REPORT

ASSIGNMENT ML

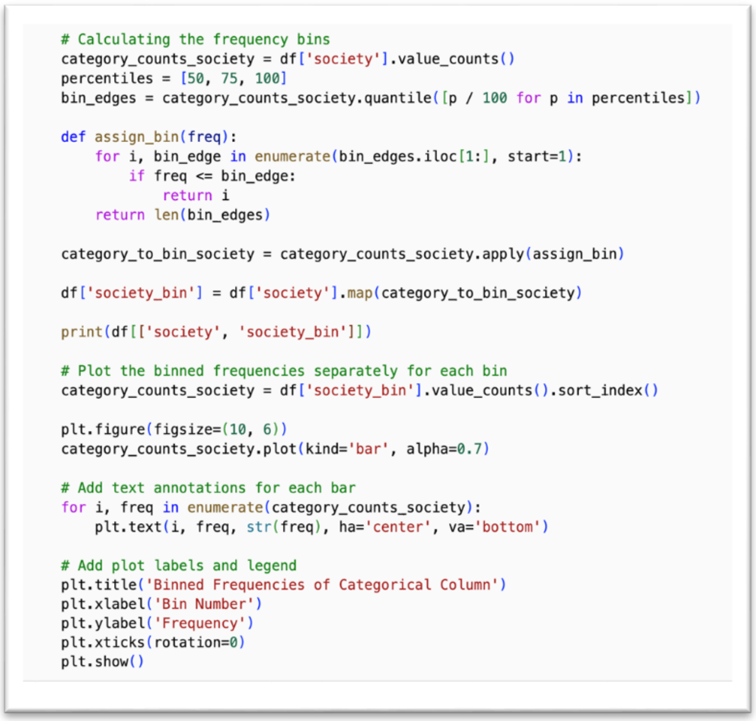
G_286_250_268_310

Objectives of the assignment:

- Identify missing values in the dataset and implement appropriate strategies for handling them.
- Detect outliers in the dataset using statistical methods and visualization techniques.
- Apply suitable techniques to handle outliers and ensure data quality for further analysis

FINDINGS FROM TASK 1-PRE-PROCESSING OF DATA (view the colab file for observations)

- After identifying the features and their data types and removing duplicates we started in-depth processing of each column.
- Due to the high frequency of the society column, we used binning to reduce the frequency of society column. We did the same for sector column and wherever necessary.

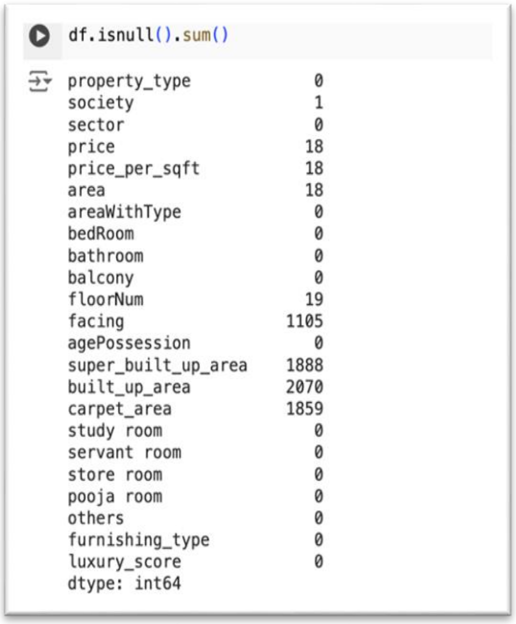


```python
# Calculating the frequency bins
category_counts_society = df['society'].value_counts()
percentiles = [50, 75, 100]
bin_edges = category_counts_society.quantile([p / 100 for p in percentiles])

def assign_bin(freq):
    for i, bin_edge in enumerate(bin_edges.iloc[1:], start=1):
        if freq <= bin_edge:
            return i
    return len(bin_edges)

category_to_bin_society = category_counts_society.apply(assign_bin)

df['society_bin'] = df['society'].map(category_to_bin_society)

print(df[['society', 'society_bin']])

# Plot the binned frequencies separately for each bin
category_counts_society = df['society_bin'].value_counts().sort_index()

plt.figure(figsize=(10, 6))
category_counts_society.plot(kind='bar', alpha=0.7)

# Add text annotations for each bar
for i, freq in enumerate(category_counts_society):
    plt.text(i, freq, str(freq), ha='center', va='bottom')

# Add plot labels and legend
plt.title('Binned Frequencies of Categorical Column')
plt.xlabel('Bin Number')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.show()
```

- Then we made box plots to check for outliers, and plotted histograms. We checked the skewness and kurtosis for the columns to justify the outliers noticed through the box plot.
- We finished task 1 by doing multivariate analysis of price with every other column.

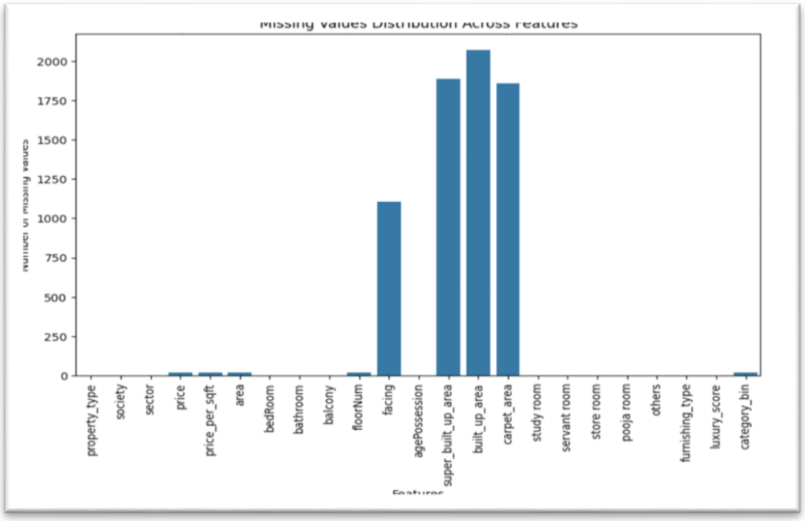
TASK-2: MISSING VALUES

- Firstly, we ran the command to check for missing values.



```
df.isnull().sum()

property_type          0
society                1
sector                 0
price                 18
price_per_sqft        18
area                  18
areaWithType           0
bedRoom                0
bathroom               0
balcony                0
floorNum              19
facing              1105
agePossession          0
super_built_up_area 1888
built_up_area       2070
carpet_area         1859
study room             0
servant room           0
store room             0
pooja room             0
others                 0
furnishing_type        0
luxury_score           0
dtype: int64
```

- Plot of missing values and columns.



- Upon detecting the missing values there are several ways in which we could handle the missing values:

1. Removing/deletion of missing values:

- This method removes the rows containing the missing values but upon doing so the size of the dataset reduces which reduces the size of the data which can be a problem for large datasets as it can remove valuable information.
- It can also lead to biased results if the nature of missingness is not random.

2. **Mean, Median, and Mode Imputation:**

- We did not use this method as it does not consider the relationships between variables which could lead to inaccuracy.

3. KNN Imputation and so on..

- KNN Imputation is the method we used to handle missing values in our dataset.
- KNN imputation considers the nearest neighbors which means the chance of introducing bias is lesser as compared to other methods.

HANDLING MISSING VALUES:

- We can drop a few columns that are unnecessary to us interms of predictions.
- We dropped the facing column as in Gurgaon people usually do not care about the facing of the houses/flats.

```
df.drop('facing', axis=1, inplace=True)
```

- **Then we observed that super_built_up_area,built_up_area, and carpet_area are dependent columns so we found out the ratios and handled the missing values of built_up_area.**

- **We calculated the ratio of carpet_area to built_up_area:**

```
carpet_to_built_up_ratio = (all_df['carpet_area']/all_df['built_up_area']).median()
```

- **We calculated the ratio of super_built_up_area to built_up_area:**

```
super_to_built_up_ratio = (all_df['super_built_up_area']/all_df['built_up_area']).median()
```

- Upon doing these we get the ratios.
- We then observed the rows which have **values of super_built_up_area and carpet_area but missing values for built_up_area information.**
- All the rows that fit the above conditions are put in **compare1_df.**

```
compare1_df = df[~(df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) &
~(df['carpet_area'].isnull())]
print(compare1_df)
```

- We use the .fillna function to handle the missing values of built_up_area. It uses the existing values in super_built_up_area and carpet_area to do this using the ratios we calculated earlier.

```
compare1_df['built_up_area'].fillna(round(((compare1_df['super_built_up_area']/1.105) +
(compare1_df['carpet_area']/0.9))/2),inplace=True)
```

- After updating our dataframe with compare1_df we observe that few missing values of builtup area have been handled but not completely
- We then observed the rows which have values of super_built_up_area but missing values for carpet_area and built_up_area information.

```
compare2_df = df[~(df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) &
(df['carpet_area'].isnull())]

compare2_df.head()
```

- After the required rows are isolated then we use the ratio of super_built_up_area to further handle the missing values of built_up_area.
- We update compare_2 also to our df.

```
compare2_df['built_up_area'].fillna(round(compare2_df['super_built_up_area']/1.105),inplace=True)
```

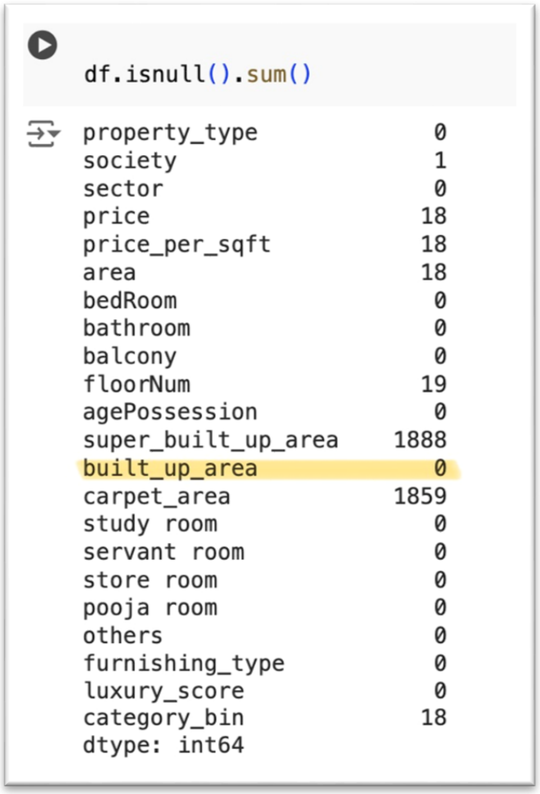
- We then observed the rows which have values of carpet_area but missing values for super_built_up_area and built_up_area information.
- All the rows are then isolated according to the conditions and stored in comp3.

```
comp3_df = df[(df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) &
~(df['carpet_area'].isnull())]
comp3_df.head()
```

- We use the ratio of carpet_area to further handle the missing values of built_up_area

```
comp3_df['built_up_area'].fillna(round(comp3_df['carpet_area']/0.9),inplace=True)
```

- After updating comp3 also to our df we then observe that all **missing values of built_up_area have become 0.**



```
df.isnull().sum()
```

```
property_type            0
society                  1
sector                   0
price                   18
price_per_sqft          18
area                    18
bedRoom                  0
bathroom                 0
balcony                  0
floorNum                19
agePossession            0
super_built_up_area   1888
built_up_area            0
carpet_area           1859
study room               0
servant room             0
store room               0
pooja room               0
others                   0
furnishing_type          0
luxury_score             0
category_bin            18
dtype: int64
```

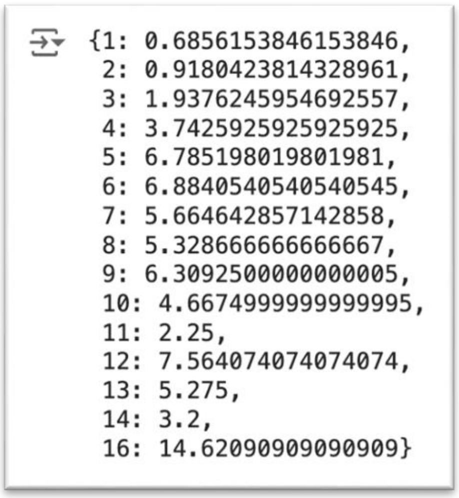
- Carpet area is the area that is used by the owner of the house whereas the built-up area includes the areas covered by walls or exclusive balconies
- Super built-up area casts the net wider to include common areas as well
- As super_built_up_area and carpert_area are not very important features that are looked at while pruchasing a house we can **drop both the columns.**

HANDLING MISSING VALUES OF PRICE

- Initially, we performed KNN imputation on the price column too, but upon observing, we found that the missing values were not replaced accurately enough, so we compared the number of bedrooms to their prices to handle the missing values.
- We took the means of prices in relation to number of bedrooms.

```
bedroom_prices = {}
for num_bedrooms in [1,2,3, 4, 5, 6,7,8, 9,10,11, 12,13,14,16]:
bedroom_prices[num_bedrooms] = df[df['bedRoom'] == num_bedrooms]['price'].mean()

bedroom_prices
```

- It can be observed in the output of the above code the means of price corresponding to the number of bedrooms. For example, the mean of prices of all 2 Bedroom properties is 0.91 so if there happens to be a missing value in the price column for the row which has 2 in the bedroom column the missing value would be replaced with 0.91.



```
{1: 0.6856153846153846,
 2: 0.9180423814328961,
 3: 1.9376245954692557,
 4: 3.7425925925925925,
 5: 6.785198019801981,
 6: 6.884054054054054‌5,
 7: 5.664642857142858,
 8: 5.328666666666667,
 9: 6.3092500000000005,
 10: 4.6674999999999995,
 11: 2.25,
 12: 7.564074074074074,
 13: 5.275,
 14: 3.2,
 16: 14.620909090909090}
```

```
for num_bedrooms in [2, 4, 5, 6, 9, 12, 16]:
mean_price = bedroom_prices.get(num_bedrooms, 0) # Get the mean price for the current number of bedrooms or 0 if not found
mean_price = round(mean_price, 2) # Round mean_price to 2 decimal places
df.loc[df['bedRoom'] == num_bedrooms, 'price'] = df.loc[df['bedRoom'] == num_bedrooms, 'price'].fillna(mean_price)
```

- Using the above code, we were able to handle all the missing values of price.

SEPERATION OF DATA INTO NUMERICAL AND CATEGORICAL:

- We separated the data into numerical and categorical to make it easier for us to handle the missing values.

NUMERICAL:

- We used KNN imputation to handle missing values of numerical data.

```
imputer = KNNImputer(n_neighbors=10)
impute_num_data = pd.DataFrame(imputer.fit_transform(num_data), columns=num_data.columns)
```

- This method checks the nearest neighbors of the missing values and updates the missing values to something that is close enough.
- We chose to use this method as it works well for large datasets.

CATEGORICAL:

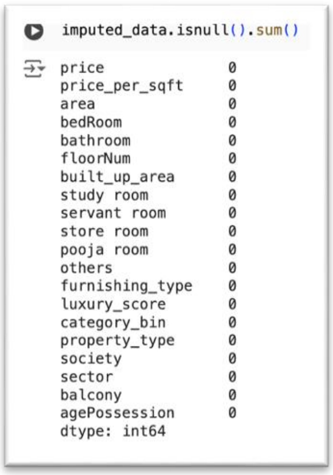
- We used the SimpleImputer. If a missing value (NaN) is encountered in a categorical column, it gets replaced with the category that appears most often within that specific column.

```
categorical_imputer= SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

- We chose to do this as the number of missing values in categorical data was very less, so this seemed the best method to handle the missing values.

In the end we concatenated both numerical and categorical data and updated our dataset to the imputed values.

→ **This was how we managed to handle the missing values present in the data set.**



```
imputed_data.isnull().sum()

price              0
price_per_sqft     0
area               0
bedRoom            0
bathroom           0
floorNum           0
built_up_area      0
study room         0
servant room       0
store room         0
pooja room         0
others             0
furnishing_type    0
luxury_score       0
category_bin       0
property_type      0
society            0
sector             0
balcony            0
agePossession      0
dtype: int64
```

TASK 3&4: OUTLIER DETECTION AND HANDLING

- We used **the IQR (INTER QUARTILE RANGE) method** for outlier detection.
- The IQR method is less sensitive to outliers compared to methods which use mean or standard deviation and is very useful for data having skewed distributions.
- IQR sets a clear boundary for the outliers. Data points falling outside range set (typically 1.5 times the IQR below Q1 or above Q3) would be considered as outliers.

1. We converted all categorical data into numerical data by using OneHotEncoder as we cannot do outlier handling on categorical data.
2. Then we did column wise detection and handling of outliers.

```
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df['price'] < lower_bound) | (df['price'] > upper_bound)]

num_outlier = outliers.shape[0]
outliers_price = outliers['price'].describe()

num_outlier,outliers_price
```

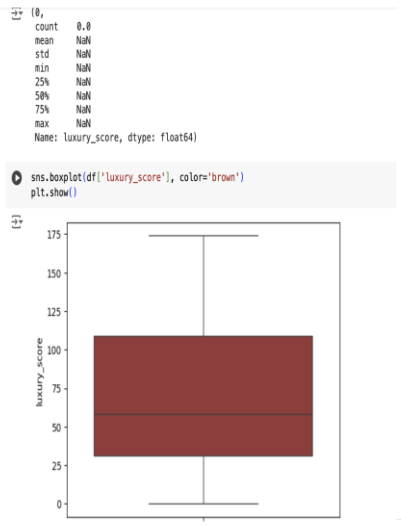
CODE USED FOR DETECTION OF OUTLIERS USING IQR METHOD

```
def handle_outliers(df, cols):
for col in cols:
Q1 = df[col].quantile(0.25)
Q3 = df[col].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df[col] = np.where((df[col] < lower_bound) | (df[col] > upper_bound), df[col].median(), df[col])
return df
```

CODE USED FOR HANDLING OUTLIERS

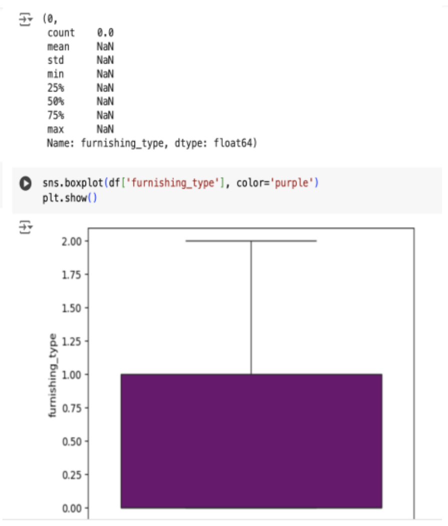
ANALYSIS OF OUTLIERS:

- In the servant room,luxury_score and furnishing type 0 outliers were detected as we can observe in their box plot as well as after running IQR method.



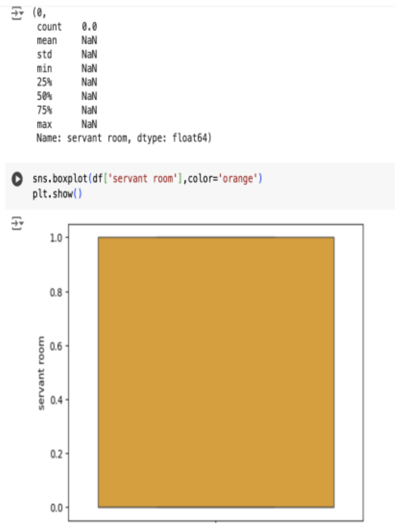
```
(0,
 count    0.0
 mean     NaN
 std      NaN
 min      NaN
 25%      NaN
 50%      NaN
 75%      NaN
 max      NaN
 Name: luxury_score, dtype: float64)
```

```
sns.boxplot(df['luxury_score'], color='brown')
plt.show()
```



```
(0,
 count    0.0
 mean     NaN
 std      NaN
 min      NaN
 25%      NaN
 50%      NaN
 75%      NaN
 max      NaN
 Name: furnishing_type, dtype: float64)
```

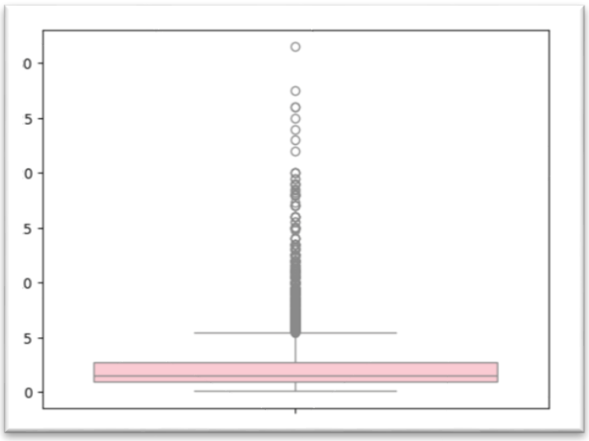
```
sns.boxplot(df['furnishing_type'], color='purple')
plt.show()
```

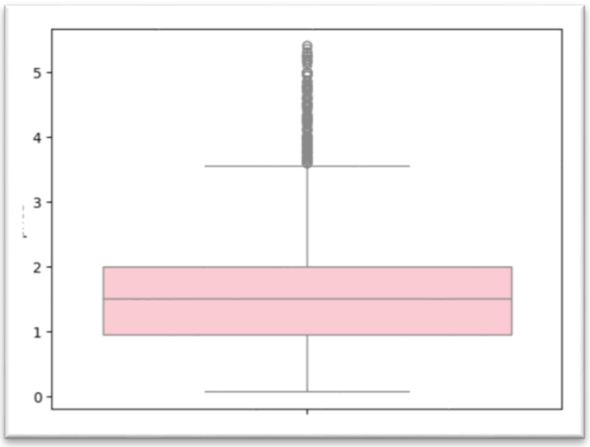


```
(0,
 count    0.0
 mean     NaN
 std      NaN
 min      NaN
 25%      NaN
 50%      NaN
 75%      NaN
 max      NaN
 Name: servant room, dtype: float64)
```

```
sns.boxplot(df['servant room'],color='orange')
plt.show()
```

- For all the other numerical data columns the outliers have been reduced but not completely 0. For example, in the price column we initially had 435 outliers after using the code for handling the number of outliers was 254.

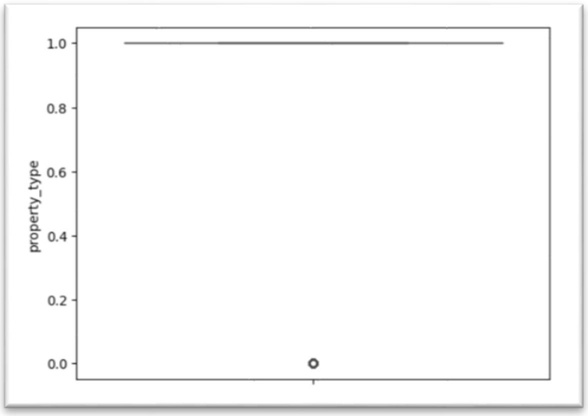


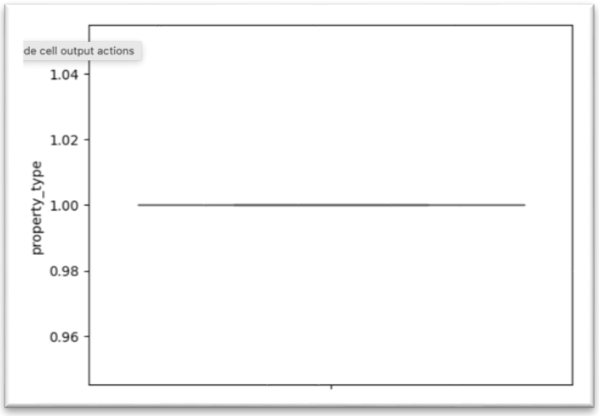


- **<u>Before and after of box plot of price column.</u>** The Price column has some genuine outliers, but we observed a few data errors as well.
- As reducing the outliers to 0 is not always the best case we felt the method we used to handle outliers was the best we were able to do.
- The same applied to the other numerical data.
- **The leftover outliers observed in the boxplot after handling are <u>mostly individual houses or villas as we did not separate houses from our dataset.</u>**

- Now speaking of **categorical data** that we converted into numerical data using **OneHotEncoder.**
- **After performing onehotencoding the dimensionality of dataset increased as number of columns increased to 805.**
- The columns being ['property_type', 'society', 'sector', 'balcony', 'agePossession']
- The columns age possession and sector did not have any outliers.
- The outliers of the other three columns were handled completely making the number of outliers to be 0.

BOX PLOT EXAMPLE OF PROPERTY TYPE:





→ **In this way we handled the outliers present in our dataset**.

Implications of Missing Values and Outliers on Data Analysis:

- Missing values and outliers have a lot of significance in impacting our data.

Missing values:

- Firstly, we opted not to delete the rows containing the missing values in our dataset as it would result in loss of information and an increase in bias.
- We chose KNN imputation to handle missing values as it was the best fit for our dataset. KNN considers the features (columns) of neighboring data points to estimate missing values. This helps us maintain the relationships between the columns intact.
- We did not use meaning or mode as they fail to capture the underlying relationships.

Outliers:

- Outliers can violate modeling assumptions, such as normality, leading to incorrect inferences and biased estimates and they can also mask relationships between the columns if any are present. Therefore, it was important to detect and handle them.
- We chose IQR method to do this over the Z-Score method as x Z-Score method relies on the mean and standard deviation, which can be significantly influenced by outliers themselves. Since our data has a lot of outliers, which were visualized using plots, the mean and standard deviation will be skewed, potentially leading to underestimating the true spread of the data and missing genuine outliers.
- We did not use standard deviation for handling outliers as all of them were becoming zero, which would not be ideal as few outliers may contain valuable information and we would have lost it.

→ **Hence these were the strategies we used for data cleaning and preprocessing.**