CSE 4/587 Data Intensive Computing
Project Phase 1

Title: Property Sales: Melbourne City

Team Details:

Harshavardhan Sivadanam -hsivadan (50478880)
Nikhil Kumar vadigala -nvadigal (50485135)
Sai Sree Depa- saisreed (50478013)

Problem Statement

The main objective of this project is to develop predictive models for forecasting real estate prices in Melbourne City accurately. The objective is to create models that can predict property sale prices by considering a wide range of factors, including the number of rooms, property type, location (suburb, postcode, latitude, longitude), building characteristics (bedroom count, bathroom count, car parking, land size, building area), year of construction, and other relevant attributes available in the dataset. This project should deliver practical insights for people and parties interested in the Melbourne housing market by forecasting property values. Making decisions on the purchase, sale, or investment of real estate in the city can be aided by these forecasts.

Background

The property market in Melbourne is known for its energy and diversity, drawing interest from both domestic and foreign markets. For a variety of stakeholders, including homebuyers looking for affordable housing, sellers hoping to maximize the value of their property, and investors looking for profitable opportunities, understanding the factors influencing property pricing and sales is essential.

Motivation

The motivation behind this project is to help different people involved in the Melbourne real estate market. For those selling homes, they want to know the best price possible by understanding what affects property prices. If you're looking to buy a house, this project provides detailed information to help you make smart choices. Investors always want to make money, and this project helps them find places for wise investments. Plus, it makes the whole market more transparent and helps everyone make better decisions.

Significance of the Problem

Affordability Concerns:
Concerns about home affordability have arisen in Melbourne as a result of growing house values. These issues have an effect on both the quality of life for locals and the allure of the city for newcomers.

Investment Potential:
The real estate market offers major investment opportunities. For investors hoping to increase their cash, spotting places with strong investment potential is essential.

Market Dynamics:
For well-informed decision-making, it is crucial to comprehend market dynamics, including property types, locations, and trends.

Contribution to the Problem Domain:

This project has the potential to significantly contribute to the domain in the following ways:

Data-Driven Insights:
This project can offer data-driven insights into the Melbourne property market by evaluating a large dataset. Different people like homebuyers, sellers, and investors can follow these insights.

Modeling Predictive:
The project can create models that predict the pricing of properties based on a variety of features. This promotes price transparency and aids in decision-making for the general public.

Finding Investment Hotspots:
By locating places with rapid increases in property value, the project can help real estate investors make wise location decisions.

Policy Implications:
Understanding the problems with housing affordability can help decision makers understand the necessity of housing policies and efforts.

Data Sources

We obtained the dataset "Property Sales of Melbourne City" from Kaggle, and which can be accessed using the following link: Property sales: Melbourne City on Kaggle.

Sample Dataset:

	Unnamed: 0	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	...	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Coun
0	1	Abbotsford	85 Turner St	2	h	1480000	S	Biggin	3/12/2016	2.5	...	1.0	1.0	202.0	NaN	NaN	
1	2	Abbotsford	25 Bloomburg St	2	h	1035000	S	Biggin	4/02/2016	2.5	...	1.0	0.0	156.0	79.0	1900.0	
2	4	Abbotsford	5 Charles St	3	h	1465000	SP	Biggin	4/03/2017	2.5	...	2.0	0.0	134.0	150.0	1900.0	
3	5	Abbotsford	40 Federation La	3	h	850000	PI	Biggin	4/03/2017	2.5	...	2.0	1.0	94.0	NaN	NaN	
4	6	Abbotsford	55a Park St	4	h	1600000	VB	Nelson	4/06/2016	2.5	...	1.0	2.0	120.0	142.0	2014.0	

5 rows × 22 columns

Columns in the Dataset:

```
Index(['Unnamed: 0', 'Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method',
       'SellerG', 'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom',
       'Car', 'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea',
       'Lattitude', 'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

Data Cleaning/Processing

1. Checking for Null Values in the dataset

The dataset consists of several missing values.

```
In [9]: data.isnull().sum()

Out[9]: Unnamed: 0          0
        Suburb              0
        Address             0
        Rooms               0
        Type                0
        Price               0
        Method              0
        SellerG             0
        Date                0
        Distance            1
        Postcode            1
        Bedroom2         3469
        Bathroom         3471
        Car              3576
        Landsize         4793
        BuildingArea    10634
        YearBuilt        9438
        CouncilArea      6163
        Lattitude        3332
        Longtitude       3332
        Regionname          1
        Propertycount       1
        dtype: int64
```

The proportion of missing values in the dataset is approximately 11.91%.

```
proportion_null_values = (total_null_values / total_size) * 100
proportion_null_values
```

```
11.91266876198383
```

- **Missing values in columns: [Postcode, Distance, Regionname, Propertycount]**

  For the 'Postcode' column, we tried to fill the null value, but it was found that there were no 'Postcode' values for the specific addresses 'Footscray Lot' and '2/16 Stafford St'. And for other columns ['Distance', 'Regionname', and 'Propertycount'], the number of missing rows is only 1. So, we have deleted the row containing these features using dropna.

```
In [15]: #But there is no postcode of that address so we are deleting the column
         data[data['Suburb']=='Footscray Lot']['Postcode']
         data[data['Address']=='2/16 Stafford St']['Postcode']

Out[15]: 14440   NaN
         Name: Postcode, dtype: float64
```

```
data.dropna(subset=['Distance','Postcode','Regionname','Propertycount'], inplace=True)
```

- **Missing values in column: [Bedroom2]**

  The 'Bedroom2' and 'Rooms' columns are highly correlated, with a correlation value of 0.9487, So we fill the missing values in the 'Bedroom2' column with the values from the 'Rooms' column.

```
#Filling Bedroom2 with corresponding Rooms value
data['Bedroom2'].fillna(data['Rooms'], inplace=True)
```

- **Missing values in column: [Bathroom, Car, Landsize, BuildingArea, YearBuilt]**

  For the columns ‘Bathroom', ‘YearBuilt’ and 'Car' columns, missing values were replaced with their respective median values. For the columns 'Landsize' and 'BuildingArea,' an iterative imputer was used to estimate and fill the missing values, providing a more advanced imputation method that considers relationships between variables.

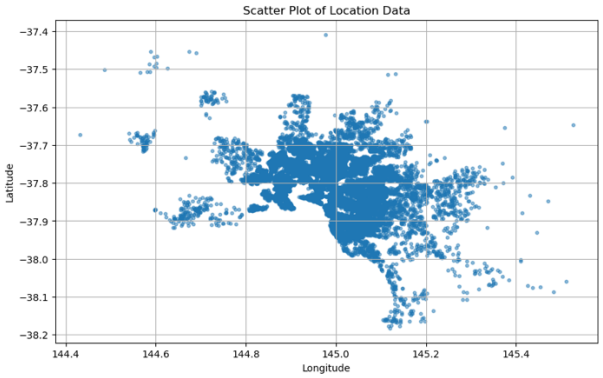
In [34]:
```
#replacing the Bathroom and Car values with median:
cols_to_impute_with_median = ['Bathroom', 'Car']
for col in cols_to_impute_with_median:
    median_value = data[col].median()
    data[col].fillna(median_value, inplace=True)
```

In [35]:
```
# Filling the Landsize, BuildingArea values using IterativeImputer
imputer = IterativeImputer()
columns_to_impute_advanced = ['Landsize', 'BuildingArea']
data[columns_to_impute_advanced] = imputer.fit_transform(data[columns_to_impute_advanced])
```

In [36]:
```
#Fixing null values in Year Built
median_year = data['YearBuilt'].median()
data['YearBuilt'].fillna(median_year, inplace=True)
```

- **Missing values in column: [Lattitude, Longtitude]**

  From observing a scatter plot between these two columns [Lattitude, Longtitude], we can observe that the dataset contains similar location data points that are close to each other.



  KNN imputation was applied to the 'Lattitude' and 'Longtitude' columns. This method estimates missing values by considering the values of the nearest neighbors in the dataset.

```
#KNN imputation estimates missing values by considering the values of the nearest neighbors in the dataset.
imputer = KNNImputer(n_neighbors=5)
data[['Lattitude', 'Longtitude']] = imputer.fit_transform(data[['Lattitude', 'Longtitude']])
```

- **Missing values in column: [CouncilArea]**

  For rows where 'CouncilArea' is missing, it assigns the 'CouncilArea' based on the suburb of that row using the mode value. This fills in the missing 'CouncilArea values based on the most common council area associated with a given suburb.

```
def get_mode_or_default(x, default='Unknown'):
    if len(x.mode()) > 0:
        return x.mode().iloc[0]
    else:
        return default
suburb_to_council = data.groupby('Suburb')['CouncilArea'].apply(get_mode_or_default).to_dict()
data['CouncilArea'] = data.apply(
    lambda row: suburb_to_council[row['Suburb']] if pd.isnull(row['CouncilArea']) else row['CouncilArea'],
    axis=1
)
```

- Finally, all the missing values are successfully cleaned. There are no null values in the dataset.

```
data.isnull().sum()
```

```
Unnamed: 0        0
Suburb            0
Address           0
Rooms             0
Type              0
Price             0
Method            0
SellerG           0
Date              0
Distance          0
Postcode          0
Bedroom2          0
Bathroom          0
Car               0
Landsize          0
BuildingArea      0
YearBuilt         0
CouncilArea       0
Lattitude         0
Longtitude        0
Regionname        0
Propertycount     0
dtype: int64
```

2. Removing Duplicates in the Dataset

We check for any duplicate rows in the dataset. The resulting shape of the data is the same as the original dataset, indicating that there were no duplicates found.

```
data = data.drop_duplicates()
data.shape
```

```
(18395, 22)
```

3. Dropping unwanted columns in the dataset

The 'Bedroom2' column is being dropped from the dataset because it is highly correlated with the 'Rooms' column and contains the same data. So, we can use only one column for prediction.

```
#Since Rooms and Bedroom2 are highly correlated data and have same data. We are dropping the column "Bedroom2"
data.drop(columns=['Bedroom2'], inplace=True)
```

4. Changing Data Type

```
# Changing Date datatype to datetime type
data['Date'] = pd.to_datetime(data['Date'])
```

```
/var/folders/kp/xfqjcgz93rx88km955rkxtjm0000gn/T/ipykernel_17689/1482365235.py:2: UserWarning: Parsing dates in DD
M/YYYY format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Speci
a format to ensure consistent parsing.
  data['Date'] = pd.to_datetime(data['Date'])
```

```
# Converting columns to int as Postcode, Bedroom2, YearBuilt and Property count are int values rather than float
columns_int = ['Postcode', 'Bathroom','Car','YearBuilt', 'Propertycount']
for col in columns_int:
    data[col] = data[col].astype(int)
```

```
#Converting price column datatype to float
data['Price'] = data['Price'].astype('float64')
```

```
# Converting the categorical columns to the 'category' data type
columns_obj = data.select_dtypes(["object"]).columns
data[columns_obj] = data[columns_obj].astype('category')
```

- The 'Date' column is converted to a datetime data type for accurate date representation.
- Columns 'Postcode,' 'Bathroom,' 'Car,' 'YearBuilt,' and 'Propertycount' are converted to int data types since they contain integer values.
- The 'Price' column is converted to a float data type to handle decimal values.
- Categorical columns in the dataset are converted to the 'category' data type for more efficient storage and handling of categorical data.

5. Datetime Column: Extracting features

Three new columns, 'Year,' 'Month,' and 'Day,' are added to the dataset based on the 'Date' column.

```
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
print(data.shape)
```

```
(18395, 37)
```

6. Adding a new column Price per Square Meter

A new column 'Price_per_sqm' is created, representing the price per square meter, which is calculated by dividing the 'Price' by the 'Landsize.' This additional column provides insights into property values based on area.

```
data['Landsize'] = data['Landsize'].replace(0, 1)
data['Price_per_sqm'] = data['Price'] / data['Landsize']
data.head()
```

7. Adding a new column to know the Age of the building

A new column 'BuildingAge' is created in the dataset, which calculates the age of buildings.

```
current_year = pd.Timestamp.now().year
data['BuildingAge'] = current_year - data['YearBuilt']
data.head()
```

8. Removing Miscellaneous Values

The dataset has a minimum 'YearBuilt' value of 1196, which appeared to be invalid. Values below 1800 were filtered and analyzed, and the 'YearBuilt' value was used to replace the ambiguous values.

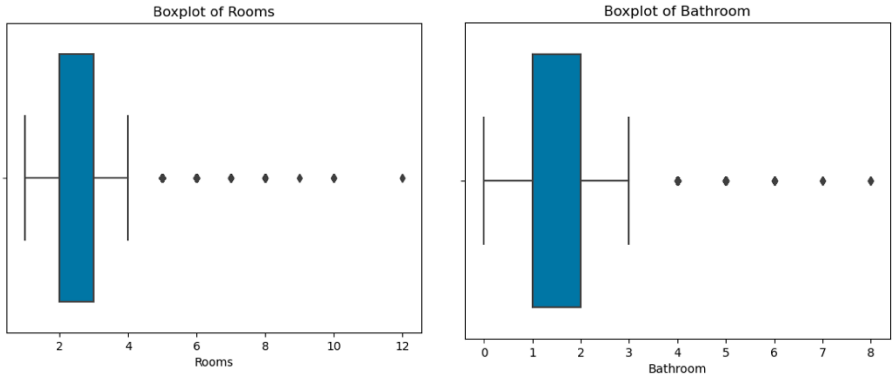
```
invalid_years = data[data['YearBuilt'] < 1800]
print(invalid_years.YearBuilt)
```

```
12754    1196
Name: YearBuilt, dtype: int64
```

```
mode_year = data['YearBuilt'].mode().values[0]
data.loc[data['YearBuilt'] == 1196, 'YearBuilt'] = mode_year
mode_year
```

9. Removing noisy data from columns Rooms and Bathroom2

The presence of noisy data in the dataset columns is identified through boxplot analysis.



For the 'Rooms' column, values that appeared highly unusual, such as 12, were removed. We assumed that the maximum reasonable number of rooms in a property is 5. Similarly, for the 'Bathroom' column, values like 8 were considered unlikely and removed. A maximum reasonable number of four bathrooms is assumed.

```
max_rooms = 5
data['Rooms'] = data['Rooms'].apply(lambda x: max_rooms if x > max_rooms else x)

# Assuming the maximum reasonable number of bathrooms as 4
max_bathrooms = 4
data['Bathroom'] = data['Bathroom'].apply(lambda x: max_bathrooms if x > max_bathrooms else x)
```

10. Label Encoder

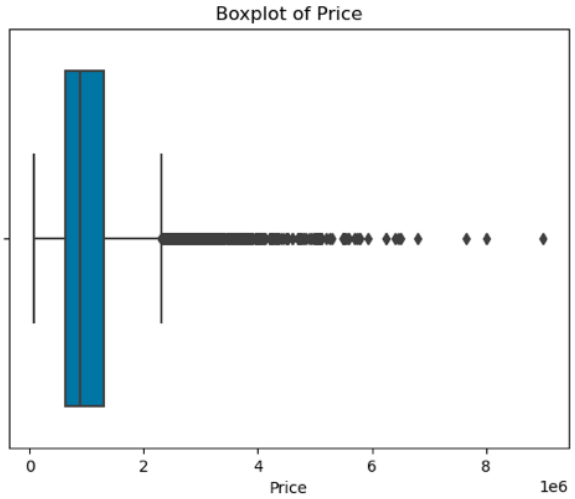
Label encoding is applied to three categorical columns - 'Suburb,' 'SellerG,' and 'CouncilArea.' This process converts the categorical values in these columns into numerical representations for use in machine learning models.

```
label_encoder = LabelEncoder()
data['Suburb_encoded'] = label_encoder.fit_transform(data['Suburb'])
data['SellerG_encoded'] = label_encoder.fit_transform(data['SellerG'])
data['CouncilArea_encoded'] = label_encoder.fit_transform(data['CouncilArea'])
data.drop(['Suburb',  'SellerG', 'CouncilArea'], axis=1, inplace=True)
```

11. Detecting Outliers

A boxplot was used to identify the presence of outliers in the 'Price' column.

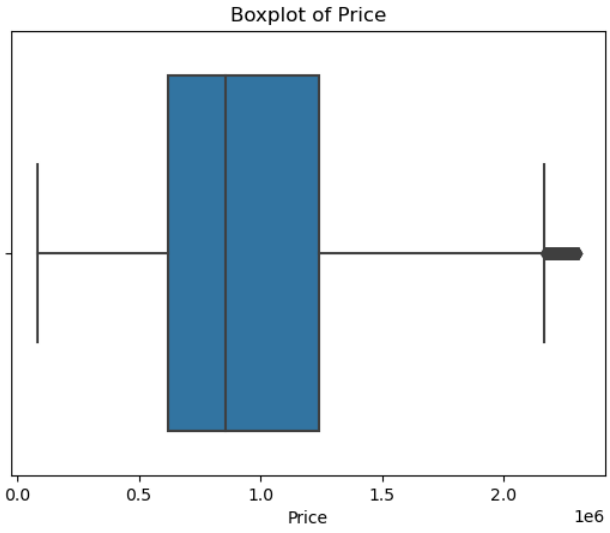
```
In [75]: sns.boxplot(x=data['Price'])
         plt.title('Boxplot of Price')
         plt.show()
```



Outliers in the 'Price' column are removed using the Interquartile Range (IQR) method.

```
#Removing the outliers in Price using Quantile
Q1 = data['Price'].quantile(0.25)
Q3 = data['Price'].quantile(0.75)
IQR = Q3 - Q1
filter = (data['Price'] >= Q1 - 1.5 * IQR) & (data['Price'] <= Q3 + 1.5 *IQR)
data = data.loc[filter]
```

After removing the Outliers, the boxplot for the ‘Price’ column is as follows:



12. Resetting the index

The dataset's index is being reset because rows and columns have been removed, causing gaps in the index. The 'Unnamed: 0' column has been removed from the dataset.

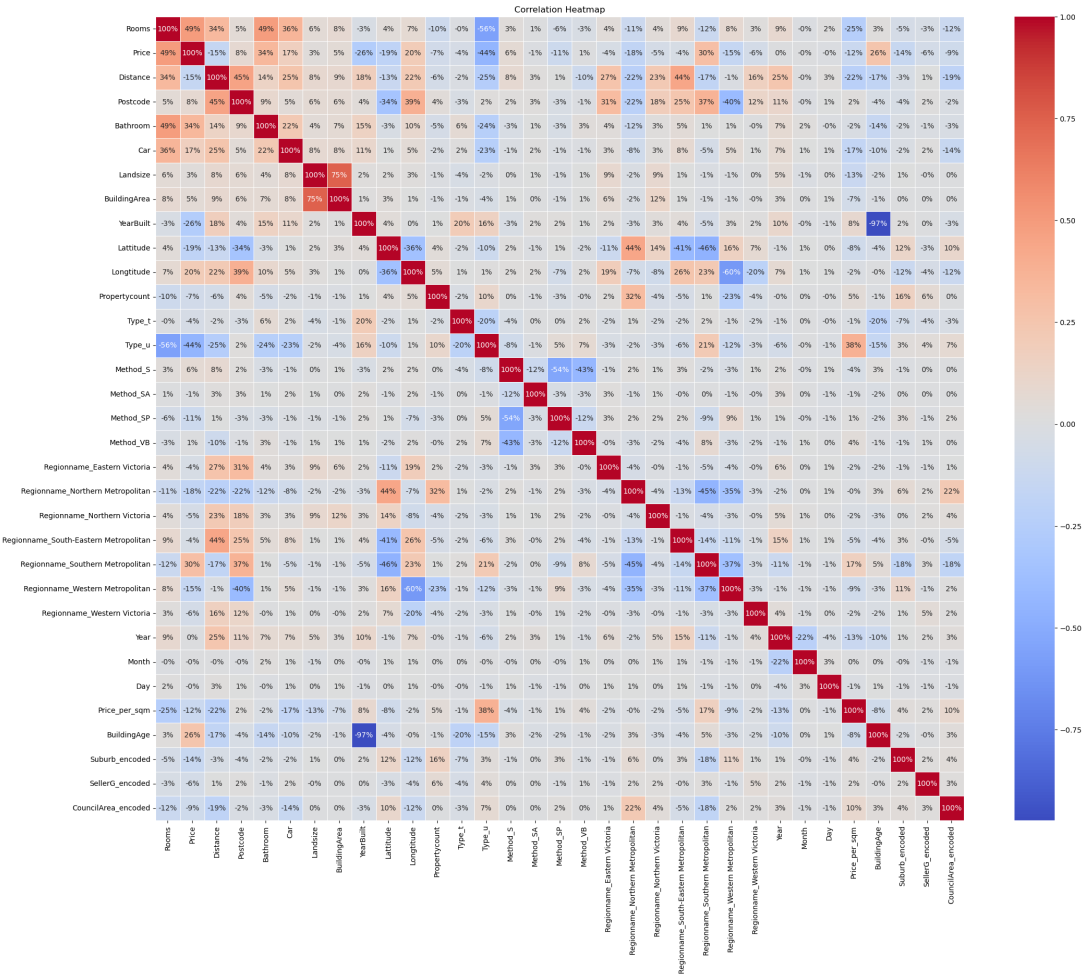
```
#Resetting the index
data = data.reset_index(drop=True)
data
```

Exploratory Data Analysis (EDA)

1. Correlation Matrix Using Heatmap

A correlation heatmap is generated to visualize the relationships between variables in the dataset. It helps in understanding how different attributes are related to each other.

```
correlation_matrix = data.corr()
plt.figure(figsize=(25, 20))
sns.heatmap (correlation_matrix, annot=True, cmap='coolwarm', fmt='.0%' ,linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



2. CountPlot for the Dataset:

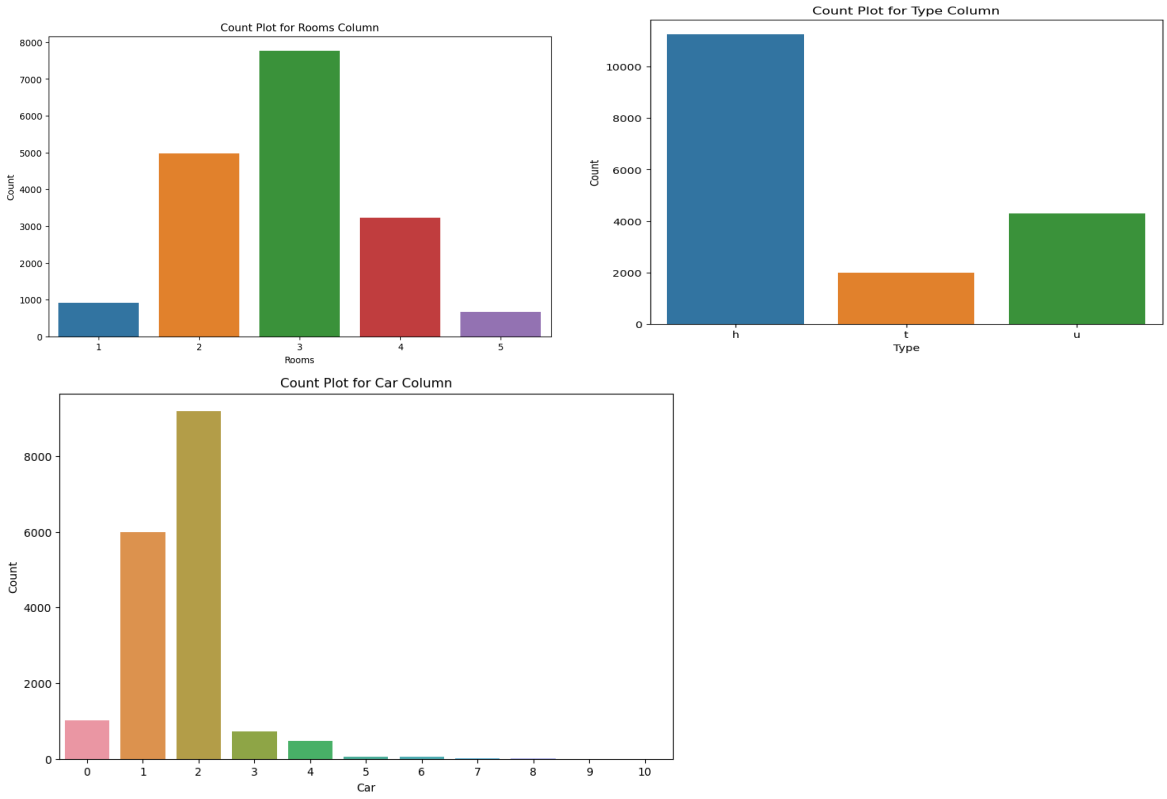
We examine the distribution of the number of rooms, property types, and car parking options in the Melbourne real estate market. The count plots show the frequency of different values within each attribute.

```
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='Rooms')
plt.title('Count Plot for Rooms Column')
plt.xlabel('Rooms')
plt.ylabel('Count')
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='Type')
plt.title('Count Plot for Type Column')
plt.xlabel('Type')
plt.ylabel('Count')
plt.show()
```

```
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='Car')
plt.title('Count Plot for Car Column')
plt.xlabel('Car')
plt.ylabel('Count')
plt.show()
```

The Count Plots are as follows:



3. Univariate Non-Graphical EDA:

In this non-graphical exploratory data analysis (EDA), key statistics and metrics for the 'Price' column are calculated and presented. The skewness of the 'Price' data is determined which is 0.8342. This value indicates the degree of asymmetry in the distribution of property prices within the dataset.

```
# Summary statistics for numeric columns
numeric_summary = data.describe()
numeric_summary
```

	Rooms	Price	Distance	Postcode	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	...	Regi
count	17543.000000	1.754300e+04	1.754300e+04	17543.000000	17543.000000	17543.000000	1.754300e+04	1.754300e+04	17543.000000	17543.000000	...	
mean	2.872770	9.597913e+05	1.620117e-16	3105.848031	1.395998	1.666876	3.564257e-17	6.561473e-17	1968.599042	-37.808498	...	
std	0.899018	4.438280e+05	1.000029e+00	96.607316	0.596134	0.857715	1.000029e+00	1.000029e+00	24.581302	0.074284	...	
min	1.000000	8.500000e+04	-1.727967e+00	3000.000000	0.000000	0.000000	-4.292649e-01	-3.541768e-01	1830.000000	-38.182550	...	
25%	2.000000	6.220000e+05	-6.923690e-01	3044.000000	1.000000	1.000000	-2.270790e-01	-1.377194e-01	1970.000000	-37.844280	...	
50%	3.000000	8.550000e+05	-1.005989e-01	3083.000000	1.000000	2.000000	2.436279e-02	4.495810e-03	1970.000000	-37.809849	...	
75%	3.000000	1.240000e+06	4.911712e-01	3149.000000	2.000000	2.000000	5.221522e-02	2.138254e-02	1970.000000	-37.764620	...	
max	5.000000	2.305000e+06	6.178740e+00	3978.000000	4.000000	10.000000	5.856461e+01	9.898194e+01	2018.000000	-37.408530	...	

8 rows × 33 columns

```
#skewness of the Price column in the data
price_skewness = data['Price'].skew()
price_skewness
```

```
0.8341881662892019
```

```
#Calculating Central Tendency
mean_price = data['Price'].mean()
median_price = data['Price'].median()
mode_price = data['Price'].mode().values[0]
print(mean_price)
print(median_price)
print(mode_price)
```

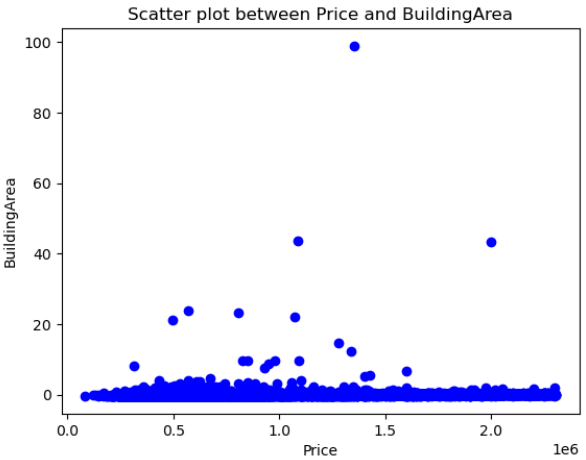
```
959791.2901442171
855000.0
600000.0
```

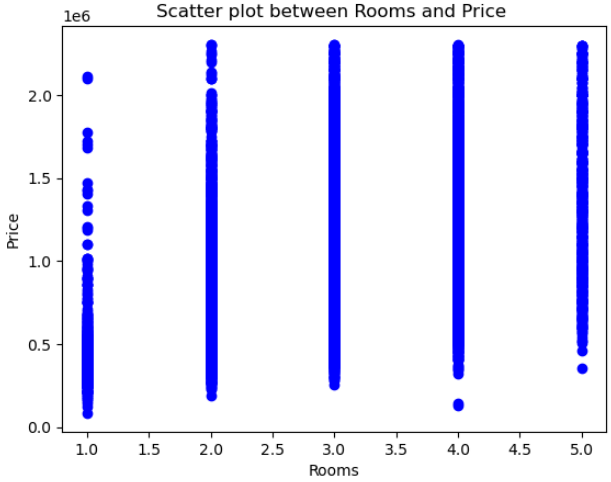
4. Scatter Plot:

Two scatter plots were created to explore relationships within the dataset. The first scatter plot illustrates the relationship between property prices ('Price') and the building area ('BuildingArea'). The second scatter plot shows the association between the number of rooms ('Rooms') and property prices ('Price').

```
#Scatter plot between Price and Buildingarea
plt.scatter(x=data['Price'],y=data['BuildingArea'], label='Scatter Plot', color='blue', marker='o')
plt.title("Scatter plot between Price and BuildingArea")
plt.xlabel("Price")
plt.ylabel("BuildingArea")
plt.show()
```

```
plt.scatter(x=data['Rooms'],y=data['Price'], label='Scatter Plot', color='blue', marker='o')
plt.title("Scatter plot between Rooms and Price ")
plt.xlabel("Rooms")
plt.ylabel("Price")
plt.show()
```

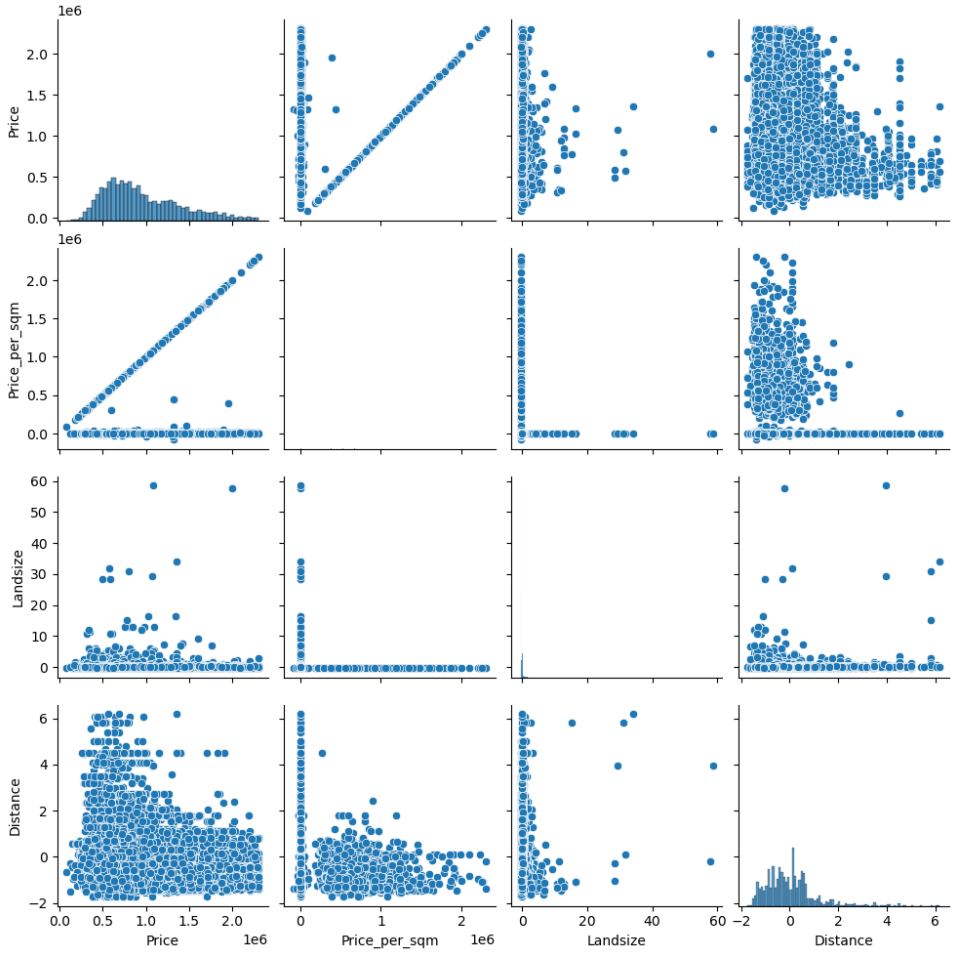




5. Pair Plot:

A pair plot was created to visualize the relationships between selected variables: 'Price,' 'Price_per_sqm,' 'Landsize,' and 'Distance.' The main diagonal of the pair plot represents the distribution of each variable. The scatterplots in the lower triangle illustrate the relationships between the variables. We examine how 'Price' relates to 'Price_per_sqm,' 'Landsize,' and 'Distance.'

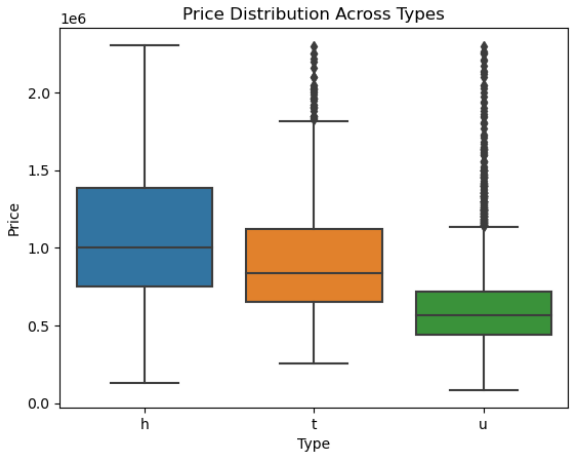
```
sns.pairplot(data[['Price', 'Price_per_sqm', 'Landsize', 'Distance']])
plt.show()
```



6. Box Plot:

A box plot was created to visualize the distribution of property prices ('Price') across different property types ('Type'). Each box represents a property type, and its height illustrates the price distribution. The lower and upper bounds of the box represent the interquartile range (IQR), indicating the middle 50% of the data. The horizontal line within each box is the median price.

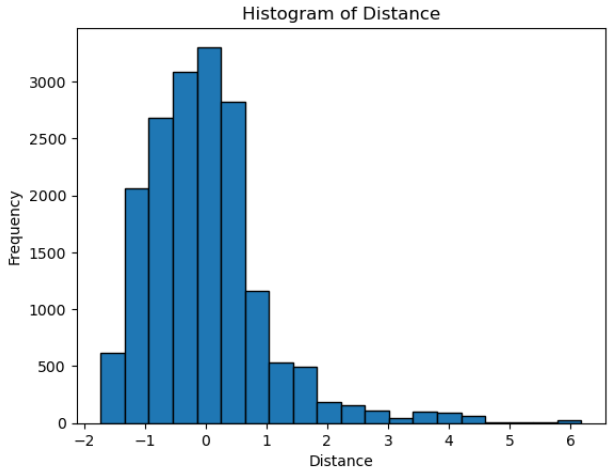
```
sns.boxplot(x=data['Type'], y=data['Price'])
plt.title('Price Distribution Across Types')
plt.xlabel("Type")
plt.ylabel("Price")
plt.show()
```



7. Histogram:

The histogram visualization shows the distribution of 'Distance' values in the dataset. It will show the number of occurrences with different ranges of distances in the dataset.

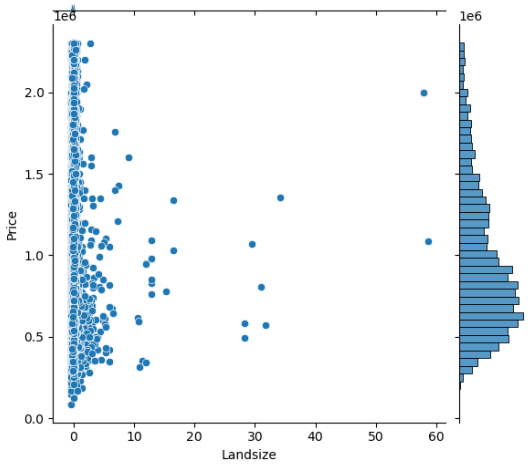
```
plt.hist(data['Distance'], bins=20, edgecolor='black')
plt.title('Histogram of Distance')
plt.xlabel('Distance')
plt.ylabel('Frequency')
plt.show()
```



8. JointPlot:

The association between "Landsize" and "Price" in the dataset is seen in this joint plot. It's a scatter plot with additional features that shows each data point and the distribution of the values for "Landsize" and "Price." The distributions of 'Landsize' are shown separately in the histograms on the right.

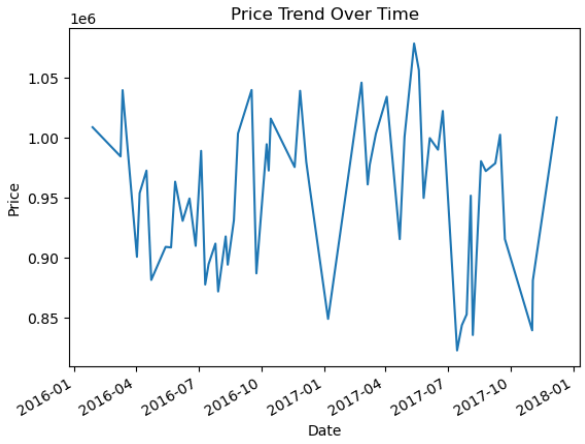
```
sns.jointplot(x='Landsize', y='Price', data=data, kind='scatter')
plt.xlabel('Landsize')
plt.ylabel('Price')
plt.show()
```



9. LinePlot:

The line graph illustrates how property prices have changed throughout time. The plot's zigzag pattern depicts changes in real estate values over the course of the period. These fluctuations can be influenced by various factors, including market dynamics, seasonality, and economic conditions.

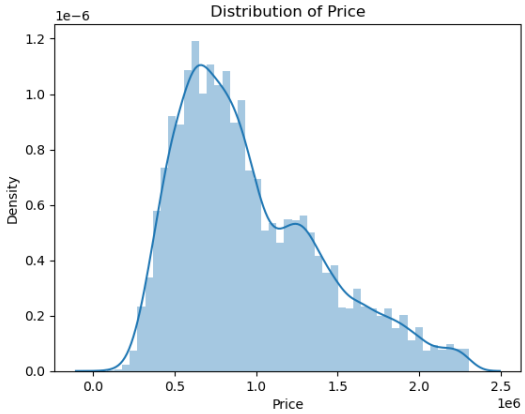
```
In [93]: data.groupby('Date')['Price'].mean().plot()
         plt.title('Price Trend Over Time')
         plt.xlabel('Date')
         plt.ylabel('Price')
         plt.show()
```



10. DistPlot:

This Distplot shows how property prices are spread out in the dataset. It's a histogram that covers a probability density curve, providing insights into the shape and characteristics of the price distribution. This shows us to understand the central tendencies, spread, and potential skewness in property prices.

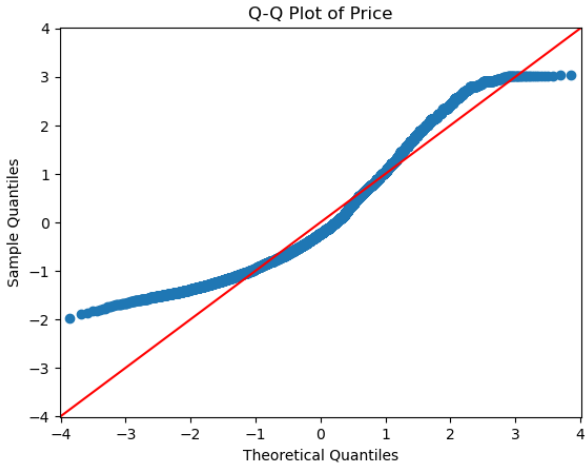
```
[94]: sns.distplot(data['Price'])
      plt.title('Distribution of Price')
      plt.show()
```



11. Q-Q Plot to Assess the Normality of Price Distribution:

A Quantile-Quantile (Q-Q) plot is created to evaluate the normality of the 'Price' data distribution. The quantiles of the actual data and the quantiles of a theoretical normal distribution are compared in a Q-Q display. The data points show that they follow a normal distribution if they are near the 45-degree reference line.

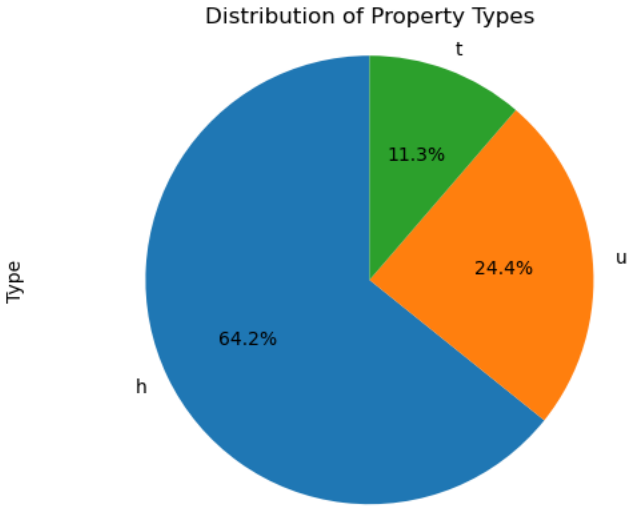
```
In [97]: scaler = StandardScaler()
         columns_to_scale = ['Price']
         data[columns_to_scale] = scaler.fit_transform(data[columns_to_scale])
         sm.qqplot(data['Price'], line ='45')
         plt.title('Q-Q Plot of Price')
         plt.show()
```



12. Pie Chart:

To show the distribution of property types in the dataset, a pie chart is created. Each part of the pie represents a specific property type [h = house, t = townhouse, u = unit/apartment], and its size corresponds to the proportion of that property type in the dataset.

```
data['Type'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Property Types')
plt.axis('equal')
plt.show()
```



References

- https://www.kaggle.com/datasets/amalab182/property-salesmelbourne-city/data
- https://pandas.pydata.org/
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
- https://numpy.org/doc/stable/
- https://seaborn.pydata.org/tutorial.html
- https://www.geeksforgeeks.org/matplotlib-tutorial/