In [37]:
```python
import pandas as pd

# Load the dataset
melbourne_data = pd.read_csv('C:/Users/PhD Scholar/Downloads/archive(3)/Melbourne_housing_FULL.csv')
```

In [38]:
```python
# Display basic information about the dataset
print(melbourne_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         34857 non-null  object 
 1   Address        34857 non-null  object 
 2   Rooms          34857 non-null  int64  
 3   Type           34857 non-null  object 
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object 
 6   SellerG        34857 non-null  object 
 7   Date           34857 non-null  object 
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object 
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object 
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
None
```

In [39]:
```python
# Describe the dataset
print(melbourne_data.describe())
```

```
              Rooms         Price      Distance      Postcode      Bedroom2  \
count  34857.000000  2.724700e+04  34856.000000  34856.000000  26640.000000
mean       3.031012  1.050173e+06     11.184929   3116.062859      3.084647
std        0.969933  6.414671e+05      6.788892    109.023903      0.980690
min        1.000000  8.500000e+04      0.000000   3000.000000      0.000000
25%        2.000000  6.350000e+05      6.400000   3051.000000      2.000000
50%        3.000000  8.700000e+05     10.300000   3103.000000      3.000000
75%        4.000000  1.295000e+06     14.000000   3156.000000      4.000000
max       16.000000  1.120000e+07     48.100000   3978.000000     30.000000

           Bathroom           Car       Landsize  BuildingArea     YearBuilt  \
count  26631.000000  26129.000000   23047.000000   13742.00000  15551.000000
mean       1.624798      1.728845     593.598993     160.25640   1965.289885
std        0.724212      1.010771    3398.841946     401.26706     37.328178
min        0.000000      0.000000       0.000000       0.00000   1196.000000
25%        1.000000      1.000000     224.000000     102.00000   1940.000000
50%        2.000000      2.000000     521.000000     136.00000   1970.000000
75%        2.000000      2.000000     670.000000     188.00000   2000.000000
max       12.000000     26.000000  433014.000000   44515.00000   2106.000000

          Lattitude    Longtitude  Propertycount
count  26881.000000  26881.000000   34854.000000
mean     -37.810634    145.001851    7572.888306
std        0.090279      0.120169    4428.090313
min      -38.190430    144.423790      83.000000
25%      -37.862950    144.933500    4385.000000
50%      -37.807600    145.007800    6763.000000
75%      -37.754100    145.071900   10412.000000
max      -37.390200    145.526350   21650.000000
```

In [40]:

```python
# Data analysis (Example: Mean price by type)
mean_price_by_type = melbourne_data.groupby('Type')['Price'].mean()
print(mean_price_by_type)
```

```
Type
h    1.203718e+06
t    9.310772e+05
u    6.279434e+05
Name: Price, dtype: float64
```

In [41]:

```python
# Find missing values and count operations
missing_values_count = melbourne_data.isnull().sum().sum()  # Total count of missing values
print("Total missing values:", missing_values_count)
```

```
Total missing values: 100975
```

Ways to handle Missing Data

Option 1: Remove rows with missing values

In [42]:
```
# clean_melbourne_data = melbourne_data.dropna()
```

Option 2: Fill missing values with mean

In [43]:
```
# mean_filled_melbourne_data = melbourne_data.fillna(melbourne_data.mean())
```

Option 3: Interpolate missing values

In [44]:
```
# interpolated_melbourne_data = melbourne_data.interpolate()
```

Option 4: Forward fill missing values

In [45]:
```
# forward_filled_melbourne_data = melbourne_data.ffill()
```

Option 5: Backward fill missing values

In [46]:
```
# backward_filled_melbourne_data = melbourne_data.bfill()
```

Display information about cleaned datasets

In [47]:
```
# print(clean_melbourne_data.info())
```

Save cleaned datasets to CSV files

In [48]:
```
# clean_melbourne_data.to_csv('clean_melbourne_data.csv', index=False)
```

To convert object data type into float

For single column conversion

In [49]:
```
# # Select the object column you want to convert to float64
# column_to_convert = 'Method'
```

In [50]:
```
# # Convert the selected object column to float64
# melbourne_data[column_to_convert] = pd.to_numeric(melbourne_data[column_to_convert], errors='coerce')

# # Print the first few rows of the converted column
# print(f"Converted column '{column_to_convert}' to float64:")
# print(melbourne_data[column_to_convert].head())
```

For multiple object columns conversion into float

In [51]:
```
# Convert all columns to float64
# melbourne_data = melbourne_data.astype(float)
```