20325583 - Colm Mooney

CS401 - Assignment 1

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

In [ ]:
```
!jupyter nbconvert --to html /CS401Lab1a.ipynb
```

In [ ]:
```
import pandas as pd

file_path = '/content/drive/MyDrive/Melbourne_housing.csv'
df = pd.read_csv(file_path)

#1(a) Provide information on the dataset, including the number of rows and columns
#Rows, Columns, Column list, First row.
print(df.shape[0])
print(df.shape[1])
print(df.head(0))
#First row
#print(df.head(1)


#(b) Briefly describe the target variable (e.g., 'Price') and its distribution.
NullPrice = df['Price'].isnull().sum()
print("There is", NullPrice, "houses with missing prices.")
print(df.shape[0] - NullPrice,"include their prices.")
print("The average price of a house is:", round(df['Price'].mean()),"$")
print("The most expensive house is: ", df.loc[df['Price'].idxmax()])
print("The cheapest house is: ", df.loc[df['Price'].idxmin()])
```

```
34857
22
Empty DataFrame
Columns: [Suburb, Address, Rooms, Type, Method, SellerG, Date, Distance, Postcode,
Bedroom, Bathroom, Car, Landsize, BuildingArea, YearBuilt, CouncilArea, Latitude,
Longtitude, Regionname, Propertycount, ParkingArea, Price]
Index: []

[0 rows x 22 columns]
There is 7610 houses with missing prices.
27247 include their prices.
The average price of a house is: 1050173 $
The most expensive house is:  Suburb                        Brighton
Address                        6 Cole St
Rooms                                  4
Type                                   h
Method                                VB
SellerG                    hockingstuart
Date                          28/10/2017
Distance                            10.5
Postcode                          3186.0
Bedroom                              4.0
Bathroom                             3.0
Car                                  2.0
Landsize                          1400.0
BuildingArea                         NaN
YearBuilt                            NaN
CouncilArea          Bayside City Council
Latitude                       -37.89335
Longtitude                     144.98643
Regionname         Southern Metropolitan
Propertycount                    10579.0
ParkingArea                       Indoor
Price                         11200000.0
Name: 32774, dtype: object
The cheapest house is:  Suburb                        Footscray
Address                     202/51 Gordon St
Rooms                                      1
Type                                       u
Method                                    PI
SellerG                              Burnham
Date                                3/9/2016
Distance                                 6.4
Postcode                              3011.0
Bedroom                                  1.0
Bathroom                                 1.0
Car                                      0.0
Landsize                                 0.0
BuildingArea                             NaN
YearBuilt                             2007.0
CouncilArea         Maribyrnong City Council
Latitude                            -37.7911
Longtitude                            144.89
Regionname              Western Metropolitan
Propertycount                         7570.0
ParkingArea                  Detached Garage
Price                                85000.0
Name: 127, dtype: object
```

```
<ipython-input-38-a3fb6c02e015>:4: DtypeWarning: Columns (13) have mixed types. Sp
ecify dtype option on import or set low_memory=False.
  df = pd.read_csv(file_path)
```

(a) From this piece of code, we can see that there is 34857 rows in total & 22 columns.

The 22 columns include: Suburb, Address, Rooms, Type, method, SellerG, Date, Distance, Postcode, Bedroom, bathroom, Car, Landsize, BuildingArea, YearBuilt, CouncilArea, Latitude, Longitude, Regionname, Propertycount, ParkingArea & Price. This is to expected of a housing dataset as these are area's of interest when looking to buy/rent a house.

(b) Looking at the target variable 'Price', we can see that there are many houses that do not have a price listed. 7610 have not included their price, 27247 houses have their price listed. The average price of a house in this dataset is 1050173 Australian dollars. The most expensive house costs 11200000 Australian dollars & the cheapest house is worth 85000 Australian dollars.

In [ ]:
```
#(c) Display summary statistics and data types of the features.
print(df.describe())
print(df.dtypes)
```

```
              Rooms      Distance      Postcode       Bedroom      Bathroom  \
count  34857.000000  34856.000000  34856.000000  26640.000000  26631.000000
mean       3.031012     11.184929   3116.062859      3.084647      1.624798
std        0.969933      6.788892    109.023903      0.980690      0.724212
min        1.000000      0.000000   3000.000000      0.000000      0.000000
25%        2.000000      6.400000   3051.000000      2.000000      1.000000
50%        3.000000     10.300000   3103.000000      3.000000      2.000000
75%        4.000000     14.000000   3156.000000      4.000000      2.000000
max       16.000000     48.100000   3978.000000     30.000000     12.000000

                Car       Landsize     YearBuilt      Latitude    Longtitude  \
count  26129.000000   23047.000000  15551.000000  26881.000000  26881.000000
mean       1.728845     593.598993   1965.289885    -37.810634    145.001851
std        1.010771    3398.841946     37.328178      0.090279      0.120169
min        0.000000       0.000000   1196.000000    -38.190430    144.423790
25%        1.000000     224.000000   1940.000000    -37.862950    144.933500
50%        2.000000     521.000000   1970.000000    -37.807600    145.007800
75%        2.000000     670.000000   2000.000000    -37.754100    145.071900
max       26.000000  433014.000000   2106.000000    -37.390200    145.526350

       Propertycount         Price
count   34854.000000  2.724700e+04
mean     7572.888306  1.050173e+06
std      4428.090313  6.414671e+05
min        83.000000  8.500000e+04
25%      4385.000000  6.350000e+05
50%      6763.000000  8.700000e+05
75%     10412.000000  1.295000e+06
max     21650.000000  1.120000e+07
Suburb            object
Address           object
Rooms              int64
Type              object
Method            object
SellerG           object
Date              object
Distance         float64
Postcode         float64
Bedroom          float64
Bathroom         float64
Car              float64
Landsize         float64
BuildingArea      object
YearBuilt        float64
CouncilArea       object
Latitude         float64
Longtitude       float64
Regionname        object
Propertycount    float64
ParkingArea       object
Price            float64
dtype: object
```

(c) Above, we can see .describe() working it's magic, counting all the elements in for each column. As you can see, it does not include the null elements, this is why each have different values.

There also includes, the average, minimum, maximum, quartiles and standard deviation.

.dtypes shows the data types of each column in this Data Frame. As shown above, this includes float64, object & int64

In [ ]:
```
#(d) Identify any missing values and outline a plan to handle them
print("The amount of missing values is:", df.isnull().sum())
```

```
The amount of missing values is: Suburb                0
Address              0
Rooms                0
Type                 0
Method               0
SellerG              0
Date                 0
Distance             1
Postcode             1
Bedroom           8217
Bathroom          8226
Car               8728
Landsize         11810
BuildingArea     21097
YearBuilt        19306
CouncilArea          3
Latitude          7976
Longtitude        7976
Regionname           0
Propertycount        3
ParkingArea          0
Price             7610
dtype: int64
```

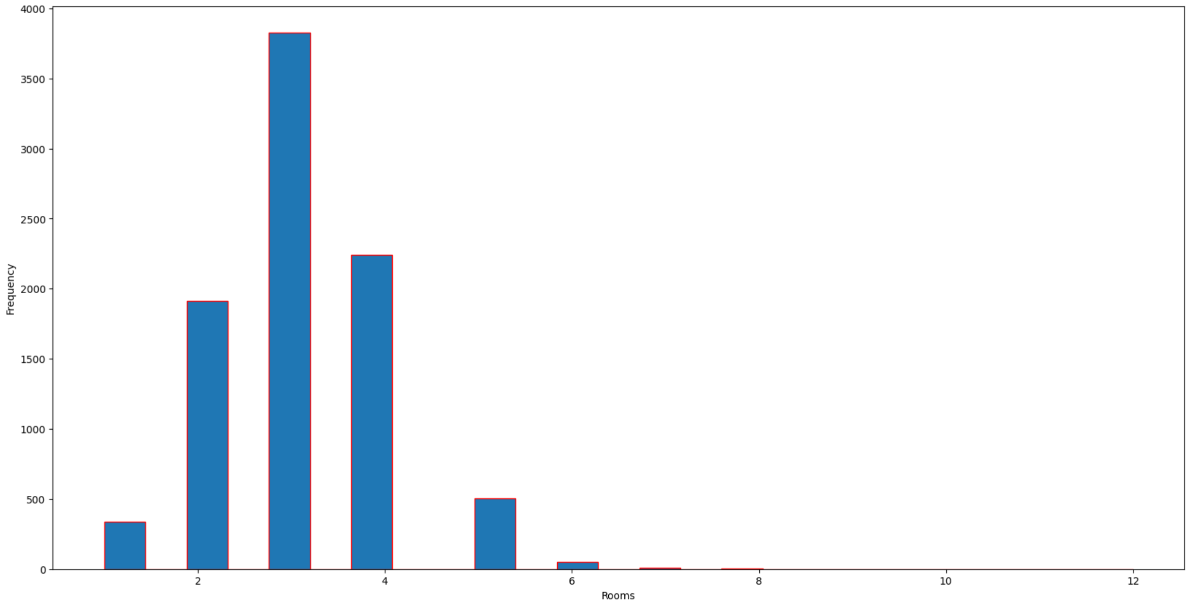
In [ ]:
```
#RemovedValues = df.fillna(0, inplace=True) #Replace nulls with 0
RemovedValues2 = df.dropna(inplace=True) #Drop rows with the null cells
#RemovedValues3 = df.fillna(mean, inplace = True)
print(df.describe())
```

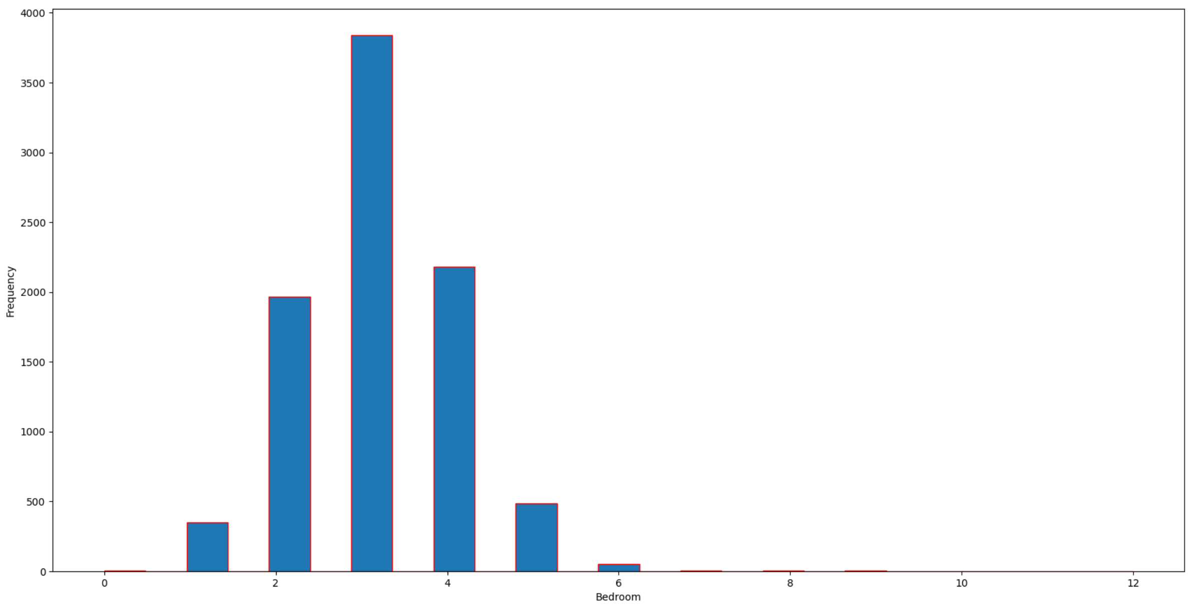
```
             Rooms     Distance     Postcode      Bedroom     Bathroom  \
count  8890.000000  8890.000000  8890.000000  8890.000000  8890.000000   
mean      3.098650    11.199663  3111.673228     3.077953     1.646344   
std       0.963765     6.812478   112.600711     0.966242     0.721565   
min       1.000000     0.000000  3000.000000     0.000000     1.000000   
25%       2.000000     6.400000  3044.000000     2.000000     1.000000   
50%       3.000000    10.200000  3084.000000     3.000000     2.000000   
75%       4.000000    13.900000  3150.000000     4.000000     2.000000   
max      12.000000    47.400000  3977.000000    12.000000     9.000000   

               Car      Landsize    YearBuilt     Latitude   Longtitude  \
count  8890.000000   8890.000000  8890.000000  8890.000000  8890.000000   
mean      1.692238    523.415411  1965.757818   -37.804519   144.991415   
std       0.975338   1061.156056    37.038495     0.090544     0.118907   
min       0.000000      0.000000  1196.000000   -38.174360   144.423790   
25%       1.000000    212.000000  1945.000000   -37.858713   144.920012   
50%       2.000000    478.000000  1970.000000   -37.798700   144.998515   
75%       2.000000    652.000000  2000.000000   -37.748978   145.064580   
max      10.000000  42800.000000  2019.000000   -37.407200   145.526350   

       Propertycount         Price  
count    8890.000000  8.890000e+03  
mean     7474.755906  1.092841e+06  
std      4374.918196  6.792854e+05  
min       249.000000  1.310000e+05  
25%      4380.000000  6.410000e+05  
50%      6567.000000  9.000000e+05  
75%     10331.000000  1.345000e+06  
max     21650.000000  9.000000e+06  
```

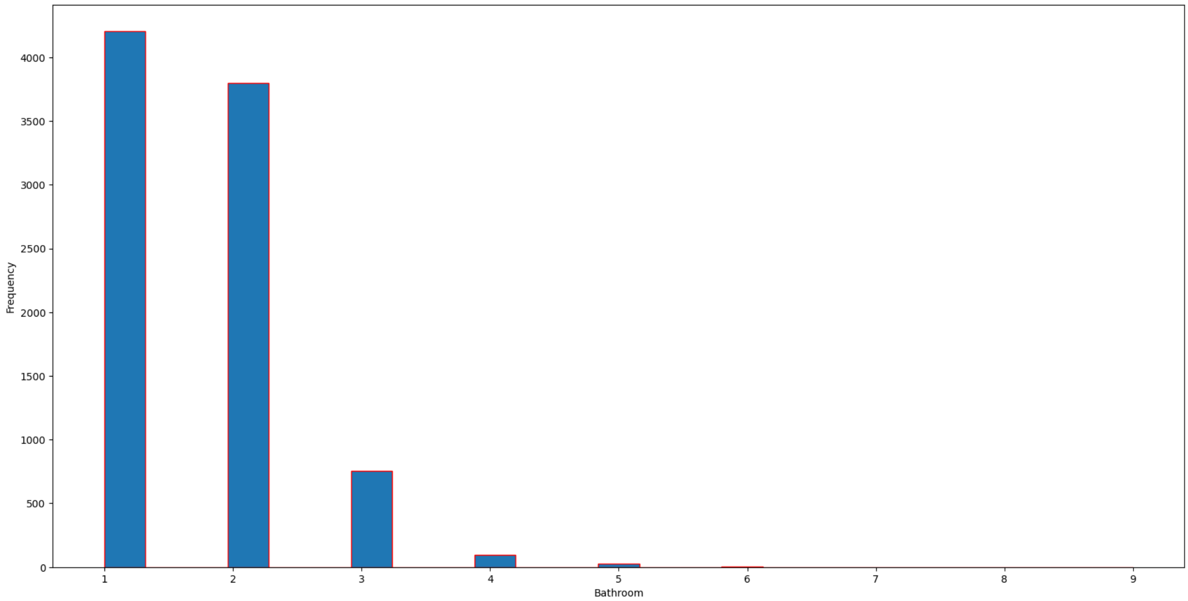
(d) The amount of missing values for each column can be seen with df.describe() but df.isnull().sum() works just as well. Now that we know they are they, we just need to handle them. This can be done being filling in the cells using .fillna or .dropna. The first only works for numbers. The second works for all, therefore, the better choice.

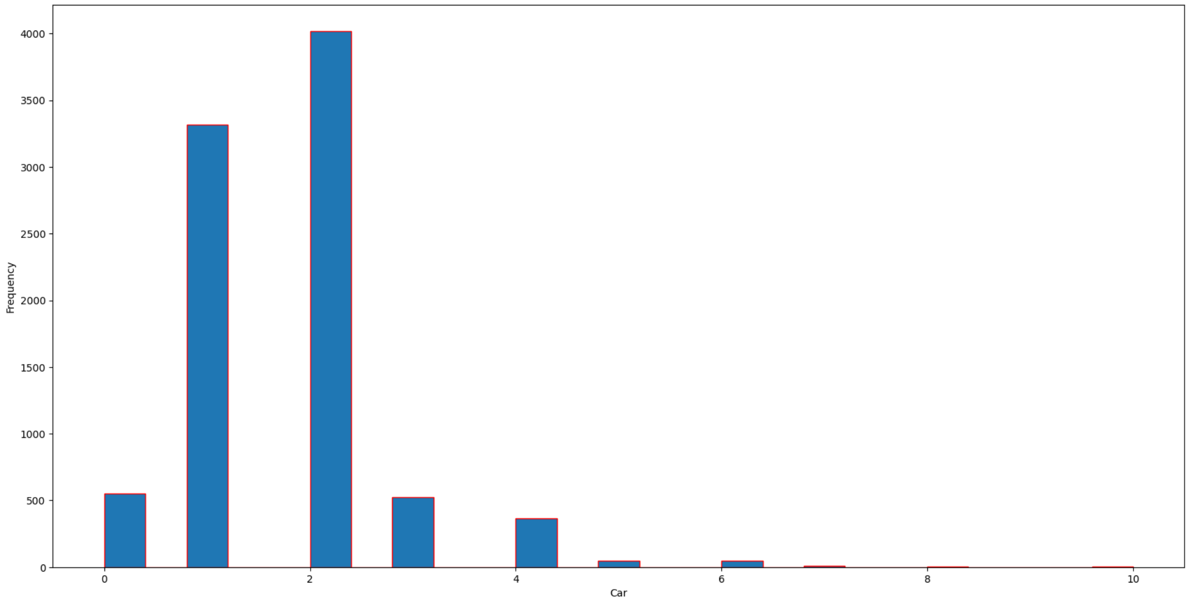
In [ ]:

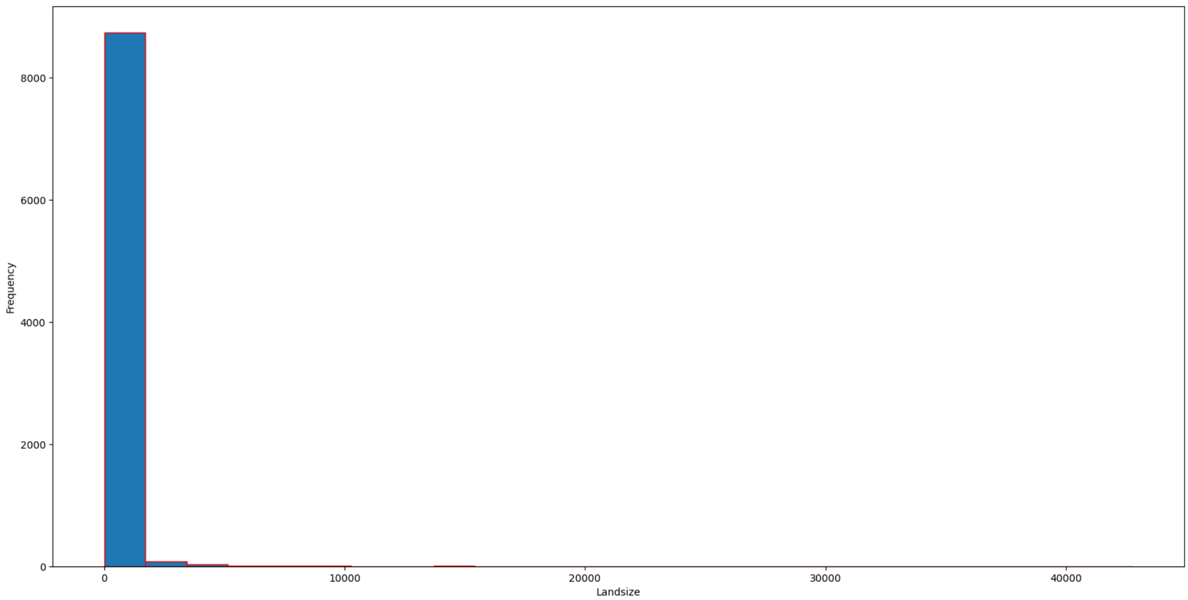
```
#2 Exploratory Data Analysis (EDA) (30 points):
#2 (a) Visualize the distribution of numeric variables using histograms and box plo
import matplotlib.pyplot as plt
import seaborn as sns

Importants = ['Rooms', 'Bedroom', 'Bathroom', 'Car','Landsize', 'Price']

for x in Importants:
    plt.figure(figsize=(20, 10))
    plt.hist(df[x], bins = 25, edgecolor = "red")
    plt.xlabel(x)
    plt.ylabel("Frequency")
    plt.show()
```

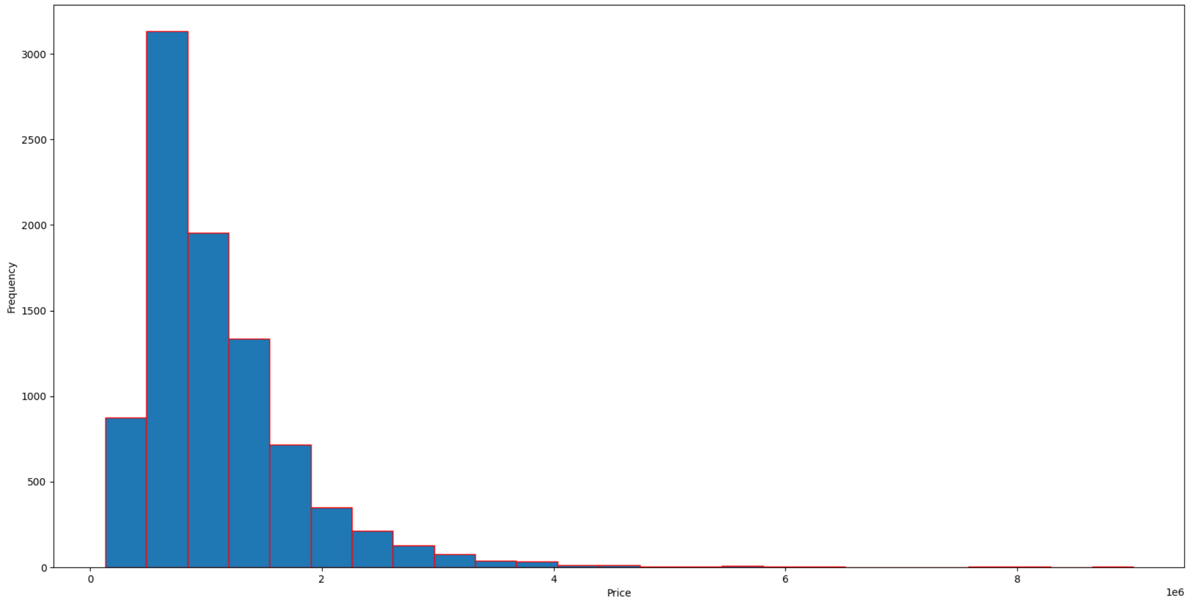




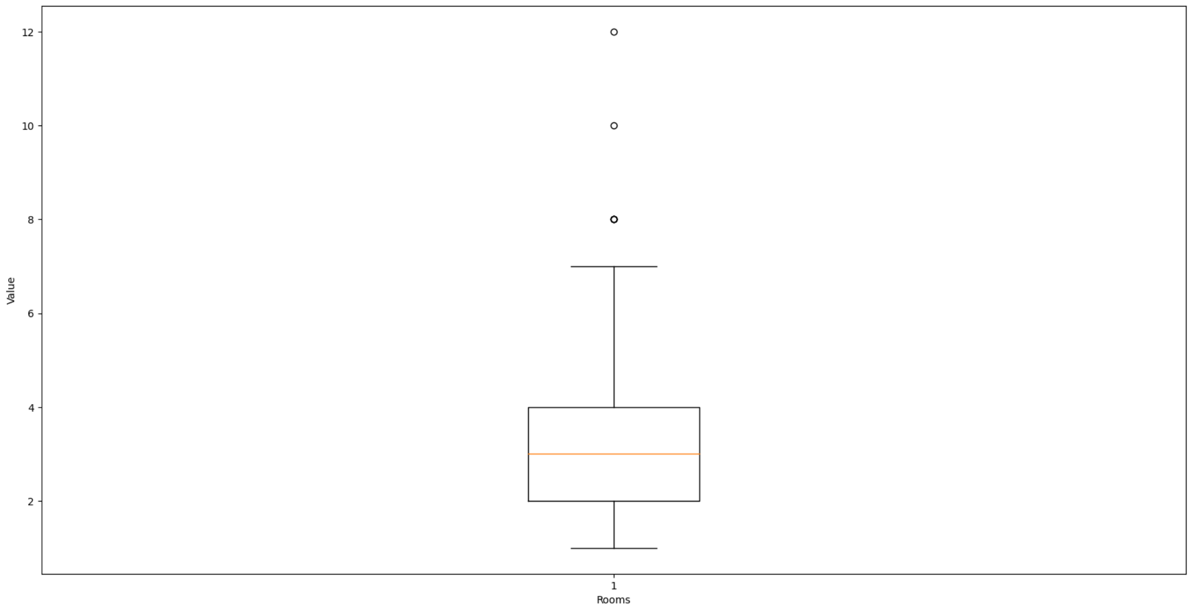

Frequency
Bathroom

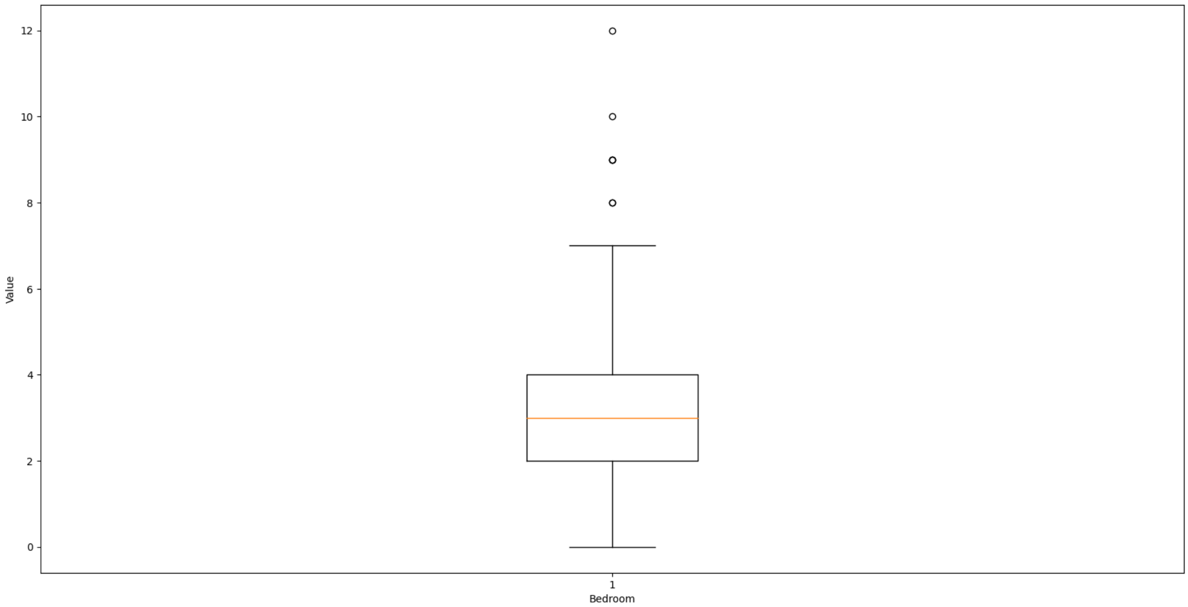

Frequency
Car

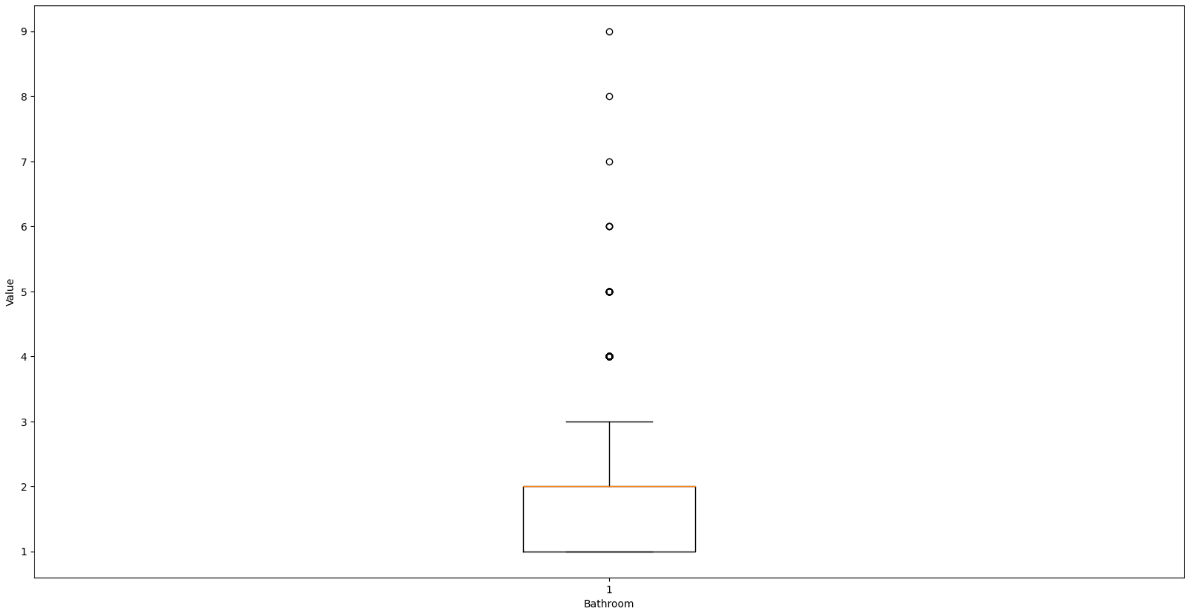

Frequency
Landsize

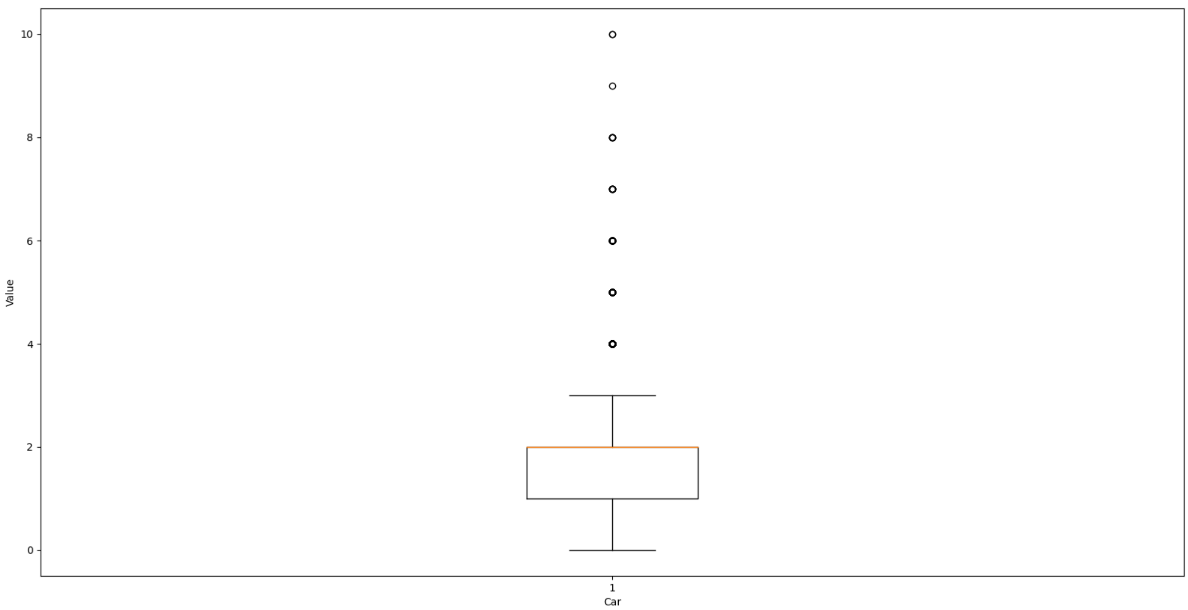


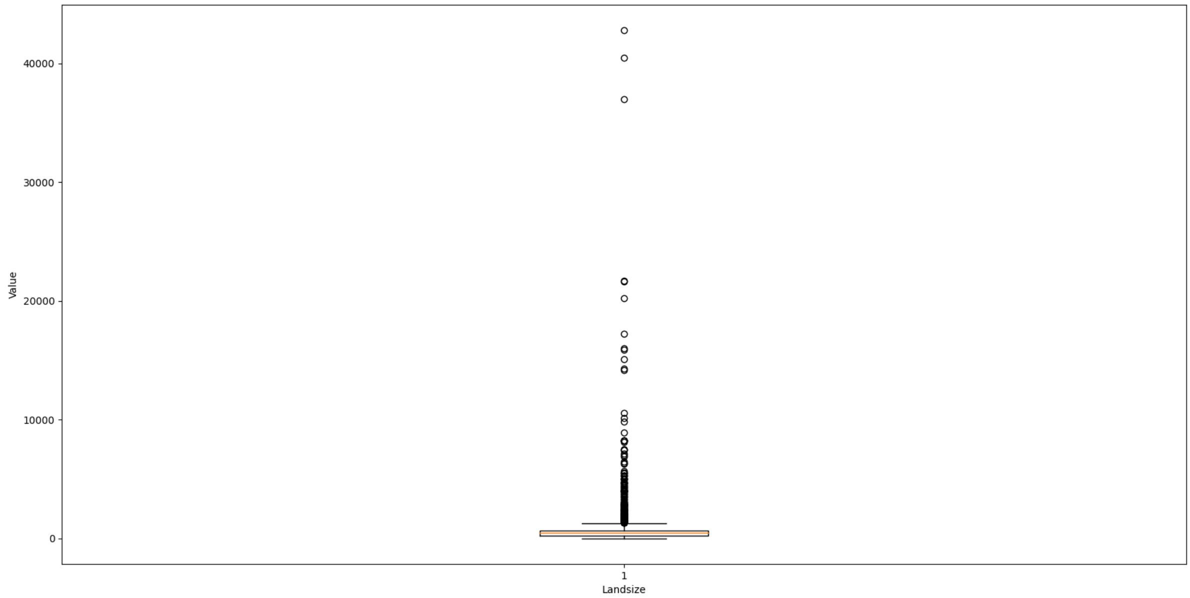
In [ ]:
```
for y in Importants:
    plt.figure(figsize=(20, 10))
    plt.boxplot(df[y])
    plt.xlabel(y)
    plt.ylabel("Value")
    plt.show()
```

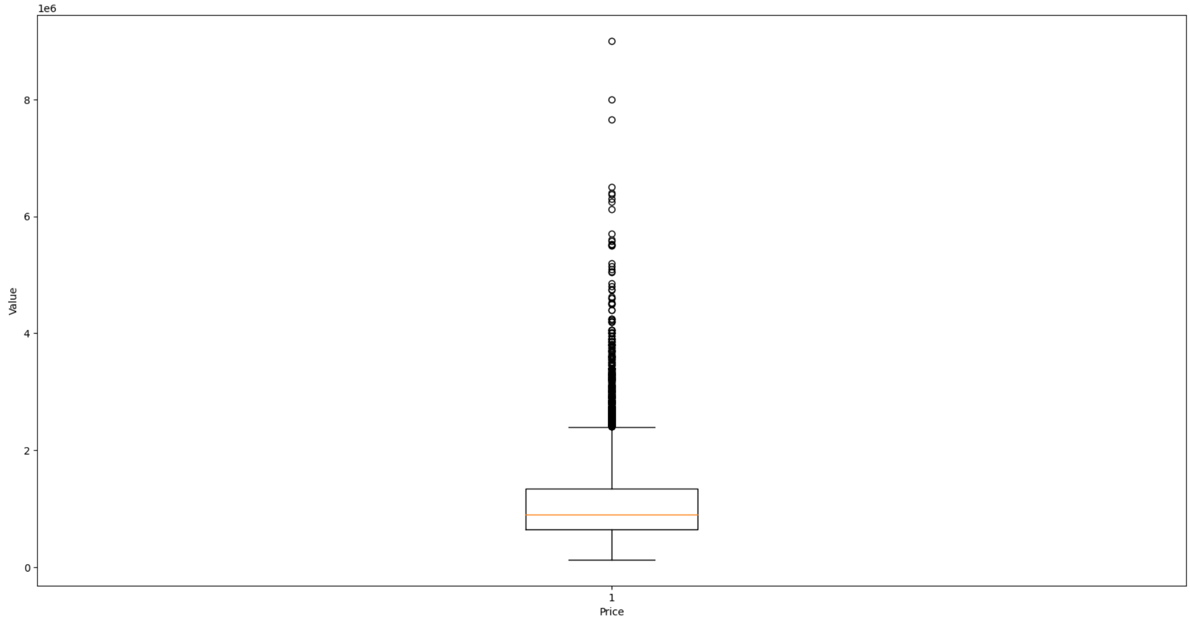



Value
12
10
8
6
4
2
0
1
Bedroom


Value
9
8
7
6
5
4
3
2
1
1
Bathroom


Value
10
8
6
4
2
0
1
Car





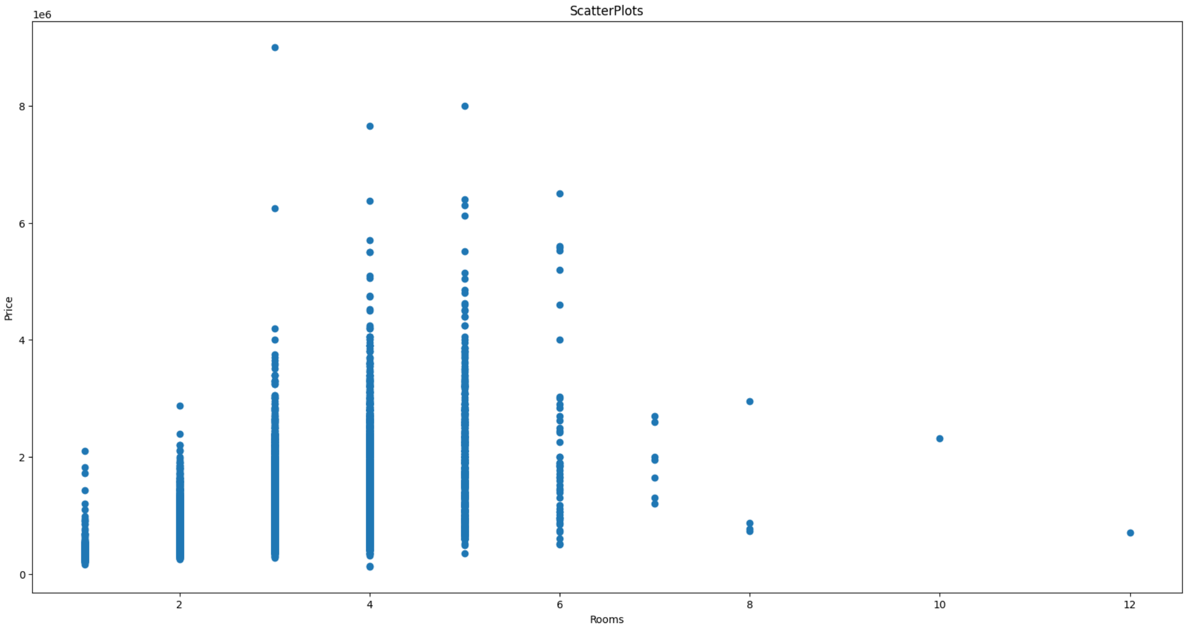
To include boxplots & histograms, we need to import the relevant libraries.

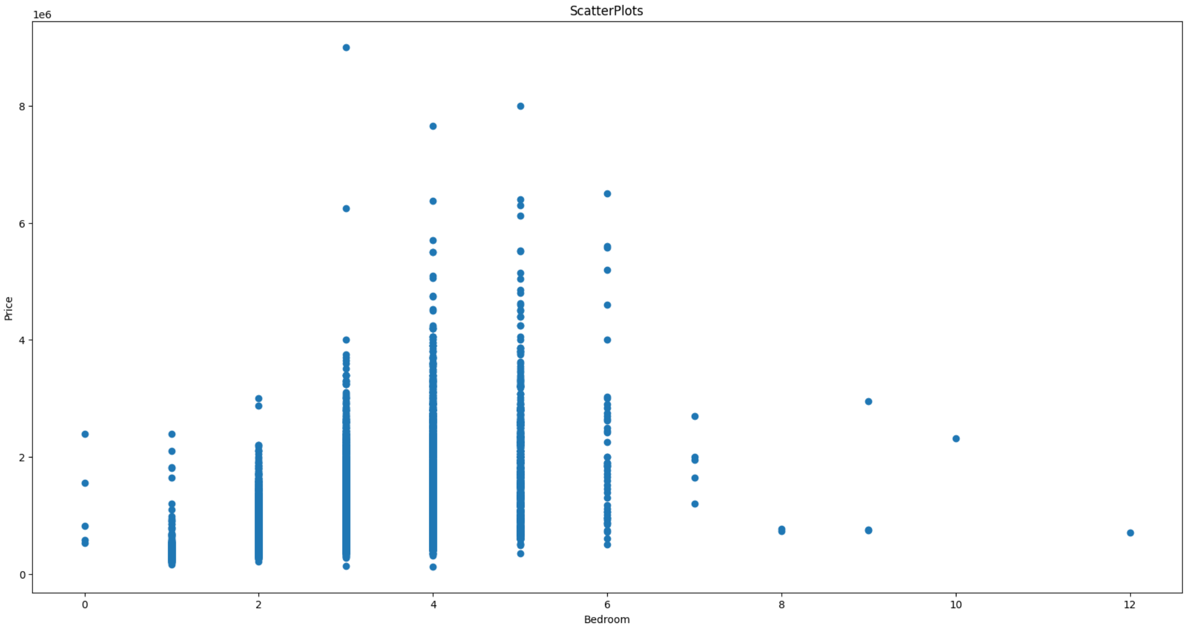
I've included Rooms, bedroom, bathroom, regionname & price as the things I would be most interested in when buying a home.

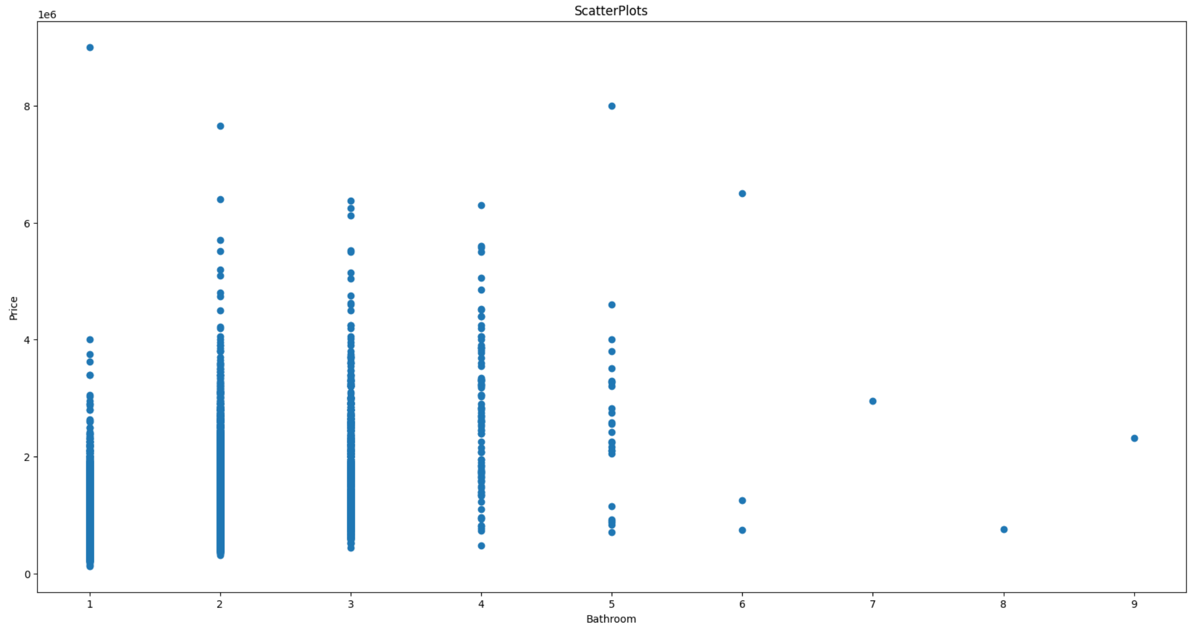
The boxplots and histogram show the distributions of the labeled variables. Again, I didn't include some things like CouncilArea or Longtitude and Latitude because I deemed them unneccessary for this question.

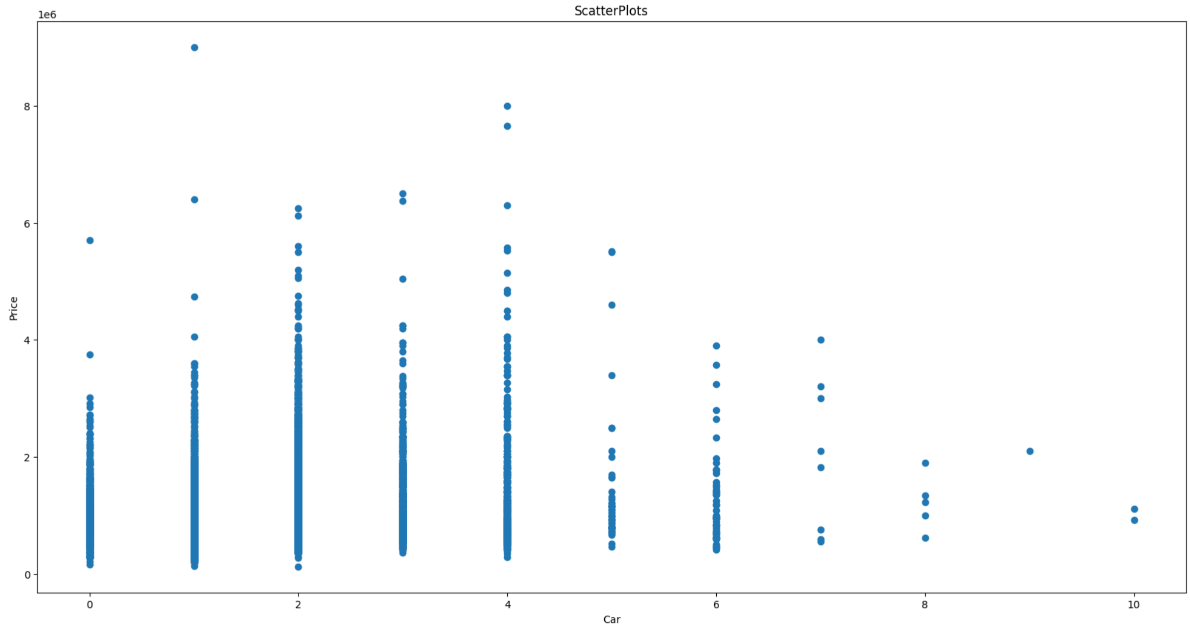
In [ ]:
```
#2(b) Explore relationships between features and the target variable using scatter

Importants2 = ['Rooms', 'Bedroom', 'Bathroom', 'Car', 'Landsize']

for z in Importants2:
    plt.figure(figsize=(20, 10))
    plt.scatter(df[z], df['Price'])
    plt.xlabel(z)
    plt.ylabel("Price")
    plt.title("ScatterPlots")
    plt.show()
```

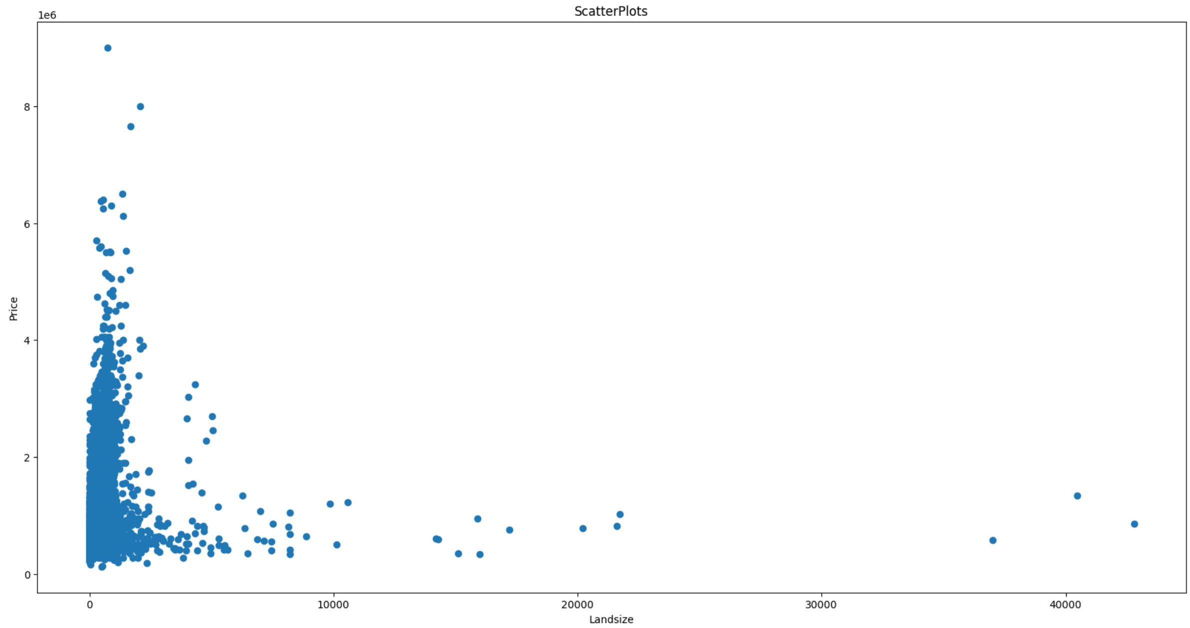
```python
#Explore relationships - Correlation
#sns.pairplot(df[Importants])
plt.figure(figsize=(20, 10))
sns.heatmap(df.corr(), annot=True, cmap='plasma', fmt='0.2f', cbar=True)
plt.title("Correlation Matrix")
plt.show()
```



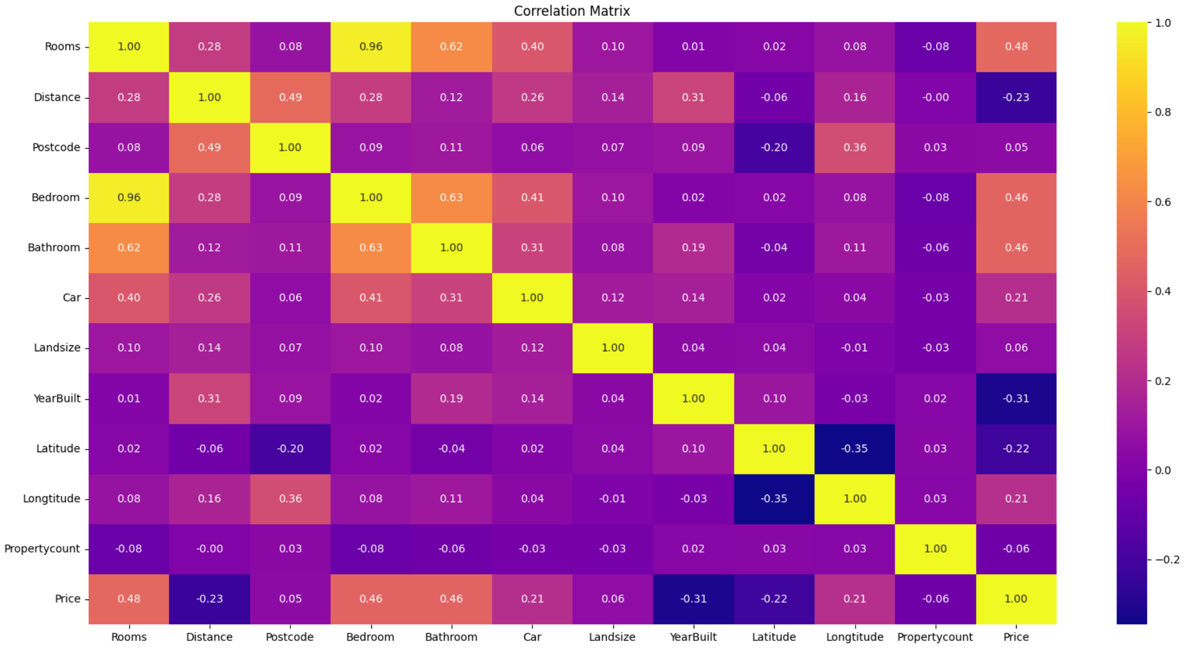



ScatterPlots
1e6
Price
Bathroom
0
2
4
6
8
1
2
3
4
5
6
7
8
9


ScatterPlots
1e6
Price
Car
0
2
4
6
8
0
2
4
6
8
10


ScatterPlots
1e6
Price
Landsize
0
2
4
6
8
0
10000
20000
30000
40000

```
<ipython-input-44-530d29665541>:19: FutureWarning: The default value of numeric_on
ly in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this war
ning.
  sns.heatmap(df.corr(), annot=True, cmap='plasma', fmt='0.2f', cbar=True)
```



(b) Here we can see the relationships between the target variable using scatter plots & correlation matrices.

I personally believe the heatmap does a lot better job, as it shows to the the user how much correlation 2 variables have.

Yellow hav the most correlation while the dark blue have the least correlation. Numbers are there to easily see the level of correlation exists.

In [ ]:

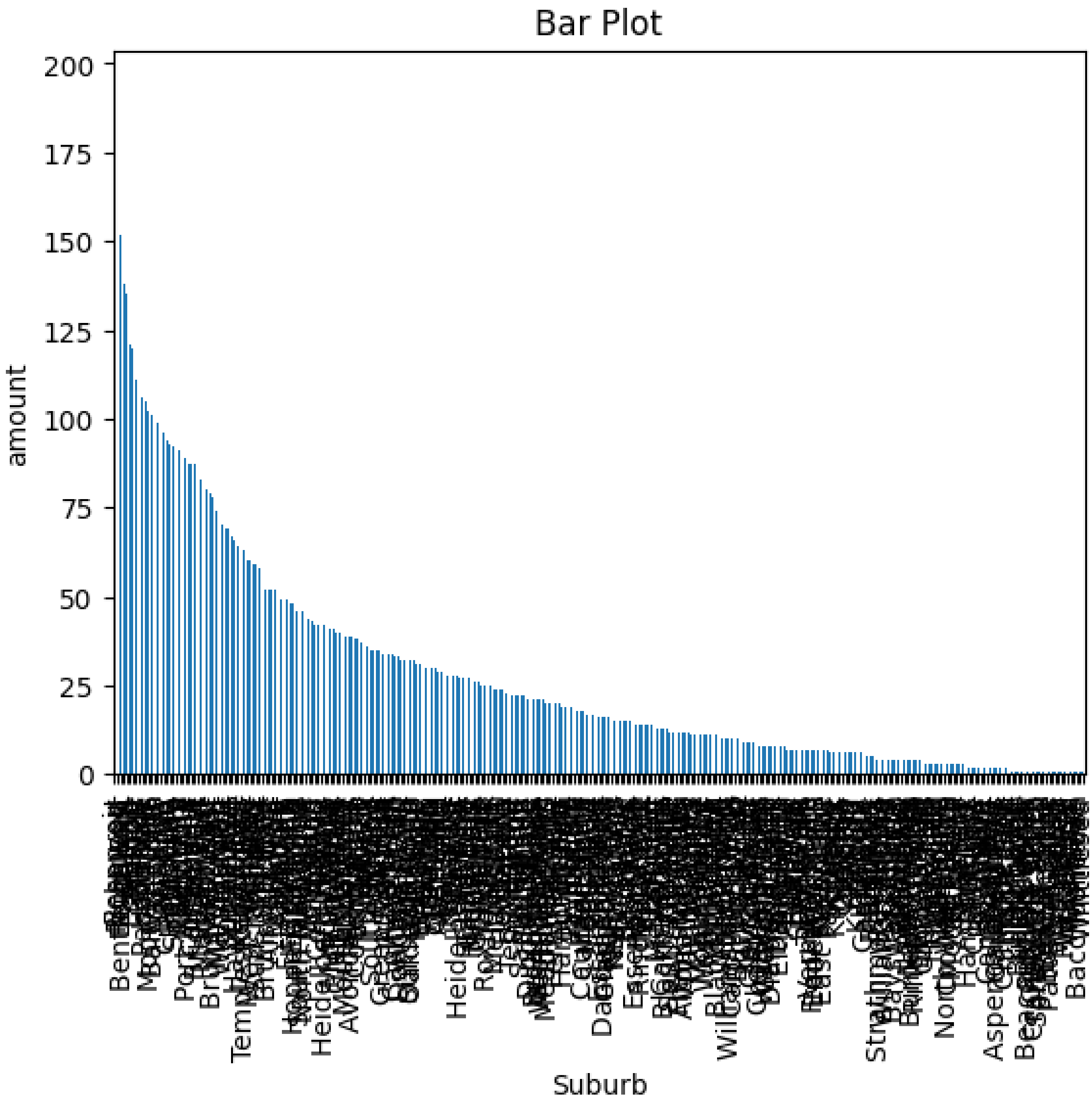
```
#(c) Examine categorical variables with bar plots and frequency tables.
CatVars = ['Suburb', 'Address', 'Type', 'Method', 'SellerG', 'Date', 'CouncilArea',

#Bar Plots
plt.figure(figsize=(20, 10))
for all in CatVars:
  plt.figure()
  df[all].value_counts().plot(kind='bar')
  plt.xlabel(all)
  plt.ylabel('amount')
  plt.title('Bar Plot')

#Frequency Table
for all2 in CatVars:
    print(f'Frequency table for {all2}:')
    print(df[all2].value_counts())
    print('\n')
```

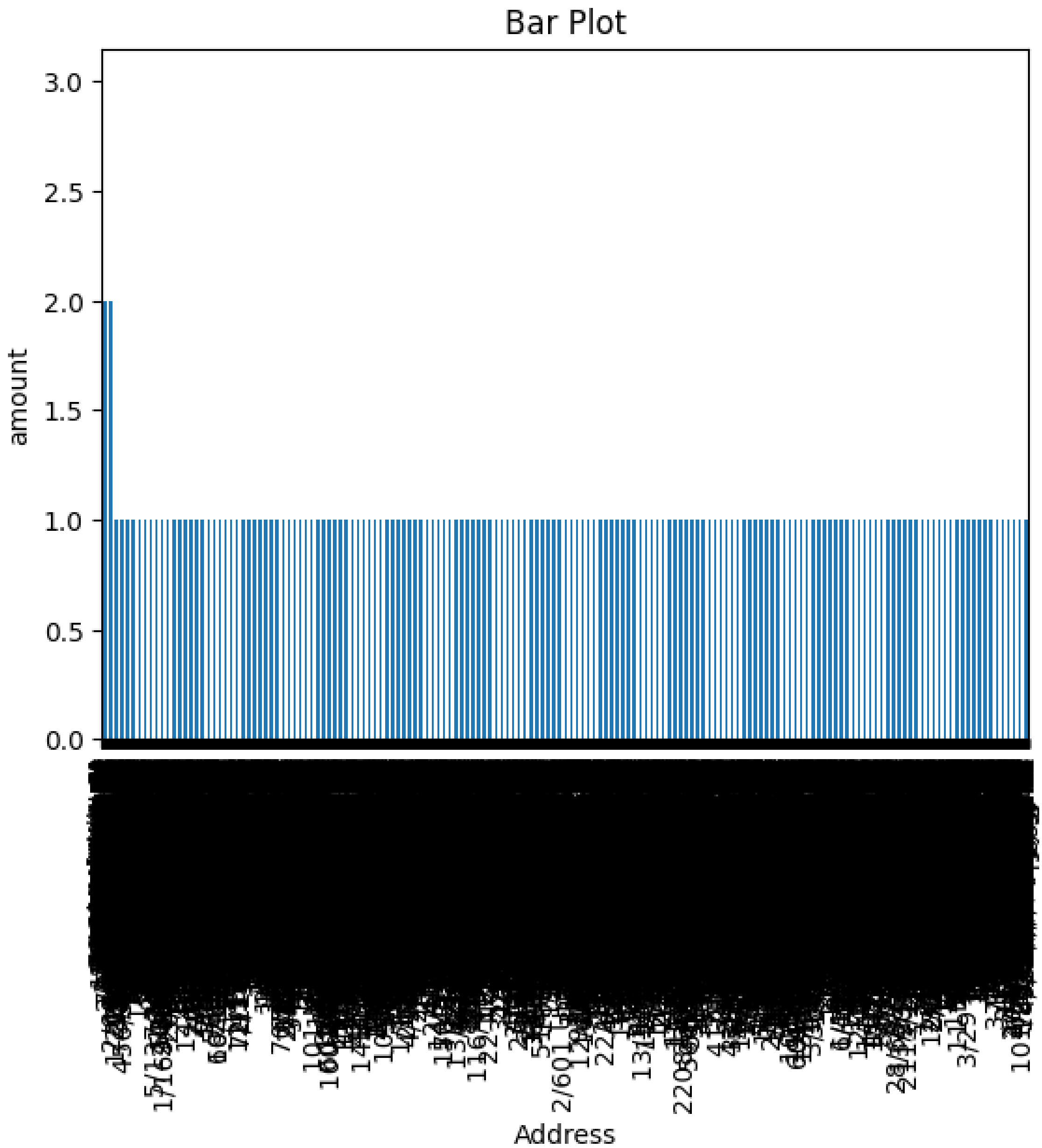
```
Frequency table for Suburb:
Reservoir         194
Richmond          155
Brunswick         152
Bentleigh East    138
Coburg            135
                 ...
Waterways           1
The Basin           1
Montrose            1
Bacchus Marsh       1
Whittlesea          1
Name: Suburb, Length: 315, dtype: int64


Frequency table for Address:
1/1 Clarendon St    3
36 Aberfeldie St    3
12 Mirams St        3
14 Northcote St     3
25 William St       3
                   ..
11/1419 High St     1
26 Audrey Cr        1
245 Carrick Dr      1
42 Kilmore Rd       1
42 Pascoe St        1
Name: Address, Length: 8767, dtype: int64


Frequency table for Type:
h    6627
u    1541
t     722
Name: Type, dtype: int64


Frequency table for Method:
S     5605
SP    1292
PI    1084
VB     846
SA      63
Name: Method, dtype: int64


Frequency table for SellerG:
Nelson                 986
Jellis                 874
Barry                  741
hockingstuart          684
Ray                    511
                      ...
Munn                     1
hockingstuart/Biggin     1
Upside                   1
Calder                   1
Weston                   1
Name: SellerG, Length: 250, dtype: int64


Frequency table for Date:
24/02/2018    227
27/05/2017    225
```

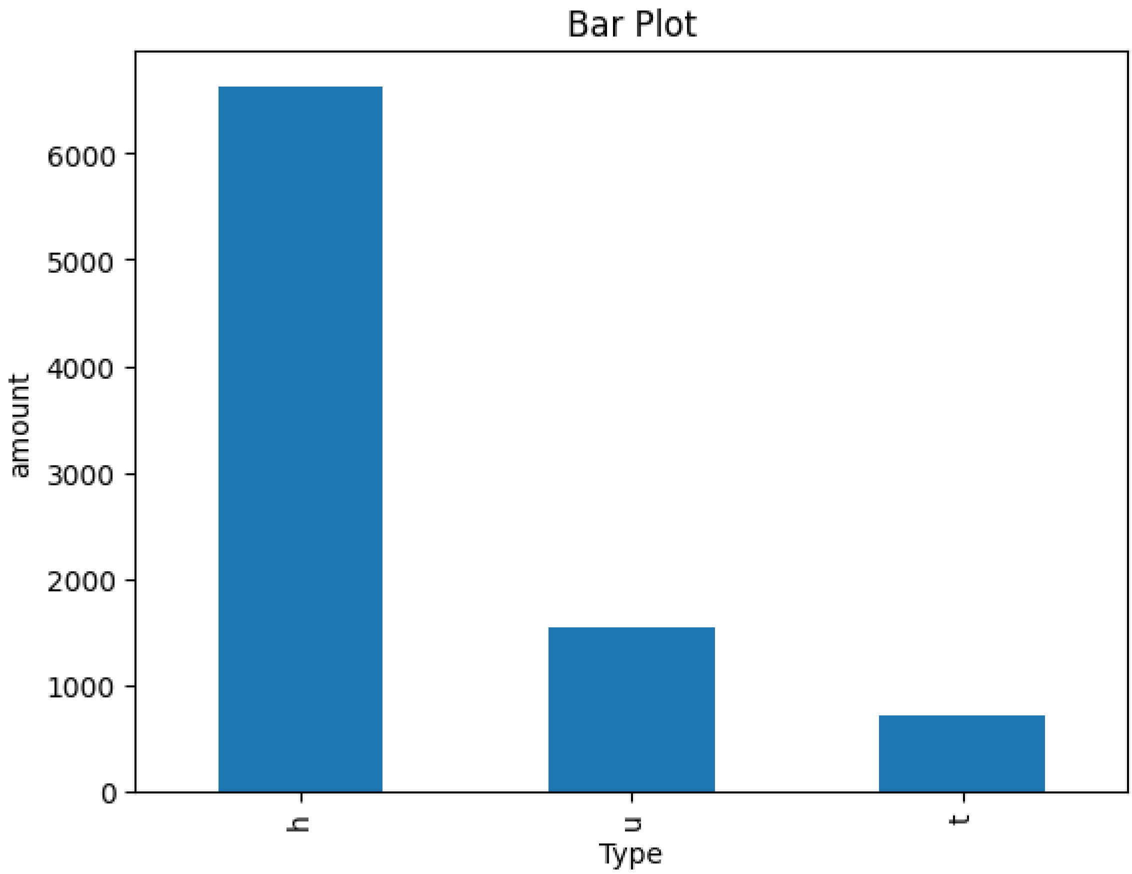
```
17/03/2018    214
3/3/2018      204
3/6/2017      202
             ...
4/2/2016       16
11/3/2017       9
20/01/2018      7
30/09/2017      5
27/01/2018      2
Name: Date, Length: 77, dtype: int64


Frequency table for CouncilArea:
Boroondara City Council           810
Darebin City Council              730
Moreland City Council             647
Moonee Valley City Council        556
Glen Eira City Council            520
Maribyrnong City Council          490
Melbourne City Council            456
Brimbank City Council             416
Banyule City Council              413
Hume City Council                 390
Bayside City Council              362
Port Phillip City Council         329
Yarra City Council                323
Monash City Council               300
Hobsons Bay City Council          289
Stonnington City Council          280
Manningham City Council           267
Whittlesea City Council           242
Kingston City Council             209
Wyndham City Council              169
Whitehorse City Council           126
Melton City Council               107
Maroondah City Council            107
Knox City Council                 103
Frankston City Council             87
Greater Dandenong City Council     51
Casey City Council                 35
Nillumbik Shire Council            28
Yarra Ranges Shire Council         20
Cardinia Shire Council             12
Macedon Ranges Shire Council       11
Mitchell Shire Council              4
Moorabool Shire Council             1
Name: CouncilArea, dtype: int64


Frequency table for Regionname:
Southern Metropolitan         2709
Northern Metropolitan         2613
Western Metropolitan          2059
Eastern Metropolitan           982
South-Eastern Metropolitan     371
Northern Victoria               62
Eastern Victoria                51
Western Victoria                43
Name: Regionname, dtype: int64


Frequency table for ParkingArea:
Attached Garage    1652
Carport            1617
```

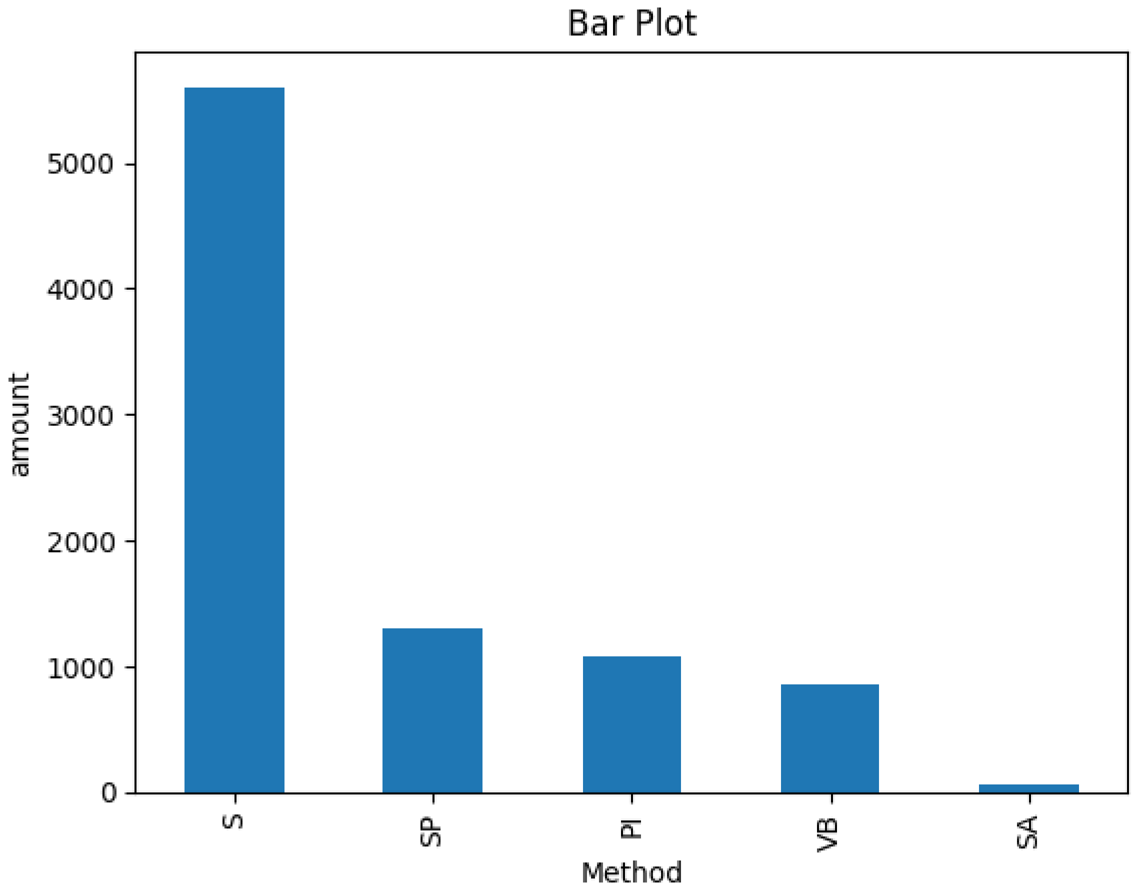
```
Detached Garage    1577
Indoor             1426
Parkade            1171
Underground         640
Outdoor Stall       536
Parking Pad         271
Name: ParkingArea, dtype: int64
```

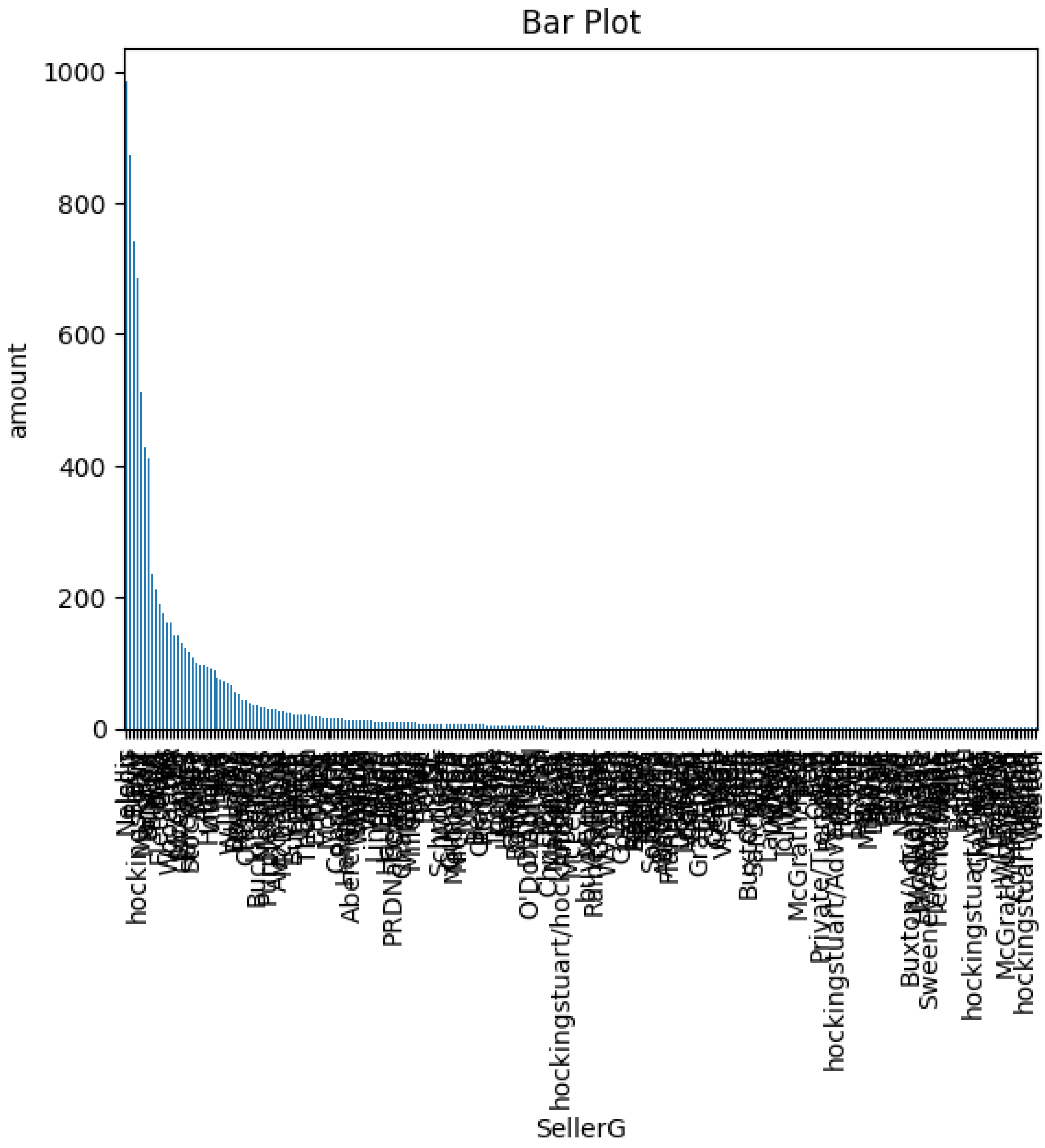
```
<Figure size 2000x1000 with 0 Axes>
```

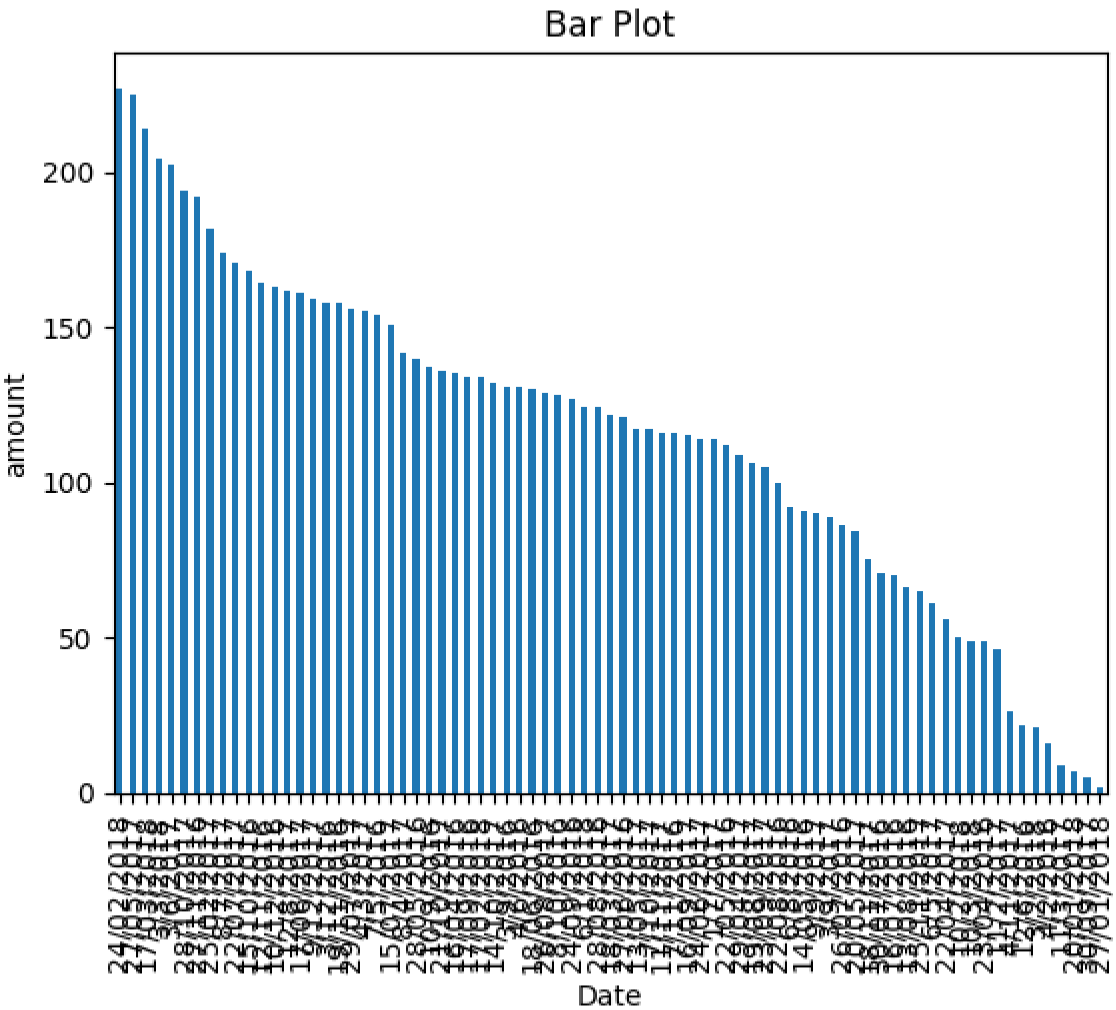


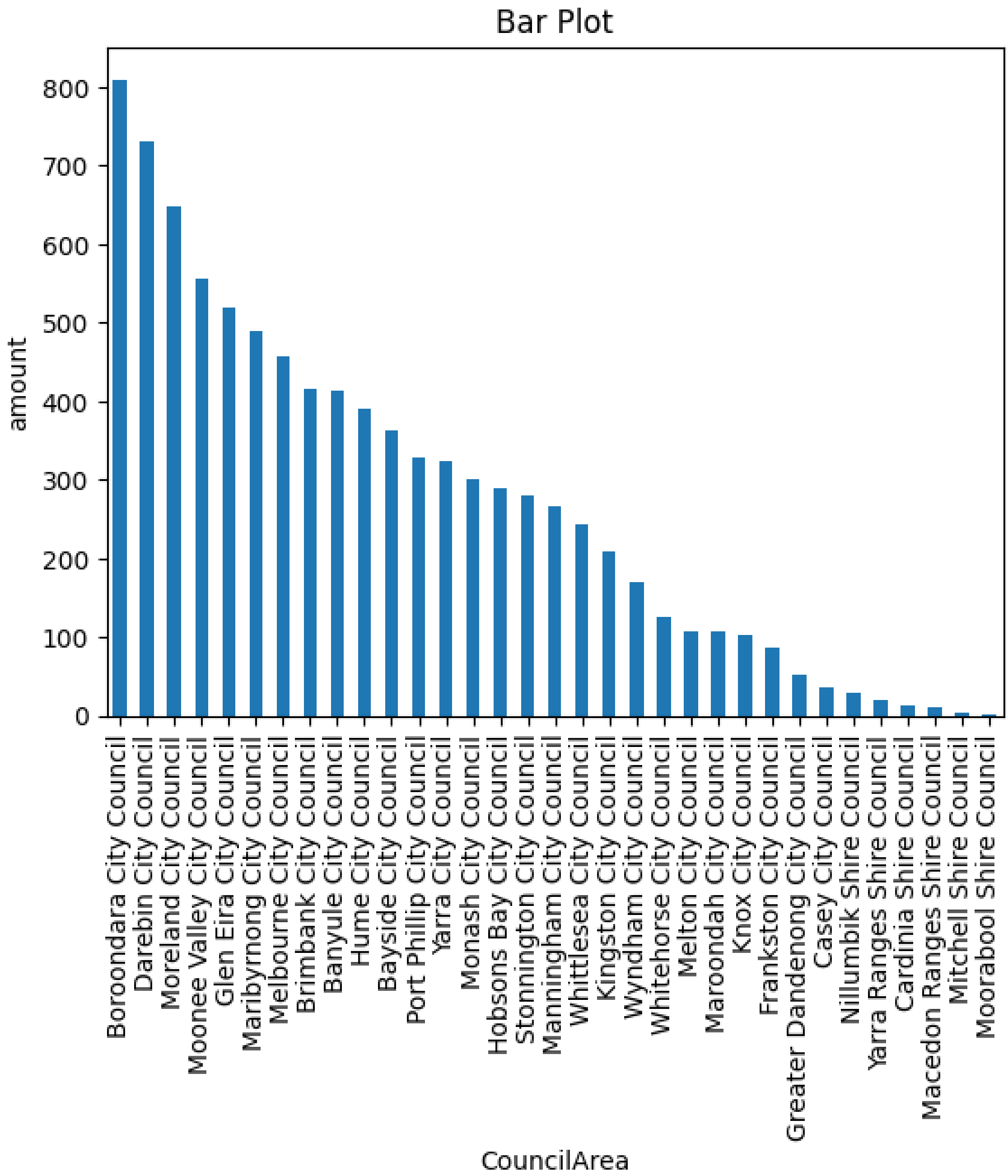

Bar Plot
amount
Address
3.0
2.5
2.0
1.5
1.0
0.5
0.0

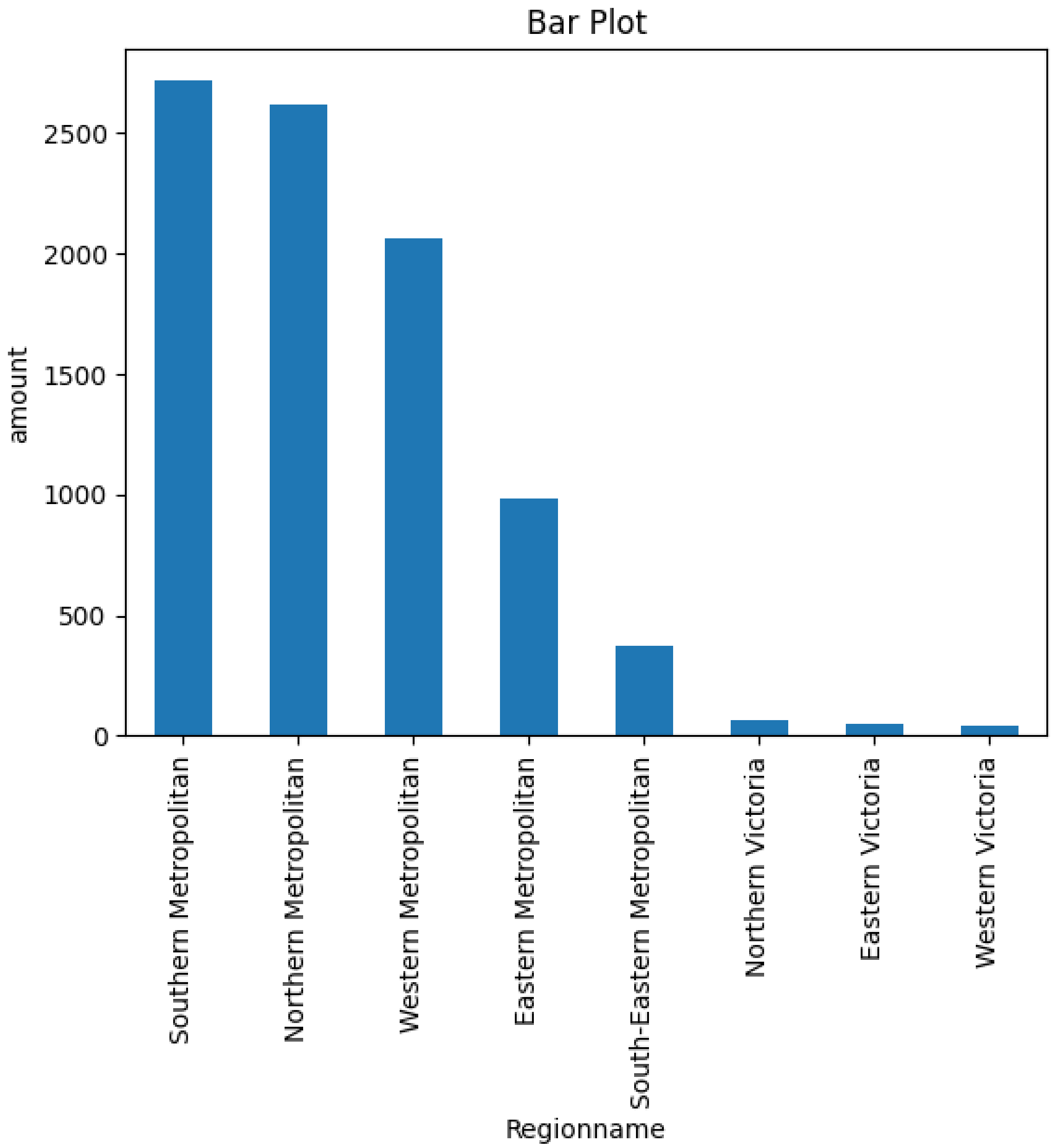

Bar Plot
amount
0
1000
2000
3000
4000
5000
6000
h
u
t
Type

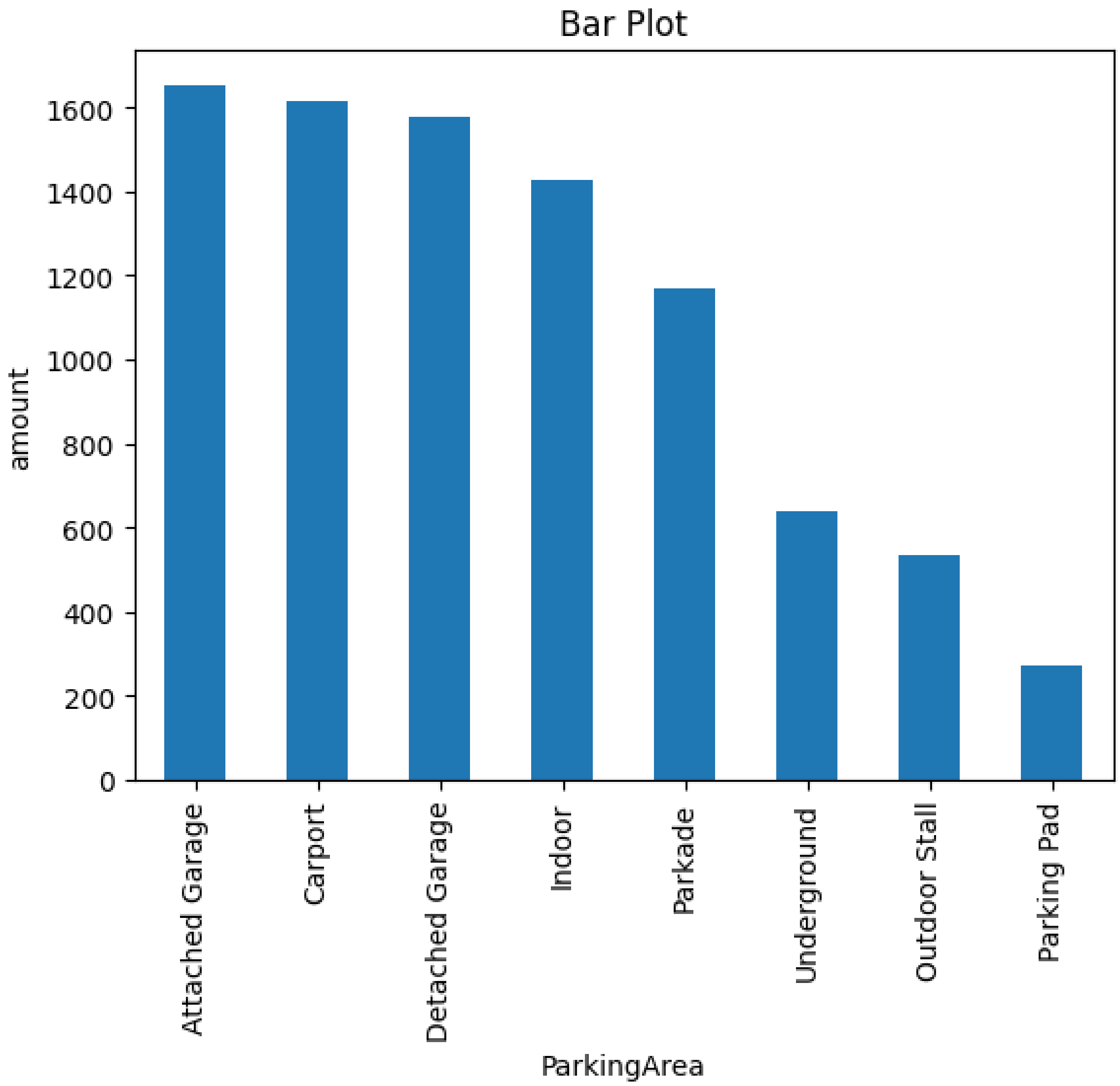

Bar Plot
amount
0
1000
2000
3000
4000
5000
S
SP
PI
VB
SA
Method


Bar Plot
amount
1000
800
600
400
200
0
SellerG


Bar Plot
amount
200
150
100
50
0
Date


Bar Plot
amount
800
700
600
500
400
300
200
100
0
Boroondara City Council
Darebin City Council
Moreland City Council
Moonee Valley City Council
Glen Eira City Council
Maribyrnong City Council
Melbourne City Council
Brimbank City Council
Banyule City Council
Hume City Council
Bayside City Council
Port Phillip City Council
Yarra City Council
Monash City Council
Hobsons Bay City Council
Stonnington City Council
Manningham City Council
Whittlesea City Council
Kingston City Council
Wyndham City Council
Whitehorse City Council
Melton City Council
Maroondah City Council
Knox City Council
Frankston City Council
Greater Dandenong City Council
Casey City Council
Nillumbik Shire Council
Yarra Ranges Shire Council
Cardinia Shire Council
Macedon Ranges Shire Council
Mitchell Shire Council
Moorabool Shire Council
CouncilArea


Bar Plot
amount
0
500
1000
1500
2000
2500
Southern Metropolitan
Northern Metropolitan
Western Metropolitan
Eastern Metropolitan
South-Eastern Metropolitan
Northern Victoria
Eastern Victoria
Western Victoria
Regionname



(c) Bar Plots show distribution of each of the categorical variables.

Starting with Suburbs, we can see the areas that have the most amount of houses on sale. Sadly, since it is such a large amount of suburbs, it makes it very hard to read all the different bars.

Address is interesting because the address is listed for all houses across the board, each having a value of 1 except for a few that have 2 listed. This is most likely someone selling one house but using 2 different agents to sell from.

Type only has 3 bar charts, making it clear and visible what type of house is being sold, with 'h' being the most popular.

The most popular method is S, the least popular is SA.

The most popular Seller is Nelson.

Funnily enought the Dates do, a lot of people putting their houses on sale on the same day.

The Boundary City Council seem to have the most houses on sale while Morabool Shire Council have the least amount of houses ons sale for CouncilArea

The Southern, Western & Northern Metropolitan Area have a considerable amount of houses for sale, WHile Northern, Eastern & Western Victoria have a very low amount on sale in comparison.

Many of these houses have Attached Garages, Carports, Detached Garages & Indoor parking. The least common is the ParkingPad.

Thanks to the frequency table, we can get the exact values, for example, we can see Nelson has 986 houses on sale or S is the most frequent house type having 5605 in total

In [ ]:
```
#(d) Identify potential outliers and discuss their impact on the dataset.

IQR = df[Importants].quantile(.75) - df[Importants].quantile(.25)
Q1 = df[Importants].quantile(.25)
Q3 = df[Importants].quantile(.75)
Q1A = (Q1- 1.5 * IQR)
Q3A =  (Q3 + 1.5 * IQR)

# Identify potential outliers
Outliers = ((df[Importants] < Q1A) | (df[Importants] > Q3A))

# Count Outliers
OutliersCount = Outliers.sum()
print(OutliersCount)
```

```
Rooms          6
Bedroom        7
Bathroom     129
Car          479
Landsize     209
Price        420
dtype: int64
```

(d) Potential Outliers can be found using the Interquartile range. This is the middle 50% of values between the first quarter point (25th Percentile) and the third quarter point, (75th percientile.)

Outliers can skew the data set, it can affect the mean & median values of a variable. However, it is best to understand these outliers and why they are there.

In this case, we can see the outliers. Rooms have the least amount outliers, followed by Bedroom then Bathroom.

Car has the highest amount of Outliers with 479 in total, followed by Price with 420.

420 Price outliers mean there is a high amount of extremely pricey houses or low cost homes. Understanding this is crucial because the cost could be proportional to the amount of bedrooms, bathrooms, etc.

129 outliers for Bathroom is notable because people do want to know how many bathrooms a house has. It could have 1 or it could have 0 for all the buyer knows.

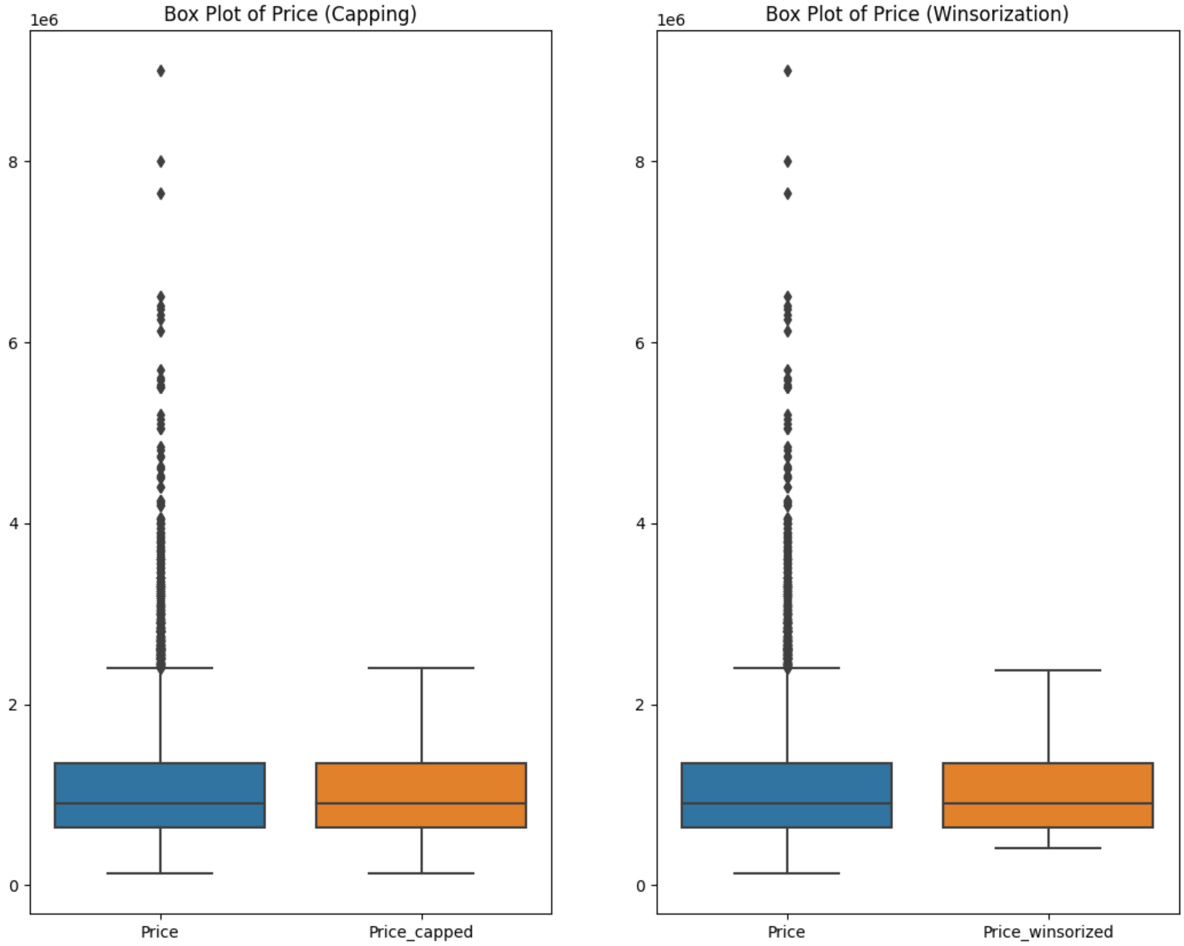
Landsize is also notable as they might not even being selling just a house at that point, but a house with a proper amount of land. Or the 'house' could literally be a single room being both the bedroom & bathroom.

Car having as much outliers as it does could indicate that there are no parking spaces or there could be 'enough space' for more parking spots.

It's important to understand why these outliers exist. They should be investigated because these outliers could be their on purpose or reveal important informaton. Data Validation can help determine its true nature and how these outliers affect the dataset

In [ ]:

```
#Feature Engineering (40 points):
#Apply at least five feature engineering techniques to improve the dataset for mode
#Some ideas include:
#Handling missing data (e.g., imputation methods). (DONE)
#Encoding categorical variables (e.g., one-hot encoding or label encoding).
#Creating interaction features or polynomial features. (INTERACTION - DONE)
#Scaling or normalizing numeric features. (DONE)
#Handling Outliers (DONE)
#Provide clear explanations and justifications for each feature engineering step.
from scipy.stats.mstats import winsorize

#Capping
Q1_price = df['Price'].quantile(0.25)
Q3_price = df['Price'].quantile(0.75)
IQR_price = Q3_price - Q1_price
UpperP = (Q3_price+1.5*IQR_price)
LowerP = (Q1_price-1.5*IQR_price)

df['Price_capped'] = df['Price'].clip(lower=LowerP, upper=UpperP)
df['Price_winsorized'] = winsorize(df['Price'], limits=(0.05, 0.05))

# BP - Capped
plt.figure(figsize=(20, 10))
plt.subplot(1, 3, 1)
sns.boxplot(data=df[['Price', 'Price_capped']])
plt.title('Box Plot of Price (Capping)')
# BP - Winning
plt.subplot(1, 3, 2)
sns.boxplot(data=df[['Price', 'Price_winsorized']])
plt.title('Box Plot of Price (Winsorization)')
plt.show()
```



3 (a) Using the IQR is a good way of handling outliers. As done in 2(d), we grab the outliers but instead go for Price only ths time. Boxplot shows how the outliers are changed as they show the quartiles, outliers & the final product.

Comparing the Price capped to the Winsorized data, it's hard to see if the outliers are non-existent. That's what winsorizing does. It replaces the extremes with more regular outliers. If there aren't extremely far from the majority of the data, it's very hard to notice.

The approach is correct and provides a clear comparison of the 'Price' variable before and after both capping and winsorization using box plots. It allows you to observe how these outlier handling techniques affect the distribution and representation of the 'Price' data. Handling Outliers is useful for this dataset as we can visualize changes using graphs.

In [ ]:
```
#3(b) Handling missing data (e.g., imputation methods).

#Shown Here:

#print("The amount of missing values is:", df.isnull().sum())
#RemovedValues = df.fillna(0, inplace=True) #Replace nulls with 0
#RemovedValues2 = df.dropna(inplace=True) #Drop rows with the null cells
#RemovedValues3 = df.fillna(mean, inplace = True)
```

This is something we have already tried when we were told to identify any missing values & outline a plan to handle them.

They have been handled, and have been identified. This was done by using isnull() & sum(). Missing data can give false information to the user. It must be handled correctly using one of the methods above or a smimilar method.

The values missing in the dataset can be missed intentionally, randomly, or missed out for a reason. So missing data is considered a problem and needs to be handled before proceeding to the next pipeline of model development.

Missing values present in the dataset can impact the performance of the model by creating a bias in the dataset. This bias can create a lack of relatability and trustworthiness in the dataset. The loss in values might contain crucial insights or information for model development.

In [ ]:

```
#3(c)  Iterate through pairs of numeric columns and create interaction features
for i in range(len(Importants2)):
    for j in range(i+1, len(Importants2)):
        brag1 = Importants2[i]
        brag2 = Importants2[j]
        checkerville = f'{brag1}_{brag2}_interaction'
        df[checkerville] = df[brag1] * df[brag2]

print(df.head())
```

```
       Suburb          Address  Rooms Type Method        SellerG      Date  \
1  Airport West    154 Halsey Rd      3    t     PI         Nelson  3/9/2016
2   Albert Park   105 Kerferd Rd      2    h      S  hockingstuart  3/9/2016
5    Alphington       6 Smith St      4    h      S          Brace  3/9/2016
6    Alphington  5/6 Yarralea St      3    h      S         Jellis  3/9/2016
7        Altona     158 Queen St      3    h     VB           Greg  3/9/2016

   Distance  Postcode  Bedroom  ...  Rooms_Bedroom_interaction  \
1      13.5    3042.0      3.0  ...                        9.0
2       3.3    3206.0      2.0  ...                        4.0
5       6.4    3078.0      3.0  ...                       12.0
6       6.4    3078.0      3.0  ...                        9.0
7      13.8    3018.0      3.0  ...                        9.0

   Rooms_Bathroom_interaction  Rooms_Car_interaction  \
1                         6.0                    3.0
2                         2.0                    0.0
5                         8.0                   16.0
6                         6.0                    6.0
7                         6.0                    3.0

  Rooms_Landsize_interaction  Bedroom_Bathroom_interaction  \
1                      909.0                           6.0
2                      240.0                           2.0
5                     3412.0                           6.0
6                      624.0                           6.0
7                     1056.0                           6.0

  Bedroom_Car_interaction  Bedroom_Landsize_interaction  \
1                     3.0                         909.0
2                     0.0                         240.0
5                    12.0                        2559.0
6                     6.0                         624.0
7                     3.0                        1056.0

   Bathroom_Car_interaction Bathroom_Landsize_interaction  \
1                       2.0                         606.0
2                       0.0                         120.0
5                       8.0                        1706.0
6                       4.0                         416.0
7                       2.0                         704.0

   Car_Landsize_interaction
1                     303.0
2                       0.0
5                    3412.0
6                     416.0
7                     352.0

[5 rows x 34 columns]
```

(c) Iterative the dataset is an important Feature Engineering technique for machine learning. It can transform the data to better represent the data & improve the models overall performance. In this case it makes the model easier to interpret.

In [ ]:

```
#3(d) Scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Minmax
df[Importants2] = scaler.fit_transform(df[Importants2])
```

```
print("Scaled DF:")
print(df.head())
```

```
Scaled DF:
          Suburb           Address     Rooms Type Method        SellerG  \
1  Airport West     154 Halsey Rd  0.181818    t     PI         Nelson
2   Albert Park   105 Kerferd Rd  0.090909    h      S  hockingstuart
5    Alphington       6 Smith St  0.272727    h      S          Brace
6    Alphington  5/6 Yarralea St  0.181818    h      S         Jellis
7        Altona      158 Queen St  0.181818    h     VB           Greg

       Date  Distance  Postcode   Bedroom  ...  Rooms_Bedroom_interaction  \
1  3/9/2016      13.5    3042.0  0.250000  ...                        9.0
2  3/9/2016       3.3    3206.0  0.166667  ...                        4.0
5  3/9/2016       6.4    3078.0  0.250000  ...                       12.0
6  3/9/2016       6.4    3078.0  0.250000  ...                        9.0
7  3/9/2016      13.8    3018.0  0.250000  ...                        9.0

   Rooms_Bathroom_interaction  Rooms_Car_interaction  \
1                         6.0                    3.0
2                         2.0                    0.0
5                         8.0                   16.0
6                         6.0                    6.0
7                         6.0                    3.0

  Rooms_Landsize_interaction  Bedroom_Bathroom_interaction  \
1                      909.0                           6.0
2                      240.0                           2.0
5                     3412.0                           6.0
6                      624.0                           6.0
7                     1056.0                           6.0

  Bedroom_Car_interaction  Bedroom_Landsize_interaction  \
1                     3.0                         909.0
2                     0.0                         240.0
5                    12.0                        2559.0
6                     6.0                         624.0
7                     3.0                        1056.0

   Bathroom_Car_interaction Bathroom_Landsize_interaction  \
1                       2.0                          606.0
2                       0.0                          120.0
5                       8.0                         1706.0
6                       4.0                          416.0
7                       2.0                          704.0

   Car_Landsize_interaction
1                     303.0
2                       0.0
5                    3412.0
6                     416.0
7                     352.0

[5 rows x 34 columns]
```

(d) Scaling the target value is a good idea in regression modelling; scaling of the data makes it easy for a model to learn and understand the problem.

Scaling ensures there is an equal amount of influence on the model. Scaling prevents larger numbers from affecting the dataset too much. In this case, it can handle outliers and allow for multivariate analysis. Like the many other features listed, Scaling is useful here as improves the model performance and its interpretability.

In [ ]:
```python
#(e) Normalizing
from sklearn.preprocessing import StandardScaler
scaler2 = StandardScaler()

df[Importants2] = scaler.fit_transform(df[Importants2])
print("Normalized DF:")
print(df.head())
```

```
Normalized DF:
         Suburb           Address     Rooms Type Method        SellerG  \
1  Airport West    154 Halsey Rd  0.181818    t     PI         Nelson
2   Albert Park   105 Kerferd Rd  0.090909    h      S  hockingstuart
5    Alphington       6 Smith St  0.272727    h      S          Brace
6    Alphington  5/6 Yarralea St  0.181818    h      S         Jellis
7        Altona     158 Queen St  0.181818    h     VB           Greg

       Date  Distance  Postcode   Bedroom  ...  Rooms_Bedroom_interaction  \
1  3/9/2016      13.5    3042.0  0.250000  ...                        9.0
2  3/9/2016       3.3    3206.0  0.166667  ...                        4.0
5  3/9/2016       6.4    3078.0  0.250000  ...                       12.0
6  3/9/2016       6.4    3078.0  0.250000  ...                        9.0
7  3/9/2016      13.8    3018.0  0.250000  ...                        9.0

   Rooms_Bathroom_interaction  Rooms_Car_interaction  \
1                         6.0                    3.0
2                         2.0                    0.0
5                         8.0                   16.0
6                         6.0                    6.0
7                         6.0                    3.0

  Rooms_Landsize_interaction  Bedroom_Bathroom_interaction  \
1                      909.0                           6.0
2                      240.0                           2.0
5                     3412.0                           6.0
6                      624.0                           6.0
7                     1056.0                           6.0

  Bedroom_Car_interaction  Bedroom_Landsize_interaction  \
1                     3.0                         909.0
2                     0.0                         240.0
5                    12.0                        2559.0
6                     6.0                         624.0
7                     3.0                        1056.0

   Bathroom_Car_interaction Bathroom_Landsize_interaction  \
1                       2.0                          606.0
2                       0.0                          120.0
5                       8.0                         1706.0
6                       4.0                          416.0
7                       2.0                          704.0

   Car_Landsize_interaction
1                     303.0
2                       0.0
5                    3412.0
6                     416.0
7                     352.0

[5 rows x 34 columns]
```

(e) Normalization can have various meanings, in the simplest case normalization means adjusting all the values measured in the different scales, in a common scale.

This is the method of rescaling data where we try to fit all the data points between the range of 0 to 1 so that the data points can become closer to each other.

It is a very common approach to scaling the data. In this method of scaling the data, the minimum value of any feature gets converted into 0 and the maximum value of the feature gets converted into 1.

We can represent the normalization as: xnorm = (x - min(x))/(max(x)-min(x))

In [ ]:
```
#(f) Encoding Categorical Variables - Labels

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

Var3 = 'Address'
df[Var3 + '_encoded'] = label_encoder.fit_transform(df[Var3])
print("DataFrame with Encoded Categorical Variable:")
print(df[[Var3, Var3 + '_encoded']].head())
```

```
DataFrame with Encoded Categorical Variable:
           Address  Address_encoded
1    154 Halsey Rd             2026
2   105 Kerferd Rd              821
5       6 Smith St             7244
6  5/6 Yarralea St             6693
7     158 Queen St             2044
```

(f) This is the most useful encoding in my opinion as it can represent categorical data as numbers.

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project.

(4) Summarize the key findings from the EDA and feature engineering processes.

This dataset is about housing in Melbourne. It is made up of 34857 rows and 22 columns.

There are 7610 houses with their prices missing, 27247 have them included. The average price of a house is 1050173$

The values that have not been filled in is: Distance 1 Postcode 1 Bedroom 8217 Bathroom 8226 Car 8728 Landsize 11810 BuildingArea 21097 YearBuilt 19306 CouncilArea 3 Latitude 7976 Longtitude 7976 Propertycount 3 Price 7610

Statistical capping was used to handle outliers for the Price. Label Encoding was useful MinMax & StandardScaling was applied to the data Missing Values were identified and dealt with swiftly. I These steps help prepare the data for machine learning modeling and improve model performance.

Overall, the EDA & feature engineering processes are both insightful and essentiatal for this dataset.

The EDA and feature engineering processes provide valuable insights and essential preprocessing steps for building predictive models for this dataset.