Predictive Analytics Project Report

- **Module Code:** IS6052
- **Module Title:** Predictive Analytics
- **Student Name:** Punit Sharma
- **Student ID:** 123115328
- **Submission Date:** 02-Dec-2024

Course: IS6052 Project Report

Contents

Introduction

With focus on the Irish context and the particularity of the Irish homebuyers, this report starts by outlining the difficulties and decisions related to purchasing homes in Dublin, with an emphasis on the most relevant criteria of price, location, BER and size. These variables, which are located within a global frame, have added some uncertainty in the Irish property market, most significantly in Dublin city's different housing segments. These factors must be explored in the study and assessed quantitatively using predictive analytics strategies such that stakeholders may make informed decisions that promote innovation.

This project uses data science approaches such as data cleaning, data feature creation, and data prediction using a structured dataset of the property characteristics and prices. Linear Regression, Random Forest, Neutral Network models are applied and calibrated to reveal promising trend analysis and make accurate predictions on sales of properties.

The findings of this report include analysis of the dataset, relationship analysis of property prices, and an assessment of the performance of the various models. As a result, Random Forest models turn out as the best performing models with a good level of interpretability and accuracy. After identifying critical paths to predictive analytics applications in the Dublin housing market context, several recommendations for further research and ideas for future work to refine the methods are provided in the report.

Dataset Description and Initial Exploration

1. Description of the Dataset

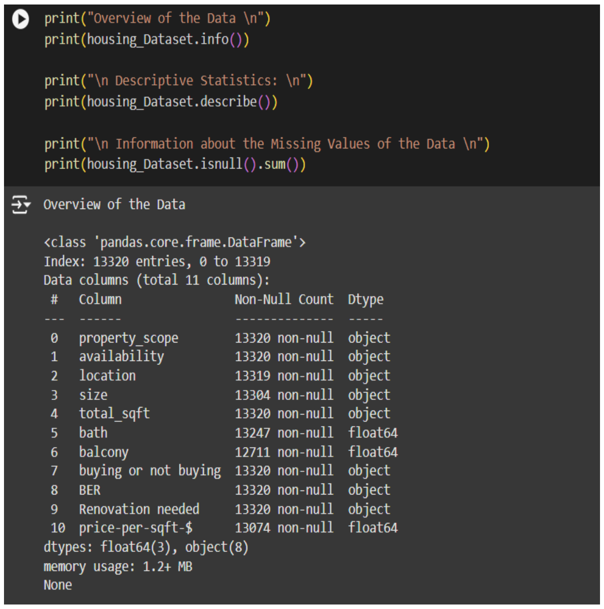
The information from the dataset used in this project includes but is not limited to house prices in Ireland, number of rooms, number of balconies, availability, need for renovation and BER, size etc. These variables were chosen since they had been acknowledged to exert influence on the buying behaviour of potential buyers in Dublin. Another advantage offered by the dataset is its separation into localities – the Dublin City Council (DCC), Fingal, Dun Laoghaire, and South Dublin – in which geographical effects on house pricing can be compared and studied.

Besides, the dataset also includes quantitative features such as size of a property in square ft. and the cost and qualitative features such as availability and location. The structure of it is ideal for tasks of a predictive nature where many types of machine learning algorithms can be applied.

2. Exploratory Data Analysis (EDA)

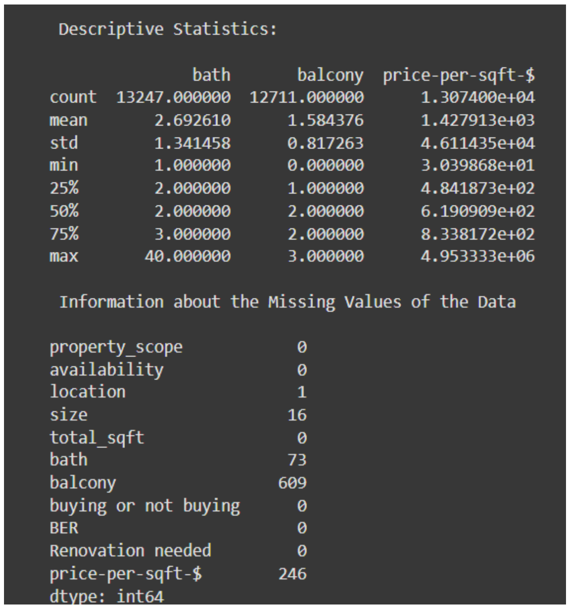
Exploratory data analysis was done so as to gain insights regarding the data and hidden structure within a set. Quantitative descriptions offered mean measures of size, range of prices, and geographical dispersion. It was easy to identify clusters by performing exercises like a boxplot to present prices of the houses and how they fell within the €400,000 – €600,000 bracket.

In particular, the properties of the objects were compared using the diagrams that revealed linear relationship indicators such as size, price, etc. Heatmaps were used to address issues with multicollinearity, which indicate how closely related specific features are. For example, the study established that location was an important factor in determining the price, which indicated significant differences between the observed local authorities.



```
print("Overview of the Data \n")
print(housing_Dataset.info())

print("\n Descriptive Statistics: \n")
print(housing_Dataset.describe())

print("\n Information about the Missing Values of the Data \n")
print(housing_Dataset.isnull().sum())
```

```
Overview of the Data

<class 'pandas.core.frame.DataFrame'>
Index: 13320 entries, 0 to 13319
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   property_scope      13320 non-null  object
 1   availability        13320 non-null  object
 2   location            13319 non-null  object
 3   size                13304 non-null  object
 4   total_sqft          13320 non-null  object
 5   bath                13247 non-null  float64
 6   balcony             12711 non-null  float64
 7   buying or not buying  13320 non-null  object
 8   BER                 13320 non-null  object
 9   Renovation needed   13320 non-null  object
 10  price-per-sqft-$    13074 non-null  float64
dtypes: float64(3), object(8)
memory usage: 1.2+ MB
None
```



```
 Descriptive Statistics:

               bath       balcony  price-per-sqft-$
count  13247.000000  12711.000000      1.307400e+04
mean       2.692610      1.584376      1.427913e+03
std        1.341458      0.817263      4.611435e+04
min        1.000000      0.000000      3.039868e+01
25%        2.000000      1.000000      4.841873e+02
50%        2.000000      2.000000      6.190909e+02
75%        3.000000      2.000000      8.338172e+02
max       40.000000      3.000000      4.953333e+06

 Information about the Missing Values of the Data

property_scope            0
availability              0
location                  1
size                     16
total_sqft                0
bath                     73
balcony                 609
buying or not buying      0
BER                       0
Renovation needed         0
price-per-sqft-$        246
dtype: int64
```

Figure 1: Data Description and Information, it explains the data type, number of null values and Column Datatype

Figure 1 illustrates the information of a dataset named housing_Dataset. This dataset is a pandas DataFrame and has 13320 events with 11 features. The first image shows the use of the **info()** method of the DataFrame, which gives an initial description of data in the dataframe and describes the data type of each column along with telling about the non-null value count. It then prints out the **describe()** method which gives a general statistics of numerical features in the dataset. Last of all, it displays the count of the missing valuers in each of the Features using the **isnull().sum** function.

Here is a summary of the information:

- It has been seen that the dataset used in this study contains 13320 entries.
- In the dataset there are 11 columns in total.
- The columns are: property location, availability, area, total area, number of bath, whether to buy or not to buy BER, balcony needed for renovation, and price per square $ dollar.
- The data types of the columns are: Object (8 attributes), float (3 attributes).
- Records of location, size, bath and price per sqft $ contain some missing values.

Some of the columns are property scope, availability, property location, size, total floor area, no of bathrooms, BER rating, balcony, if the house requires renovation and price per sqft.

There might be values missing in location, size, bath, and price-per-sqft-$ features that might require prior sanitation once it has been used in analysis.

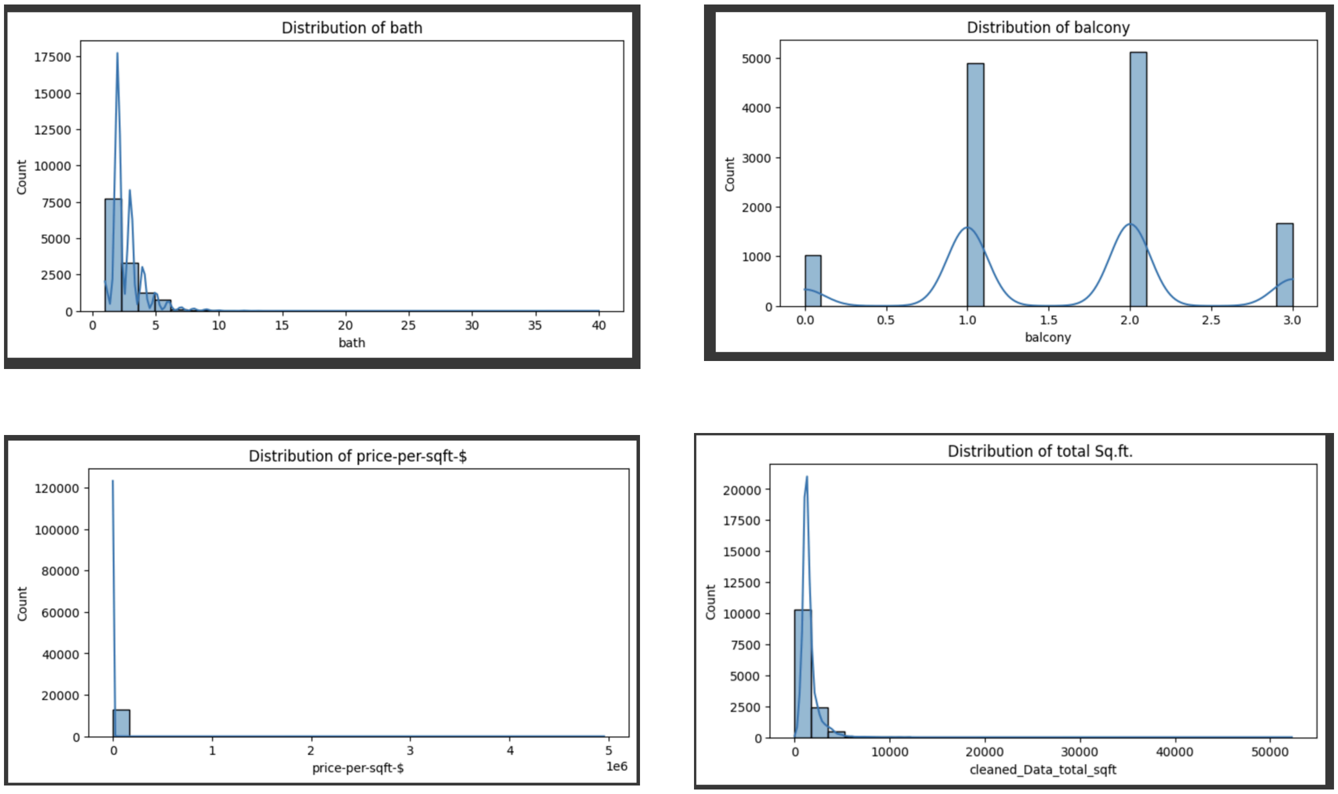


Figure 2: Shows the distribution of the data using the histplot

Figure 2, illustrates the histograms for four numerical columns in a housing dataset: bath, balcony, price-per-sqft-s, and total_sqft.

Distribution of bath:

- The histogram shows the number of houses according to the number of bathroom.
- The largest range is from 1 to 5 bathrooms in most of the houses.
- There are several houses that have more bathrooms than the average, but this declines very sharply thereafter.

Distribution of balcony:

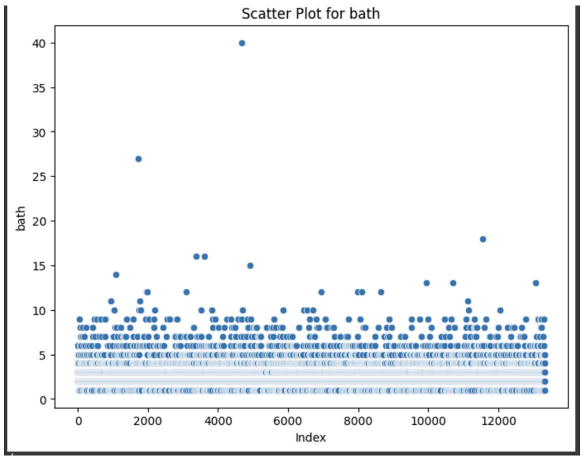
- This histogram shows how many of the surveyed houses have a balcony, using a 0 for no and a 1 for yes.
- Most of the houses have a balcony – all have a value of 1.
- Of them, there are a few houses that have a balcony value of 0.

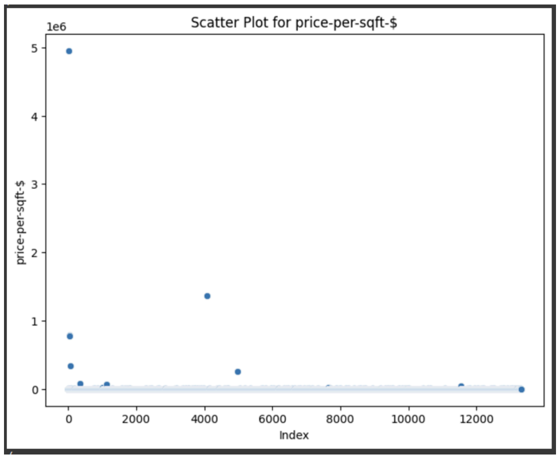
Distribution of price-per-sqft-$:

- This histogram shows the distribution of the price per square foot of the houses.
- In this case of data distribution skew is very high on the right side which means the majority of the houses will be priced lower per square foot area and only a very few will be very highly prized per square foot.

Distribution of total Sq.ft.:

- This histogram shows the distribution of the total square footage of the houses.
- The distribution is also heavily skewed to the right, indicating that most houses have a smaller total square footage, while a few houses have a much larger total square footage.





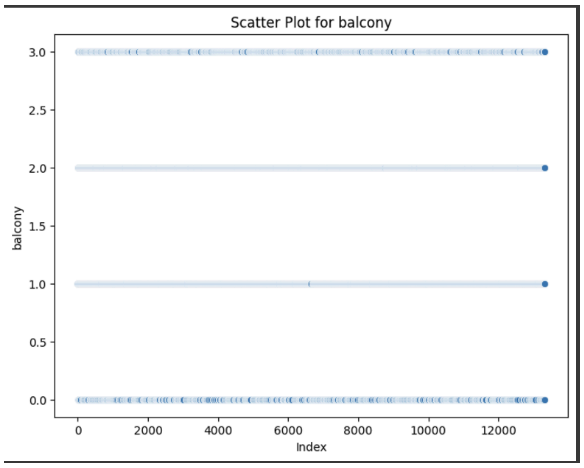


Figure 3: Shows the Scatter plot of the numerical Data

Figure 3, illustrates the scatter plots for three numerical columns in a housing dataset: bath, balcony, and price-per-sqft-s. Let's analyze each scatter plot individually:

Scatter Plot for bath:

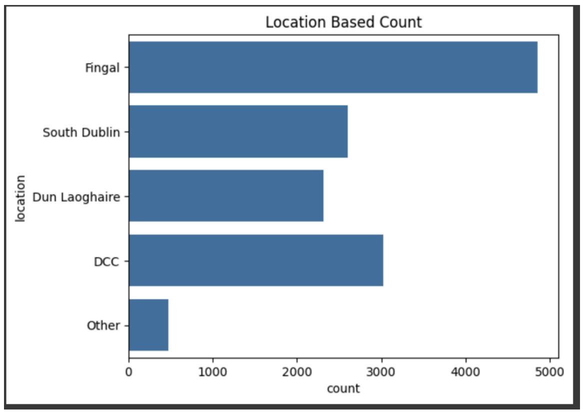
- According to the scatter plot, the index increases as the number of bathrooms also increases.
- This implies that as the index rises then the number of bathrooms in the different houses also tends to rise.
- But the correlation is not very close as the points are scattered a lot.

Scatter Plot for balcony:

- There are four clear horizontal hyperplane values indicated at the axes 0, 1, 2 and 3 on the scatter plot.
- This means that the balcony column is a dummy variable which can only take value from 0 to 3.
- Most of the houses have a balcony, which have been valued 1.

Scatter Plot for price-per-sqft-$:

- Looking at the scatter plot of the index and the price per square in feet, it shows that as the index increases the price advances proportionally.
- From this, it can be deduced that more so, as the index rises, the price per square feet of the houses also rises.
- As shown the dependence is not very close because the points are scattered quite a lot.



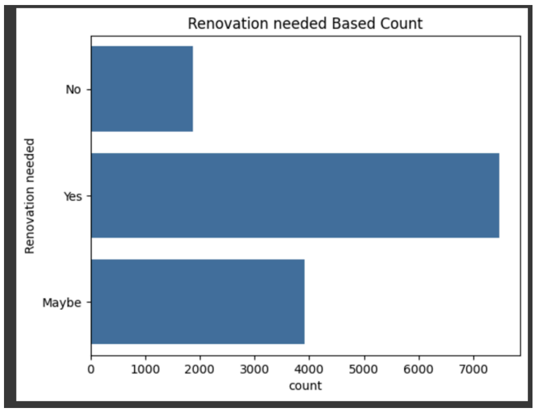


Figure 4: Horizontal bar graph for Location and Renovation needed for houses

In Figure 4, The bar chart titled **"Renovation needed Based Count"** displays the distribution of houses based on whether they need renovation.

- Majority Need Renovation: The scale at the top of each bar represents percentage and the largest bar belongs to 'Yes' meaning that many of the houses need some form of renovation in the dataset.

- Some Need Maybe: The second largest bar is of ‘Maybe’ implying that as per the houses that might require a renovation; there could be a large number, but the extent is not clearly known.
- Few Don't Need Renovation: The shortest bar is for ‘No’ meaning that compared to the other categories, only relatively small proportion houses are in good condition and do not need to be renovated.

Therefore, the chart shows that renewal is a widespread requirement for the houses in the dataset, and a significant number of them require large-scale operations.

The bar chart titled **"Location Based Count"** shows the distribution of houses across different locations in Dublin, Ireland.

Here are the observations from the chart:

- Fingal Dominates: Fingal has the largest number of houses, and it has a higher number compared to other geographical locations.
- South Dublin and Dun Laoghaire: There is considerable representation of first and second generation immigrants in both areas with the representation slightly lower in Dun Laoghaire than in South Dublin.
- DCC and Other: The counts for DCC and the “Other” category are comparatively small to the top three places.

In conclusion, the chart presented below shows that Fingal is the most frequent location within the dataset and is followed by South Dublin and Dun Laoghaire. This distribution suggests that there is a higher housing stock concentration in these areas compared to DCC and other areas.

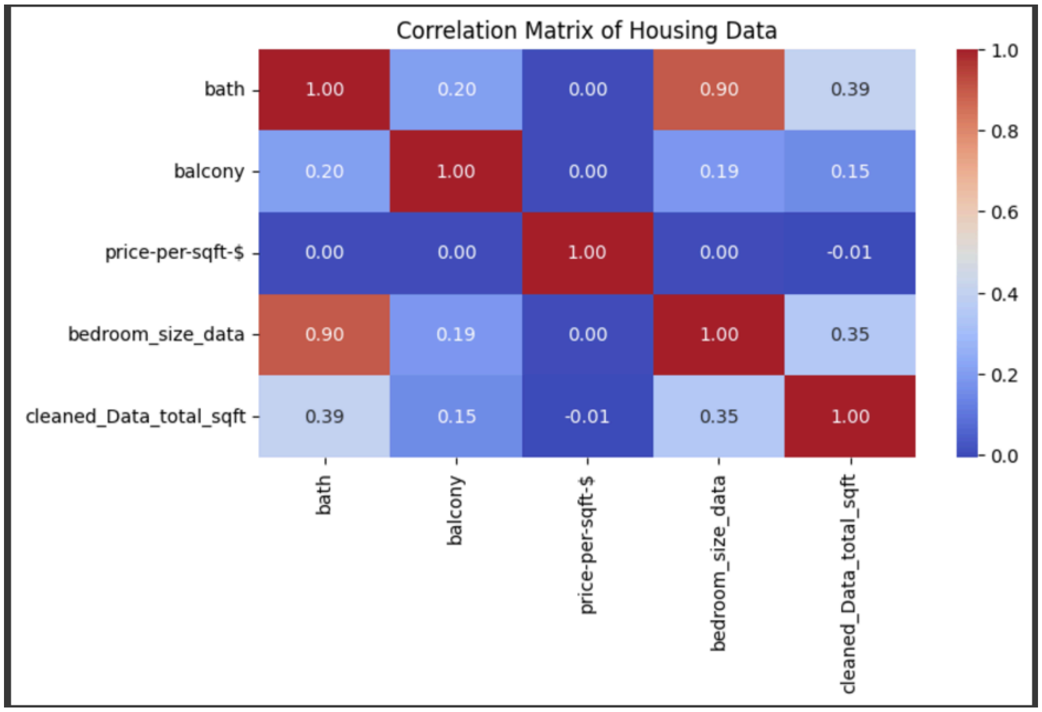


Figure 5: Correlation matrix for numerical columns of the dataset

Figure 5, illustrates the visual representation of the relation of different numerical variables in the housing dataset between each other. Here's what we can interpret from the matrix:

Positive Correlations:

bath and bedroom_size_data: In fact, houses with more bathrooms have larger bedrooms (0.90 correlation is strong positive). It's intuitive: the larger the house, the more rooms there will be.

bath and cleaned_Data_total_sqft: It is a moderate positive correlation (0.39) of the total square footage being greater on houses with more bathrooms. That's probably because larger houses have more rooms, like bathrooms.

Negative Correlations:

price-per-sqft-$ and cleaned_Data_total_sqft: This indicates very weak negative correlation (-0.01) which means that price per square foot and the total square footage does not have any noise. There could be many reasons behind the price of the house, like location, it's age and amenities.

Weak or No Correlations:

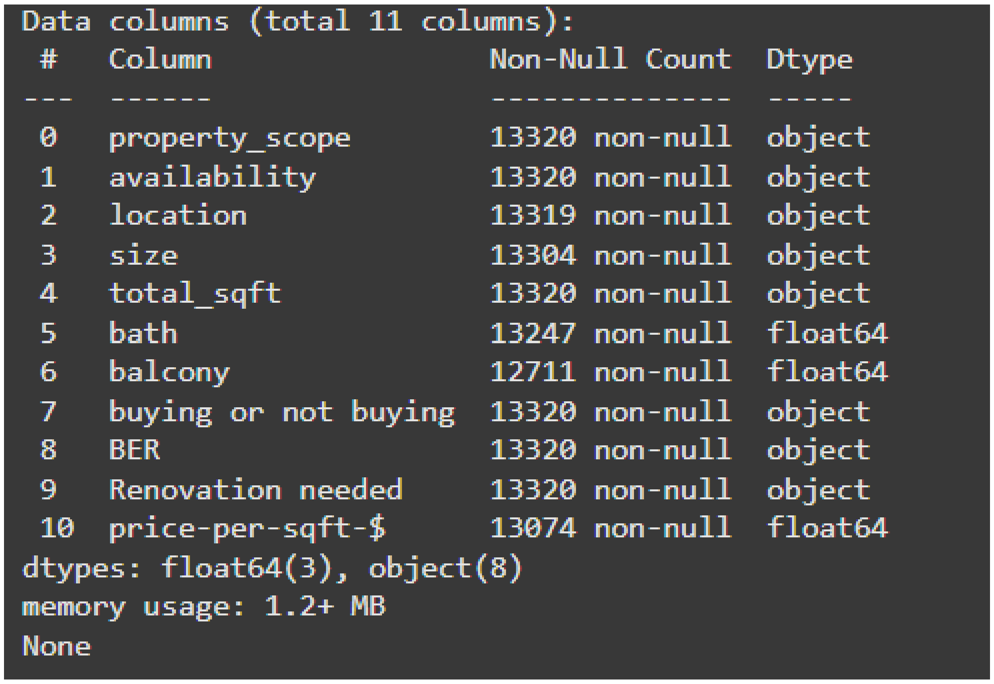
balcony and other variables: Very weak correlations are seen between the balcony variable and all other variables indicating that having a balcony does not have a large impact on the number of bathrooms, bedrooms, the total square footage, or the price per square foot.

Data Preparation and Feature Engineering

1. Data Cleansing Steps

Data cleansing was a critical step in preparing the dataset for analysis to ensure data quality and reliability for predictive modeling. The following steps were undertaken:

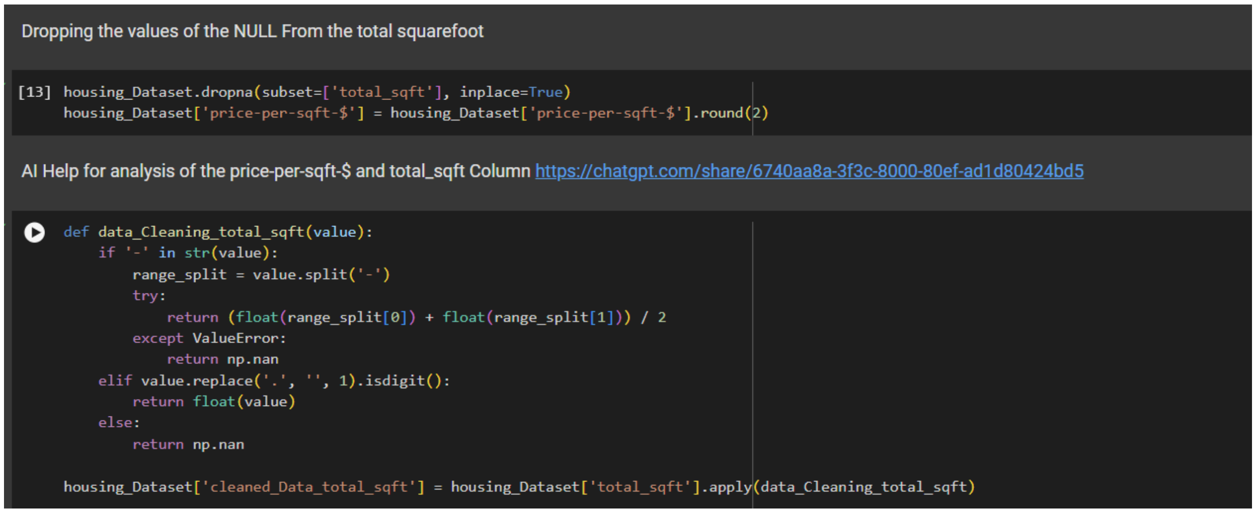
a. Standardizing Data Formats:



```
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   property_scope        13320 non-null  object
 1   availability          13320 non-null  object
 2   location              13319 non-null  object
 3   size                  13304 non-null  object
 4   total_sqft            13320 non-null  object
 5   bath                  13247 non-null  float64
 6   balcony               12711 non-null  float64
 7   buying or not buying  13320 non-null  object
 8   BER                   13320 non-null  object
 9   Renovation needed     13320 non-null  object
 10  price-per-sqft-$      13074 non-null  float64
dtypes: float64(3), object(8)
memory usage: 1.2+ MB
None
```

Figure 6: Details about columns and Datatype of columns

Figure 6, From the observations from figure 6, the columns like bath, balcony and price-per-sqft-$ are float type and after observations of the head() of the dataset we found that the size (represents the number of rooms) and total_sqft is also float type. First we converted both columns to a float datatype.



Dropping the values of the NULL From the total squarefoot

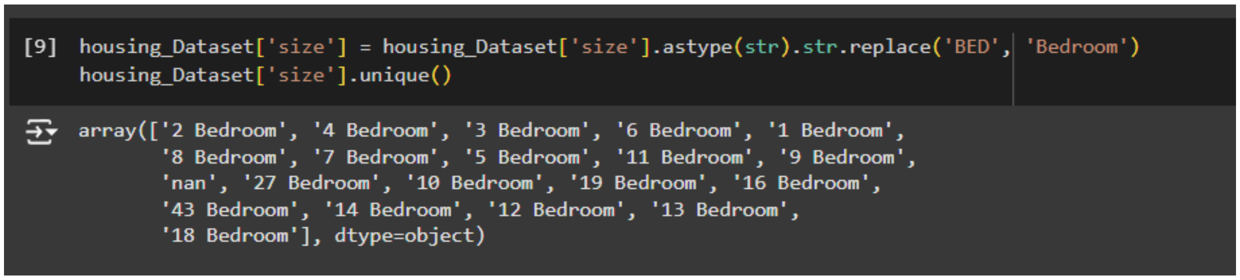
```
[13] housing_Dataset.dropna(subset=['total_sqft'], inplace=True)
     housing_Dataset['price-per-sqft-$'] = housing_Dataset['price-per-sqft-$'].round(2)
```

AI Help for analysis of the price-per-sqft-$ and total_sqft Column https://chatgpt.com/share/6740aa8a-3f3c-8000-80ef-ad1d80424bd5

```
def data_Cleaning_total_sqft(value):
    if '-' in str(value):
        range_split = value.split('-')
        try:
            return (float(range_split[0]) + float(range_split[1])) / 2
        except ValueError:
            return np.nan
    elif value.replace('.', '', 1).isdigit():
        return float(value)
    else:
        return np.nan

housing_Dataset['cleaned_Data_total_sqft'] = housing_Dataset['total_sqft'].apply(data_Cleaning_total_sqft)
```

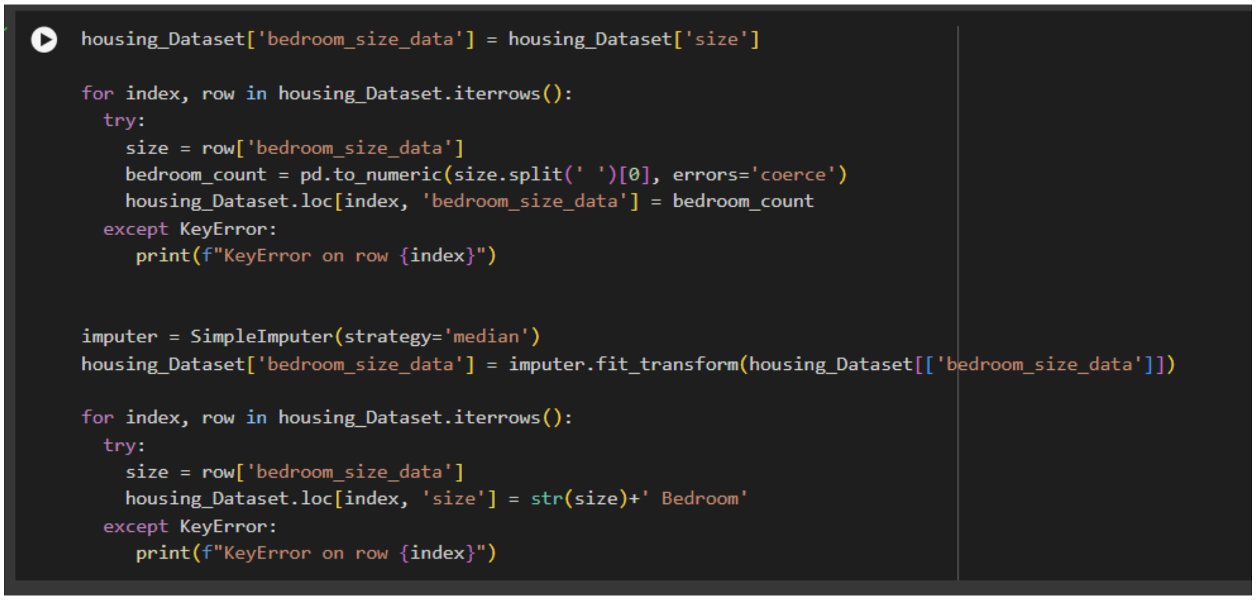
Figure 7: Cleaning of total_sqft. Column

As illustrated in the figure 7, the issue with this column was that the column contained the string type data which was difficult to use in numerical operations, we used string function to remove the (“-”) from the data and took the mean to replace the (“404 - 504”) type of the data and change the datatype of the data to float from string. Here I was facing difficulties in cleaning this column data so I took help of chatGPT for this column logic (*ChatGPT - Random Forest Prediction Guide*, no date).



```
[9] housing_Dataset['size'] = housing_Dataset['size'].astype(str).str.replace('BED', 'Bedroom')
    housing_Dataset['size'].unique()
```

```
array(['2 Bedroom', '4 Bedroom', '3 Bedroom', '6 Bedroom', '1 Bedroom',
       '8 Bedroom', '7 Bedroom', '5 Bedroom', '11 Bedroom', '9 Bedroom',
       'nan', '27 Bedroom', '10 Bedroom', '19 Bedroom', '16 Bedroom',
       '43 Bedroom', '14 Bedroom', '12 Bedroom', '13 Bedroom',
       '18 Bedroom'], dtype=object)
```



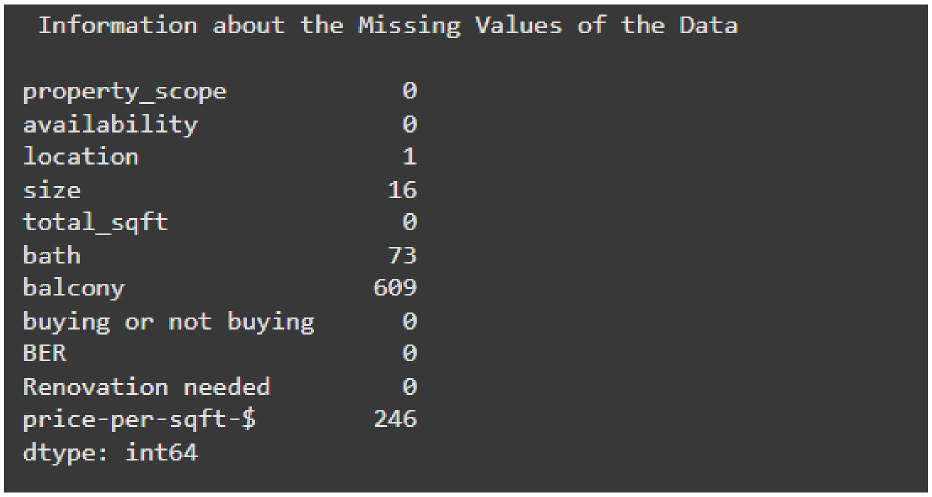
```
housing_Dataset['bedroom_size_data'] = housing_Dataset['size']

for index, row in housing_Dataset.iterrows():
  try:
    size = row['bedroom_size_data']
    bedroom_count = pd.to_numeric(size.split(' ')[0], errors='coerce')
    housing_Dataset.loc[index, 'bedroom_size_data'] = bedroom_count
  except KeyError:
     print(f"KeyError on row {index}")


imputer = SimpleImputer(strategy='median')
housing_Dataset['bedroom_size_data'] = imputer.fit_transform(housing_Dataset[['bedroom_size_data']])

for index, row in housing_Dataset.iterrows():
  try:
    size = row['bedroom_size_data']
    housing_Dataset.loc[index, 'size'] = str(size)+' Bedroom'
  except KeyError:
     print(f"KeyError on row {index}")
```

Figure 8: Cleaning of Size Column

As seen in Figure 8, the "size" column contained both numeric and string components. We extracted the numeric part for the cleaning and standardization of this column and converted it into a numerical type. This numeric part had missing values which were imputed with the median value. We then made a new column “bedroom_size_data” to save these numeric value. We also standardized the size column by replacing the “BED” and “bedroom” with a standardizing “Bedroom” string.

b. Handling Missing Data:

Handling missing values is a really difficult task and the most time consuming for me. To to handle the missing values I used a unique approach.



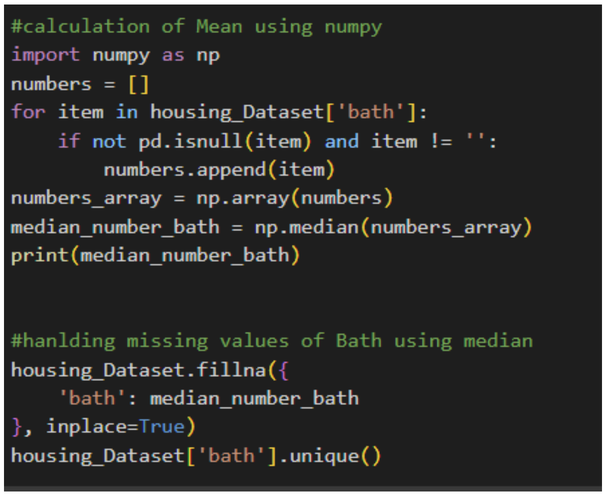
```
 Information about the Missing Values of the Data

property_scope          0
availability            0
location                1
size                   16
total_sqft              0
bath                   73
balcony               609
buying or not buying    0
BER                     0
Renovation needed       0
price-per-sqft-$      246
dtype: int64
```

Figure 9: Data about the missing values

Size column:

As illustrated in figure 8, I used the median value to handle the missing value of the size column.

Bath Column:



```
#calculation of Mean using numpy
import numpy as np
numbers = []
for item in housing_Dataset['bath']:
    if not pd.isnull(item) and item != '':
        numbers.append(item)
numbers_array = np.array(numbers)
median_number_bath = np.median(numbers_array)
print(median_number_bath)


#hanlding missing values of Bath using median
housing_Dataset.fillna({
    'bath': median_number_bath
}, inplace=True)
housing_Dataset['bath'].unique()
```

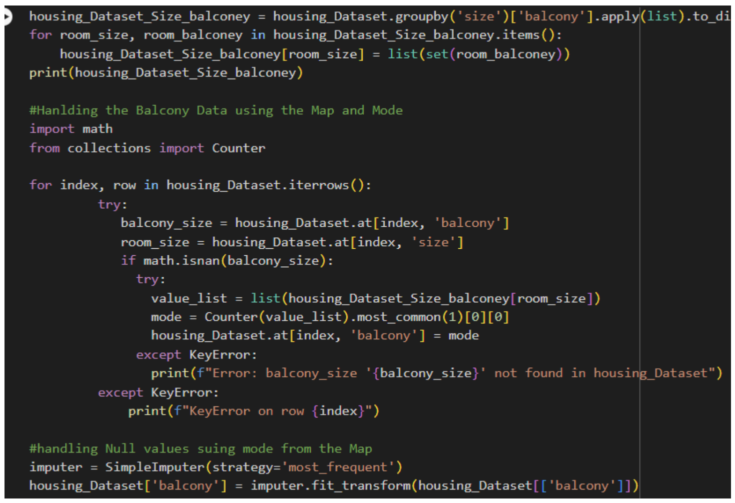
Figure 10: Handling of missing values of the bath

As shown in the figure 10, I used the median to fill the missing values of the bath column.

Location Column:

As location has only one missing value I simply dropped the null location record.

Balcony Column:

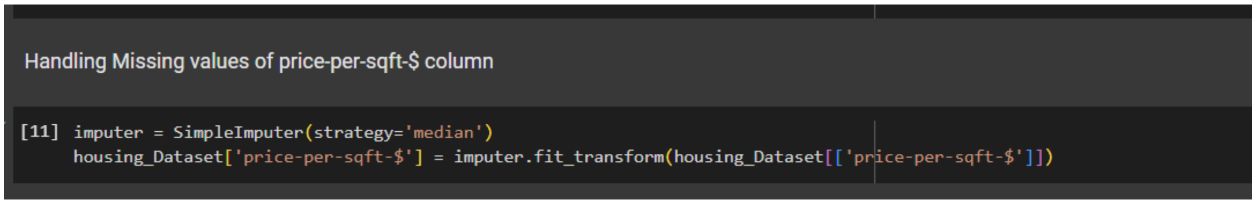


```
housing_Dataset_Size_balconey = housing_Dataset.groupby('size')['balcony'].apply(list).to_di
for room_size, room_balconey in housing_Dataset_Size_balconey.items():
    housing_Dataset_Size_balconey[room_size] = list(set(room_balconey))
print(housing_Dataset_Size_balconey)

#Hanlding the Balcony Data using the Map and Mode
import math
from collections import Counter

for index, row in housing_Dataset.iterrows():
         try:
            balcony_size = housing_Dataset.at[index, 'balcony']
            room_size = housing_Dataset.at[index, 'size']
            if math.isnan(balcony_size):
              try:
                value_list = list(housing_Dataset_Size_balconey[room_size])
                mode = Counter(value_list).most_common(1)[0][0]
                housing_Dataset.at[index, 'balcony'] = mode
              except KeyError:
                print(f"Error: balcony_size '{balcony_size}' not found in housing_Dataset")
         except KeyError:
             print(f"KeyError on row {index}")

#handling Null values suing mode from the Map
imputer = SimpleImputer(strategy='most_frequent')
housing_Dataset['balcony'] = imputer.fit_transform(housing_Dataset[['balcony']])
```

Figure 10: Handling Balcony Column

As for the balcony I changed the logic, as the balcony has 4 different values i.e. 0, 1, 2 and 3 only predicting and putting the median or mode was not possible. So I used a map depending upon the room size and number of balconies for that house. I put the value of the balcony by mapping the room number of that house and the mode of the balcony value from the map of room and balcony. I know it's difficult to explain in words, please refer to the code for this. This really helped me to improve the accuracy of my model.

Price per Sq.ft. Column:



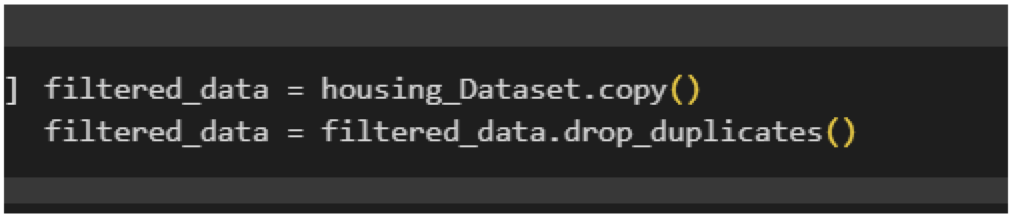
Handling Missing values of price-per-sqft-$ column

```
[11] imputer = SimpleImputer(strategy='median')
     housing_Dataset['price-per-sqft-$'] = imputer.fit_transform(housing_Dataset[['price-per-sqft-$']])
```

Figure 11: Handling missing values for price-per-sqft-$ Column

To handle the missing value for this column I simply put the median as the price can vary depending upon alot of factors so I use median which can be near to real value for this column.

c. Removing Duplicates:

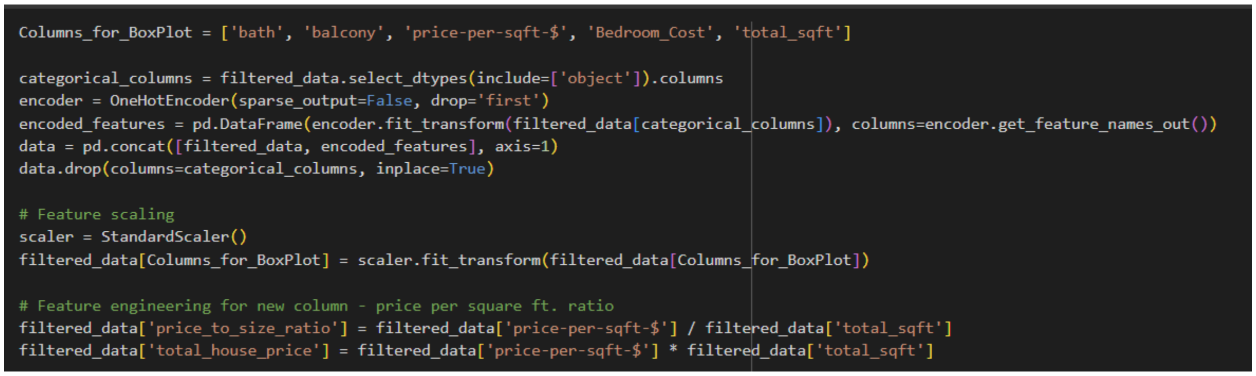


```
] filtered_data = housing_Dataset.copy()
  filtered_data = filtered_data.drop_duplicates()
```

Figure 12: Handling the duplicate records.

To handle the duplicate records, I created a separate dataset filtered_data which keeps the unique and filtered data.

2. Feature Engineering Techniques



```
Columns_for_BoxPlot = ['bath', 'balcony', 'price-per-sqft-$', 'Bedroom_Cost', 'total_sqft']

categorical_columns = filtered_data.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(sparse_output=False, drop='first')
encoded_features = pd.DataFrame(encoder.fit_transform(filtered_data[categorical_columns]), columns=encoder.get_feature_names_out())
data = pd.concat([filtered_data, encoded_features], axis=1)
data.drop(columns=categorical_columns, inplace=True)

# Feature scaling
scaler = StandardScaler()
filtered_data[Columns_for_BoxPlot] = scaler.fit_transform(filtered_data[Columns_for_BoxPlot])

# Feature engineering for new column - price per square ft. ratio
filtered_data['price_to_size_ratio'] = filtered_data['price-per-sqft-$'] / filtered_data['total_sqft']
filtered_data['total_house_price'] = filtered_data['price-per-sqft-$'] * filtered_data['total_sqft']
```

Figure 13: Feature Scaling and Engineering

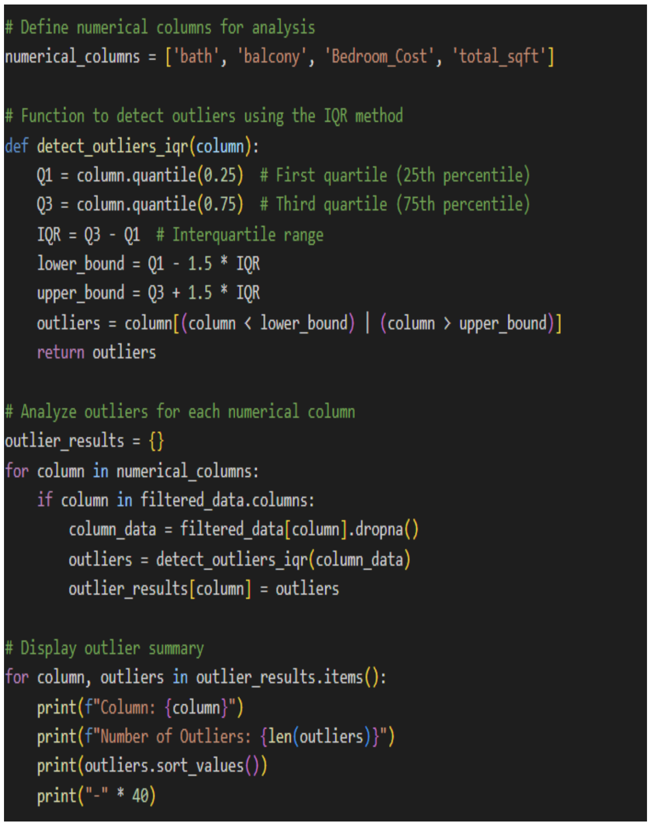
For feature engineering to support the accuracy of the dataset I transformed the scale of the numerical columns and created three new columns one is price_to_size_ratio, Bedroom_Cost and total_house_price. Both columns helped to improve the accuracy of the model.

Outlier Detection and Handling

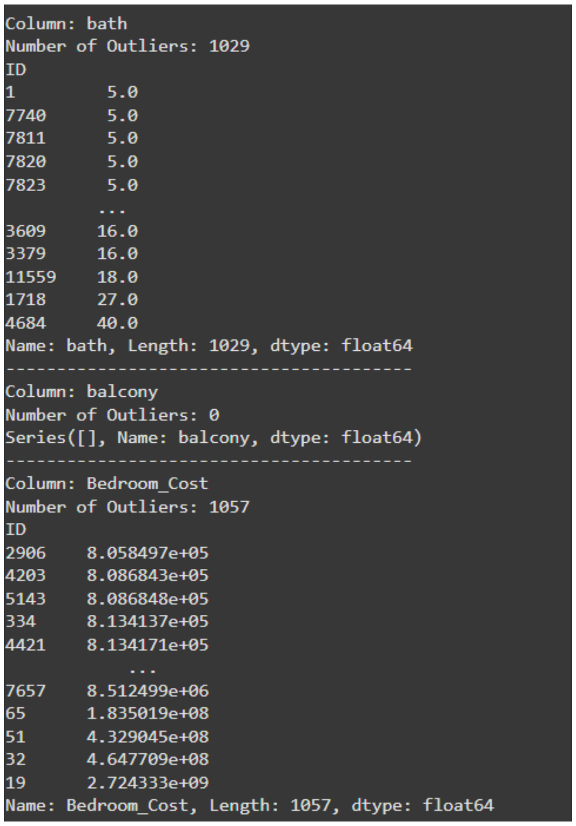
1. Methods for Outlier Detection

Data points that have significant deviants from the remaining data have a huge influence on the performance of prediction models. Here's how outliers can affect predictions: Biased Model, Reduced Model Performance and Overfitting.

Statistical Method used:



```python
# Define numerical columns for analysis
numerical_columns = ['bath', 'balcony', 'Bedroom_Cost', 'total_sqft']

# Function to detect outliers using the IQR method
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)  # First quartile (25th percentile)
    Q3 = column.quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1  # Interquartile range
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers

# Analyze outliers for each numerical column
outlier_results = {}
for column in numerical_columns:
    if column in filtered_data.columns:
        column_data = filtered_data[column].dropna()
        outliers = detect_outliers_iqr(column_data)
        outlier_results[column] = outliers

# Display outlier summary
for column, outliers in outlier_results.items():
    print(f"Column: {column}")
    print(f"Number of Outliers: {len(outliers)}")
    print(outliers.sort_values())
    print("-" * 40)
```



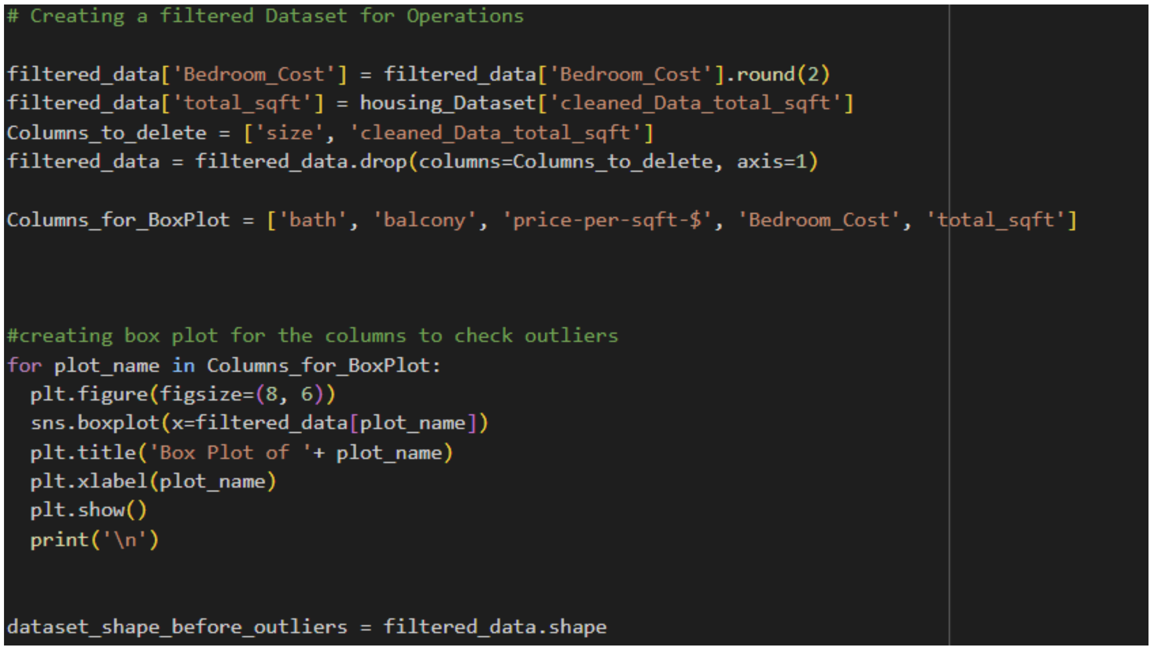
```
Column: bath
Number of Outliers: 1029
ID
1         5.0
7740      5.0
7811      5.0
7820      5.0
7823      5.0
         ...
3609     16.0
3379     16.0
11559    18.0
1718     27.0
4684     40.0
Name: bath, Length: 1029, dtype: float64
----------------------------------------
Column: balcony
Number of Outliers: 0
Series([], Name: balcony, dtype: float64)
----------------------------------------
Column: Bedroom_Cost
Number of Outliers: 1057
ID
2906    8.058497e+05
4203    8.086843e+05
5143    8.086848e+05
334     8.134137e+05
4421    8.134171e+05
            ...
7657    8.512499e+06
65      1.835019e+08
51      4.329045e+08
32      4.647709e+08
19      2.724333e+09
Name: Bedroom_Cost, Length: 1057, dtype: float64
```

Figure 14: IQR Method for outlier detection

IQR (Interquartile Range): Identifies outliers based on their position relative to the quartiles. Here in image 14, shows the method used for showing outliers of the numerical column

Visualization:

Box plots: Visually identify outliers by their position outside the whiskers.



```
# Creating a filtered Dataset for Operations

filtered_data['Bedroom_Cost'] = filtered_data['Bedroom_Cost'].round(2)
filtered_data['total_sqft'] = housing_Dataset['cleaned_Data_total_sqft']
Columns_to_delete = ['size', 'cleaned_Data_total_sqft']
filtered_data = filtered_data.drop(columns=Columns_to_delete, axis=1)

Columns_for_BoxPlot = ['bath', 'balcony', 'price-per-sqft-$', 'Bedroom_Cost', 'total_sqft']



#creating box plot for the columns to check outliers
for plot_name in Columns_for_BoxPlot:
  plt.figure(figsize=(8, 6))
  sns.boxplot(x=filtered_data[plot_name])
  plt.title('Box Plot of '+ plot_name)
  plt.xlabel(plot_name)
  plt.show()
  print('\n')


dataset_shape_before_outliers = filtered_data.shape
```

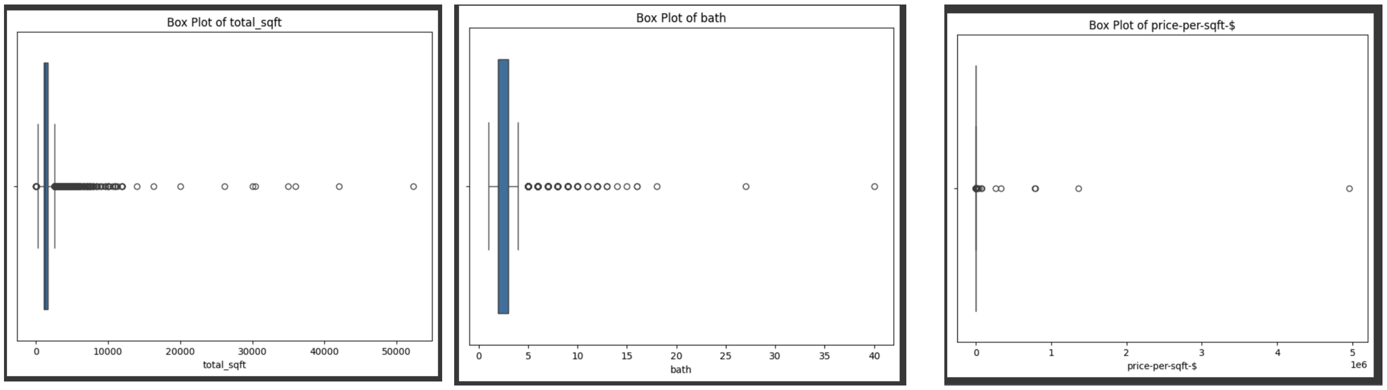


Figure 15: Outliers detection using Box Plot

In figure 25, we can see the outliers which are detected using the Box Plot. I used box plot on all the numerical columns but removed the outliers only from the needed one. Columns like Bath don't have any outliers.

Scatter plots: Identify outliers as points that are far from the main clusters. I have performed a scatter plot as displayed in figure 3.

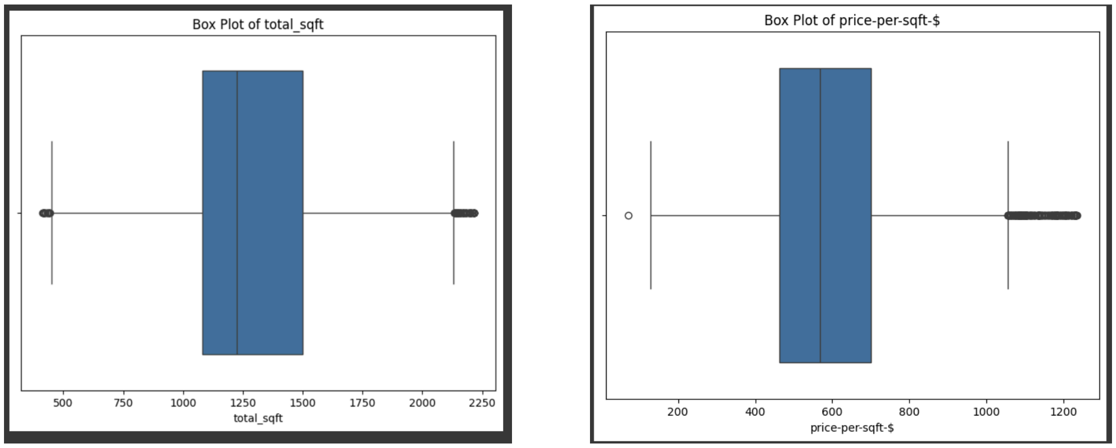


Figure 16: Box plot after removing outliers

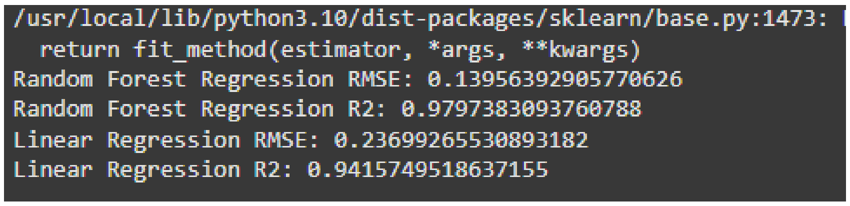
Figure 16, shows the box plot after removing outliers from the filtered dataset. We utilised this filtered dataset for predictions (*Frost, J.,2019)*.

Predictive Analysis

I performed the predictive analysis on the two columns: “price-per-sqft-$” and “buying or not buying” (*ChatGPT - Random Forest Prediction Guide*, no date b).

1. **Predictive Techniques Applied:**

For price-per-sqft-$:



```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473:
  return fit_method(estimator, *args, **kwargs)
Random Forest Regression RMSE: 0.13956392905770626
Random Forest Regression R2: 0.9797383093760788
Linear Regression RMSE: 0.23699265530893182
Linear Regression R2: 0.9415749518637155
```

Figure 17: Result for prediction of price-per-sqft-$

1. Random Forest Regression
   - I used the Random forest algorithm for prediction of this column as it does not get impacted by the outliers and the accuracy is high in this model. As random forest is an upgraded version for Decision Tree I dropped the idea of using a decision tree algorithm.
2. Linear Regression
   - From the observation of the data I found that the price was increasing linearly with the size of the house and it also depends on the location of the house. So I tried to use Linear Regression as well.

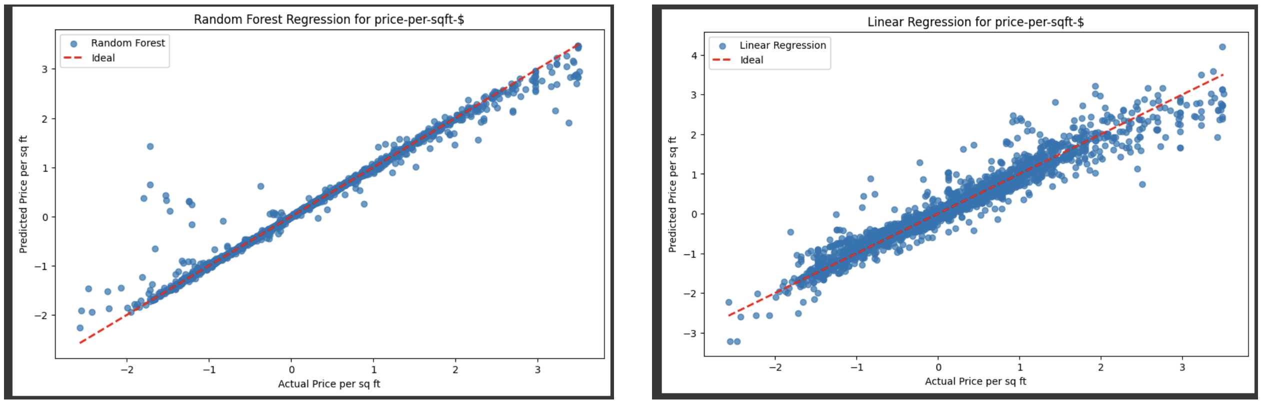
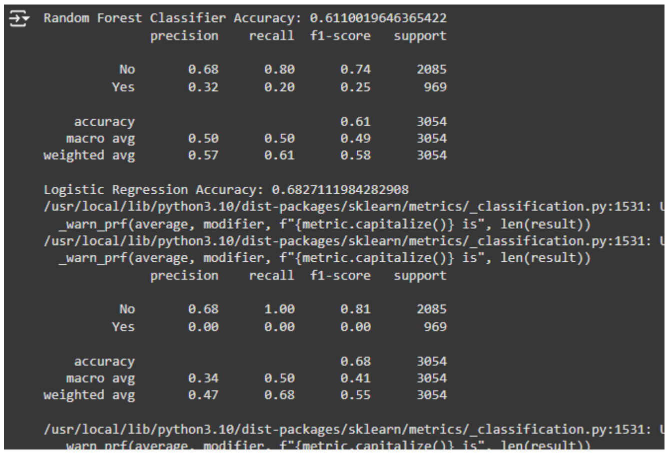
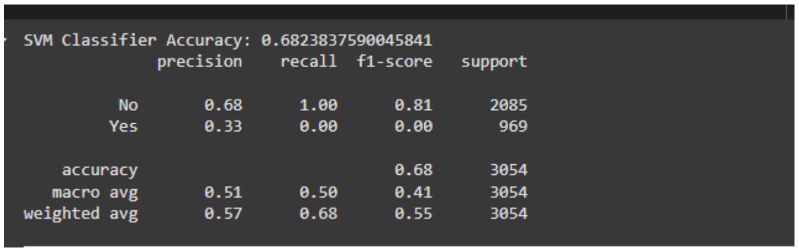


Figure 18: Graph to show the Prediction values and Actual Values relation

For buying or not buying:



```
Random Forest Classifier Accuracy: 0.6110019646365422
              precision    recall  f1-score   support

          No       0.68      0.80      0.74      2085
         Yes       0.32      0.20      0.25       969

    accuracy                           0.61      3054
   macro avg       0.50      0.50      0.49      3054
weighted avg       0.57      0.61      0.58      3054

Logistic Regression Accuracy: 0.6827111984282908
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: U
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: U
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

          No       0.68      1.00      0.81      2085
         Yes       0.00      0.00      0.00       969

    accuracy                           0.68      3054
   macro avg       0.34      0.50      0.41      3054
weighted avg       0.47      0.68      0.55      3054

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: U
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



```
SVM Classifier Accuracy: 0.6823837590045841
              precision    recall  f1-score   support

          No       0.68      1.00      0.81      2085
         Yes       0.33      0.00      0.00       969

    accuracy                           0.68      3054
   macro avg       0.51      0.50      0.41      3054
weighted avg       0.57      0.68      0.55      3054
```

Figure 19: Result for Buying and not Buying using Random Forest classifier, SVM and Logistic Regression

1. Random Forest Classifier
    - It is an ensemble learning method of multiple decision trees which is built and used to classify or regress the target variable during training by giving output from all decision trees.
    - Averaging multiple tree predictions reduces overfitting and improves accuracy.
2. Logistic Regression
    - The Logistic Regression is a linear classification algorithm which predicts the probability of binary outcomes (Buy or Not Buy).
    - Rather than map predictions to probabilities, it uses the sigmoid function.
3. Support vector machine (SVM)
    - SVM is a supervised machine learning algorithm and finds the best hyperplane that separates the classes of a feature space.
    - It's good for high dimensional spaces and can be made to deal with an unknown nonlinearity using kernel functions.

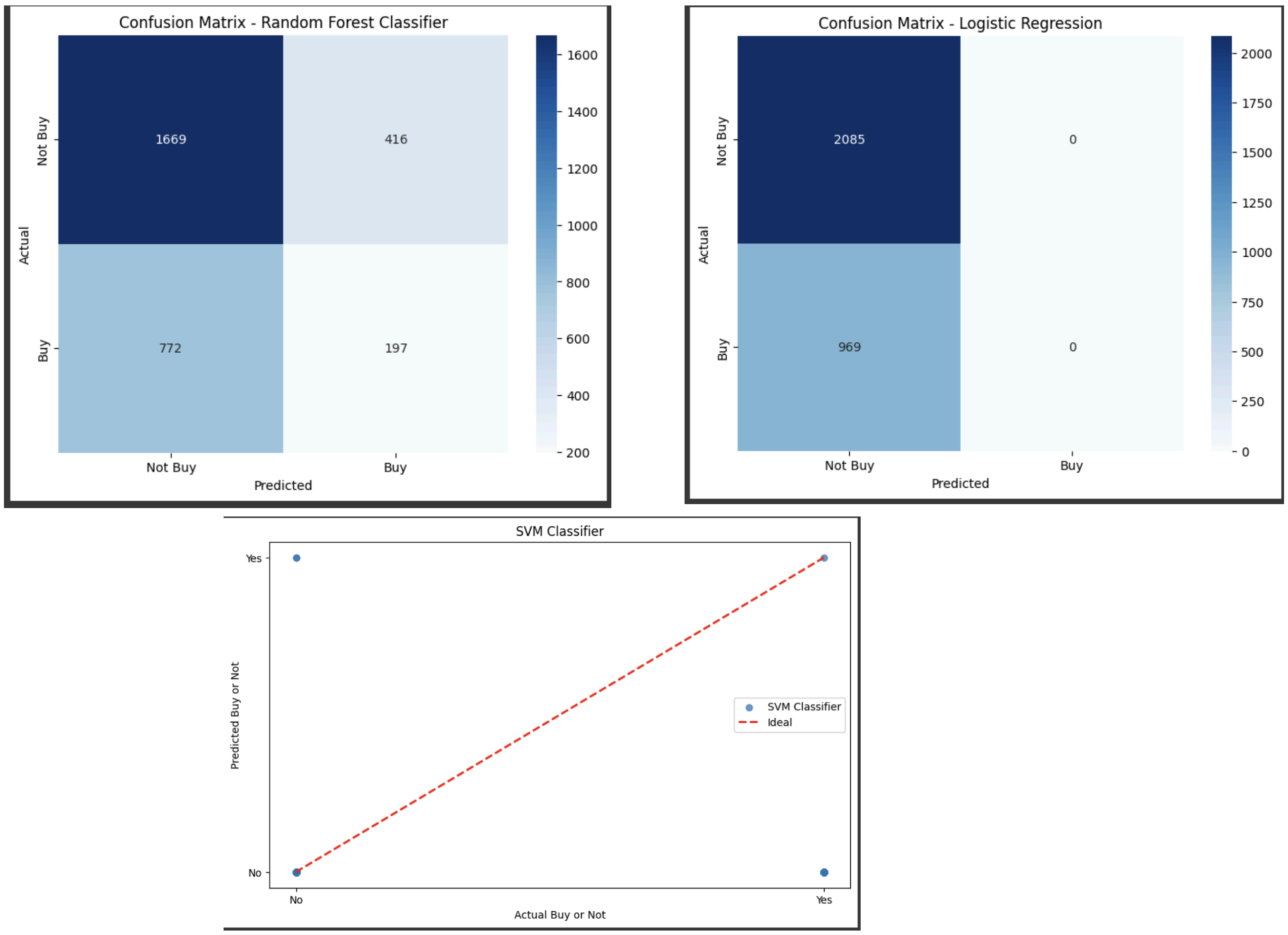
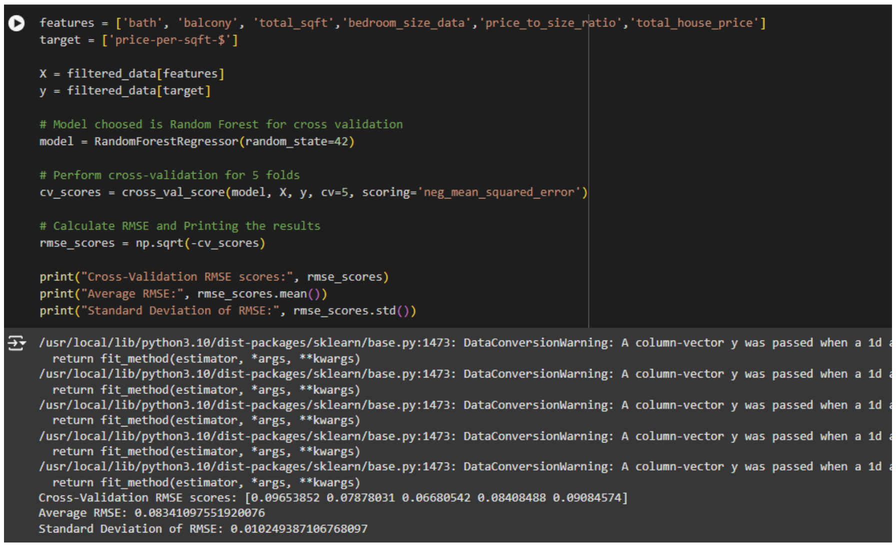


Figure 20: Confusion Matrix and SVM Classifier Visual Representation

Figure 20, represents the Confusion Matrix for visual analysis of the predicted value and Actual values.

2. Model Tuning and Performance Improvements

Hyperparameter tuning was performed using grid search and cross-validation to optimize performance.

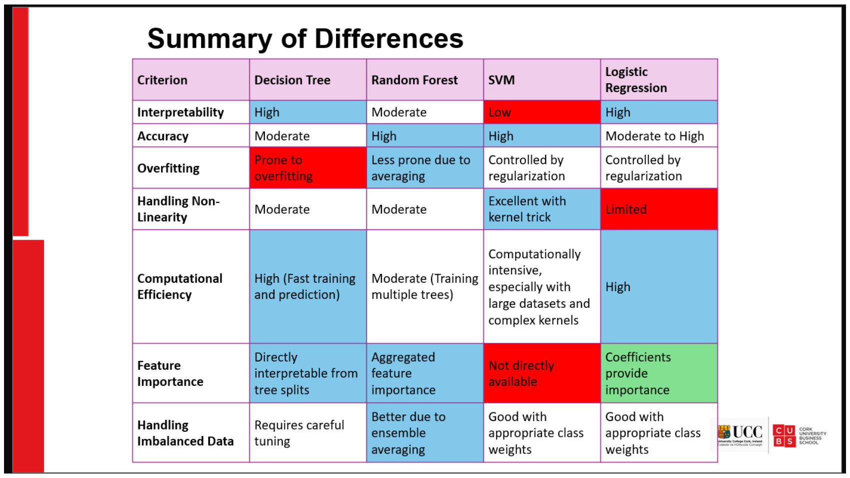


```
features = ['bath', 'balcony', 'total_sqft','bedroom_size_data','price_to_size_ratio','total_house_price']
target = ['price-per-sqft-$']

X = filtered_data[features]
y = filtered_data[target]

# Model choosed is Random Forest for cross validation
model = RandomForestRegressor(random_state=42)

# Perform cross-validation for 5 folds
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')

# Calculate RMSE and Printing the results
rmse_scores = np.sqrt(-cv_scores)

print("Cross-Validation RMSE scores:", rmse_scores)
print("Average RMSE:", rmse_scores.mean())
print("Standard Deviation of RMSE:", rmse_scores.std())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d
  return fit_method(estimator, *args, **kwargs)
Cross-Validation RMSE scores: [0.09653852 0.07878031 0.06680542 0.08408488 0.09084574]
Average RMSE: 0.08341097551920076
Standard Deviation of RMSE: 0.010249387106768097
```

Figure 21: Cross-Validation for Random Forest for price-per-sq-ft-$ Column

Figure 21, Shows the cross validation result for the price-per-sqft-$. This helps to understand the accuracy and performance of the Random Forest model which is best suited for prediction. Figure 22, helps to understand the best suitable model for the prediction and analysis (*3.1. Cross-validation: evaluating estimator performance*, no date).



Summary of Differences

Criterion	Decision Tree	Random Forest	SVM	Logistic Regression
Interpretability	High	Moderate	Low	High
Accuracy	Moderate	High	High	Moderate to High
Overfitting	Prone to overfitting	Less prone due to averaging	Controlled by regularization	Controlled by regularization
Handling Non-Linearity	Moderate	Moderate	Excellent with kernel trick	Limited
Computational Efficiency	High (Fast training and prediction)	Moderate (Training multiple trees)	Computationally intensive, especially with large datasets and complex kernels	High
Feature Importance	Directly interpretable from tree splits	Aggregated feature importance	Not directly available	Coefficients provide importance
Handling Imbalanced Data	Requires careful tuning	Better due to ensemble averaging	Good with appropriate class weights	Good with appropriate class weights

Figure 22: Comparison Table for different methods

Results, Evaluation, and Discussion

1. For price-per-sqft-$:

Random Forest Regression:

Rationale: Since the Random Forest algorithm was robust to outliers and has high accuracy, it was chosen. For this task, it is an enhanced version of Decision Trees and thus, it is a good choice.

Linear Regression:

Rationale: The idea behind that was that the bigger the house, the more it is going to cost per sqft, and the more it will cost is dependent upon location. This linear relationship was then modeled using Linear Regression.

Model Performance

1. **Random Forest Regression:**
   a. RMSE: 0.13956392905770626
   b. R-squared: 0.9797383093760788
2. **Linear Regression:**
   a. RMSE: 0.23699265530893182
   b. R-squared: 0.9415749518637155

Comparison:

Linear Regression tends to perform worse than Random Forest with RMSE and R-Squared. This means that Random Forest is more accurate and fits the data better. (Figure 18: Prediction vs. Actual Values).

Random Forest: Here, the scatter plot shows strong correlation between predicted and actual values with the points almost clumped closely to the diagonal line. These results suggest accurate predictions.

Linear Regression: The linear regression model, too, displays a positive correlation, however, the scatter plot reveals more dispersion suggesting deviation from the linear assumption.

By comparing the results and analysis, Random Forest Regression is the selected model to predict the price-per-sqft-$ column. It shows superior accuracy in addition to robustness to outliers. But in the case of this task, simple linear relationships can be covered by Linear Regression, while the complexity of patterns that Random Forest could handle allow it to be a more versatile and more trusted solution for this task.

2. For buying or not buying:

Random Forest Classifier:

1. Accuracy: ~61.01%
2. Precision, Recall, and F1-Score metrics indicate better performance in predicting "Not Buying" compared to "Buying."
3. Confusion Matrix:
    a. Correctly predicted "Not Buying": 1669
    b. Correctly predicted "Buying": 197
    c. Misclassifications for "Buying" are significant.

Logistic Regression:

1. Accuracy: ~68.28%
2. The model struggles to predict the "Buying" class (F1-Score of 0 for "Yes").
3. Confusion Matrix:
    a. Correctly predicted "Not Buying": 2085
    b. Completely failed to predict "Buying."

Support Vector Machine (SVM):

1. Accuracy: ~68.23%
2. Similar to Logistic Regression, it struggles significantly with "Buying."
3. Confusion Matrix:
    a. Correctly predicted "Not Buying": 2085

    b. Completely failed to predict "Buying."
    c. Visualization confirms the inability to separate classes effectively for "Buying."

Confusion Matrix Observations:

1. **Random Forest**:
    a. Performs better in predicting both classes compared to Logistic Regression and SVM.
    b. Still struggles with imbalanced performance between "Buying" and "Not Buying."
2. **Logistic Regression and SVM**:
    a. Heavily biased toward predicting "Not Buying."
    b. No meaningful classification for the "Buying" class.

Performance Comparison:

1. **Random Forest**:
    a. Best overall classifier, although still not ideal for predicting "Buying."
    b. Benefits from ensemble learning, reducing overfitting, and improving generalization.
2. **Logistic Regression and SVM**:
    a. Fail to capture class imbalance effectively.
    b. Struggles with non-linearity in the data.

Conclusion and Recommendations

The predictive analytics shines light on Dublin's housing market through actionable insights only useful to both buyers and developers. Most reliable model, accurate and computationally efficient, Random Forests emerged.

We can use more models and hypertuning to improve the accuracy of our model and predictions can be improved further.

References:

1. *Frost, J. (2019). 5 Ways to Find Outliers in Your Data. [online] Statistics By Jim. Available at: https://statisticsbyjim.com/basics/outliers/.*
2. *ChatGPT - Random Forest Prediction Guide (no date). https://chatgpt.com/share/674cfa4d-4ce4-8000-b858-10e0b0074269.*
3. *Redirect Notice (no date). https://www.google.com/url?q=https%3A%2F%2Fchatgpt.com%2Fshare%2F6740aa8a-3f3c-8000-80ef-ad1d80424bd5.*
4. *ChatGPT - Random Forest Prediction Guide (no date c). https://chatgpt.com/share/674cfa4d-4ce4-8000-b858-10e0b0074269.*
5. *numpy (2024). https://pypi.org/project/numpy/.*
6. *Quick start guide — Matplotlib 3.9.3 documentation (no date). https://matplotlib.org/stable/users/explain/quick_start.html.*
7. *GeeksforGeeks (2024) Python String Methods. https://www.geeksforgeeks.org/python-string-methods/.*
8. *An introduction to seaborn — seaborn 0.13.2 documentation (no date). https://seaborn.pydata.org/tutorial/introduction.*
9. *3.1. Cross-validation: evaluating estimator performance (no date). https://scikit-learn.org/1.5/modules/cross_validation.html.*