REPORT - CS2202  
TITLE: DATA ANALYSIS AND DATA PREPROCESSING FOR GURGAON REAL ESTATE DATASET

G\_241\_211\_239\_076

horizontal line

# 

[**INTRODUCTION**](#_rn1w2m8521yk) **2**

[OBJECTIVES:](#_j4kqqqpd0qhz) 2

[**DATA EXPLORATION**](#_ga25u6kunb99) **3**

[Duplicate Rows:](#_6631r46vbp3v) 3

[Societies with less flats/houses:](#_dt0p5uihipdm) 3

[**IDENTIFYING MISSING VALUES AND HANDLING**](#_vsbp8y2lgg4u) **7**

[IDENTIFYING:](#_t7lailatptgx) 7

[HANDLING:](#_gvhwmwck0g0z) 8

[1. built\_up\_area and carpet\_area](#_cuj0prkxt229) 8

[2. super\_built\_up\_area](#_f23htb416ksg) 9

[3. facing](#_jfbsaj9lf5g2) 10

[4. floorNum](#_76y2cw5cukg9) 12

[5. price, price\_per\_sqft, area](#_28oxv1ckx1hl) 12

[**OUTLIER DETECTION AND HANDLING**](#_qg5405vb43yw) **15**

[DEFINITION AND DETECTION:](#_u8kup2caahph) 15

[HANDLING:](#_d6255oixgtt0) 19

[Use of the Winsorization method to handle the outliers:](#_cntjtazalaiy) 19

[Trimming the remaining outliers:](#_abvvrnxay60) 22

[**FEATURE SELECTION:**](#_ewafqa1ggoab) **23**

[RANDOM FOREST:](#_m34wflty3v9d) 23

[MACHINE LEARNING MODEL:](#_z4t2s7ryh62) 24

[Random Forest Regressor:](#_6dao6y2y7r1n) 24

[Linear Regression Model:](#_xwh7zkf0l64s) 24

# 

# 

# INTRODUCTION

The report presents findings and actions taken in handling missing values and outliers in the Gurgaon real estate dataset. The dataset contains information about various attributes of real estate properties in Gurgaon, including property type, location, price, area, etc. Our objective is to ensure the integrity and quality of the dataset for further analysis and modelling.

The objective of this project is to investigate and rectify any anomalies within the Gurgaon real estate dataset, specifically focusing on missing values and outliers. Addressing missing values and outliers not only enhances the dataset's reliability but also fortifies subsequent analytical endeavours and modelling processes.

## OBJECTIVES:

* Identify missing values and implement appropriate strategies
* Detect outliers using statistical methods and visualization techniques
* Apply techniques to handle outliers and ensure data quality

# DATA EXPLORATION

The "Gurgaon\_RealEstate.csv" dataset is loaded into a Pandas DataFrame, providing a foundation for subsequent data analysis and manipulation. It retrieves the dimensions of the DataFrame, displaying the number of columns and rows.

dataset = pd.read\_csv('Gurgaon\_RealEstate.csv')

df = pd.DataFrame(dataset)

[rows, columns] = df.shape

print(f"Columns: {columns}, Rows: {rows}")

Rows: 3803, Columns: 23

### Duplicate Rows:

For the initial exploration, the duplicate rows in the data set are removed.

duplicate\_rows = df[df.duplicated()]

row\_index = duplicate\_rows.index.tolist()

df.drop(index=row\_index, inplace=True)

.duplicated() method returns a series with True and False values that describe which rows in the DataFrame are duplicated or not.

.drop() method removes the specified row or column.

By which the number of rows is decreased - Rows: 3677, Columns: 23

### Societies with less flats/houses:

We use the concept of Frequency Bins to remove the societies with less number of flats/ houses. Each bin with the society name holds the number of occurrences of the society as a flat/ house.

The number of bins is decided by the number of societies existing in the data set. In "Gurgaon\_RealEstate.csv" dataset, there are 676 different societies.

Say, we remove all the societies with <7 flats/ houses for simplicity.

#1. Count the frequency of each society name

frequency\_count = pd.Series(df['society']).value\_counts()

#2. Decide the number of bins

num\_bins = 676

#3. Frequency data in the bins

frequency\_bins = pd.cut(frequency\_count, bins = num\_bins)

# Getting society names along with frequencies

society\_frequencies = pd.DataFrame({'society': frequency\_count.index, 'frequency': frequency\_count.values})

# Drop the societies that have less frequency count

filtered\_societies = society\_frequencies[society\_frequencies['frequency'] > 6.74] #only selecting societies with >=7 flats/houses

print(filtered\_societies)

.value\_counts() method counts the frequency of each unique value in a series

Then we bin the frequency count data into the specified number of bins (num\_bins) using the pd.cut() function.

Filtered\_societies containing society names with frequencies greater than 6.74, are printed to display the selected societies that have a higher presence in the dataset.

society frequency

0 independent 486

1 tulip violet 75

2 ss the leaf 73

3 dlf new town heights 42

4 shapoorji pallonji joyville gurugram 42

.. ... …

129 vipul lavanya 7

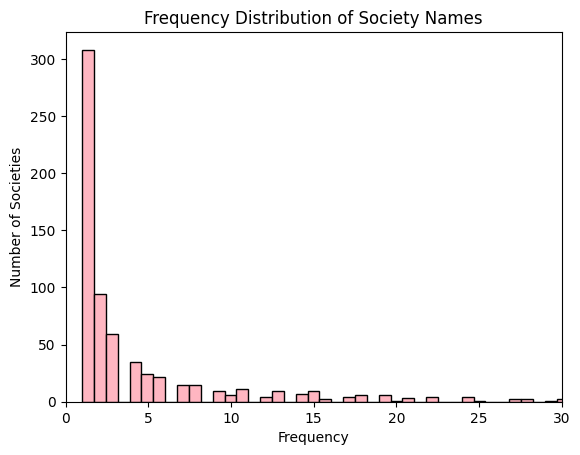
130 tulip ivory 7

131 signature andour heights 7

132 satya the hermitage 7

133 emaar digihomes 7

The following is a histogram Number of societies vs Frequency: where frequency represents the number of flats/ houses.



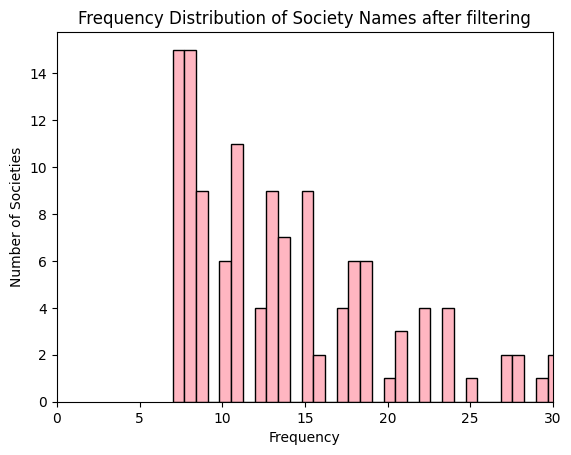
Now, we drop the societies with less frequency and alter or DataFrame.

filtered\_society\_names = filtered\_societies['society']

filtered\_df = df[df['society'].isin(filtered\_society\_names)]

[columns, rows] = filtered\_df.shape

print(f"Column:{columns}, Rows: {rows}")



Filtered\_df is the new DataFrame with the duplicates and the societies with less number of flats/houses removed. We will proceed to use this data frame for further analyses.

Filtered\_df contains Rows: 2611, Columns: 23.

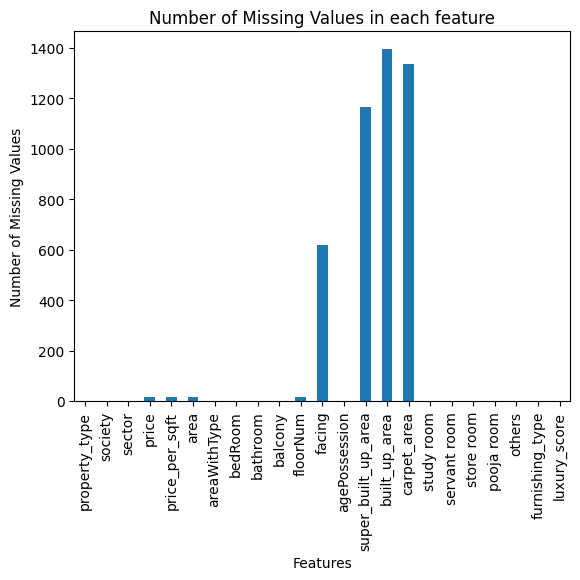
# IDENTIFYING MISSING VALUES AND HANDLING

## IDENTIFYING:

To identify the number of missing values in each column (feature), we use .isnull().sum() methods. .isnull().sum() identifies the null elements in each column and counts them.

missing\_values\_per\_column = filtered\_df.isnull().sum()

Graphically,



## HANDLING:

The blog [Effective Strategies for Handling Missing Values in Data Analysis](https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/) gives an idea of how to handle missing values. Here are the two main approaches:

1. **Deletion:** This involves removing rows or columns with missing values.
2. **Imputation:** This replaces missing values with estimates.
3. Mean/ Median/ Mode imputation
4. K- Nearest Neighbours imputation
5. Model-based imputation

The following ways are adopted to handle the missing values in each feature.

1. The missing values in built\_up\_area and carpet\_area can be imputed by finding the ratio between built\_up\_area:super\_built\_up\_area and carpet\_area:built\_up\_area.
2. Super\_built\_up\_area can be deleted as it is highly irrelevant to the dataset and contains many missing values.
3. The missing values in facing can be imputed by drawing a relationship between facing and any other column in the dataset using heatmaps.
4. The missing values in floorNum can be imputed with mode value
5. As price\_per\_sqft\*area gives the price, we can find price\_per\_sqft and area by —------ and multiply both values to predict the price.

### 

### 1. built\_up\_area and carpet\_area

#identify rows with all filled values

valid\_rows = filtered\_df.dropna(subset=['super\_built\_up\_area', 'built\_up\_area', 'carpet\_area'])

#ratios

ratio1 = filtered\_df['built\_up\_area']/filtered\_df['super\_built\_up\_area']

ratio2 = filtered\_df['carpet\_area']/filtered\_df['built\_up\_area']

#predicting

filtered\_df.loc[filtered\_df['super\_built\_up\_area'].notna() & filtered\_df['built\_up\_area'].isna(), 'built\_up\_area'] = filtered\_df['super\_built\_up\_area'] \* ratio1.mean()

filtered\_df.loc[filtered\_df['carpet\_area'].notna() & filtered\_df['built\_up\_area'].isna(), 'built\_up\_area'] = filtered\_df['carpet\_area'] / ratio2.mean()

filtered\_df.loc[filtered\_df['built\_up\_area'].notna() & filtered\_df['carpet\_area'].isna(), 'carpet\_area'] = filtered\_df['built\_up\_area'] \* ratio2.mean()

.dropna() method drops all the rows with null values in the specified columns and creates valid\_rows that contain the rows where all three columns have a value.

We then find ratio1 and ratio2 where ratio1 is built\_up\_area/super\_built\_up\_area and ratio2 is carpet\_area/built\_up\_area.

Then we predict the missing values in built\_up\_area and carpet\_area by giving the following conditions:

* If super\_built\_up\_area is not null and built\_up\_area is null, we find the built\_up\_area by multiplying super\_built\_up\_area and ratio1.mean() – where .mean() method finds the mean value of ratio1.
* If carpet\_area is not null and built\_up\_area is null, we find the carpet\_area by dividing carpet\_area with ratio2.mean().
* If built\_up\_area is not null and carpet\_area is null, we find the built\_up\_area by multiplying built\_up\_area and ratio2.mean().

With this, all the missing values in built\_up\_area and carpet\_area are imputed.

### 2. super\_built\_up\_area

We delete the super\_built\_up\_area column as it is only used to find the values of built\_up\_area and carpet\_area.

filtered\_df = filtered\_df.drop(['super\_built\_up\_area'], axis=1)

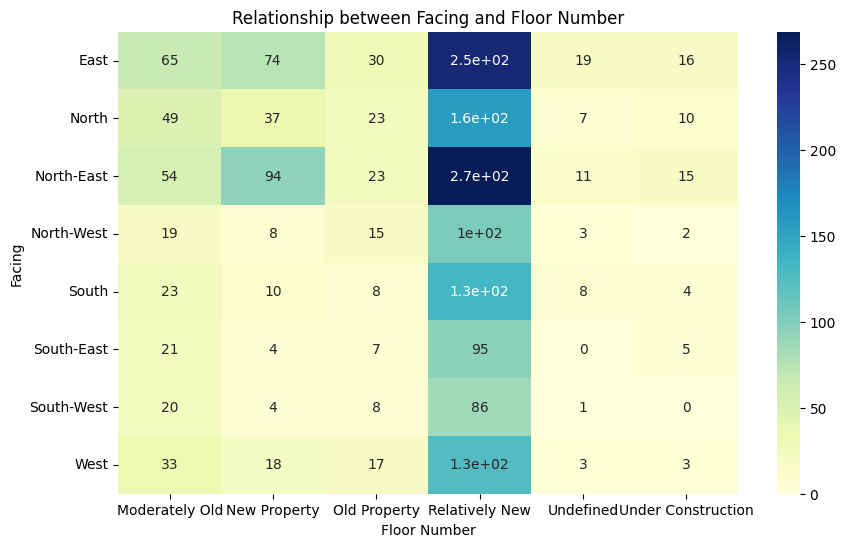
filtered\_df.shape

We are left with,

Rows: 2611, Columns: 22

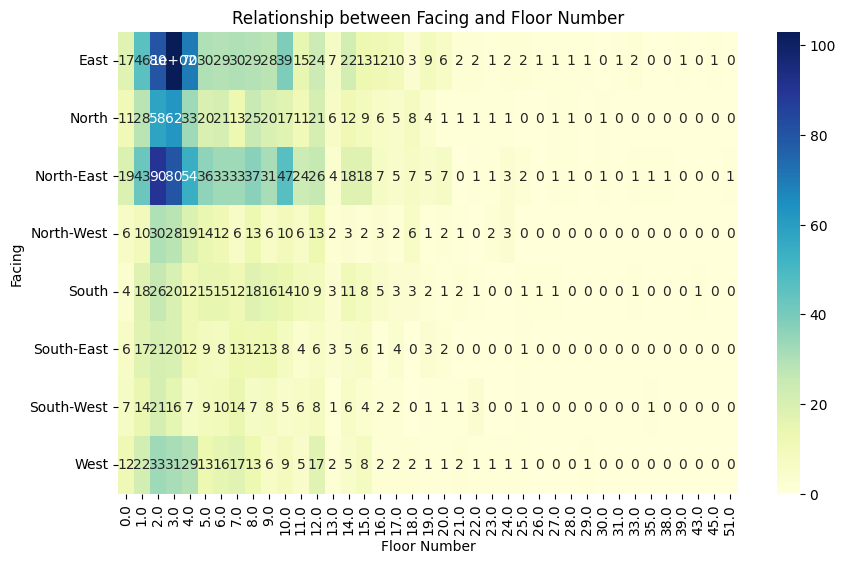
### 3. facing

When we draw a heatmap between facing and agePossession, we infer,



For every character like ‘Moderately old’, ‘Relatively New’ etc, the frequently occurring facing is East and North-East. More particularly, flats/houses under ‘Moderately Old’, ‘Old Property’, ‘Undefined’ and ‘Under Construction’, majority is East facing and flats/houses which are classified under ‘New Property’ and ‘Relatively New’ are North-East faced.

We infer the same i.e. most flats/houses are classified under East facing and North-East facing when a heat map is drawn between facing and floorNum.



We now impute the following observations into the facing column

filtered\_df['facing'] = np.where(filtered\_df['facing'].isna() & filtered\_df['agePossession'].isin(['Moderately Old', 'Old Property', 'Undefined', 'Under Construction']), 'east', filtered\_df['facing'])

filtered\_df['facing'] = np.where(filtered\_df['facing'].isna() & filtered\_df['agePossession'].isin(['New Property', 'Relatively New']), 'North-East', filtered\_df['facing'])

# Print the new DataFrame

print(filtered\_df['facing'], filtered\_df['agePossession'])

We use np.where() function to replace the missing values in the facing column based on the value in the agePossession column. If agePossession is

Moderately Old, Old Property, Undefined, Under Construction - East

New Property, Relatively New - North-East

### 4. floorNum

For floor number, mean/ median/ mode imputation can be used.

Mean imputation is used when the distribution is normal. Therefore mean does not represent the central tendency well due to the presence of outliers. A median is a better estimate in such cases.

from sklearn.impute import SimpleImputer

missing\_floorNum = filtered\_df['floorNum'].isnull()

imputer = SimpleImputer(strategy='median')

filtered\_df['floorNum'] = imputer.fit\_transform(filtered\_df['floorNum'].values.reshape(-1, 1))

We create a SimpleImputer instance with the strategy set to median

.fit\_transform() method is used to impute the missing values with median after the floorNum values are reshaped into a 2D array.

### 5. price, price\_per\_sqft, area

To impute the missing values in three different columns in the data frame, we use kNNimputer from scikit-learn

KNNimputer uses the K-Nearest Neighbours approach to impute the missing values.

from sklearn.impute import KNNImputer

# Define the columns for imputation

cols = ['luxury\_score', 'bedRoom', 'bathroom', 'floorNum', 'study room', 'servant room', 'pooja room', 'others', 'furnishing\_type']

# KNNImputer instance

imputer = KNNImputer(n\_neighbors=2)

# Impute missing values for 'price'

filtered\_df['price'] = imputer.fit\_transform(filtered\_df[cols + ['price']])[:, -1]

# Impute missing values for 'price\_per\_sqft'

filtered\_df['price\_per\_sqft'] = imputer.fit\_transform(filtered\_df[cols + ['price\_per\_sqft']])[:, -1]

# Impute missing values for 'area'

filtered\_df['area'] = imputer.fit\_transform(filtered\_df[cols + ['area']])[:, -1]

print(filtered\_df['price'], filtered\_df['price\_per\_sqft'], filtered\_df['area'])

cols list contains the names of columns to be used for imputation in each step.

A KNNimputer object is created with n\_neighbours=2, indicating that it will consider two nearest neighbours when imputing missing values.

We then use .fit\_transform() method to fit the imputer to the data and impute missing values.

All the missing values throughout the dataframe are either imputed/ deleted. Finally, .isnull().sum() gives the following output.

**property\_type 0**

**society 0**

**sector 0**

**price 0**

**price\_per\_sqft 0**

**area 0**

**areaWithType 0**

**bedRoom 0**

**bathroom 0**

**balcony 0**

**floorNum 0**

**facing 0**

**agePossession 0**

**built\_up\_area 0**

**carpet\_area 0**

**study room 0**

**servant room 0**

**store room 0**

**pooja room 0**

**others 0**

**furnishing\_type 0**

**luxury\_score 0**

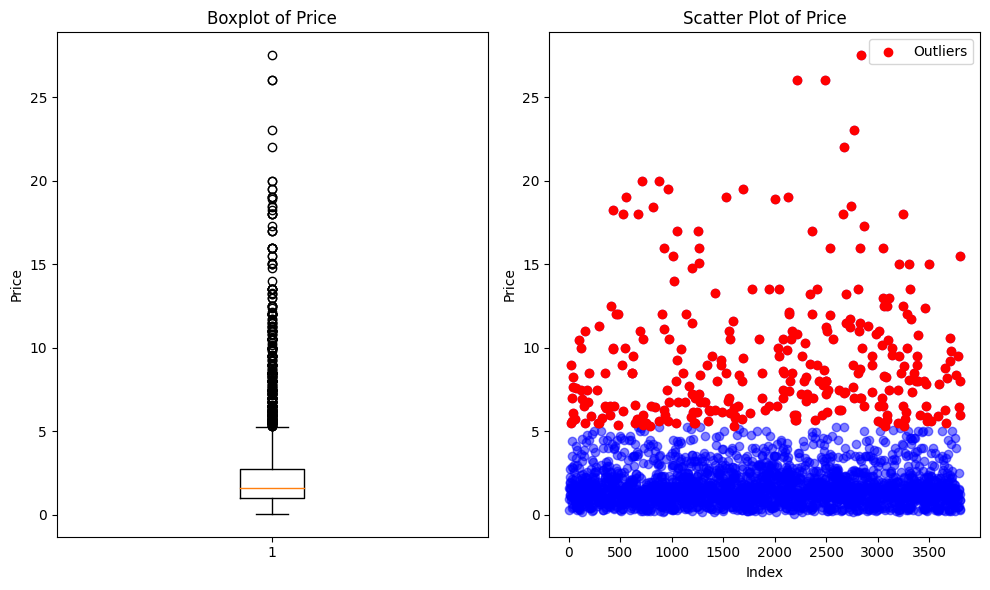
# OUTLIER DETECTION AND HANDLING

## DEFINITION AND DETECTION:

Outliers are the data points that lie far outside the range of the majority of the data points.

We use graphical methods like box plots and statistical methods like IQR to identify the outliers in all the numeric features.

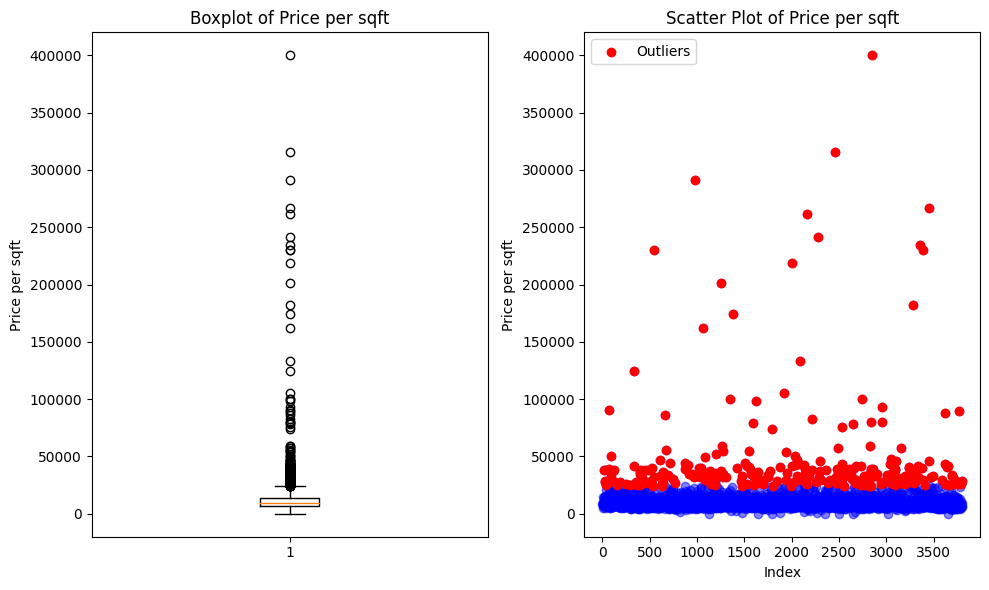
The following are the boxplots representation of the data

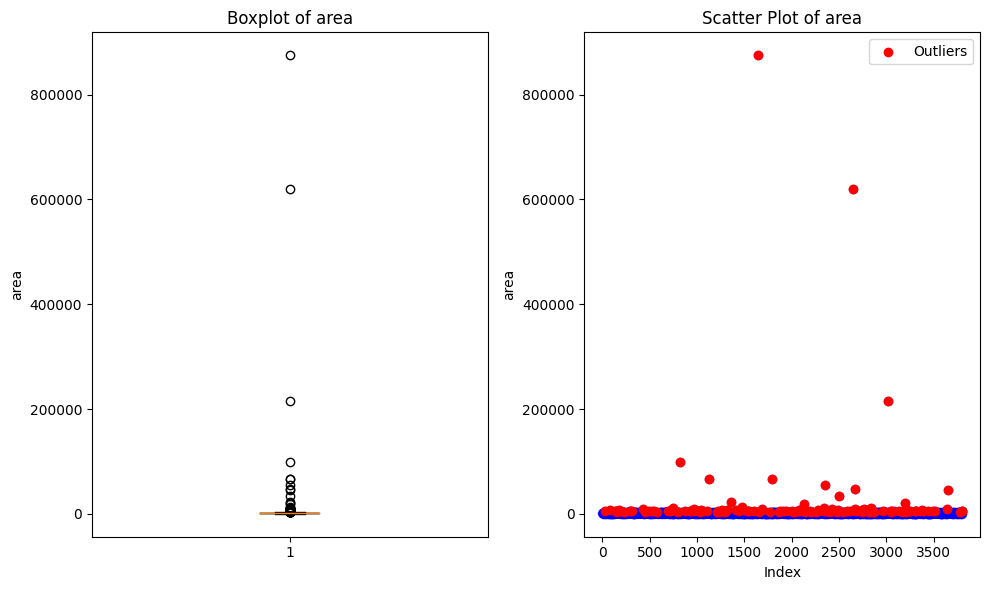


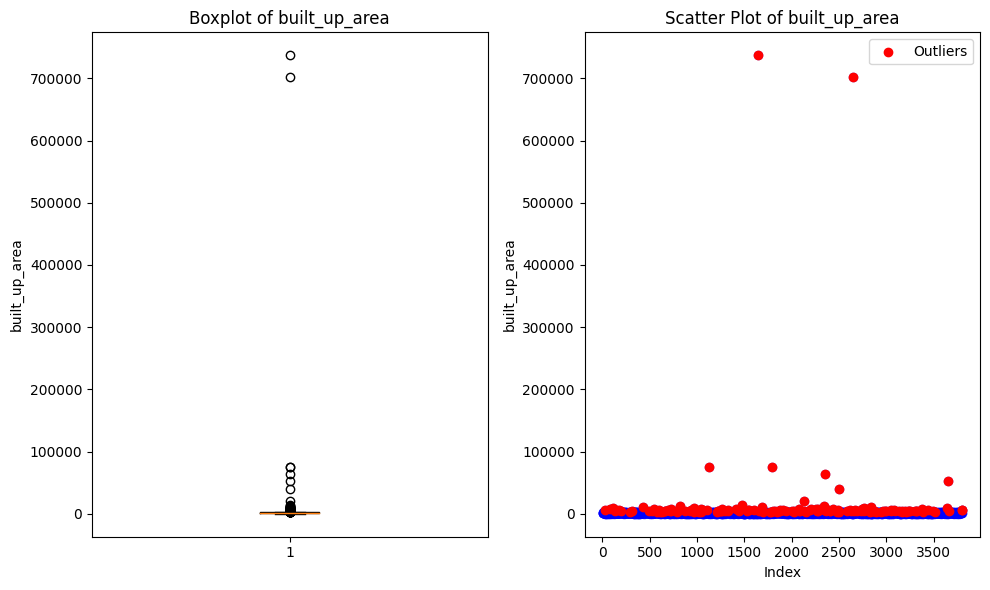
**Box** - The box represents the interquartile range (IQR), which is the range between the first quartile (Q1) and the third quartile (Q3). The length of the box indicates the spread of the middle 50% of the data.

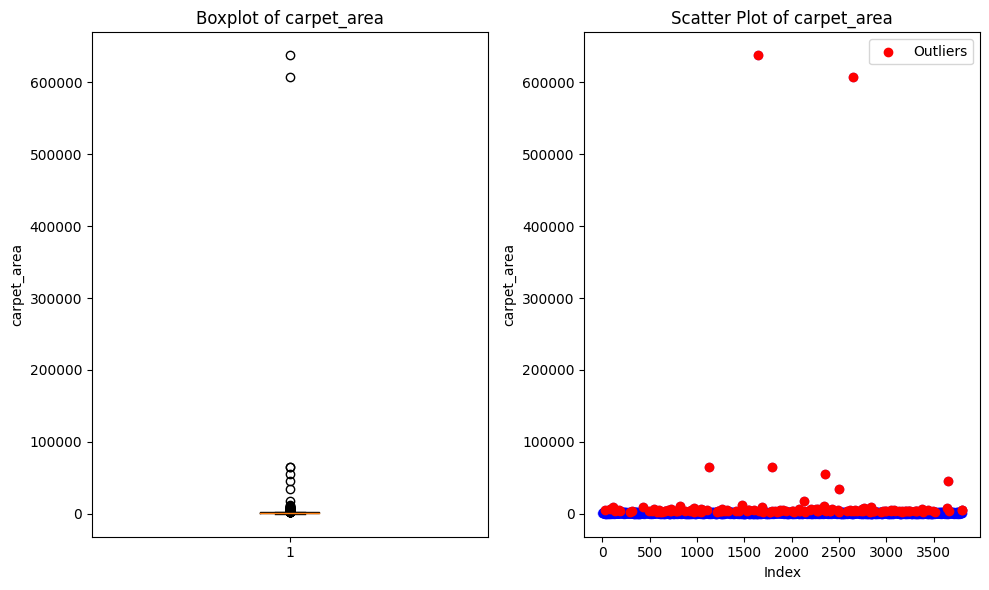
**Whiskers** - The lines extending from the box show the range of the data excluding outliers. They extend up to 1.5 times the IQR from Q1 and Q2.

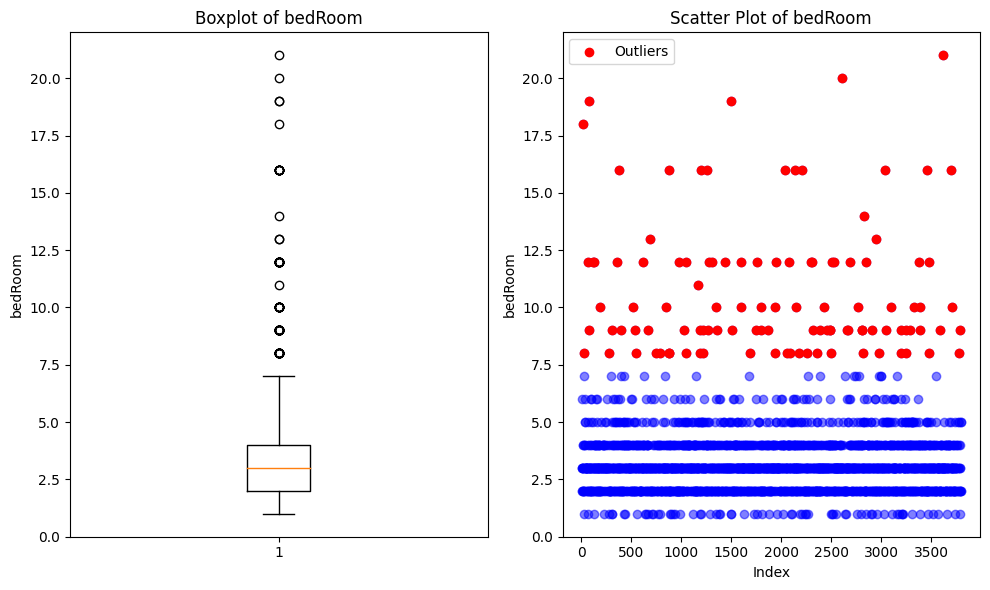
**Individual points** - Any data point beyond the whiskers are potential outliers or extreme values of the dataset.

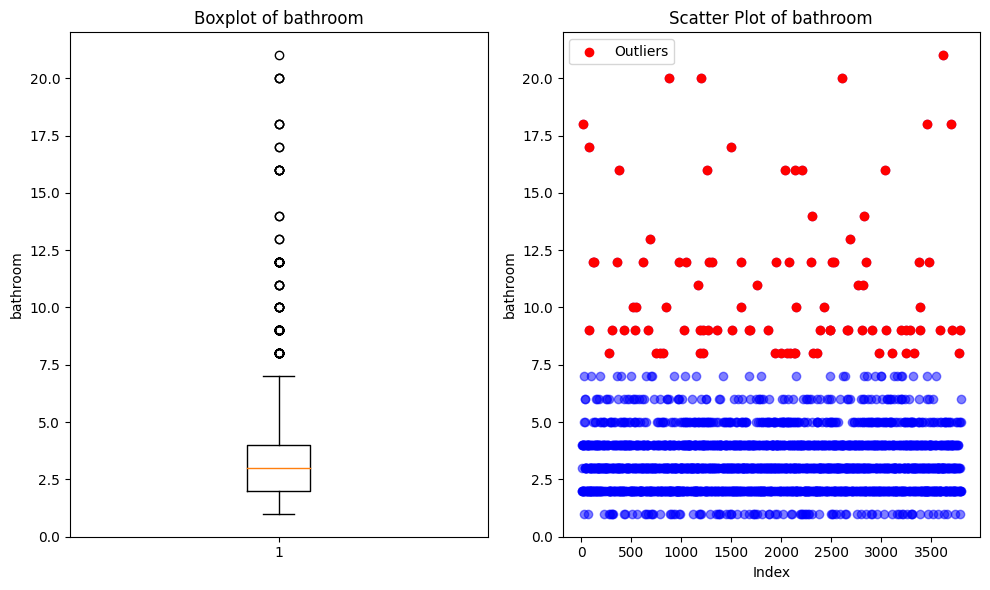


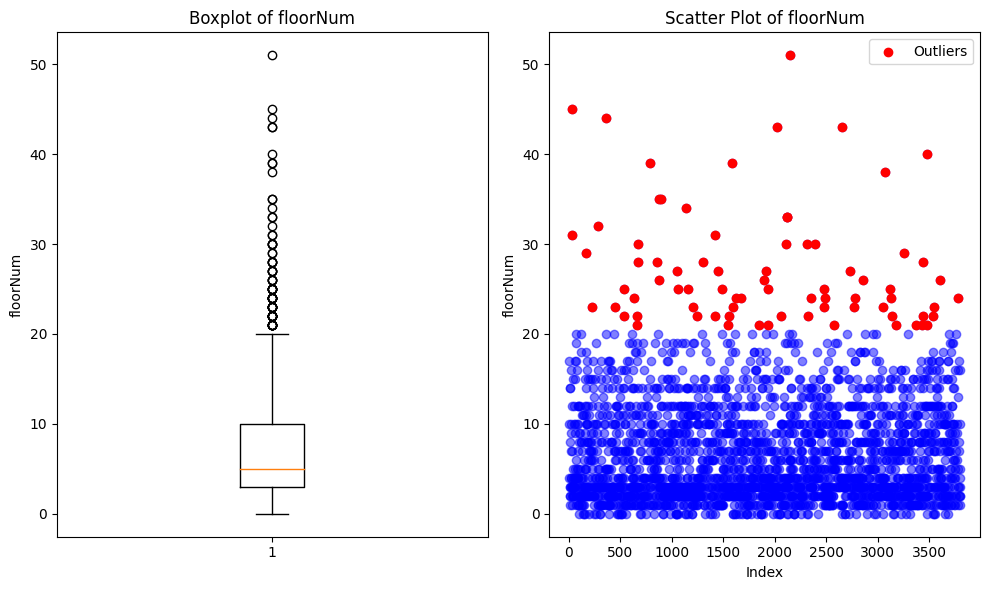












### HANDLING:

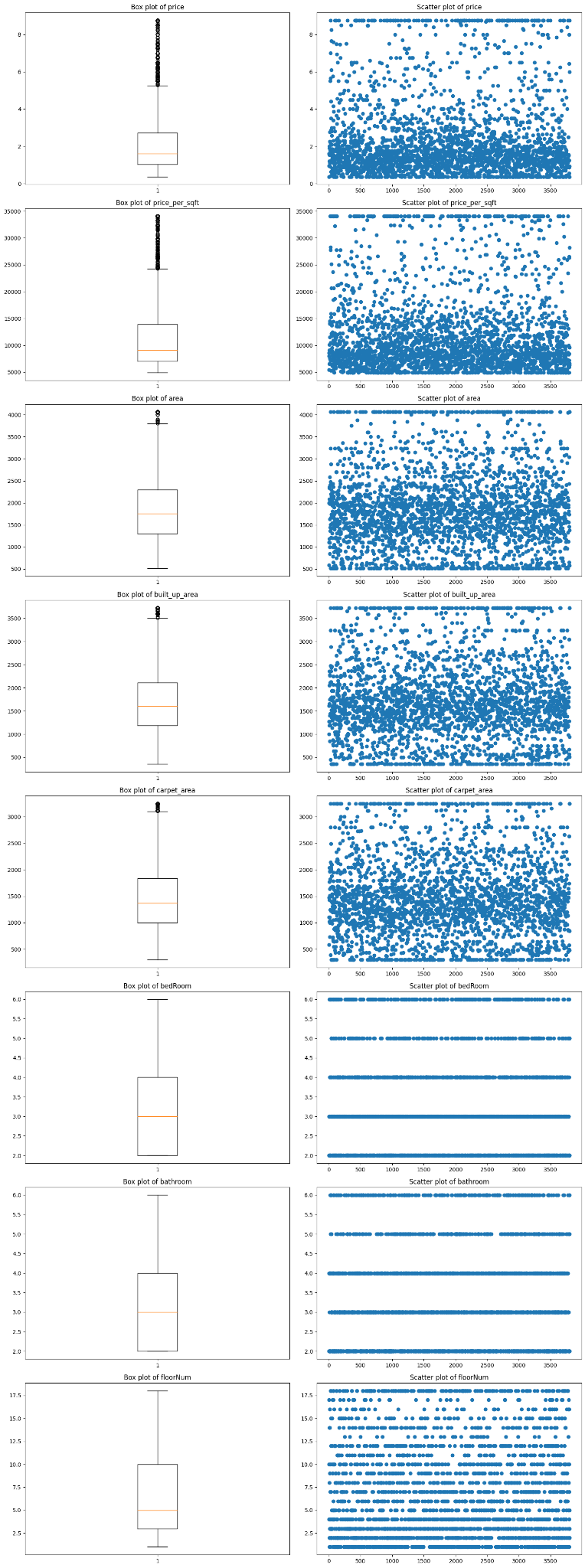
The blog [How to Remove Outliers for Machine Learning?](https://medium.com/analytics-vidhya/how-to-remove-outliers-for-machine-learning-24620c4657e8) gives an idea of how to handle outliers. Here are the few main approaches:

1. Trimming/ Deleting observations: This can be done when the outliers are believed to be due to errors in data collection or outlier observations are very small in number.
2. Transforming values:
3. Winsorization: This is a method to replace the extreme values with the less extreme values, it adjusts them to a specified percentile value

### Use of the Winsorization method to handle the outliers:

Winsorization method is chosen for the following reasons:

1. It Preserves the Data
2. Minimal Data Manipulation
3. Reduces Skewness



### 

### Trimming the remaining outliers:

Since some outliers still exist in the data frame after the winsorization method, we use the method of trimming to get rid of these outliers.

# Detect and remove rows with outliers

outlier\_indices = []

# Define the columns to check for outliers

columns\_to\_check = ['price', 'price\_per\_sqft', 'area', 'built\_up\_area', 'carpet\_area', 'bedRoom', 'bathroom', 'floorNum']

# Iterate over each column

for col in columns\_to\_check:

# Calculate the interquartile range

Q1 = np.percentile(filtered\_df[col], 25)

Q3 = np.percentile(filtered\_df[col], 75)

IQR = Q3 - Q1

# Define the lower and upper bounds

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Identify outliers and append their indices to the list

outliers = filtered\_df[(filtered\_df[col] < lower\_bound) | (filtered\_df[col] > upper\_bound)].index

outlier\_indices.extend(outliers)

# Remove duplicate indices

outlier\_indices = list(set(outlier\_indices))

# Remove rows with outliers

filtered\_df = filtered\_df.drop(outlier\_indices)

# Reset index

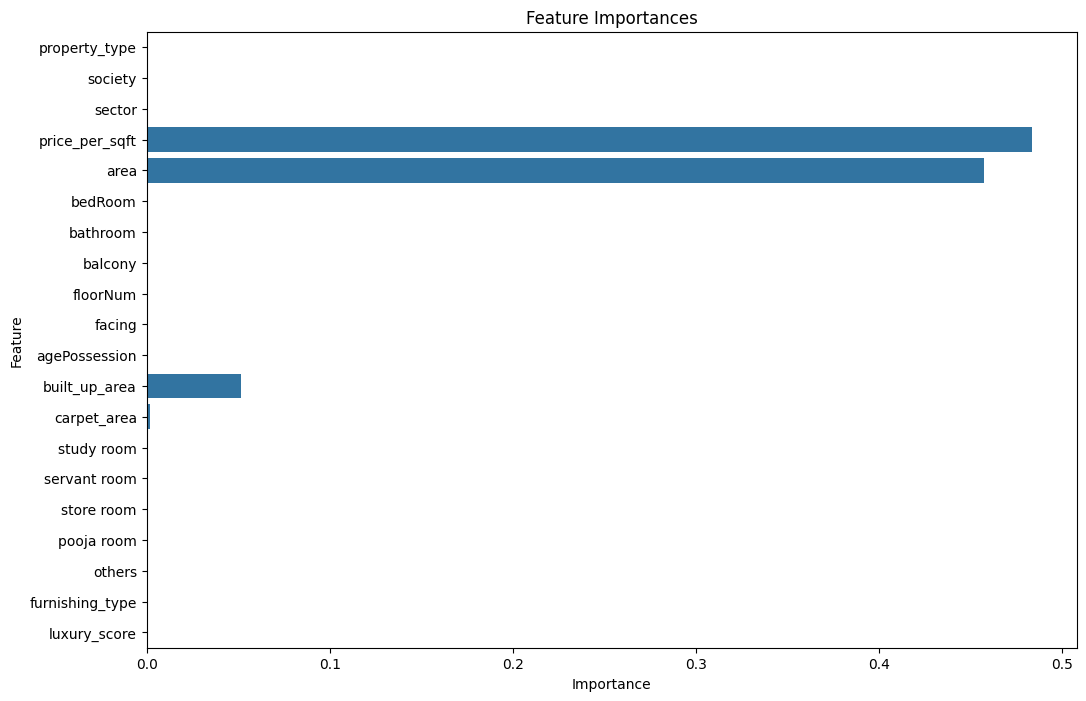
filtered\_df = filtered\_df.reset\_index(drop=True)

print('Rows with outliers removed. Filtered DataFrame updated.')

Therefore all the outliers in the dataframe are handled

# FEATURE SELECTION:

## RANDOM FOREST:



## 

## MACHINE LEARNING MODEL:

### Random Forest Regressor:

Mean Squared Error: 0.006707365046505068

### Linear Regression Model:

Mean Squared Error: 0.07121716866396541

Therefore Random Forest Regressor Model is a better model as it’s Mean Squared Error is comparatively lesser than that of Linear Regression Model.