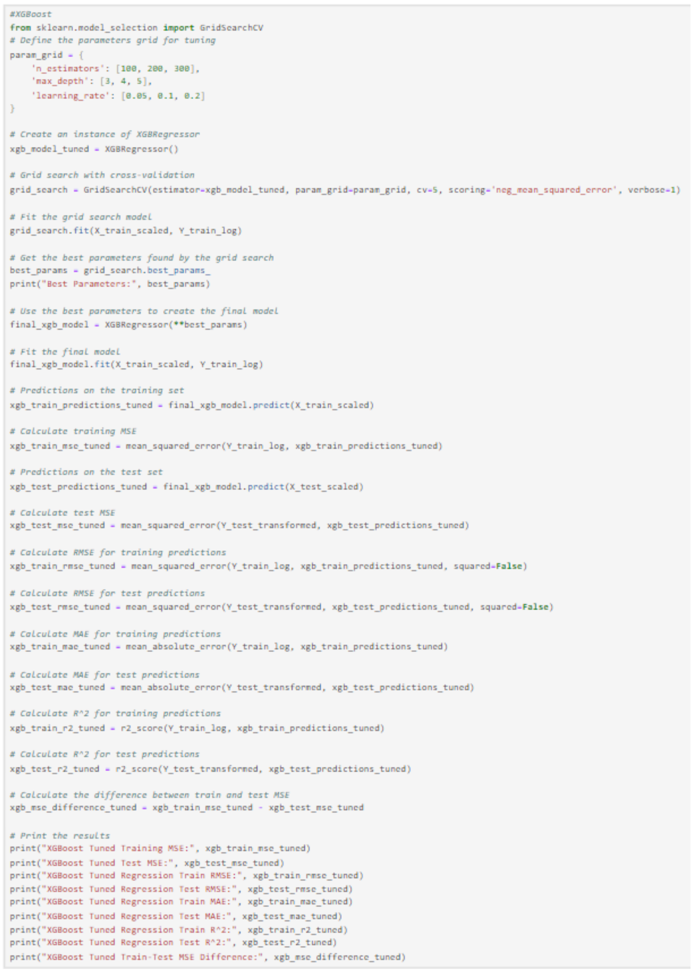
```
#ELASTICNET
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_grid_elasticnet = {
    'alpha': [0.1, 1, 10],
    'l1_ratio': [0.1, 0.5, 0.9]  # Ratio of L1 to L2 penalty
}
elasticnet_model_tuned = ElasticNet()
grid_search_elasticnet = GridSearchCV(elasticnet_model_tuned, param_grid_elasticnet, cv=5, scoring='neg_mean_squared_error')
grid_search_elasticnet.fit(X_train_scaled, Y_train_log)
best_params_elasticnet = grid_search_elasticnet.best_params_
best_score_elasticnet = grid_search_elasticnet.best_score_

print("Best parameters for Elastic Net Regression:", best_params_elasticnet)
print("Best score for Elastic Net Regression:", best_score_elasticnet)

elasticnet_model_best = ElasticNet(**best_params_elasticnet)
elasticnet_model_best.fit(X_train_scaled, Y_train_log)

# Predictions on the training set
elasticnet_train_predictions = elasticnet_model_best.predict(X_train_scaled)

# Training MSE
elasticnet_train_mse = mean_squared_error(Y_train_log, elasticnet_train_predictions)

# Predictions on the test set
elasticnet_test_predictions = elasticnet_model_best.predict(X_test_scaled)

# Test MSE
elasticnet_test_mse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions)

# Calculate RMSE for training predictions
elasticnet_train_rmse = mean_squared_error(Y_train_log, elasticnet_train_predictions, squared=False)

# Calculate RMSE for test predictions
elasticnet_test_rmse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions, squared=False)

# Calculate MAE for training predictions
elasticnet_train_mae = mean_absolute_error(Y_train_log, elasticnet_train_predictions)

# Calculate MAE for test predictions
elasticnet_test_mae = mean_absolute_error(Y_test_transformed, elasticnet_test_predictions)

# Calculate R^2 for training predictions
elasticnet_train_r2 = r2_score(Y_train_log, elasticnet_train_predictions)

# Calculate R^2 for test predictions
elasticnet_test_r2 = r2_score(Y_test_transformed, elasticnet_test_predictions)

# Calculate the difference between train and test MSE
elasticnet_mse_difference = elasticnet_train_mse - elasticnet_test_mse

print("Elastic Net Train MSE:", elasticnet_train_mse)
print("Elastic Net Test MSE:", elasticnet_test_mse)
print("Elastic Net Regression Train RMSE:", elasticnet_train_rmse)
print("Elastic Net Regression Test RMSE:", elasticnet_test_rmse)
print("Elastic Net Regression Train MAE:", elasticnet_train_mae)
print("Elastic Net Regression Test MAE:", elasticnet_test_mae)
print("Elastic Net Regression Train R^2:", elasticnet_train_r2)
print("Elastic Net Regression Test R^2:", elasticnet_test_r2)
print("Elastic Net Train-Test MSE Difference:", elasticnet_mse_difference)
```

25 RANDOM FOREST



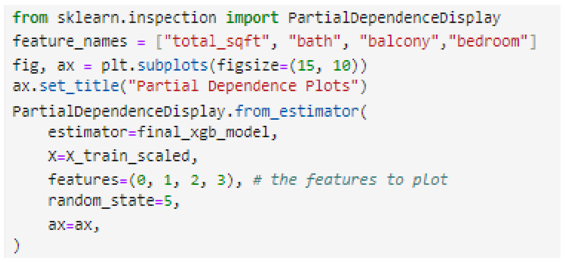
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score


param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5]
}
rf_model_tuned = RandomForestRegressor()
grid_search_rf = GridSearchCV(rf_model_tuned, param_grid_rf, cv=5, scoring='neg_mean_squared_error')
grid_search_rf.fit(X_train_scaled, Y_train_log)

# Access the best parameters and best score
best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_

print("Best parameters for RandomForest:", best_params_rf)
print("Best score for RandomForest:", best_score_rf)

rf_model_best = RandomForestRegressor(**best_params_rf)
rf_model_best.fit(X_train_scaled, Y_train_log)

# Predictions on the training set
rf_train_predictions = rf_model_best.predict(X_train_scaled)

# Training MSE
rf_train_mse = mean_squared_error(Y_train_log, rf_train_predictions)

# Predictions on the test set
rf_test_predictions = rf_model_best.predict(X_test_scaled)

# Test MSE
rf_test_mse = mean_squared_error(Y_test_transformed, rf_test_predictions)

# RMSE for training predictions
rf_train_rmse = mean_squared_error(Y_train_log, rf_train_predictions, squared=False)

# RMSE for test predictions
rf_test_rmse = mean_squared_error(Y_test_transformed, rf_test_predictions, squared=False)

# MAE for training predictions
rf_train_mae = mean_absolute_error(Y_train_log, rf_train_predictions)

# MAE for test predictions
rf_test_mae = mean_absolute_error(Y_test_transformed, rf_test_predictions)

# R^2 for training predictions
rf_train_r2 = r2_score(Y_train_log, rf_train_predictions)

# R^2 for test predictions
rf_test_r2 = r2_score(Y_test_transformed, rf_test_predictions)

# Difference between train and test MSE
rf_mse_difference = rf_train_mse - rf_test_mse

# Calculate the difference between train and test MSE
rf_mse_difference= rf_train_mse - rf_test_mse
print("Random Forest Train MSE:", rf_train_mse)
print("Random Forest Test MSE:", rf_test_mse)
print("Random Forest Regression Train RMSE:", rf_train_rmse)
print("Random Forest Regression Test RMSE:", rf_test_rmse)
print("Random Forest Regression Train MAE:", rf_train_mae)
print("Random Forest Regression Test MAE:", rf_test_mae)
print("Random Forest Regression Train R^2:", rf_train_r2)
print("Random Forest Regression Test R^2:", rf_test_r2)
print("Random Forest Train-Test MSE Difference:", rf_mse_difference)
```

26 XGBOOST



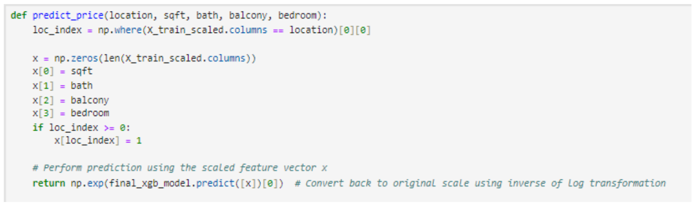
```
#XGBoost
from sklearn.model_selection import GridSearchCV
# Define the parameters grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.05, 0.1, 0.2]
}

# Create an instance of XGBRegressor
xgb_model_tuned = XGBRegressor()

# Grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model_tuned, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)

# Fit the grid search model
grid_search.fit(X_train_scaled, Y_train_log)

# Get the best parameters found by the grid search
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Use the best parameters to create the final model
final_xgb_model = XGBRegressor(**best_params)

# Fit the final model
final_xgb_model.fit(X_train_scaled, Y_train_log)

# Predictions on the training set
xgb_train_predictions_tuned = final_xgb_model.predict(X_train_scaled)

# Calculate training MSE
xgb_train_mse_tuned = mean_squared_error(Y_train_log, xgb_train_predictions_tuned)

# Predictions on the test set
xgb_test_predictions_tuned = final_xgb_model.predict(X_test_scaled)

# Calculate test MSE
xgb_test_mse_tuned = mean_squared_error(Y_test_transformed, xgb_test_predictions_tuned)

# Calculate RMSE for training predictions
xgb_train_rmse_tuned = mean_squared_error(Y_train_log, xgb_train_predictions_tuned, squared=False)

# Calculate RMSE for test predictions
xgb_test_rmse_tuned = mean_squared_error(Y_test_transformed, xgb_test_predictions_tuned, squared=False)

# Calculate MAE for training predictions
xgb_train_mae_tuned = mean_absolute_error(Y_train_log, xgb_train_predictions_tuned)

# Calculate MAE for test predictions
xgb_test_mae_tuned = mean_absolute_error(Y_test_transformed, xgb_test_predictions_tuned)

# Calculate R^2 for training predictions
xgb_train_r2_tuned = r2_score(Y_train_log, xgb_train_predictions_tuned)

# Calculate R^2 for test predictions
xgb_test_r2_tuned = r2_score(Y_test_transformed, xgb_test_predictions_tuned)

# Calculate the difference between train and test MSE
xgb_mse_difference_tuned = xgb_train_mse_tuned - xgb_test_mse_tuned

# Print the results
print("XGBoost Tuned Training MSE:", xgb_train_mse_tuned)
print("XGBoost Tuned Test MSE:", xgb_test_mse_tuned)
print("XGBoost Tuned Regression Train RMSE:", xgb_train_rmse_tuned)
print("XGBoost Tuned Regression Test RMSE:", xgb_test_rmse_tuned)
print("XGBoost Tuned Regression Train MAE:", xgb_train_mae_tuned)
print("XGBoost Tuned Regression Test MAE:", xgb_test_mae_tuned)
print("XGBoost Tuned Regression Train R^2:", xgb_train_r2_tuned)
print("XGBoost Tuned Regression Test R^2:", xgb_test_r2_tuned)
print("XGBoost Tuned Train-Test MSE Difference:", xgb_mse_difference_tuned)
```

27 PDP



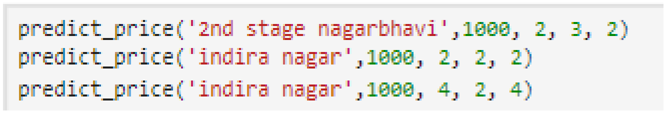
```
from sklearn.inspection import PartialDependenceDisplay
feature_names = ["total_sqft", "bath", "balcony","bedroom"]
fig, ax = plt.subplots(figsize=(15, 10))
ax.set_title("Partial Dependence Plots")
PartialDependenceDisplay.from_estimator(
    estimator=final_xgb_model,
    X=X_train_scaled,
    features=(0, 1, 2, 3), # the features to plot
    random_state=5,
    ax=ax,
)
```

28



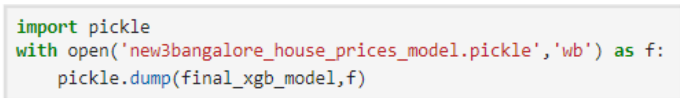
```
def predict_price(location, sqft, bath, balcony, bedroom):
    loc_index = np.where(X_train_scaled.columns == location)[0][0]

    x = np.zeros(len(X_train_scaled.columns))
    x[0] = sqft
    x[1] = bath
    x[2] = balcony
    x[3] = bedroom
    if loc_index >= 0:
        x[loc_index] = 1

    # Perform prediction using the scaled feature vector x
    return np.exp(final_xgb_model.predict([x])[0])  # Convert back to original scale using inverse of log transformation
```

29



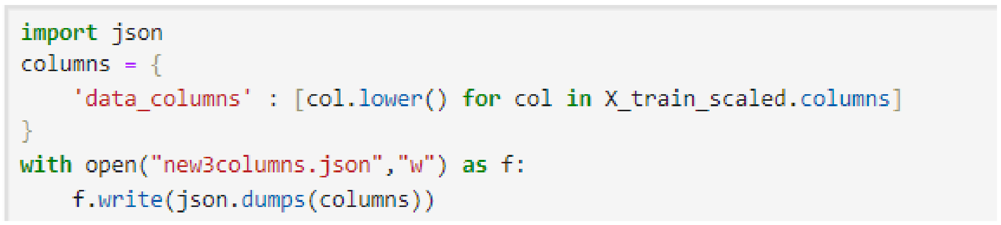
```
predict_price('2nd stage nagarbhavi',1000, 2, 3, 2)
predict_price('indira nagar',1000, 2, 2, 2)
predict_price('indira nagar',1000, 4, 2, 4)
```

30



```
import pickle
with open('new3bangalore_house_prices_model.pickle','wb') as f:
    pickle.dump(final_xgb_model,f)
```

31



```
import json
columns = {
    'data_columns' : [col.lower() for col in X_train_scaled.columns]
}
with open("new3columns.json","w") as f:
    f.write(json.dumps(columns))
```

32 CODE REGARDING PRODUCT

https://drive.google.com/drive/folders/1-LY75DKPVWzskRVAzaAwTJvchB99yBx0?usp=sharing