In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
```

Importing Required libraries and Loading Dataset

In [63]:

```
os.chdir('C:\\Users\\ANIL YADAV\\Downloads')
df=pd.read_csv('RealEstateAU_1000_Samples.csv')
df
```

Out[63]:

	index	TID	breadcrumb	category_name	property_type	building_size	land_size	preferred_size	open_date	listing_agency	price	locatio
0	0	1350988	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Added 2 hours ago	Professionals - DARWIN CITY	$435,000	
1	1	1350989	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m²	NaN	171m²	Added 7 hours ago	Nick Mousellis Real Estate - Eview Group Member	Offers Over $320,000	
2	2	1350990	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	NaN	NaN	Added 22 hours ago	Habitat Real Estate - THE GARDENS	$310,000	
3	3	1350991	Buy>NT>DARWIN	Real Estate & Property for sale	House	NaN	NaN	NaN	Added	Ray White -	$259,000	

Let's First Check all the Columns contained in the DataFrame.

In [4]:

```
df.columns
```

Out[4]:

```
Index(['index', 'TID', 'breadcrumb', 'category_name', 'property_type',
       'building_size', 'land_size', 'preferred_size', 'open_date',
       'listing_agency', 'price', 'location_number', 'location_type',
       'location_name', 'address', 'address_1', 'city', 'state', 'zip_code',
       'phone', 'latitude', 'longitude', 'product_depth', 'bedroom_count',
       'bathroom_count', 'parking_count', 'RunDate'],
      dtype='object')
```

Removing unwanted columns

In [64]:

```
df.drop(["index", "TID", "breadcrumb", "open_date", "phone", "RunDate"], axis = 1, inplace = True)
```

In [6]:

```
df.head()
```

Out[6]:

'y_name	property_type	building_size	land_size	preferred_size	listing_agency	price	location_number	location_type	location_name	...	address_1	
Estate & / for sale DARWIN CITY...	House	NaN	NaN	NaN	Professionals - DARWIN CITY	$435,000	139468611	Buy	$435,000	...	44 Woods Street	Dar
Estate & / for sale DARWIN CITY...	Apartment	171m²	NaN	171m²	Nick Mousellis Real Estate - Eview Group Member	Offers Over $320,000	139463755	Buy	Offers Over $320,000	...	14/14 Dashwood Place	Dar
Estate & / for sale DARWIN CITY...	Unit	NaN	NaN	NaN	Habitat Real Estate - THE GARDENS	$310,000	139462495	Buy	$310,000	...	13/86 Woods Street	Dar
Estate & / for sale DARWIN CITY...	House	NaN	NaN	NaN	Ray White - NIGHTCLIFF	$259,000	139451679	Buy	$259,000	...	1309/43B Knuckey Street	Dar
Estate & / for sale DARWIN CITY...	Unit	201m²	NaN	201m²	Carol Need Real Estate - Fannie Bay	$439,000	139433803	Buy	$439,000	...	3/10 McLachlan Street	Dar

1 columns

Checking count of null values

In [7]:

```
nan_val=df.isnull().sum()
```

In [8]:

```
nan_val
```

Out[8]:

```
category_name         0
property_type         0
building_size       720
land_size           467
preferred_size      391
listing_agency        0
price                 0
location_number       0
location_type         0
location_name         0
address              12
address_1            12
city                  0
state                 0
zip_code              0
latitude           1000
longitude          1000
product_depth         0
bedroom_count        33
bathroom_count       33
parking_count        33
dtype: int64
```

The "Building Size", "Land Size" and "Preferred Size" are sadly columns to drop, since there are too many missing values.

In [9]:

```
df.drop(["preferred_size", "building_size", "land_size"], axis = 1, inplace = True)
```

We have got 2 main ways to approach this:

1) Remove the Rows with the missing values

2) Replace the Missing Values

Since the Missing Value Rows are just 33 out of 1000, that represent a 3.3% of the total DataFrame Size, so it won't really mess that much with the DataSet integrity.

That being said, we decide to go ahead and replace missing value with the mode, which is the most occurring value in a column, since it's the Null Filling/Replacement technique that makes the most sense.

Exploratory Data Analysis

In [10]:

```
df["bedroom_count"].fillna(df["bedroom_count"].mode()[0], inplace = True)
df["bathroom_count"].fillna(df["bedroom_count"].mode()[0], inplace = True)
df["parking_count"].fillna(df["bedroom_count"].mode()[0], inplace = True)
df.isnull().sum()
```

Out[10]:

```
category_name         0
property_type         0
listing_agency        0
price                 0
location_number       0
location_type         0
location_name         0
address              12
address_1            12
city                  0
state                 0
zip_code              0
latitude           1000
longitude          1000
product_depth         0
bedroom_count         0
bathroom_count        0
parking_count         0
dtype: int64
```

In [12]:

```
df["category_name"].value_counts()
```

Out[12]:

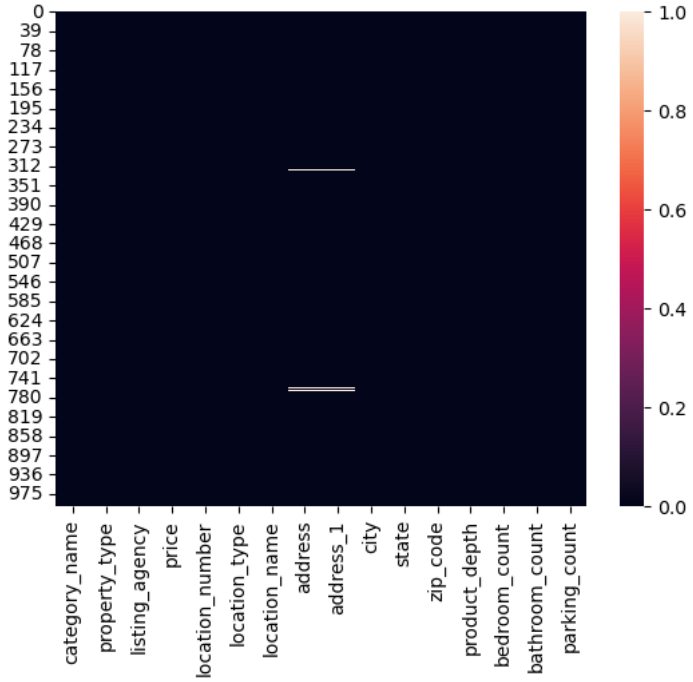
```
Real Estate & Property for sale in DARWIN, NT 0801         816
Real Estate & Property for sale in DARWIN CITY, NT 0800    184
Name: category_name, dtype: int64
```

In [11]:

```
df.drop(["longitude", "latitude"], axis = 1, inplace = True)
sns.heatmap(df.isnull())
```

Out[11]:

```
<AxesSubplot:>
```



EDA on Categorical Attributes

In [13]:

```
df["property_type"].value_counts()
```

Out[13]:

```
House                  441
Unit                   230
Apartment              212
Townhouse               38
Residential Land        33
Duplex/Semi-detached    19
Acreage                  9
Block Of Units           6
Other                    4
Villa                    4
Studio                   2
Warehouse                1
Lifestyle                1
Name: property_type, dtype: int64
```

In [14]:

```
df["listing_agency"].value_counts()
```

Out[14]:

```
Real Estate Central - DARWIN CITY                 113
Elders Real Estate - Darwin                        62
Elders Real Estate - Palmerston                    53
Raine & Horne - Darwin                             48
First National Real Estate O'Donoghues - Darwin    41
                                                 ...
Ellis Parker Real Estate - LARRAKEYAH               1
Dunvegan Real Estate - PALMERSTON                   1
Australian Home Partners                            1
buymyplace                                          1
Mercury Real Estate                                 1
Name: listing_agency, Length: 85, dtype: int64
```

In [15]:

```
df["city"].value_counts()
```

Out[15]:

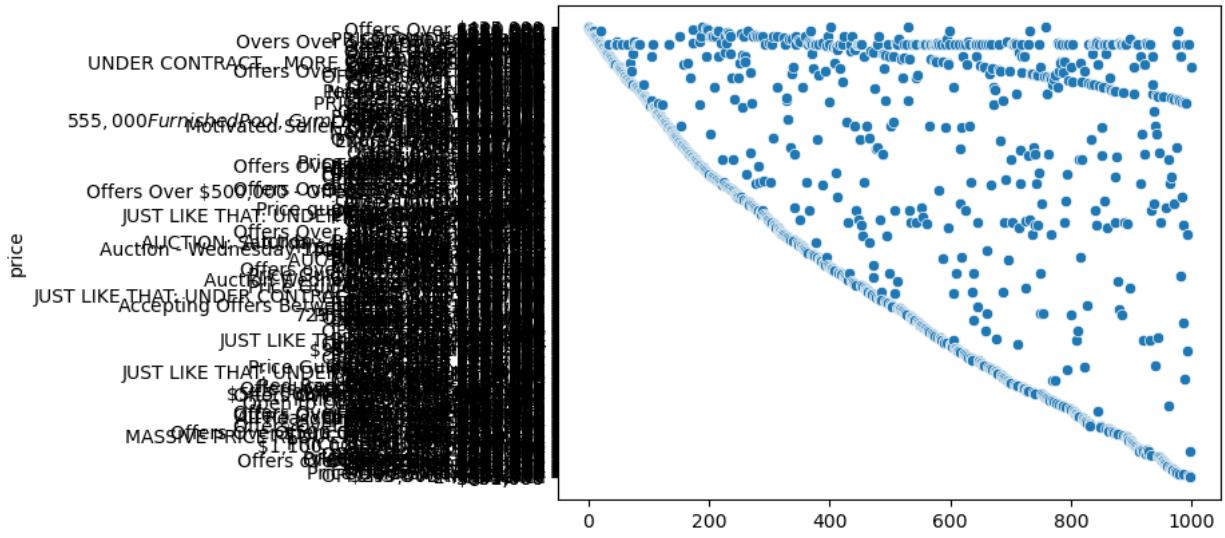
```
Darwin City         285
Stuart Park          39
Rosebery             37
Bakewell             31
Durack               30
Zuccoli              29
Woodroffe            27
Nightcliff           27
Driver               26
Parap                26
Rapid Creek          25
Bellamack            23
Humpty Doo           20
Johnston             20
Leanyer              19
Gunn                 19
Gray                 19
Karama               16
Moulden              15
Howard Springs       15
Berrimah             15
Bayview              14
Fannie Bay           14
Farrar               12
Coconut Grove        12
Muirhead             12
The Gardens          11
Lyons                10
Millner              10
Woolner               9
Jingili               9
Herbert               9
Tiwi                  9
Larrakeyah            9
Ludmilla              7
Alawa                 7
Anula                 7
Wagaman               7
Malak                 7
Wulagi                6
Virginia              6
Brinkin               6
Wanguri               6
Berry Springs         6
Moil                  5
Lee Point             4
Nakara                4
Marrara               3
Coolalinga            3
Girraween             3
Bees Creek            3
Cullen Bay            3
The Narrows           1
Knuckey Lagoon        1
Rosebery Heights      1
Marlow Lagoon         1
Name: city, dtype: int64
```

In [16]:

```
sns.scatterplot(data = df["price"])
```

Out[16]:

```
<AxesSubplot:ylabel='price'>
```



In [17]:

```
df.columns
```

Out[17]:

```
Index(['category_name', 'property_type', 'listing_agency', 'price',
       'location_number', 'location_type', 'location_name', 'address',
       'address_1', 'city', 'state', 'zip_code', 'product_depth',
       'bedroom_count', 'bathroom_count', 'parking_count'],
      dtype='object')
```

In [18]:

```
import re

def extract_price(x):
    if x != "":

        match = re.search(r'\$?\s*(\d+(?:\.\d+)?)[k|m|K|M]\s*', x)

        if match:
            price = match.group(1)
            price = float(price)
            if x[-1].lower() == 'k':
                price *= 1000
            elif x[-1].lower() == 'm':
                price *= 1000000
            return int(price)
        else:
            match = re.sub(r'[^\d]', '', x)

            try:
                return int(match)
            except:
                try:
                    return(float(match))
                except:
                    return  None
df["NumericalPrice"] = df["price"]

df["NumericalPrice"] = df["NumericalPrice"].apply(lambda x: extract_price(x))
```

In [19]:

```
pd.options.display.max_columns =  None
pd.options.display.max_rows = None

df[["price", "NumericalPrice"]]
```

14	$625,000	6.250000e+05
15	Overs Over $599,000 Considered	5.990000e+05
16	$490,000	4.900000e+05
17	$337,500	3.375000e+05
18	FASTRAK	NaN
19	$439,000	4.390000e+05
20	UNDER CONTRACT	NaN
21	Offers Over $440,000	4.400000e+05
22	$640,000	6.400000e+05
23	$500,000	5.000000e+05
24	$305,000 +	3.050000e+05
25	$295,000 +	2.950000e+05
26	$795,000	7.950000e+05

In [20]:

```
c = 0

for x in df["NumericalPrice"]:
    x = str(x)

    if len(x) <= 5:
        x = None

    elif len(x) >= 8:
        count0 = 0

        for n in x:
            if n == "0":
                count0 += 1

        if count0 <= 2:
            x = None

        else:
            x = x[0:6]

    df["NumericalPrice"][c] = x

    c += 1
```

```
C:\Users\ANIL YADAV\AppData\Local\Temp\ipykernel_16424\2997109868.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df["NumericalPrice"][c] = x
```

In [21]:

```
df[["price", "NumericalPrice"]]
```

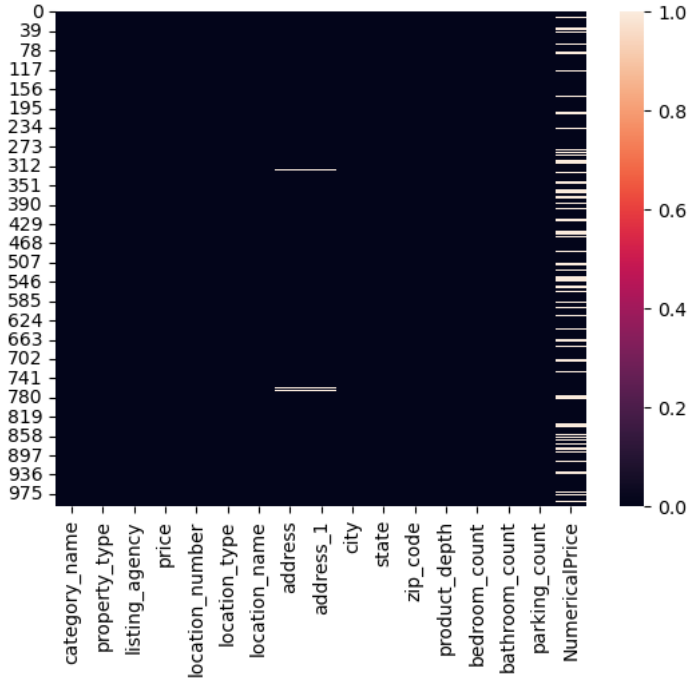
24	$305,000 +	305000
25	$295,000 +	295000
26	$795,000	795000
27	Offers Over $950,000	950000
28	Offers Over $485,000	485000
29	$250,000	250000
30	$549,000	549000
31	Offers over $299,000	299000
32	$395,000	395000
33	UNDER CONTRACT	None
34	$475,000	475000
35	Contact Agent	None
36	OFFERS INVITED	None

In [22]:

```
sns.heatmap(df.isnull())
```

Out[22]:

<AxesSubplot:>



In [23]:

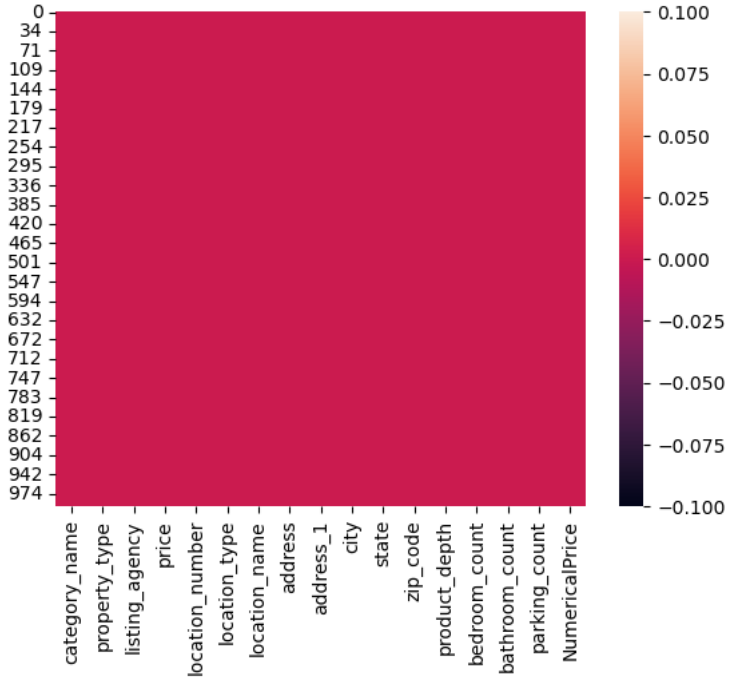
```
df.dropna(axis = 0, how = "any", inplace = True)
```

In [24]:

```
sns.heatmap(df.isnull())
```

Out[24]:

<AxesSubplot:>



In [25]:

```
df["NumericalPrice"]
```

```
30      549000
31      299000
32      395000
34      475000
37      465000
38      600000
39      980000
40      105000
42      450000
43      749000
45      289000
46      490000
48      565000
49      399000
50      469000
51      649000
52      499000
53      400000
54      800000
56      299000
```

In [26]:

```
df["NumericalPrice"].astype("float")
```

Out[26]:

```
0      435000.0
1      320000.0
2      310000.0
3      259000.0
4      439000.0
5      825000.0
6      820000.0
7      369000.0
8      439000.0
9      455000.0
10     280000.0
12     439000.0
13     775000.0
14     625000.0
15     599000.0
16     490000.0
17     337500.0
19     439000.0
```

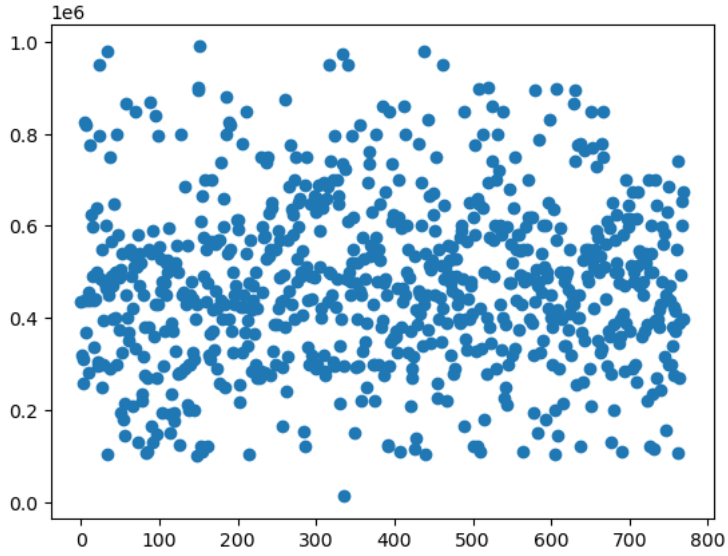
Building The Final Model

In [27]:

```
plt.scatter(x = [i for i in range(len(df))], y = df["NumericalPrice"].astype("float"))
```

Out[27]:

```
<matplotlib.collections.PathCollection at 0x1acf8836f70>
```



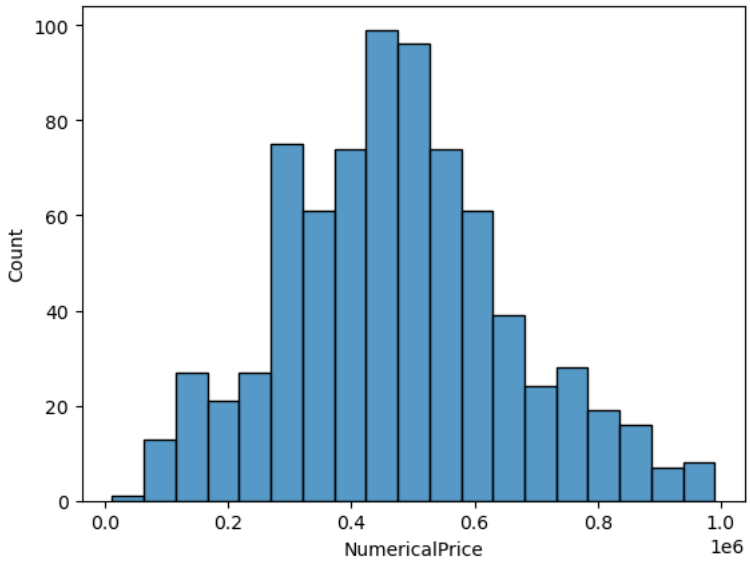
Price Data is not Linearly Distributed, however, we'll Try to Firstly Build a Model with a Linear Regression.

In [28]:

```
sns.histplot(df["NumericalPrice"].astype("float"))
```

Out[28]:

```
<AxesSubplot:xlabel='NumericalPrice', ylabel='Count'>
```



In [29]:

```
df["NumericalPrice"] = df["NumericalPrice"].astype("float")
```

In [30]:

```
df.columns
```

Out[30]:

```
Index(['category_name', 'property_type', 'listing_agency', 'price',
       'location_number', 'location_type', 'location_name', 'address',
       'address_1', 'city', 'state', 'zip_code', 'product_depth',
       'bedroom_count', 'bathroom_count', 'parking_count', 'NumericalPrice'],
      dtype='object')
```

Building a New DataFrame, containing all the Columns that we will use for our Machine Learning.

In [31]:

```
newdf = df[["property_type", "zip_code", "bedroom_count", "bathroom_count", "parking_count","NumericalPrice"]]
```

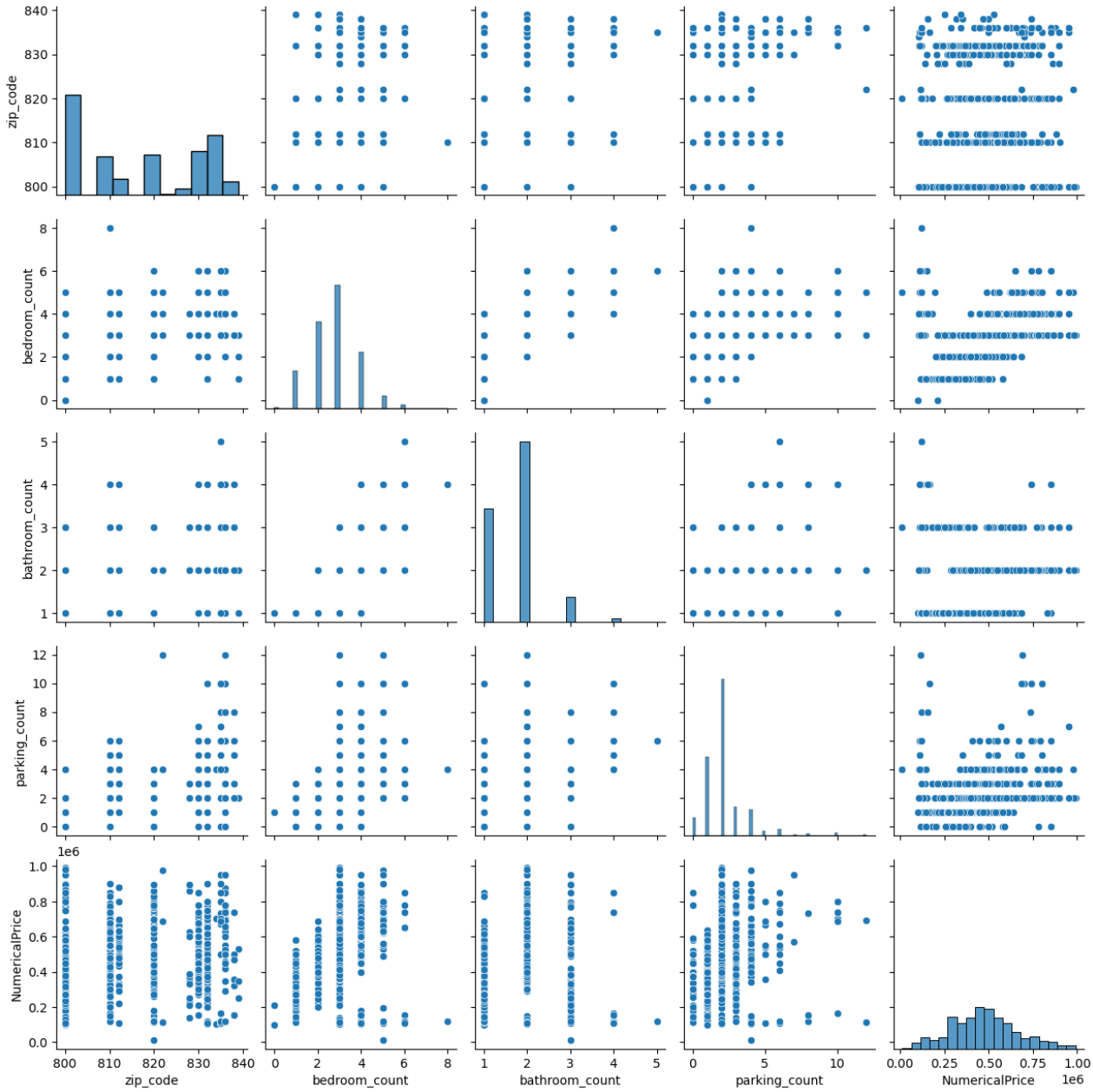
Getting Correlations

In [32]:

```
sns.pairplot(data = newdf)
```

Out[32]:

```
<seaborn.axisgrid.PairGrid at 0x1acf8a0e340>
```



Creating Dummy Variables

In [33]:

```
newdf = pd.get_dummies(newdf)
```

In [34]:

```
newdf.head()
```

Out[34]:

	zip_code	bedroom_count	bathroom_count	parking_count	NumericalPrice	property_type_Acreage	property_type_Apartment	property_type_Block Of Units	prop
0	800	2.0	1.0	1.0	435000.0	0	0	0	
1	800	3.0	2.0	2.0	320000.0	0	1	0	
2	800	2.0	1.0	1.0	310000.0	0	0	0	
3	800	1.0	1.0	0.0	259000.0	0	0	0	
4	800	3.0	2.0	2.0	439000.0	0	0	0	

The Steps for Building Our Linear Regression Model Are:

1.Create the X Features Variable and Y the Predicted Variable

2.Split the DataSet into a TrainSet and a TestSet

3.Fit the Model

4.Use the Model to make Predictions

5.Calculate Errors

6.Evaluate the Model

In [35]:

```
from sklearn.linear_model import LinearRegression as LR
from sklearn.model_selection import train_test_split as TTS
x = newdf.drop("NumericalPrice", axis = 1, inplace= False)
y = newdf["NumericalPrice"]
```

In [36]:

```
xTrain, xTest, yTrain, yTest = TTS(x, y, test_size = 0.3)
LR_ = LR()
LR_.fit(xTrain, yTrain)
```

Out[36]:

```
LinearRegression()
```

In [37]:

```
preds = LR_.predict(xTest)
```

In [38]:

```
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import r2_score as R2
```

In [39]:

```
mae = MAE(preds, yTest)
mse = MSE(preds, yTest)
rmse = MSE(preds, yTest, squared= False)
r2 = R2(preds, yTest)
```

In [40]:

```
print(mae)
```

```
111452.02466201608
```

In [41]:

```
print(mse)
```

```
32291101602.001003
```

In [42]:

```
print(rmse)
```

```
179697.2498454025
```

In [43]:

```
print(r2)
```

```
-0.6104786872958645
```

In [44]:

```
#We Resize the Unit of Our Prices for Convenience Purposes

newdf["RoundedPrice"] = newdf["NumericalPrice"].apply(lambda x: x/100000)
```

In [45]:

```
from sklearn.neighbors import KNeighborsRegressor as KNR
from sklearn.tree import DecisionTreeRegressor as DTR
from sklearn.model_selection import GridSearchCV as GSCV

x = newdf.drop("RoundedPrice", axis = 1, inplace= False)
y = newdf["RoundedPrice"]
xTrain, xTest, yTrain, yTest = TTS(x, y, test_size = 0.3)
```

In [46]:

```
Parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNR()

KNN_CV = GSCV(KNN, Parameters, cv = 10)
KNN_CV.fit(xTrain, yTrain)
KNN_CV.best_params_
```

Out[46]:

```
{'algorithm': 'auto', 'n_neighbors': 1, 'p': 1}
```

In [47]:

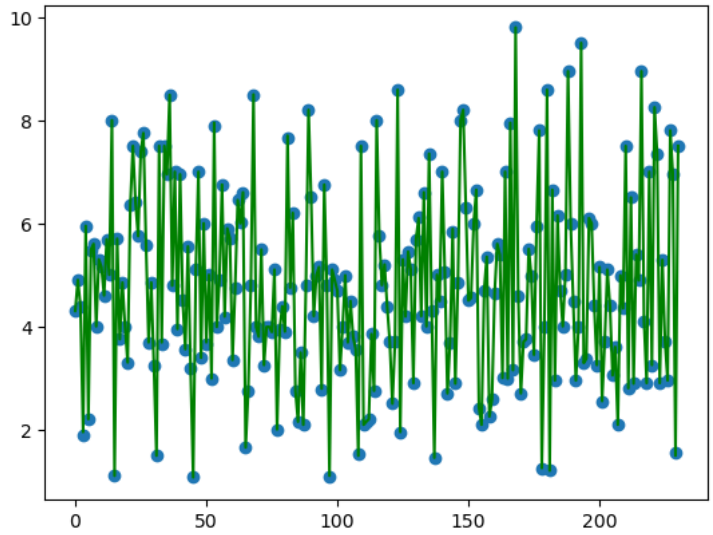
```
Preds = KNN_CV.predict(xTest)
```

In [48]:

```
plt.scatter(x = [i for i in range(len(yTest))], y = yTest)
plt.plot(Preds, color = "green")
```

Out[48]:

```
[<matplotlib.lines.Line2D at 0x1acfd1b2130>]
```



In [49]:

```
KNNMAE = MAE(yTest, Preds)
KNNMSE = MSE(yTest, Preds)
KNNRMSE = MSE(yTest, Preds, squared = False)
KNNR2 = R2(yTest, Preds)
```

In [50]:

```
Errors = pd.DataFrame(data = [KNNMAE, KNNMSE, KNNRMSE, KNNR2], index = ["MAE", "MSE", "RMSE", "R2"], columns = ["KNR"])
Errors
```

Out[50]:

	KNR
MAE	0.002706
MSE	0.000096
RMSE	0.009806
R2	0.999972

In [51]:

```
print('R2 Value : ',KNNR2)
```

```
R2 Value :  0.9999720694823959
```

In [52]:

```
from sklearn.model_selection import cross_val_score as CrossVal

S = CrossVal(KNN_CV, x, y, cv = 10)
S
```

Out[52]:

```
array([1.        , 0.99991606, 0.99997034, 0.99993879, 0.99733808,
       0.99997504, 0.99998958, 0.99997527, 0.99996503, 0.99995697])
```

In [53]:

```
Parameters = {'criterion': ["squared_error", "friedman_mse", "absolute_error", "poisson"],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt', "log2"],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

Tree = DTR()
TreeCV = GSCV(Tree, Parameters, cv = 10)
TreeCV.fit(xTrain, yTrain)
```

Out[53]:

```
GridSearchCV(cv=10, estimator=DecisionTreeRegressor(),
             param_grid={'criterion': ['squared_error', 'friedman_mse',
                                       'absolute_error', 'poisson'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

In [54]:

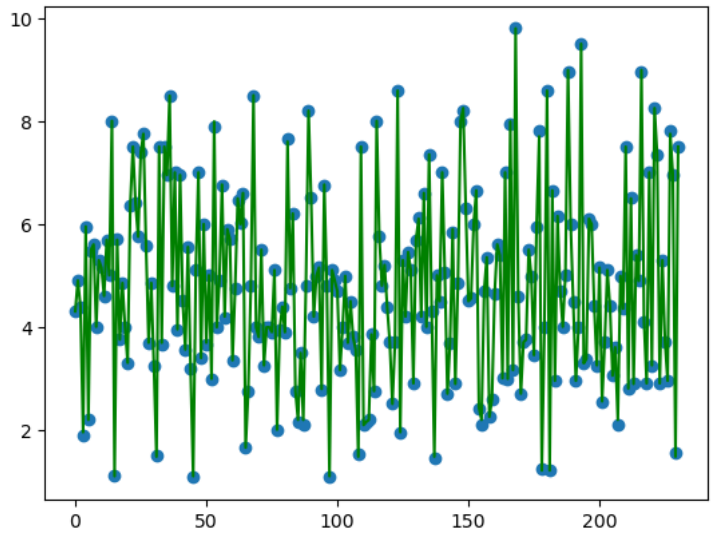
```
Preds = TreeCV.predict(xTest)
```

In [55]:

```
plt.scatter(x = [i for i in range(len(yTest))], y = yTest)
plt.plot(Preds, color = "green")
```

Out[55]:

```
[<matplotlib.lines.Line2D at 0x1acfd22da30>]
```



In [56]:

```
TreeMAE = MAE(yTest, Preds)
TreeMSE = MSE(yTest, Preds)
TreeRMSE = MSE(yTest, Preds, squared = False)
TreeR2 = R2(yTest, Preds)
```

In [57]:

```
print('R2 Value : ',TreeR2)
```

```
R2 Value :  0.9998880029465869
```

In [59]:

```
TreeMAE = MAE(yTest, Preds)
TreeMSE = MSE(yTest, Preds)
TreeRMSE = MSE(yTest, Preds, squared = False)
TreeR2 = R2(yTest, Preds)
```

In [60]:

```
Errors["DTR"] = [TreeMAE, TreeMSE, TreeRMSE, TreeR2]
Errors["LinearRegression"] = [mae/100000, mse/100000, rmse/100000, r2]
```

In [61]:

```
S = CrossVal(TreeCV, x, y, cv = 10)
S
```

Out[61]:

```
array([1.        , 0.99987079, 0.9998753 , 0.99984703, 0.997002  ,
       0.99982301, 0.99996593, 0.99996167, 0.99995262, 0.99989209])
```

In [62]:

```
Errors
```

Out[62]:

	KNR	DTR	LinearRegression
MAE	0.002706	0.007600	1.114520
MSE	0.000096	0.000386	322911.016020
RMSE	0.009806	0.019637	1.796972
R2	0.999972	0.999888	-0.610479

As we Can Conclude, Both DecisionTreeRegressor and KNearestNeighborRegressor can fit the Data with Relatively Low Errors. Please Mind that Errors for the 2 Models are with Reduced Unit, so, Divided by 100000, to have them with the Same Measuring Unit as the Linear Regression Errors, we Just Need to Multiply them by 100000.