```
      NAME             MIS
Niranjan Sonawane  732392045
Loukik Pathak      732392047
Sachin kumbhar     732392033
Prajwal Chaudhari  732392049
Siddhesh Dane      732392015
```

In [ ]:

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
#load the dataset
df = pd.read_csv('melb_data.csv')
```

In [6]:
```
df
```

Out[6]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distanc
0	Abbotsford	85 Turner St	2	h	1480000	S	Biggin	3/12/2016	2
1	Abbotsford	25 Bloomburg St	2	h	1035000	S	Biggin	04/02/2016	2
2	Abbotsford	5 Charles St	3	h	1465000	SP	Biggin	04/03/2017	2
3	Abbotsford	40 Federation La	3	h	850000	PI	Biggin	04/03/2017	2
4	Abbotsford	55a Park St	4	h	1600000	VB	Nelson	04/06/2016	2
...	...	...	...	...	...	...	...	...	
13575	Wheelers Hill	12 Strada Cr	4	h	1245000	S	Barry	26/08/2017	16
13576	Williamstown	77 Merrett Dr	3	h	1031000	SP	Williams	26/08/2017	6
13577	Williamstown	83 Power St	3	h	1170000	S	Raine	26/08/2017	6
13578	Williamstown	96 Verdon St	4	h	2500000	PI	Sweeney	26/08/2017	6
13579	Yarraville	6 Agnes St	4	h	1285000	SP	Village	26/08/2017	6

13580 rows × 21 columns

In [7]:
```
df.head()
```

Out[7]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Post
0	Abbotsford	85 Turner St	2	h	1480000	S	Biggin	3/12/2016	2.5	
1	Abbotsford	25 Bloomburg St	2	h	1035000	S	Biggin	04/02/2016	2.5	
2	Abbotsford	5 Charles St	3	h	1465000	SP	Biggin	04/03/2017	2.5	
3	Abbotsford	40 Federation La	3	h	850000	PI	Biggin	04/03/2017	2.5	
4	Abbotsford	55a Park St	4	h	1600000	VB	Nelson	04/06/2016	2.5	

5 rows × 21 columns

In [8]:
```
df.head(10)
```

Out[8]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Post
0	Abbotsford	85 Turner St	2	h	1480000	S	Biggin	3/12/2016	2.5	
1	Abbotsford	25 Bloomburg St	2	h	1035000	S	Biggin	04/02/2016	2.5	
2	Abbotsford	5 Charles St	3	h	1465000	SP	Biggin	04/03/2017	2.5	
3	Abbotsford	40 Federation La	3	h	850000	PI	Biggin	04/03/2017	2.5	
4	Abbotsford	55a Park St	4	h	1600000	VB	Nelson	04/06/2016	2.5	
5	Abbotsford	129 Charles St	2	h	941000	S	Jellis	07/05/2016	2.5	
6	Abbotsford	124 Yarra St	3	h	1876000	S	Nelson	07/05/2016	2.5	
7	Abbotsford	98 Charles St	2	h	1636000	S	Nelson	8/10/2016	2.5	
8	Abbotsford	6/241 Nicholson St	1	u	300000	S	Biggin	8/10/2016	2.5	
9	Abbotsford	10 Valiant St	2	h	1097000	S	Biggin	8/10/2016	2.5	

10 rows × 21 columns

In [9]:
```
df.tail()
```

Out[9]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
13575	Wheelers Hill	12 Strada Cr	4	h	1245000	S	Barry	26/08/2017	16.7
13576	Williamstown	77 Merrett Dr	3	h	1031000	SP	Williams	26/08/2017	6.8
13577	Williamstown	83 Power St	3	h	1170000	S	Raine	26/08/2017	6.8
13578	Williamstown	96 Verdon St	4	h	2500000	PI	Sweeney	26/08/2017	6.8
13579	Yarraville	6 Agnes St	4	h	1285000	SP	Village	26/08/2017	6.3

5 rows × 21 columns

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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               62
Landsize           0
BuildingArea    6450
YearBuilt       5375
CouncilArea     1369
Lattitude          0
Longtitude         0
Regionname         0
Propertycount      0
dtype: int64
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  object 
 4   Price          13580 non-null  int64  
 5   Method         13580 non-null  object 
 6   SellerG        13580 non-null  object 
 7   Date           13580 non-null  object 
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  int64  
 10  Bedroom2       13580 non-null  int64  
 11  Bathroom       13580 non-null  int64  
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  int64  
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object 
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object 
 20  Propertycount  13580 non-null  int64  
dtypes: float64(6), int64(7), object(8)
memory usage: 2.2+ MB
```

In [12]:
```
df.shape
```

Out[12]:
```
(13580, 21)
```

In [13]:
```
df.isnull().sum()
```

Out[13]:
```
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                62
Landsize            0
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  object 
 4   Price          13580 non-null  int64  
 5   Method         13580 non-null  object 
 6   SellerG        13580 non-null  object 
 7   Date           13580 non-null  object 
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  int64  
 10  Bedroom2       13580 non-null  int64  
 11  Bathroom       13580 non-null  int64  
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  int64  
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object 
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object 
 20  Propertycount  13580 non-null  int64  
dtypes: float64(6), int64(7), object(8)
memory usage: 2.2+ MB
```

In [15]:
```
unique_values = df['Suburb'].unique()
print(len(unique_values))
```

```
314
```

In [16]:
```
unique_values = df['Regionname'].unique()
print(len(unique_values))
```

```
8
```

In [17]:
```
unique_values = df['CouncilArea'].unique()
print(len(unique_values))
```

```
34
```

In [18]:
```
unique_values = df['Date'].unique()
print(len(unique_values))
```

```
58
```

In [19]:
```
unique_values = df['SellerG'].unique()
print(len(unique_values))
```

```
268
```

In [20]:
```
unique_values = df['Method'].unique()
print(len(unique_values))
```

```
5
```

In [21]:
```
unique_values = df['Type'].unique()
print(len(unique_values))
```

```
3
```

In [22]:
```
unique_values = df['Address'].unique()
print(len(unique_values))
```

```
13378
```

In [23]:
```
df.shape
```

Out[23]:
```
(13580, 21)
```

In [24]:
```
df.describe()
```

Out[24]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000

In [25]:
```
df.corr()
```

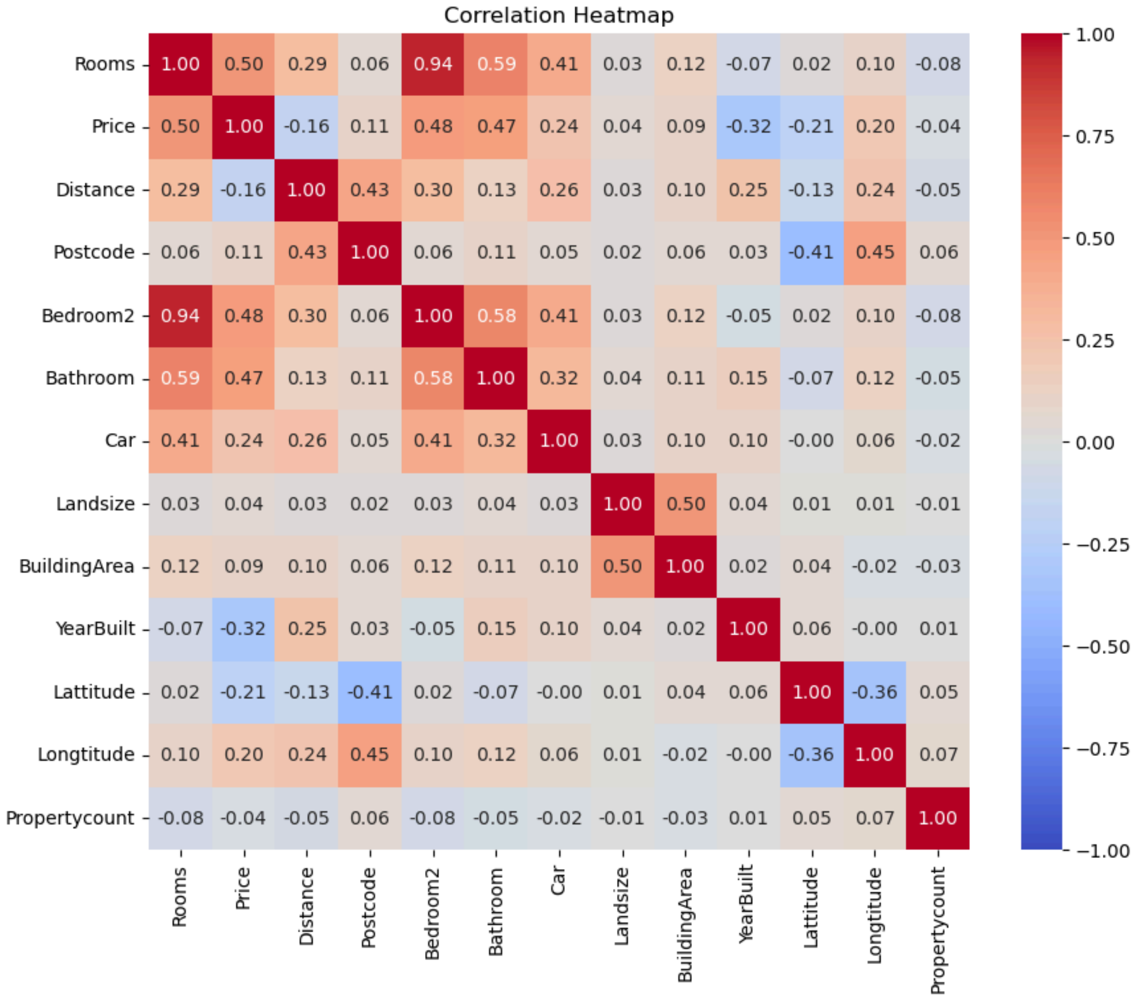
```
C:\Users\niran\AppData\Local\Temp\ipykernel_27380\1134722465.py:1: FutureW
arning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  df.corr()
```

Out[25]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	La
Rooms	1.000000	0.496634	0.294203	0.055303	0.944190	0.592934	0.408483	0
Price	0.496634	1.000000	-0.162522	0.107867	0.475951	0.467038	0.238979	0
Distance	0.294203	-0.162522	1.000000	0.431514	0.295927	0.127155	0.262994	0
Postcode	0.055303	0.107867	0.431514	1.000000	0.060584	0.113664	0.050289	0
Bedroom2	0.944190	0.475951	0.295927	0.060584	1.000000	0.584685	0.405325	0
Bathroom	0.592934	0.467038	0.127155	0.113664	0.584685	1.000000	0.322246	0
Car	0.408483	0.238979	0.262994	0.050289	0.405325	0.322246	1.000000	0
Landsize	0.025678	0.037507	0.025004	0.024558	0.025646	0.037130	0.026770	1
BuildingArea	0.124127	0.090981	0.099481	0.055475	0.122319	0.111933	0.096101	0
YearBuilt	-0.065413	-0.323617	0.246379	0.032863	-0.053319	0.152702	0.104515	0
Lattitude	0.015948	-0.212934	-0.130723	-0.406104	0.015925	-0.070594	-0.001963	0
Longtitude	0.100771	0.203656	0.239425	0.445357	0.102238	0.118971	0.063395	0
Propertycount	-0.081530	-0.042153	-0.054910	0.062304	-0.081350	-0.052201	-0.024295	-0

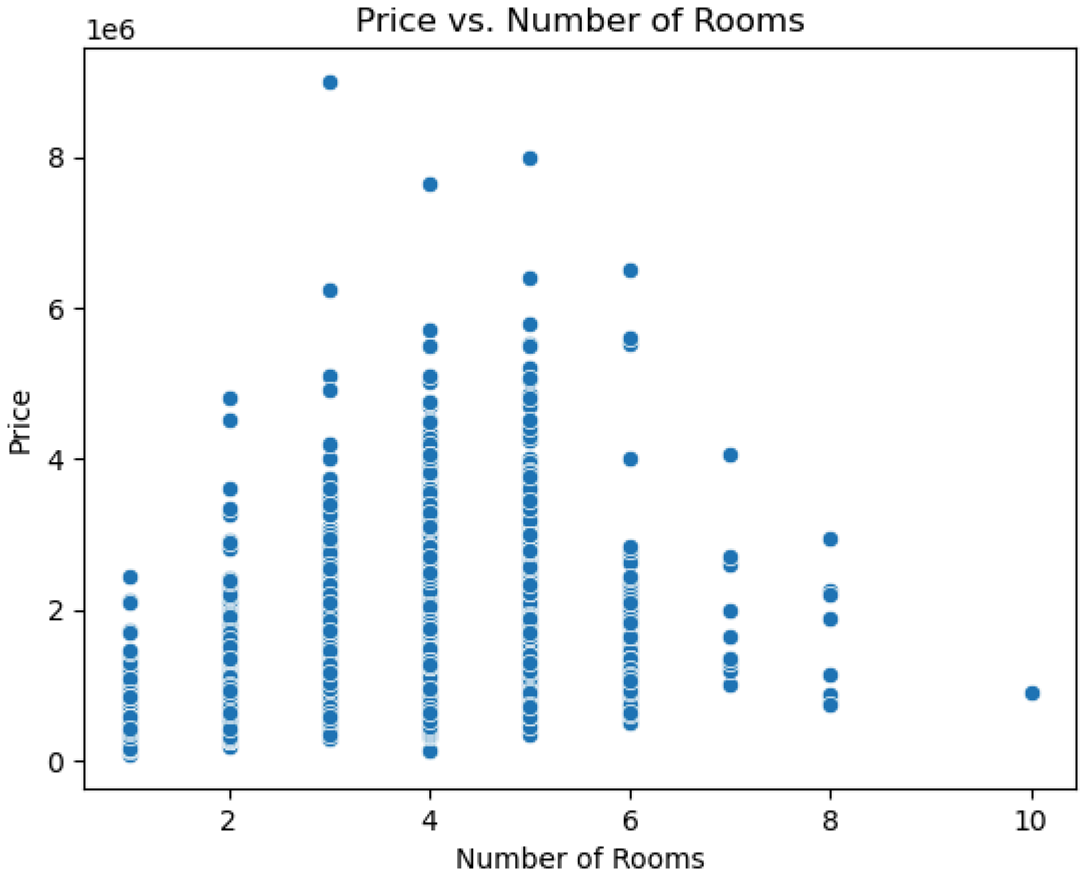
In [26]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vma
plt.title('Correlation Heatmap')
plt.show()
```

```
C:\Users\niran\AppData\Local\Temp\ipykernel_27380\4123246429.py:2: FutureW
arning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', vmin=-1,
vmax=1)
```

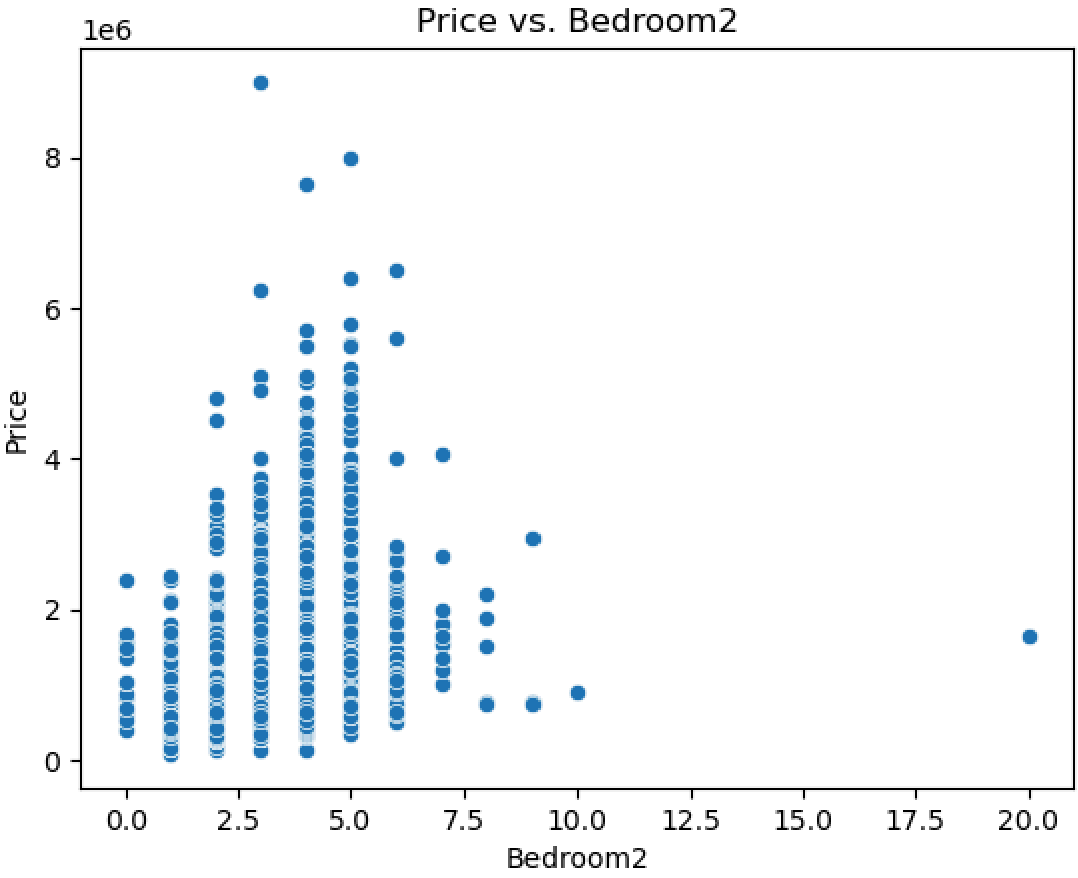


	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
Rooms	1.00	0.50	0.29	0.06	0.94	0.59	0.41	0.03	0.12	-0.07	0.02	0.10	-0.08
Price	0.50	1.00	-0.16	0.11	0.48	0.47	0.24	0.04	0.09	-0.32	-0.21	0.20	-0.04
Distance	0.29	-0.16	1.00	0.43	0.30	0.13	0.26	0.03	0.10	0.25	-0.13	0.24	-0.05
Postcode	0.06	0.11	0.43	1.00	0.06	0.11	0.05	0.02	0.06	0.03	-0.41	0.45	0.06
Bedroom2	0.94	0.48	0.30	0.06	1.00	0.58	0.41	0.03	0.12	-0.05	0.02	0.10	-0.08
Bathroom	0.59	0.47	0.13	0.11	0.58	1.00	0.32	0.04	0.11	0.15	-0.07	0.12	-0.05
Car	0.41	0.24	0.26	0.05	0.41	0.32	1.00	0.03	0.10	0.10	-0.00	0.06	-0.02
Landsize	0.03	0.04	0.03	0.02	0.03	0.04	0.03	1.00	0.50	0.04	0.01	0.01	-0.01
BuildingArea	0.12	0.09	0.10	0.06	0.12	0.11	0.10	0.50	1.00	0.02	0.04	-0.02	-0.03
YearBuilt	-0.07	-0.32	0.25	0.03	-0.05	0.15	0.10	0.04	0.02	1.00	0.06	-0.00	0.01
Lattitude	0.02	-0.21	-0.13	-0.41	0.02	-0.07	-0.00	0.01	0.04	0.06	1.00	-0.36	0.05
Longtitude	0.10	0.20	0.24	0.45	0.10	0.12	0.06	0.01	-0.02	-0.00	-0.36	1.00	0.07
Propertycount	-0.08	-0.04	-0.05	0.06	-0.08	-0.05	-0.02	-0.01	-0.03	0.01	0.05	0.07	1.00

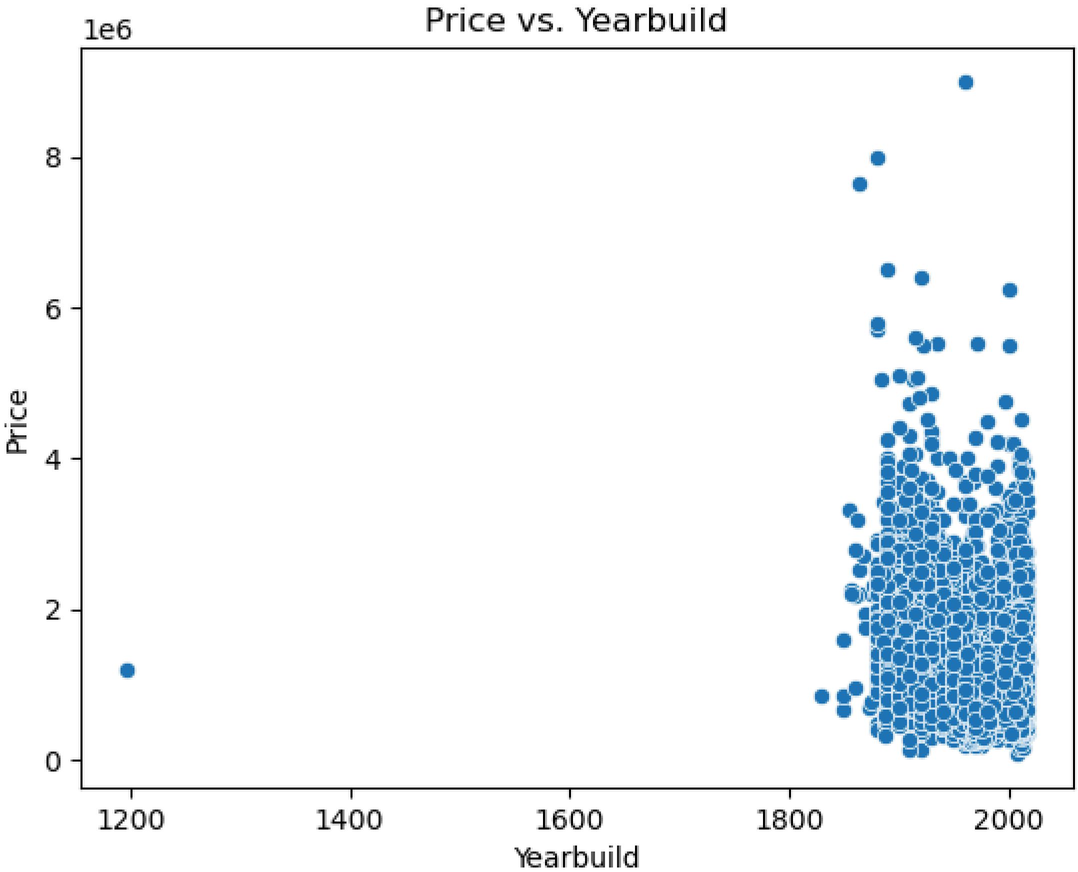
In [27]:
```
# Create scatter plot
sns.scatterplot(x=df.Rooms, y=df.Price)
plt.xlabel('Number of Rooms')
plt.ylabel('Price')
plt.title('Price vs. Number of Rooms')
plt.show()
```



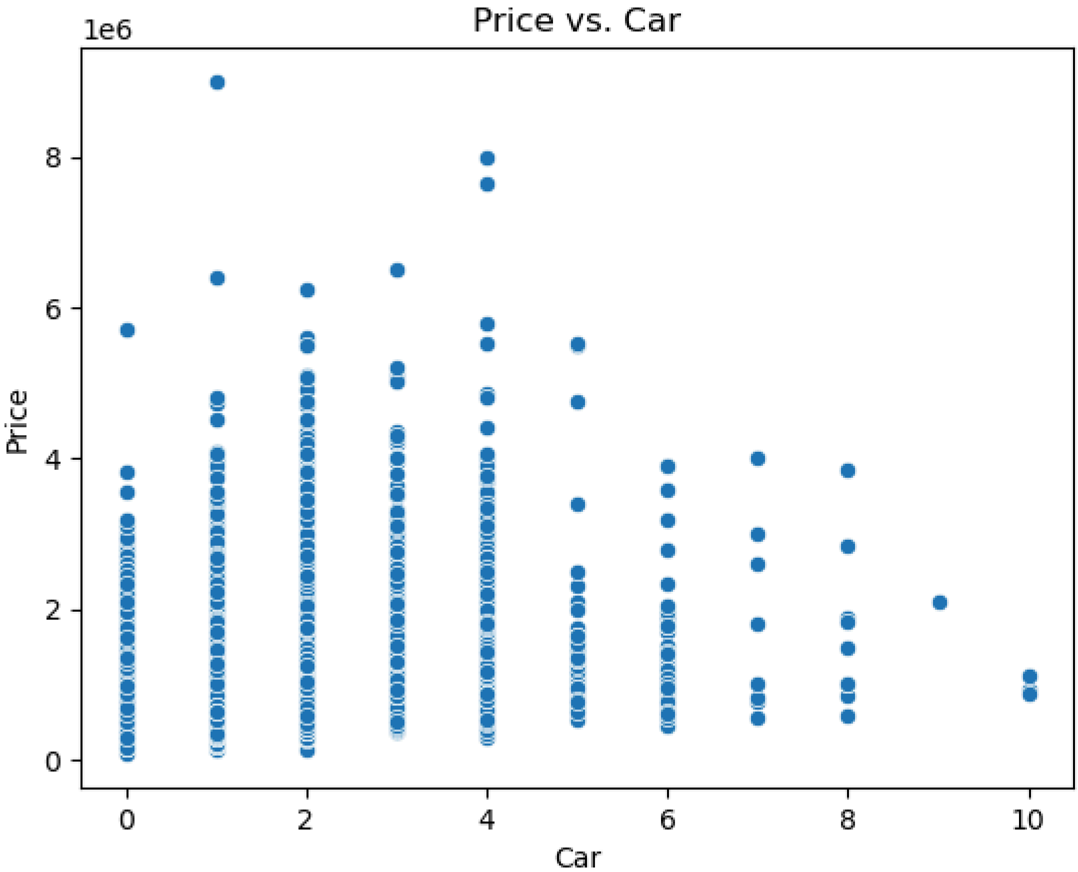
In [28]:
```
# Create scatter plot
sns.scatterplot(x=df.Bedroom2, y=df.Price)
plt.xlabel('Bedroom2')
plt.ylabel('Price')
plt.title('Price vs. Bedroom2')
plt.show()
```



In [29]:
```
# Create scatter plot
sns.scatterplot(x=df.YearBuilt , y=df.Price)
plt.xlabel('Yearbuild')
plt.ylabel('Price')
plt.title('Price vs. Yearbuild')
plt.show()
```

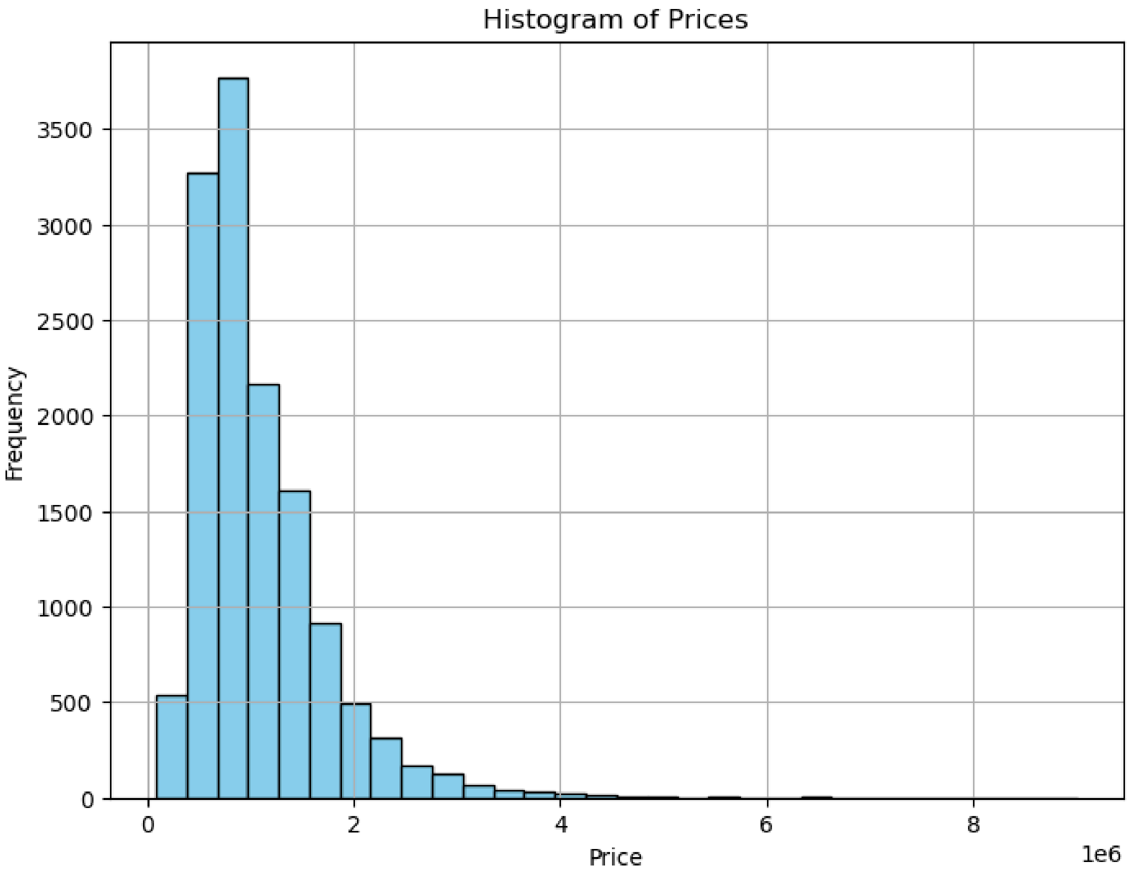


In [30]:
```
# Create scatter plot
sns.scatterplot(x=df.Car, y=df.Price)
plt.xlabel('Car')
plt.ylabel('Price')
plt.title('Price vs. Car')
plt.show()
```

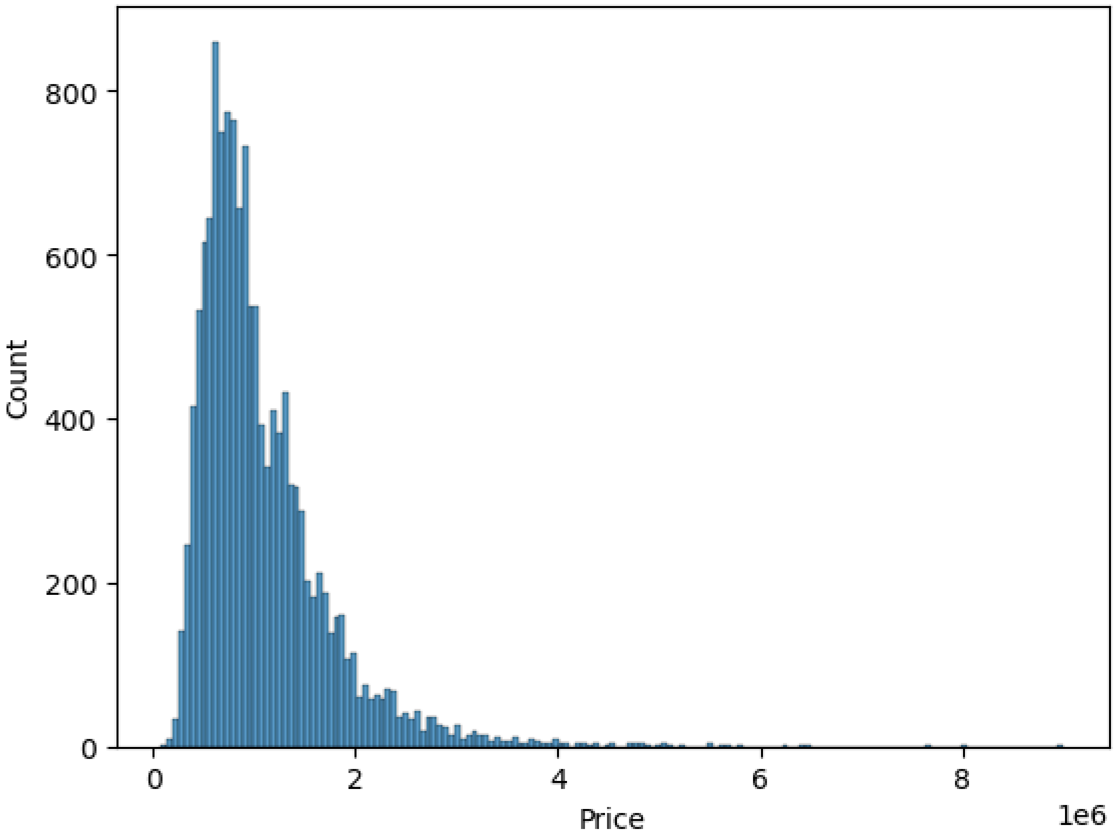


In [31]:
```
#handeling missing data
#BuildingArea have 0.09 Correlation With Price also have 6450 NaN Values so
#drop this feature_col
```

In [32]:
```
# Create a histogram for Price column
plt.figure(figsize=(8, 6))
plt.hist(df['Price'], bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



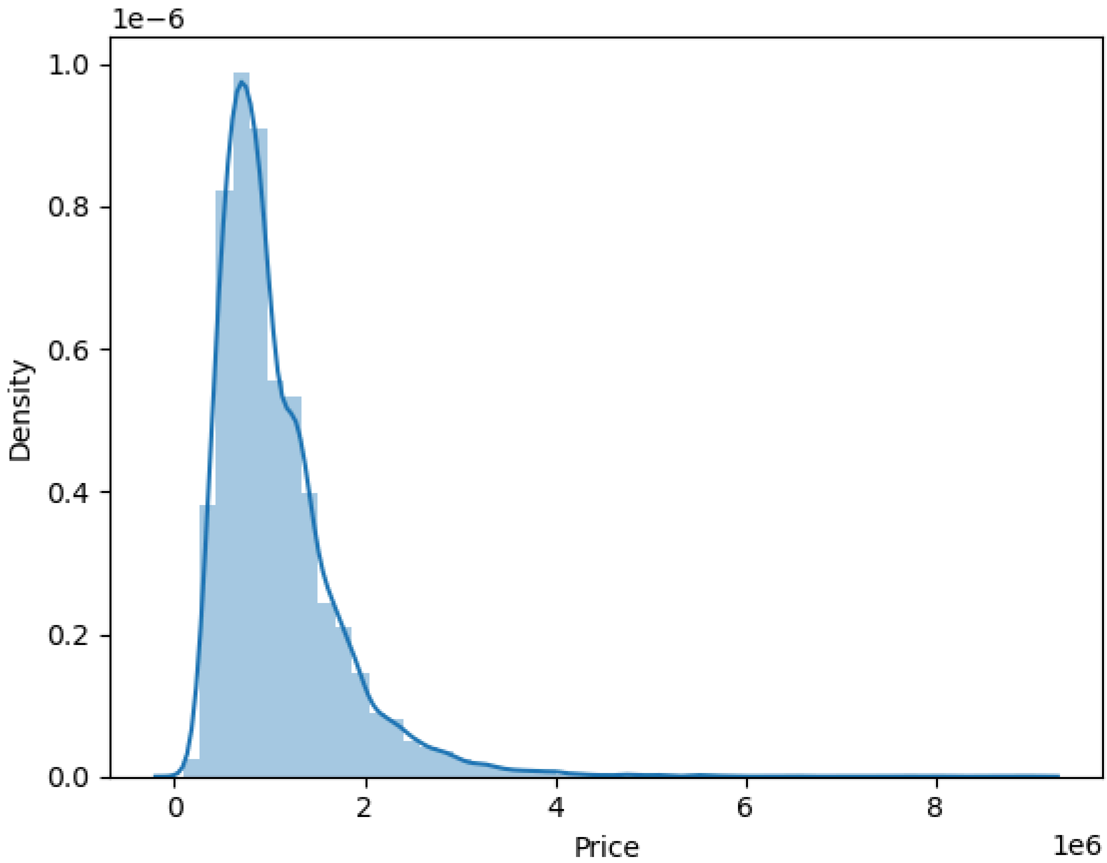
In [33]:
```
sns.histplot(df['Price'])
plt.show()
```



In [34]:
```
sns.distplot(df['Price'])
```

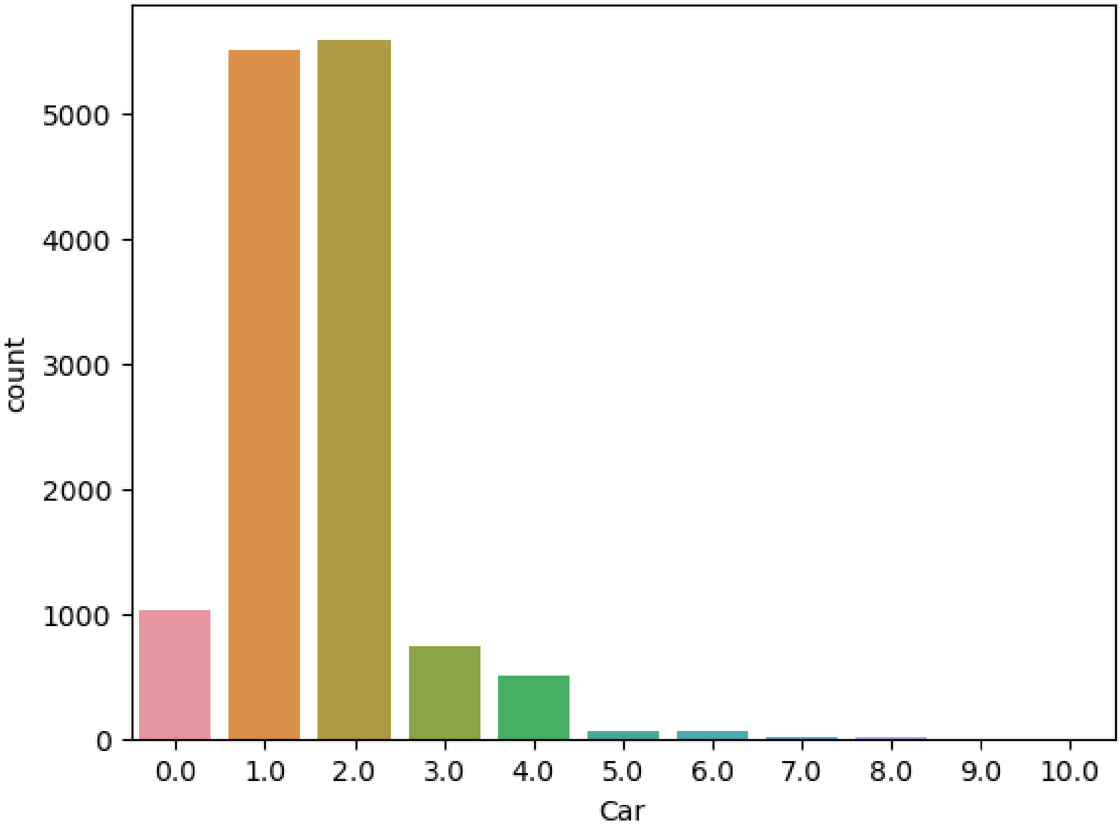
```
C:\Users\niran\AppData\Local\Temp\ipykernel_27380\834922981.py:1: UserWarn
ing:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['Price'])
```

Out[34]: <Axes: xlabel='Price', ylabel='Density'>



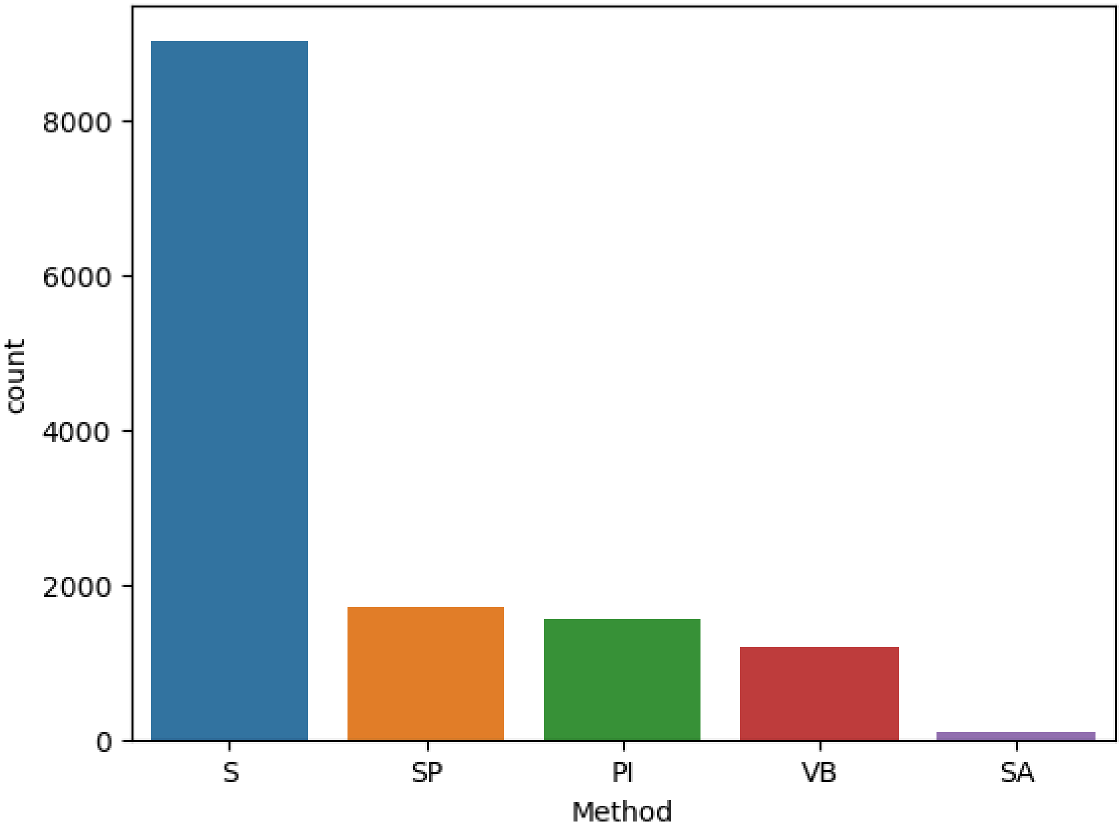
In [35]:
```
sns.countplot(x='Car',data=df)
```

Out[35]: <Axes: xlabel='Car', ylabel='count'>



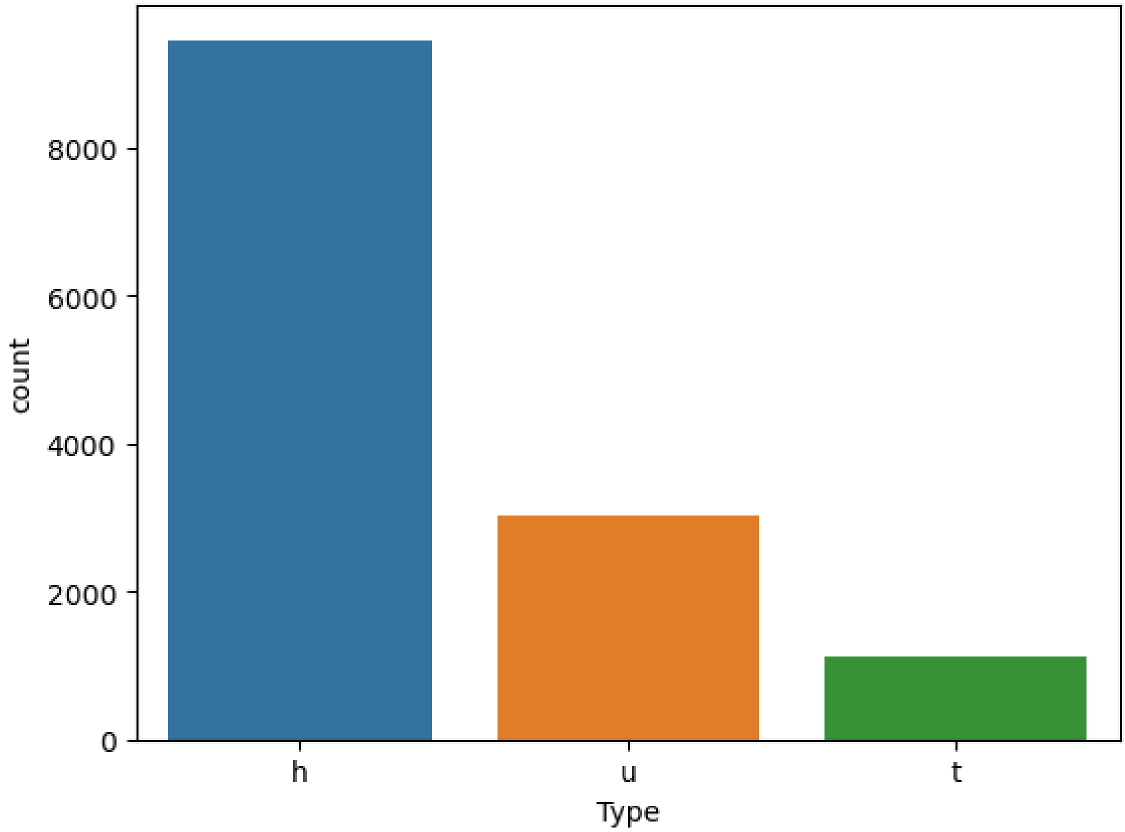
In [36]:
```
sns.countplot(x='Method',data=df)
```

Out[36]: <Axes: xlabel='Method', ylabel='count'>



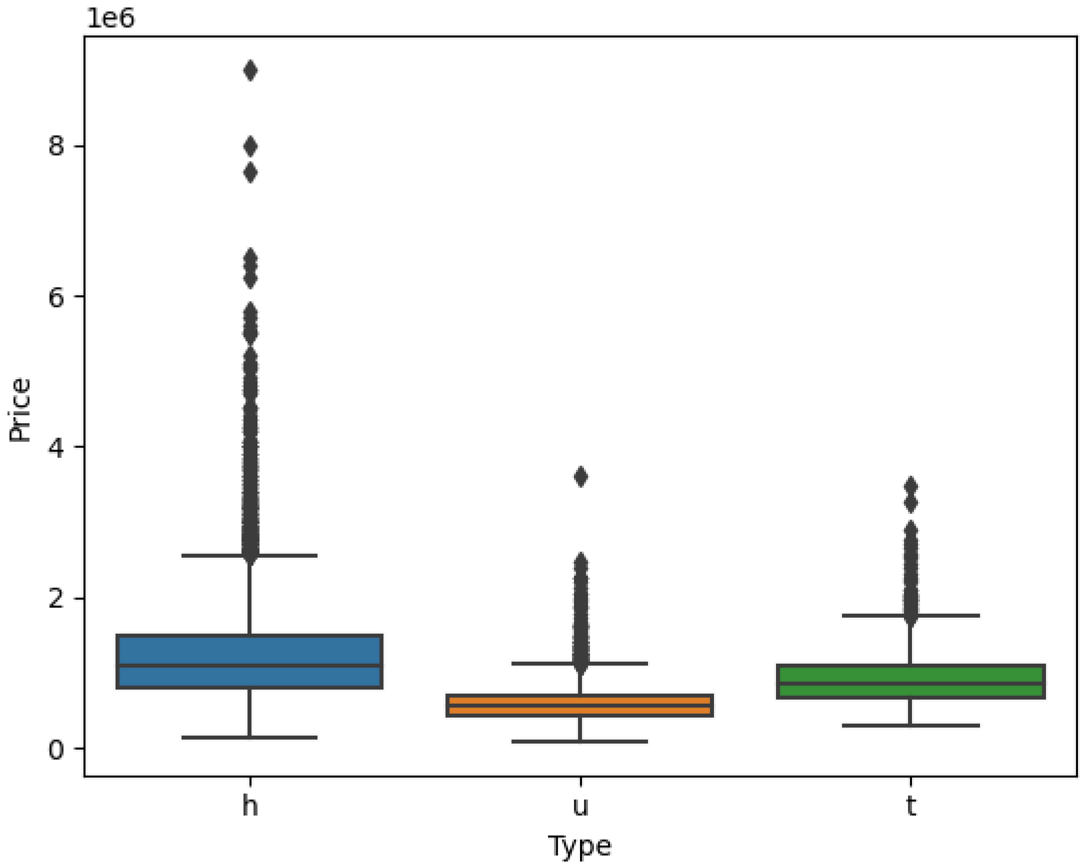
In [37]:
```
sns.countplot(x='Type',data=df)
```

Out[37]: <Axes: xlabel='Type', ylabel='count'>



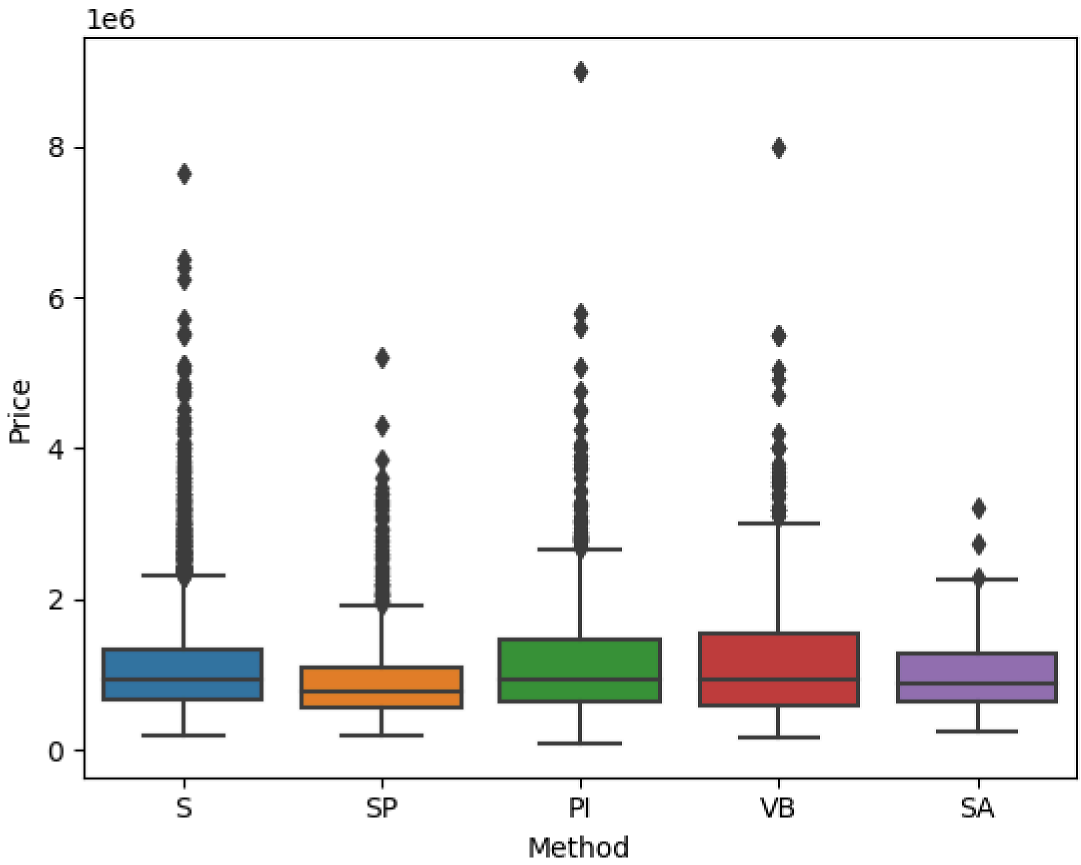
In [38]:
```
sns.boxplot(y='Price',x='Type',data=df)
```

Out[38]: <Axes: xlabel='Type', ylabel='Price'>



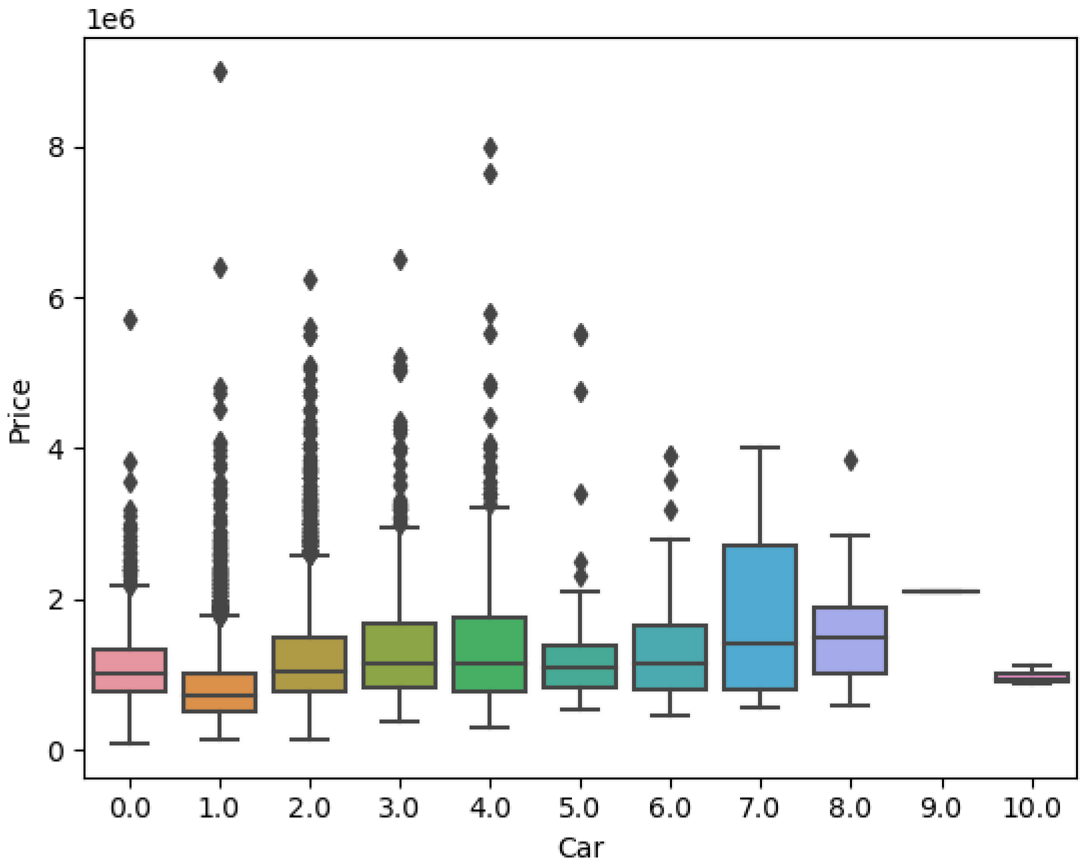
In [39]:
```
sns.boxplot(y='Price',x='Method',data=df)
```

Out[39]: <Axes: xlabel='Method', ylabel='Price'>

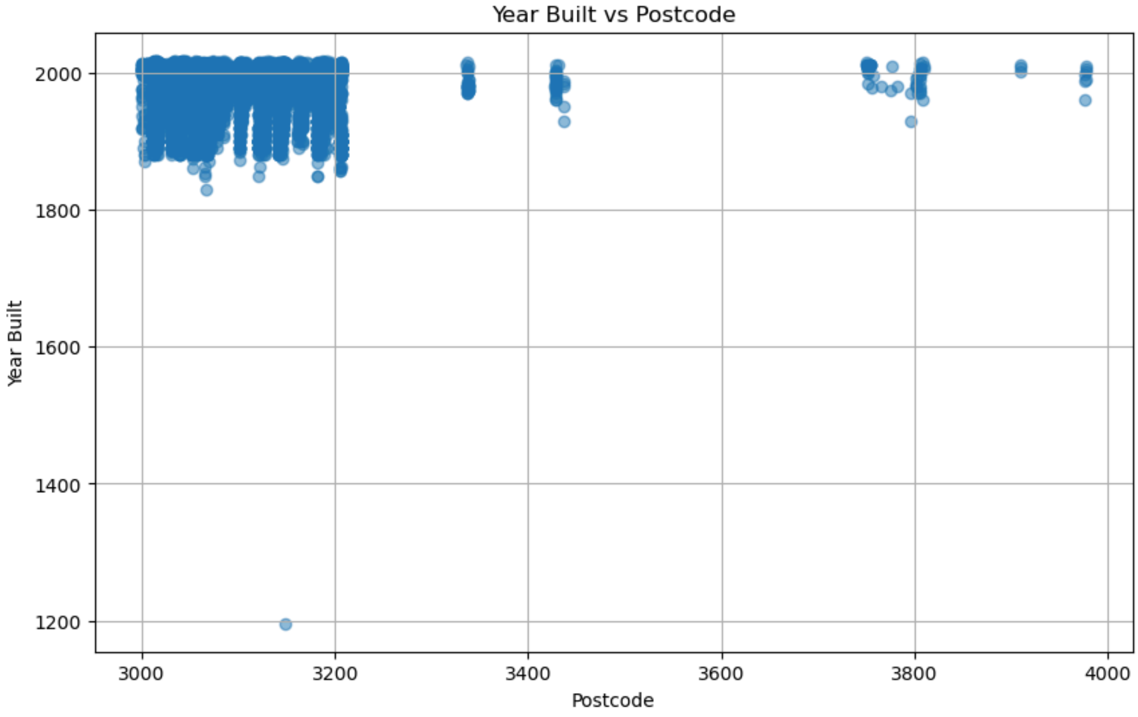


In [40]:
```
sns.boxplot(x='Car',y='Price',data=df)
```

Out[40]:
```
<Axes: xlabel='Car', ylabel='Price'>
```



In [41]:
```
# Scatter plot of YearBuilt vs Postcode
plt.figure(figsize=(10, 6))
plt.scatter(df['Postcode'], df['YearBuilt'], alpha=0.5)
plt.title('Year Built vs Postcode')
plt.xlabel('Postcode')
plt.ylabel('Year Built')
plt.grid(True)
plt.show()
```



In [42]:
```
# Calculate mean and median of 'YearBuilt' column
mean_year_built = df['YearBuilt'].mean()
median_year_built = df['YearBuilt'].median()

print("Mean YearBuilt:", mean_year_built)
print("Median YearBuilt:", median_year_built)
```

```
Mean YearBuilt: 1964.6842169408897
Median YearBuilt: 1970.0
```

In [43]:
```
unique_values = df['CouncilArea'].unique()
print(len(unique_values))
```

```
34
```

In [ ]:

In [44]:
```
#NaN values in 'Car' column
#Replace NaN values with the mode

mode_value = df['Car'].mode().iloc[0]
df['Car'].fillna(mode_value, inplace=True)

# Check the count of NaN values
print("Count of NaN values in 'Car' column:", df['Car'].isna().sum())
```

```
Count of NaN values in 'Car' column: 0
```

In [45]:
```
# Calculate mode for 'CouncilArea' column
CouncilArea_mean = df['CouncilArea'].mode()
CouncilArea_mean
```

Out[45]:
```
0    Moreland
Name: CouncilArea, dtype: object
```

In [46]:
```
# Fill NaN values in 'CouncilArea' column with the mode
mode_value = df['CouncilArea'].mode().iloc[0]
df['CouncilArea'].fillna(mode_value, inplace=True)

# Check the count of NaN values after filling
df['CouncilArea'].isna().sum()
```

Out[46]:
```
0
```

In [47]:
```
df.isnull().sum()
```

Out[47]:
```
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     6450
YearBuilt        5375
CouncilArea         0
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64
```

In [48]:
```
df.shape
```

Out[48]:
```
(13580, 21)
```

In [49]:
```
#split data as x and y
# Assuming 'price' is the column you want to exclude
#feature_col = df.drop(columns=['Price','Lattitude', 'Longtitude'])


feature_col= ['Rooms','Distance','Postcode','Bedroom2','Bathroom','Car','Reg


#High corrleation
#feature_col= ['Rooms','Bedroom2','Bathroom']

#High corrleation  with calegorical data
#feature_col= ['Rooms','Bedroom2','Bathroom','Car','Method','Type','Regionna
```

In [50]:
```
X =df[feature_col]
X
```

Out[50]:

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Regionname	CouncilArea	
0	2	2.5	3067	2	1	1.0	Northern Metropolitan	Yarra	3
1	2	2.5	3067	2	1	0.0	Northern Metropolitan	Yarra	04
2	3	2.5	3067	3	2	0.0	Northern Metropolitan	Yarra	04
3	3	2.5	3067	3	2	1.0	Northern Metropolitan	Yarra	04
4	4	2.5	3067	3	1	2.0	Northern Metropolitan	Yarra	04
...	...	...	...	...	...	...	...	...	
13575	4	16.7	3150	4	2	2.0	South-Eastern Metropolitan	Moreland	26
13576	3	6.8	3016	3	2	2.0	Western Metropolitan	Moreland	26
13577	3	6.8	3016	3	2	4.0	Western Metropolitan	Moreland	26
13578	4	6.8	3016	4	1	5.0	Western Metropolitan	Moreland	26
13579	4	6.3	3013	4	1	1.0	Western Metropolitan	Moreland	26

13580 rows × 14 columns

In [51]:
```
# Apply one-hot encoding using pd.get_dummies

df_new = pd.get_dummies(X,drop_first=True)
```

In [52]:
```
df_new
```

Out[52]:

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Lattitude	Longtitu
0	2	2.5	3067	2	1	1.0	202	-37.79960	144.998
1	2	2.5	3067	2	1	0.0	156	-37.80790	144.993
2	3	2.5	3067	3	2	0.0	134	-37.80930	144.994
3	3	2.5	3067	3	2	1.0	94	-37.79690	144.996
4	4	2.5	3067	3	1	2.0	120	-37.80720	144.994
...	...	...	...	...	...	...	...	...	
13575	4	16.7	3150	4	2	2.0	652	-37.90562	145.167
13576	3	6.8	3016	3	2	2.0	333	-37.85927	144.879
13577	3	6.8	3016	3	2	4.0	436	-37.85274	144.887
13578	4	6.8	3016	4	1	5.0	866	-37.85908	144.892
13579	4	6.3	3013	4	1	1.0	362	-37.81188	144.884

13580 rows × 111 columns

In [53]:
```
X
```

Out[53]:

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Regionname	CouncilArea	
0	2	2.5	3067	2	1	1.0	Northern Metropolitan	Yarra	;
1	2	2.5	3067	2	1	0.0	Northern Metropolitan	Yarra	04
2	3	2.5	3067	3	2	0.0	Northern Metropolitan	Yarra	04
3	3	2.5	3067	3	2	1.0	Northern Metropolitan	Yarra	04
4	4	2.5	3067	3	1	2.0	Northern Metropolitan	Yarra	04
...	...	...	...	...	...	...	...	...	
13575	4	16.7	3150	4	2	2.0	South-Eastern Metropolitan	Moreland	26
13576	3	6.8	3016	3	2	2.0	Western Metropolitan	Moreland	26
13577	3	6.8	3016	3	2	4.0	Western Metropolitan	Moreland	26
13578	4	6.8	3016	4	1	5.0	Western Metropolitan	Moreland	26
13579	4	6.3	3013	4	1	1.0	Western Metropolitan	Moreland	26

13580 rows × 14 columns

In [54]:
```
Y = df['Price']
```

In [55]:
```
Y
```

Out[55]:
```
0        1480000
1        1035000
2        1465000
3         850000
4        1600000
          ...   
13575    1245000
13576    1031000
13577    1170000
13578    2500000
13579    1285000
Name: Price, Length: 13580, dtype: int64
```

In [56]:
```
X=df_new
X
```

Out[56]:

	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	Lattitude	Longtitu
0	2	2.5	3067	2	1	1.0	202	-37.79960	144.998
1	2	2.5	3067	2	1	0.0	156	-37.80790	144.993
2	3	2.5	3067	3	2	0.0	134	-37.80930	144.994
3	3	2.5	3067	3	2	1.0	94	-37.79690	144.996
4	4	2.5	3067	3	1	2.0	120	-37.80720	144.994
...	...	...	...	...	...	...	...	...	
13575	4	16.7	3150	4	2	2.0	652	-37.90562	145.167
13576	3	6.8	3016	3	2	2.0	333	-37.85927	144.879
13577	3	6.8	3016	3	2	4.0	436	-37.85274	144.887
13578	4	6.8	3016	4	1	5.0	866	-37.85908	144.892
13579	4	6.3	3013	4	1	1.0	362	-37.81188	144.884

13580 rows × 111 columns

In [57]:
```
# for scaling the data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
X
```

Out[57]:
```
array([[-0.98146337, -1.30148498, -0.42241517, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [-0.98146337, -1.30148498, -0.42241517, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 0.06487613, -1.30148498, -0.42241517, ..., -0.31119437,
        -0.29893656, -0.53443392],
       ...,
       [ 0.06487613, -0.56876052, -0.984872  , ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 1.11121563, -0.56876052, -0.984872  , ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 1.11121563, -0.65396104, -1.01795769, ..., -0.31119437,
        -0.29893656, -0.53443392]])
```

In [58]:
```
# split data for traning and testing

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.15,random_s
```

In [59]:
```
X_train
```

Out[59]:
```
array([[ 1.11121563,  0.89668841,  0.28341301, ...,  3.21342576,
        -0.29893656, -0.53443392],
       [ 1.11121563, -1.45484592, -0.4444723 , ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 0.06487613,  0.62404675,  0.01872745, ..., -0.31119437,
        -0.29893656, -0.53443392],
       ...,
       [ 0.06487613, -0.07459751, -0.02538682, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 0.06487613,  0.64108685,  0.02975601, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [-0.98146337, -1.42076571, -0.57681508, ..., -0.31119437,
        -0.29893656, -0.53443392]])
```

In [60]:
```
X_test
```

Out[60]:
```
array([[-0.98146337, -0.39835948, -0.52167225, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 0.06487613,  0.64108685, -0.94075774, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [ 0.06487613, -0.07459751, -0.70915786, ..., -0.31119437,
        -0.29893656, -0.53443392],
       ...,
       [-0.98146337, -0.0405173 , -0.67607217, ..., -0.31119437,
        -0.29893656,  1.87113872],
       [ 1.11121563, -0.80732197, -0.04744395, ..., -0.31119437,
        -0.29893656, -0.53443392],
       [-0.98146337, -1.28444488,  0.17312736, ..., -0.31119437,
        -0.29893656, -0.53443392]])
```

In [61]:
```
Y_train
```

Out[61]:
```
12724     950000
2540     1620000
6202     1200000
8797      940000
10076     856000
           ...
635      1230000
1345     1395000
581      1500000
2169     1430000
6825      720000
Name: Price, Length: 11543, dtype: int64
```

In [62]:
```
Y_test
```

Out[62]:
```
1993      860000
8559      800000
5900      910000
9653      660000
804       830000
           ...
5444      560000
4190      776000
8213      496000
10366    2425000
5328      777500
Name: Price, Length: 2037, dtype: int64
```

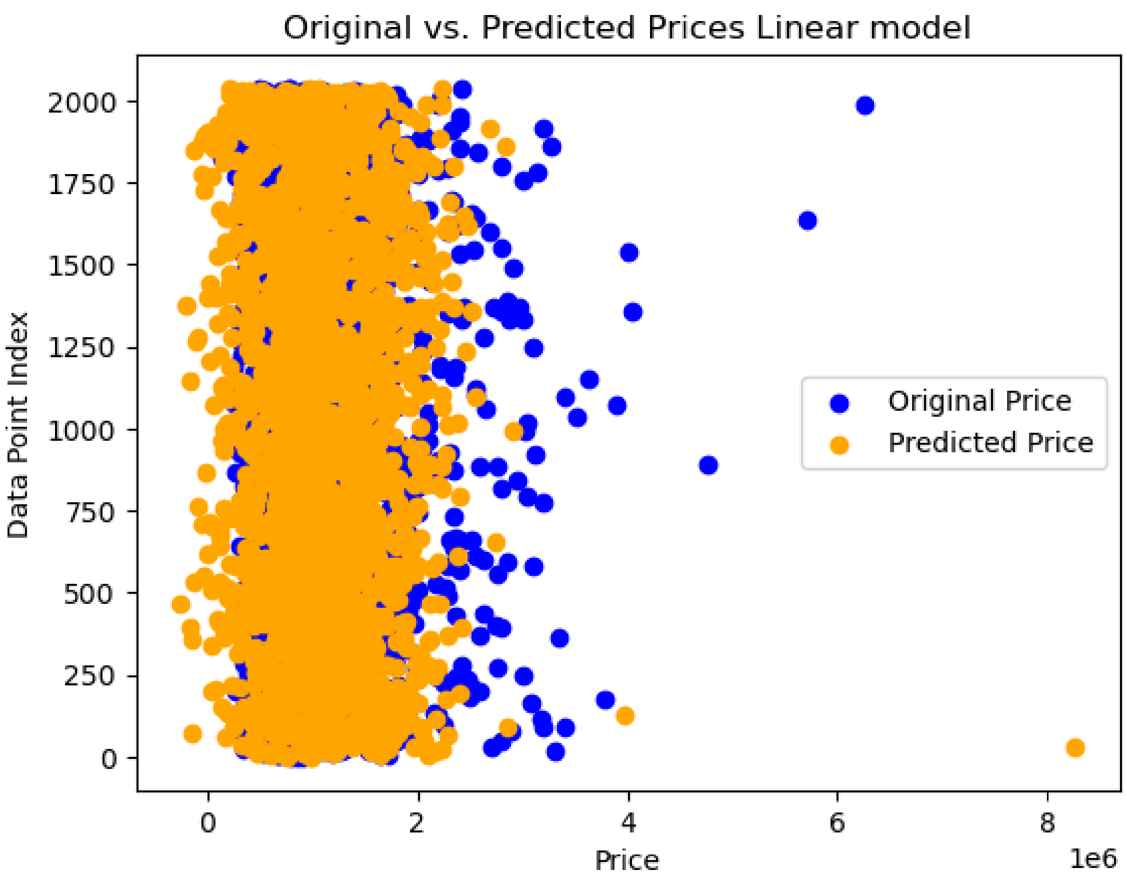
In [ ]:

In [ ]:
```
#Initialize LinearRegression model
```

In [63]:
```
#Initialize LinearRegression model

from sklearn.linear_model import LinearRegression
lr = LinearRegression()


#Fit SVR model to the training data
lr.fit(X_train,Y_train)

predictions = lr.predict(X_test)

# Print the predictions
print(predictions)



# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = predictions.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df_linear = pd.DataFrame({'Original': Y_test_rounded, 'Predicted'

# Display the DataFrame
print(comparison_df_linear)
```

```
[ 763776.75221027  979585.88387217  997121.29519901 ...  221555.14905853
 2239031.73218314 1065765.70058251]
       Original  Predicted
1993     860000     763777
8559     800000     979586
5900     910000     997121
9653     660000     603353
804      830000     746368
...         ...        ...
5444     560000     324125
4190     776000     971505
8213     496000     221555
10366   2425000    2239032
5328     777500    1065766

[2037 rows x 2 columns]
```

In [64]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Assuming comparison_df holds original and predicted prices (rounded intege

try:
  # Extract data for plotting
  original_prices = comparison_df_linear['Original']
  predicted_prices = comparison_df_linear['Predicted']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted Prices Linear model')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [65]:
```
lr.score(X_test,Y_test)
```

Out[65]: 0.5448366700938244

In [ ]:

```
#SVR model
```

In [66]:
```
#Initialize SVR model

from sklearn.svm import SVR
svr = SVR(kernel='linear')  # You can choose different kernels like 'linear

#Fit SVR model to the training data
svr.fit(X_train,Y_train)
```

Out[66]:
```
SVR
SVR(kernel='linear')
```

In [67]:
```
Y_predict = svr.predict(X_test)
```

In [68]:
```
Y_predict
```

Out[68]:
```
array([891671.33571579, 904608.57472994, 902649.62619245, ...,
       880566.04185926, 939404.44841916, 899151.79728111])
```

In [69]:
```
Y_test
```

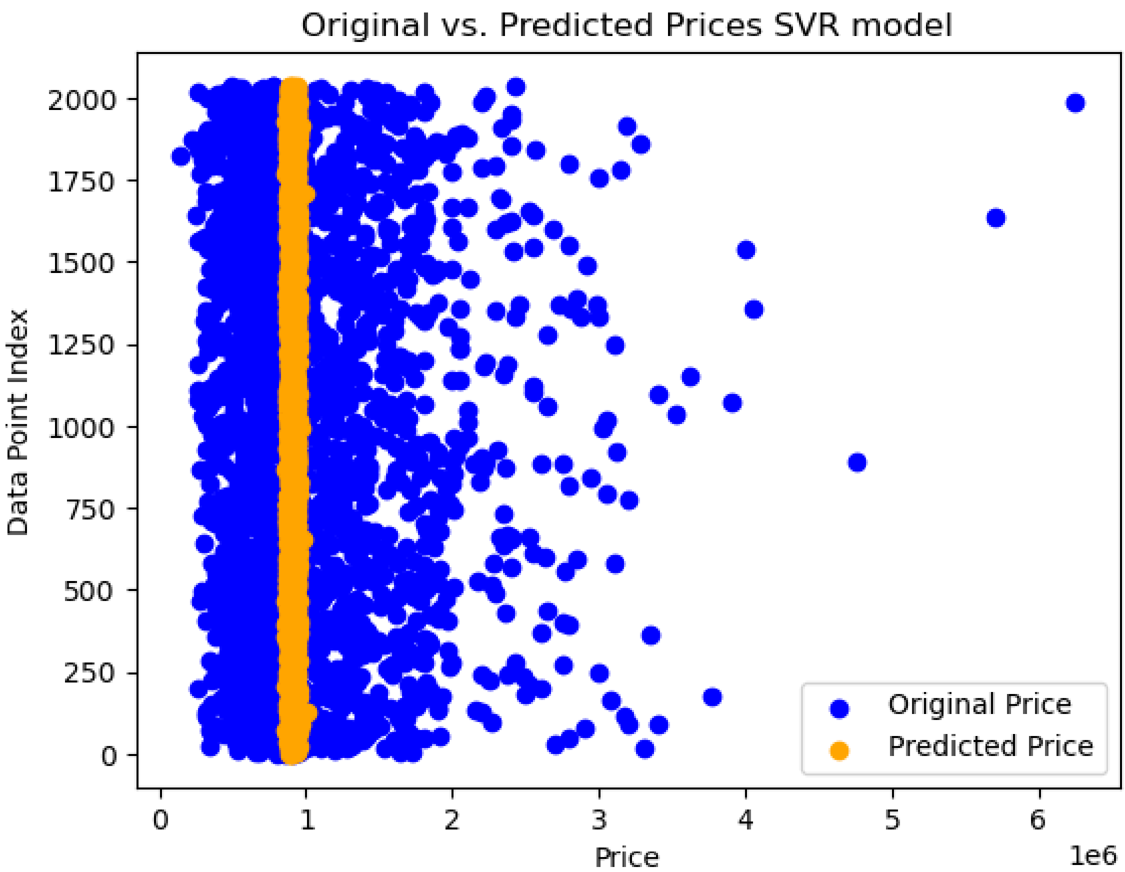
Out[69]:
```
1993      860000
8559      800000
5900      910000
9653      660000
804       830000
           ...
5444      560000
4190      776000
8213      496000
10366    2425000
5328      777500
Name: Price, Length: 2037, dtype: int64
```

```
Y_predict=Y_predict.reshape(-1, 1)
Y_predict
```

In [70]:
```
# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = Y_predict.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df = pd.DataFrame({'Original Price': Y_test_rounded, 'Predicted P

# Display the DataFrame
print(comparison_df)
```

```
       Original Price  Predicted Price
1993           860000           891671
8559           800000           904609
5900           910000           902650
9653           660000           898852
804            830000           908915
...               ...              ...
5444           560000           895039
4190           776000           910242
8213           496000           880566
10366         2425000           939404
5328           777500           899152

[2037 rows x 2 columns]
```

In [71]:
```
import matplotlib.pyplot as plt
import pandas as pd


try:
  # Extract data for plotting
  original_prices = comparison_df['Original Price']
  predicted_prices = comparison_df['Predicted Price']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted Prices SVR model')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [72]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(Y_test, Y_predict)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(Y_test, Y_predict)

# Calculate R-squared (R2)
r2 = r2_score(Y_test, Y_predict)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
```

```
Mean Squared Error (MSE): 371097765633.9623
Mean Absolute Error (MAE): 415823.97318179667
R-squared (R2): -0.014482159932799332
```

In [73]:
```
svr.score(X_test, Y_test)
```

Out[73]:
```
-0.014482159932799332
```

In [ ]:

In [ ]:

#DecisionTreeRegressor

In [74]:
```
from sklearn.tree import DecisionTreeRegressor


best_score = float('inf')  # Initialize with a high value
best_random_state = None

for random_state in range(100):  # Try different random states
    model = DecisionTreeRegressor(random_state=random_state)
    model.fit(X_train, Y_train)
    Y_test_pred = model.predict(X_test)
    score = mean_squared_error(Y_test, Y_test_pred)  # Evaluate using mean s
    if score < best_score:
        best_score = score
        best_random_state = random_state

print("Best Random State:", best_random_state)
print("Best Mean Squared Error:", best_score)
```

```
Best Random State: 49
Best Mean Squared Error: 140854137364.8002
```

In [75]:
```
#DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error


modelDecisionTreeRegressor = DecisionTreeRegressor(random_state=49)
modelDecisionTreeRegressor.fit(X_train, Y_train)

Y_test_pred = modelDecisionTreeRegressor.predict(X_test)
```

In [77]:
```
Y_test_pred
```

Out[77]:
```
array([1008000.,  625000.,  920000., ...,  485000., 2560000.,  912000.])
```

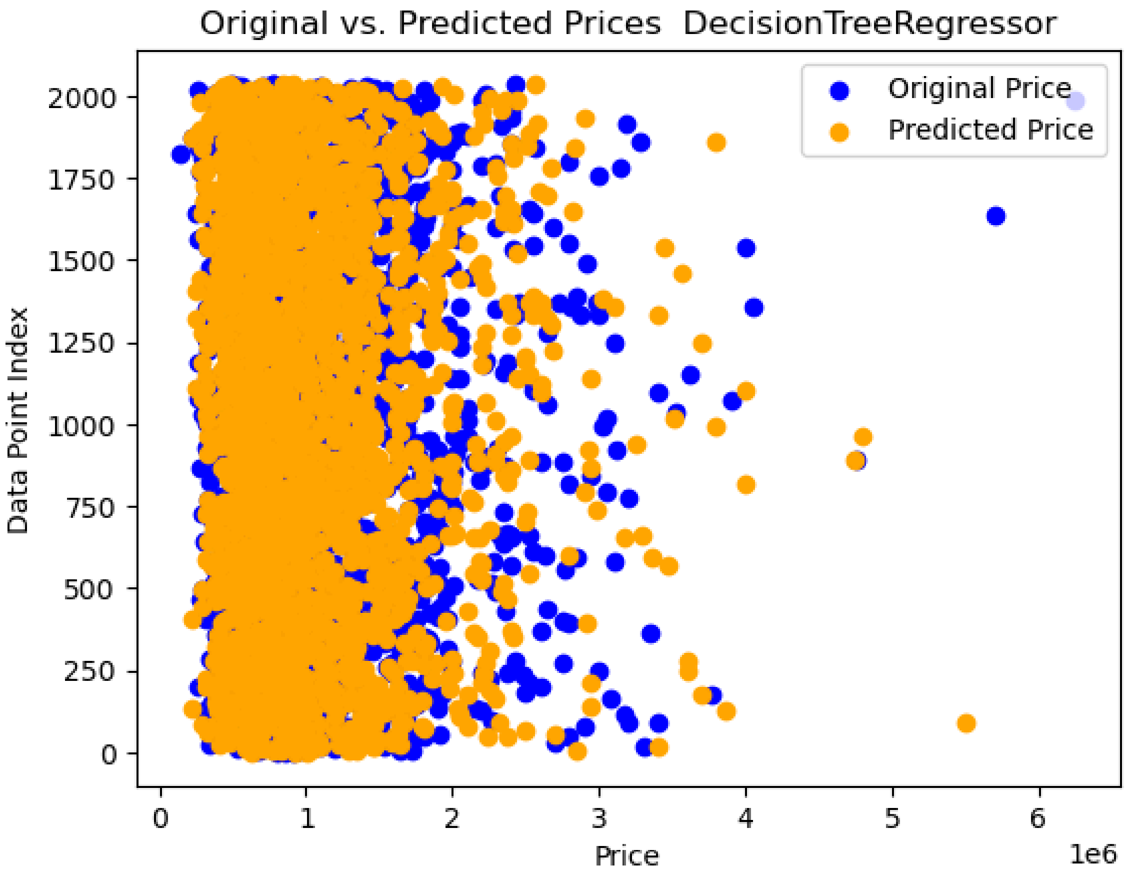
In [78]:
```
mse = mean_squared_error(Y_test, Y_test_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 140854137364.8002
```

In [79]:
```
# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = Y_test_pred.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df1 = pd.DataFrame({'Original Price': Y_test_rounded, 'Predicted

# Display the DataFrame
print(comparison_df1)
```

```
       Original Price  Predicted Price
1993           860000          1008000
8559           800000           625000
5900           910000           920000
9653           660000           830000
804            830000          1350000
...               ...              ...
5444           560000           440000
4190           776000           851000
8213           496000           485000
10366         2425000          2560000
5328           777500           912000

[2037 rows x 2 columns]
```

In [80]:
```
import matplotlib.pyplot as plt
import pandas as pd

try:
  # Extract data for plotting
  original_prices = comparison_df1['Original Price']
  predicted_prices = comparison_df1['Predicted Price']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted Prices  DecisionTreeRegressor')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [81]:
```
modelDecisionTreeRegressor.score(X_test, Y_test)
```

Out[81]: 0.6149424147968077

In [ ]:

In [ ]:
```
#LogisticRegression
```

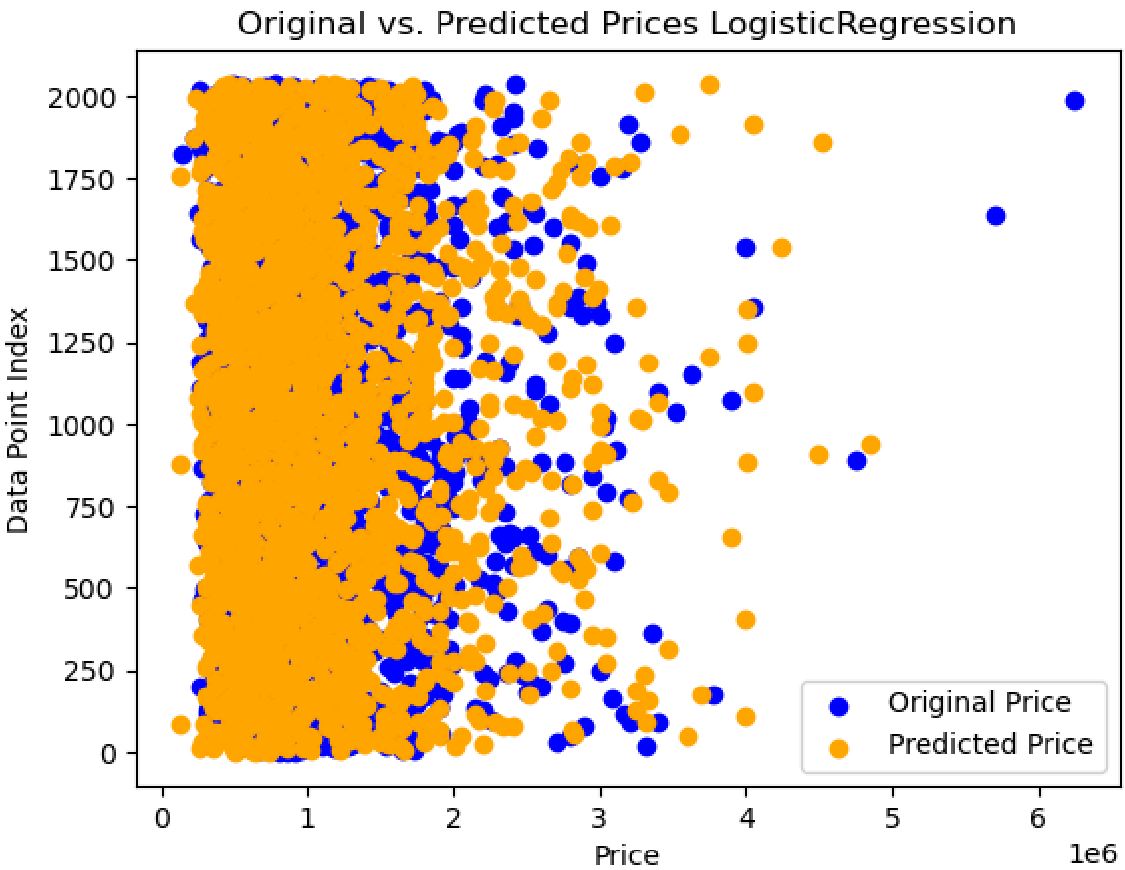
In [82]:
```
from sklearn.linear_model import LogisticRegression

# Initialize and train your logistic regression model
modelLogisticRegression = LogisticRegression()
modelLogisticRegression.fit(X_train, Y_train)

# Make predictions on the test set
y_pred_LOGISTIC = modelLogisticRegression.predict(X_test)
```

```
C:\Users\niran\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [83]:
```
# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = y_pred_LOGISTIC.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df2 = pd.DataFrame({'Original Price': Y_test_rounded, 'Predicted

# Display the DataFrame
print(comparison_df2)
```

```
       Original Price  Predicted Price
1993           860000           643000
8559           800000           511000
5900           910000           732500
9653           660000           503000
804            830000           700000
...               ...              ...
5444           560000           550000
4190           776000          1191000
8213           496000           465000
10366         2425000          3750000
5328           777500          1100000

[2037 rows x 2 columns]
```

In [84]:
```
import matplotlib.pyplot as plt
import pandas as pd


try:
  # Extract data for plotting
  original_prices = comparison_df2['Original Price']
  predicted_prices = comparison_df2['Predicted Price']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted Prices LogisticRegression')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [85]:
```
score = modelLogisticRegression.score(X_test, Y_test)
print("R-squared (score):", score)
```

```
R-squared (score): 0.005891016200294551
```

In [86]:
```
modelLogisticRegression.score(X_test, Y_test)
```

Out[86]:
```
0.005891016200294551
```

In [ ]:

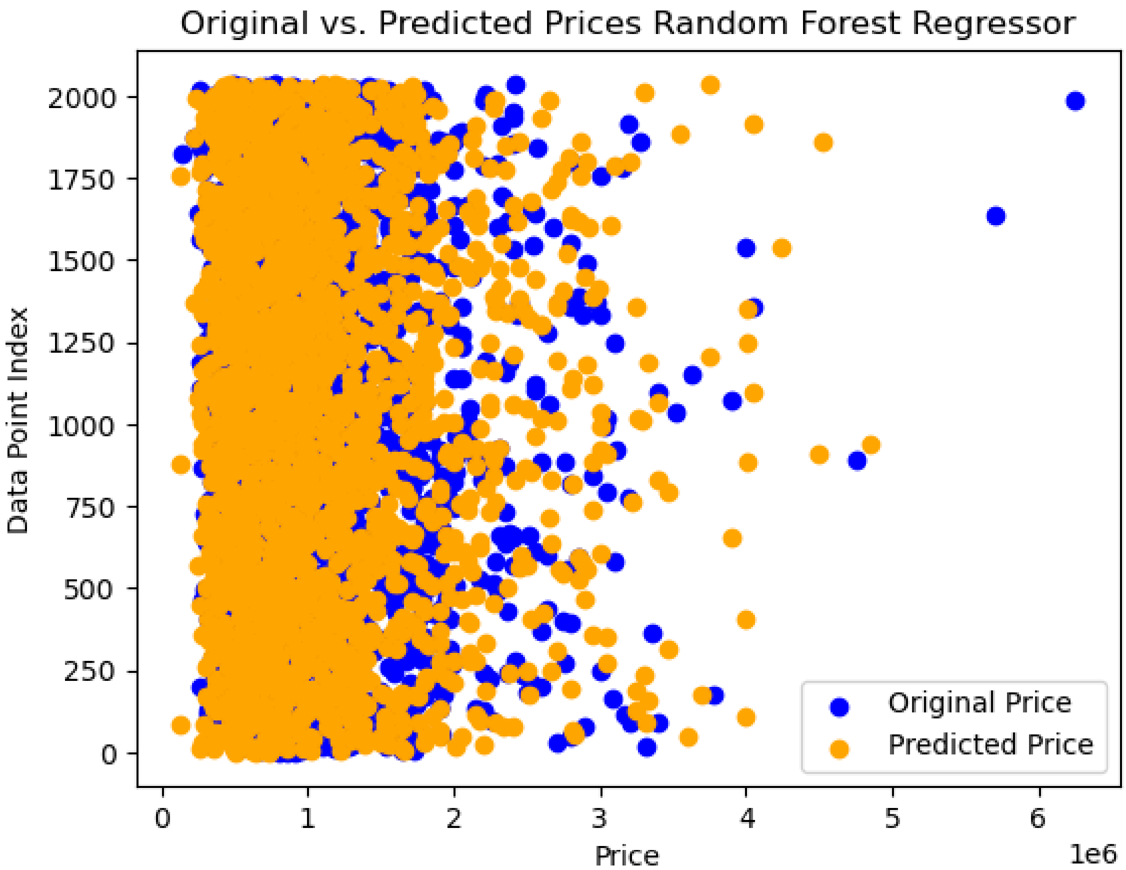
```
#RandomForestRegressor
```

In [87]:
```
from sklearn.ensemble import RandomForestRegressor



# Initialize and train the Random Forest Regressor model
rf_regressor = RandomForestRegressor(random_state=2)
rf_regressor.fit(X_train, Y_train)

# Make predictions on the test set
y_predRandomForestRegressor= rf_regressor.predict(X_test)
```

In [88]:
```
# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = y_predRandomForestRegressor.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df1 = pd.DataFrame({'Original Price': Y_test_rounded, 'Predicted

# Display the DataFrame
print(comparison_df1)
```

```
       Original Price  Predicted Price
1993           860000           972700
8559           800000           662670
5900           910000          1000230
9653           660000           673070
804            830000          1010620
...               ...              ...
5444           560000           624992
4190           776000           781065
8213           496000           480115
10366         2425000          2805650
5328           777500          1053555

[2037 rows x 2 columns]
```

In [89]:
```
import matplotlib.pyplot as plt
import pandas as pd


try:
  # Extract data for plotting
  original_prices = comparison_df2['Original Price']
  predicted_prices = comparison_df2['Predicted Price']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted Prices Random Forest Regressor')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [90]:
```
rf_regressor.score(X_test, Y_test)
```

Out[90]: 0.7496941387506733

In [ ]:

In [ ]:

```

```

HistGradientBoostingRegressor

In [92]:

```
from sklearn.ensemble import HistGradientBoostingRegressor
hist_gb_regressor = HistGradientBoostingRegressor()
hist_gb_regressor.fit(X_train, Y_train)
```

Out[92]:

```
▾ HistGradientBoostingRegressor
HistGradientBoostingRegressor()
```

In [93]:

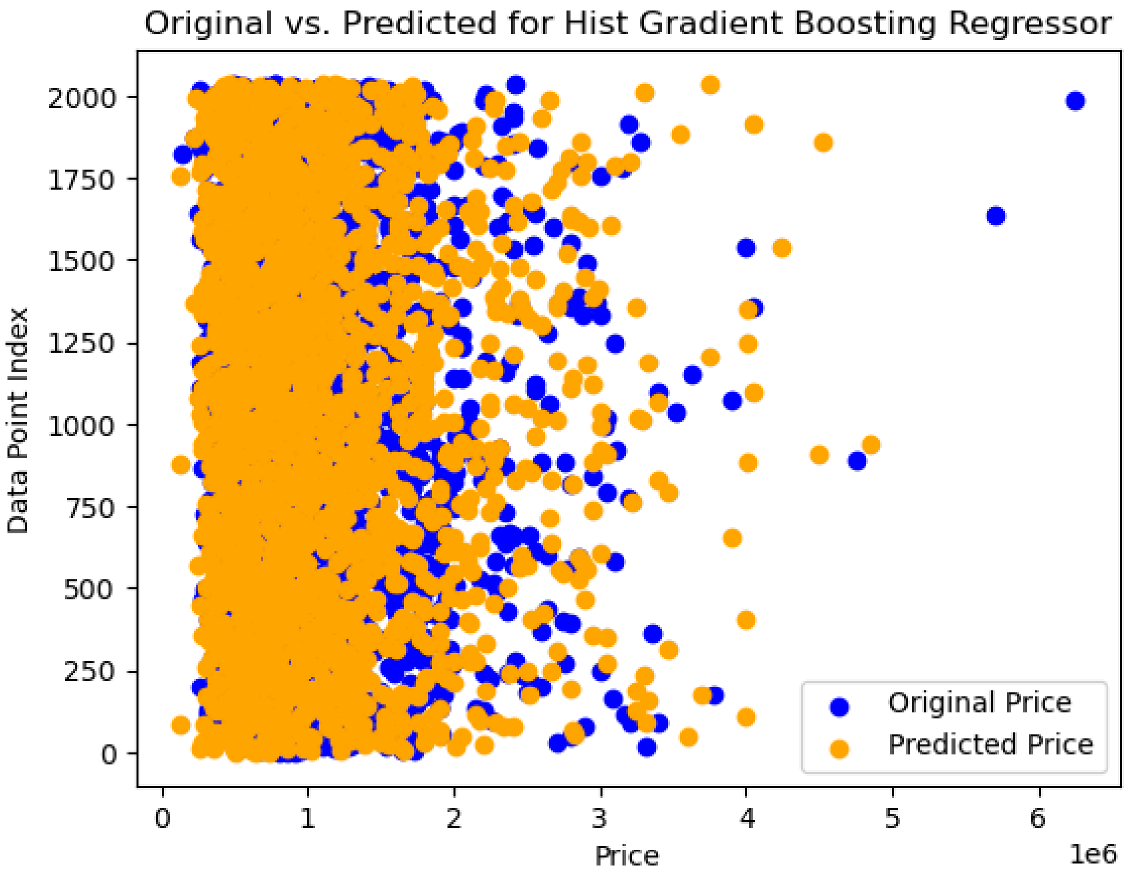
```
y_predHIST = hist_gb_regressor.predict(X_test)
```

In [94]:

```
# Round the original and predicted prices to integers
Y_test_rounded = Y_test.round().astype(int)
Y_pred_rounded = y_predHIST.round().astype(int)

# Create a DataFrame to store original and predicted prices
comparison_df1 = pd.DataFrame({'Original Price': Y_test_rounded, 'Predicted

# Display the DataFrame
print(comparison_df1)
```

```
       Original Price  Predicted Price
1993           860000           934388
8559           800000           704793
5900           910000           947312
9653           660000           577801
804            830000           772215
...               ...              ...
5444           560000           518862
4190           776000           985390
8213           496000           516257
10366         2425000          2534855
5328           777500          1003612

[2037 rows x 2 columns]
```

In [95]:
```
import matplotlib.pyplot as plt
import pandas as pd


try:
  # Extract data for plotting
  original_prices = comparison_df2['Original Price']
  predicted_prices = comparison_df2['Predicted Price']

  # Create separate scatter plots
  plt.scatter(original_prices, np.arange(len(original_prices)), color='blue'
  plt.scatter(predicted_prices, np.arange(len(predicted_prices)), color='ora

  # Adjust plot elements (optional)
  plt.xlabel('Price')  # Can be more specific if data allows
  plt.ylabel('Data Point Index')  # Using index for differentiation
  plt.title('Original vs. Predicted for Hist Gradient Boosting Regressor')
  plt.legend()

except (KeyError, TypeError) as e:
  print(f"Error: Data extraction failed. Check DataFrame structure or data t

plt.show()
```



In [96]:
```
hist_gb_regressor.score(X_test, Y_test)
```

Out[96]: 0.8059388719671446

In [ ]:

In [ ]:

In [ ]: