Importing the Libraries Needed for EDA and Data Wrangling/Cleaning.

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

%matplotlib inline
```

Importing the DataSet

```
df = pd.read_csv("/kaggle/input/australian-housing-data-1000-properties-sampled/RealEstateAU_1000_Samples.csv")
df
```

	index	TID	breadcrumb	category_name	property_type	building_size	la
0	0	1350988	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	
1	1	1350989	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m²	
2	2	1350990	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	
3	3	1350991	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	
4	4	1350992	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	201m²	
...	...	...	...	...	...	...	
995	995	1351983	Buy>NT>DARWIN	Real Estate & Property for sale in DARWIN, NT ...	House	NaN	
996	996	1351984	Buy>NT>DARWIN	Real Estate & Property for sale in DARWIN, NT ...	House	203m²	
997	997	1351985	Buy>NT>DARWIN	Real Estate & Property for sale in DARWIN, NT ...	House	209.6m²	
998	998	1351986	Buy>NT>DARWIN	Real Estate & Property for sale in DARWIN, NT ...	House	180m²	
999	999	1351987	Buy>NT>DARWIN	Real Estate & Property for sale in DARWIN, NT ...	Unit	120m²	

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

```
df.columns
```

```
df.columns
```

```
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')
```

We can Cearly see that we won't need some of the Columns, so let's just drop them.

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

Let's Check our DataFrame Again

```
df.head()
```

	category_name	property_type	building_size	land_size	preferred_size	listing_agency	price	location_number	location_type	loca
0	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Professionals - DARWIN CITY	$435,000	139468611	Buy	
1	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m²	NaN	171m²	Nick Mousellis Real Estate - Eview Group Member	Offers Over $320,000	139463755	Buy	
2	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	NaN	NaN	Habitat Real Estate - THE GARDENS	$310,000	139462495	Buy	
3	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Ray White - NIGHTCLIFF	$259,000	139451679	Buy	
4	Real Estate & Property for sale in DARWIN CITY...	Unit	201m²	NaN	201m²	Carol Need Real Estate - Fannie Bay	$439,000	139433803	Buy	

5 rows × 21 columns

Now, let's see how many Null Values we have got and how we can handle Them

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

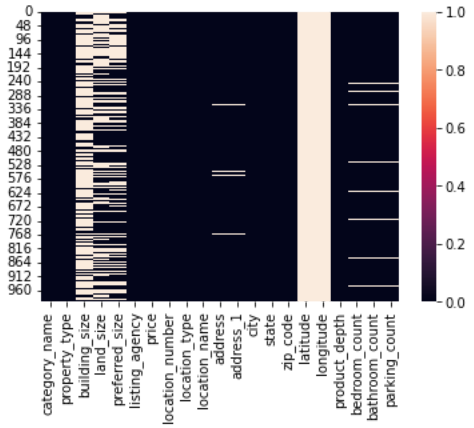
```
category_name         0
property_type         0
building_size       720
land_size           467
preferred_size      391
listing_agency        0
price                 0
location_number       0
location_type         0
location_name         0
address              12
address_1            12
city    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

```
parking_count        33
dtype: int64
```

Let's Create an Heatmap to see how these Null Values are distributed

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

```
<AxesSubplot:>
```



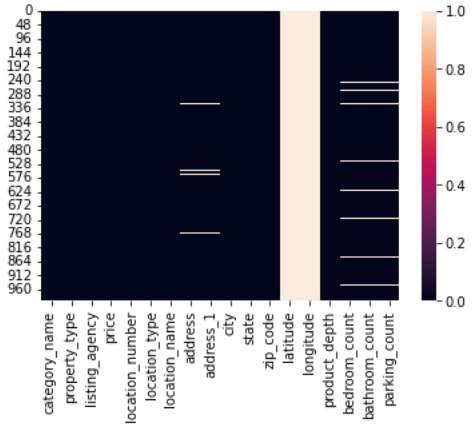
The "Building Size", "Land Size" and "Preferred Size" are sadly columns to drop, since there are too many missing values.

Alternatively, we could create a new DataFrame to be analyzed with Only the Rows that are not Null for the 3 Columns mentioned, but that would reduce the size of our samples, which is already small.

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

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

```
<AxesSubplot:>
```



Now, we See in the Columns that we have no Latitude and Longitude Data, so we are going to drop them.

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

```
<AxesSubplot:>
```

Now, some of the Columns won't be used probably, but we decide to leave them there, just in case.

We also notice that the count for bedrooms, bathrooms and parking are missing into the same Rows.

We have got 2 main ways to approach this:

1) Remove the Rows with the missing values

2) Replace the Missing Values

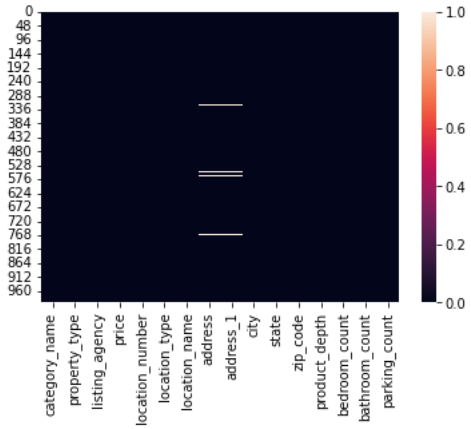
Since the Missing Value Rows are just 33 out of 1000, that represent a 3.3% of the total DataFrame Size, so it won't really mess that much with the DataSet integrity.

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.

```
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)
```

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

```
<AxesSubplot:>
```



Now, let's have a look to our Data and do some EDA

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

```
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
```

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

```
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
```

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

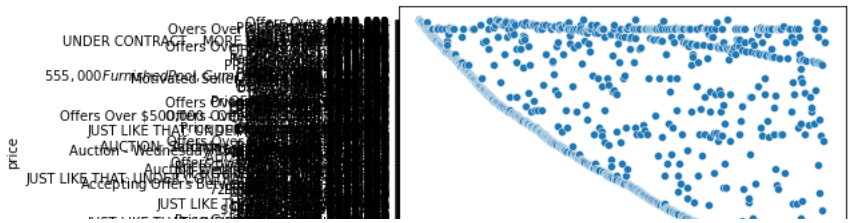
```
Real Estate Central - DARWIN CITY                  113
Elders Real Estate - Darwin                         62
Elders Real Estate - Palmerston                     53
Raine & Horne - Darwin                              48
First National Real Estate O'Donoghues - Darwin     41
                                                   ...
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
```

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

```
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
```

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

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



Well, feels like we need to clean or reorganize that messy data :D

As we can See, not every price is a continuous Value, with some prices being more like a categorical variable.

So, from now on we will work around that to gather all the numerical prices only from the "price" Column, ignoring every other "Categorical Value".

```
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))
```

```
'''
import re

# Define a function that extracts the price information from a string
def extract_price(x):
    if x != "":
        # Use a regular expression to match the pattern of the string
        #r'\$?(\d+(?:\.\d+)?)
        #match = re.search(r'\$?\s*(\d+(?:\.\d+)?)[k|K]\s*', x)

        match = re.search(r'\$?\s*(\d+(?:\.\d+)?)[k|m|K|M]\s*', x)
        if match:
            # Extract the numeric part of the string
            price = match.group(1)
            # Convert the string to a float
            price = float(price)
            # Multiply the price by 1000 if the suffix is "k" or by 1000000 if the suffix is "m"
            if x[-1].lower() == 'k':
                price *= 1000
            elif x[-1].lower() == 'm':
                price *= 1000000
            return int(price)
        else:
            # Use a regular expression to match any characters that are not digits
            match = re.sub(r'[^\d]', '', x)
            #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: re.sub(r'[^\d]', '', x))

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

#df["NumericalPrice"] = df["NumericalPrice"].apply(lambda x: re.sub(r'[^\d.]', '', x))
'''
```

```
'\nimport re\n\n# Define a function that extracts the price information from a string\ndef extract_price(x):\n    if x != "":\n
# Use a regular expression to match the pattern of the string\n        #r\'\\$?(\\d+(?:\\.\\d+)?)\n        #match = re.search(r\'\\$?
\\s*(\\d+(?:\\.\\d+)?)[k|K]\\s*\', x)\n        \n        match = re.search(r\'\\$?\\s*(\\d+(?:\\.\\d+)?)[k|m|K|M]\\s*\', x)\n        if
match:\n            # Extract the numeric part of the string\n            price = match.group(1)\n            # Convert the string to a
float\n            price = float(price)\n            # Multiply the price by 1000 if the suffix is "k" or by 1000000 if the suffix is
"m"\n            if x[-1].lower() == \'k\':\n                price *= 1000\n            elif x[-1].lower() == \'m\':\n
price *= 1000000\n            return int(price)\n        else:\n            # Use a regular expression to match any characters that are
not digits\n            match = re.sub(r\'[^\\d]\', \'\', x)\n            #match = re.sub(r\'\\.\\d+\', \'\', x)\n            \n
try:\n                return int(match)\n            except:\n                try:\n                    return(float(match))\n
except:\n                    return  None\n            \n                \ndf["NumericalPrice"] = df["price"]\n\n#df["NumericalPrice"]
= df["NumericalPrice"].apply(lambda x: re.sub(r\'[^\\d]\', \'\', x))\n\ndf["NumericalPrice"] = df["NumericalPrice"].apply(lambda x:
extract_price(x))\n\n#df["NumericalPrice"] = df["NumericalPrice"].apply(lambda x: re.sub(r\'[^\\d.]\', \'\', x))\n'
```

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

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

	price	NumericalPrice
0	$435,000	4.350000e+05
1	Offers Over $320,000	3.200000e+05
2	$310,000	3.100000e+05
3	$259,000	2.590000e+05
4	$439,000	4.390000e+05
5	$825,000	8.250000e+05
6	$820,000	8.200000e+05
7	$369 000	3.690000e+05
8	$439,000	4.390000e+05
9	$455,000	4.550000e+05
10	$280,000	2.800000e+05
11	Openn Negotiation	NaN
12	PRICE GUIDE $439,000	4.390000e+05
13	$775,000	7.750000e+05
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
27	Offers Over $950,000	9.500000e+05
28	Offers Over $485,000	4.850000e+05
29	$250,000	2.500000e+05
30	$549.000	5.490000e+05

```
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
```

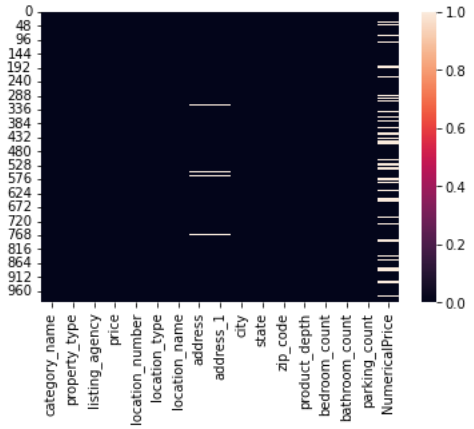
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
```

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

	price	NumericalPrice
0	$435,000	435000.0
1	Offers Over $320,000	320000.0
2	$310,000	310000.0
3	$259,000	259000.0
4	$439,000	439000.0
5	$825,000	825000.0
6	$820,000	820000.0
7	$369 000	369000.0
8	$439,000	439000.0
9	$455,000	455000.0
10	$280,000	280000.0
11	Openn Negotiation	NaN
12	PRICE GUIDE $439,000	439000.0
13	$775,000	775000.0
14	$625,000	625000.0
15	Overs Over $599,000 Considered	599000.0
16	$490,000	490000.0
17	$337,500	337500.0
18	FASTRAK	NaN
19	$439,000	439000.0
20	UNDER CONTRACT	NaN
21	Offers Over $440,000	440000.0
22	$640,000	640000.0
23	$500,000	500000.0
24	$305,000 +	305000.0
25	$295,000 +	295000.0
26	$795,000	795000.0
27	Offers Over $950,000	950000.0
28	Offers Over $485,000	485000.0
29	$250,000	250000.0
30	$549.000	549000.0

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

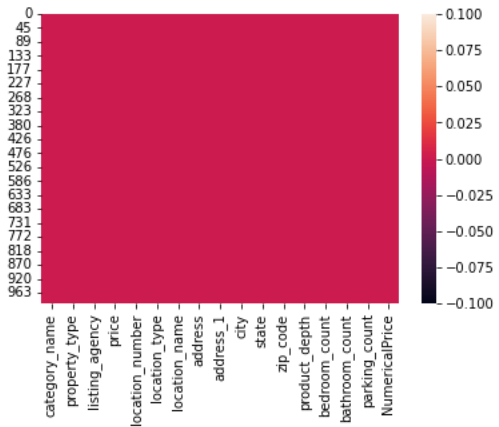
```
<AxesSubplot:>
```




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

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

```
<AxesSubplot:>
```




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

```
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
21     440000.0
22     640000.0
23     500000.0
24     305000.0
25     295000.0
26     795000.0
27     950000.0
28     485000.0
29     250000.0
30     549000.0
31     299000.0
32     395000.0
```

```
34     475000.0
37     465000.0
38     600000.0
39     980000.0
40     105000.0
42     450000.0
43     749000.0
45     289000.0
46     490000.0
48     565000.0
49     399000.0
50     469000.0
51     649000.0
52     499000.0
53     400000.0
54     800000.0
56     299000.0
57     580000.0
58     375000.0
59     489000.0
60     195000.0
61     505000.0
62     325000.0
63     539000.0
64     180000.0
66     145000.0
67     215000.0
68     450000.0
```

Ok, now that we have Cleaned our Data, let's move on with our Project.

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

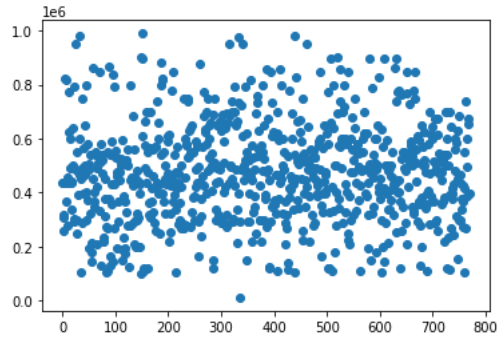
```
0      435000.0
1      320000.0
2      310000.0
3      259000.0
4      439000.0
5      825000.0
6      820000.0
7      369000.0
8      439000.0
9      455000.0
10     280000.0
12     439000.0
13     775000.0
14     625000.0
15     599000.0
16     490000.0
17     337500.0
19     439000.0
21     440000.0
22     640000.0
23     500000.0
24     305000.0
25     295000.0
26     795000.0
27     950000.0
28     485000.0
29     250000.0
30     549000.0
31     299000.0
32     395000.0
34     475000.0
37     465000.0
38     600000.0
39     980000.0
40     105000.0
42     450000.0
43     749000.0
45     289000.0
46     490000.0
48     565000.0
49     399000.0
50     469000.0
51     649000.0
52     499000.0
53     400000.0
54     800000.0
56     299000.0
57     580000.0
58     375000.0
59     489000.0
```

```
60     195000.0
61     505000.0
62     325000.0
63     539000.0
64     180000.0
66     145000.0
67     215000.0
68     450000.0
```

Ok, Now, Let's finsh our EDA and build the Final Model.

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

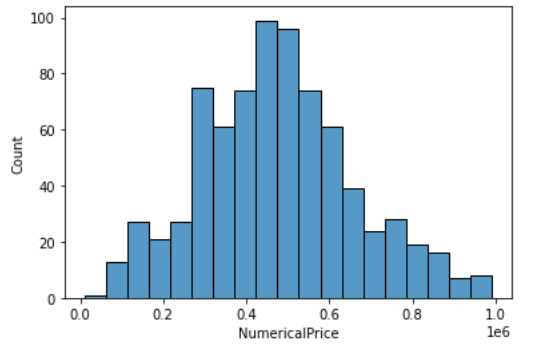
```
<matplotlib.collections.PathCollection at 0x7fadc859b350>
```



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

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

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




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

```
df.columns
```

```
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')
```

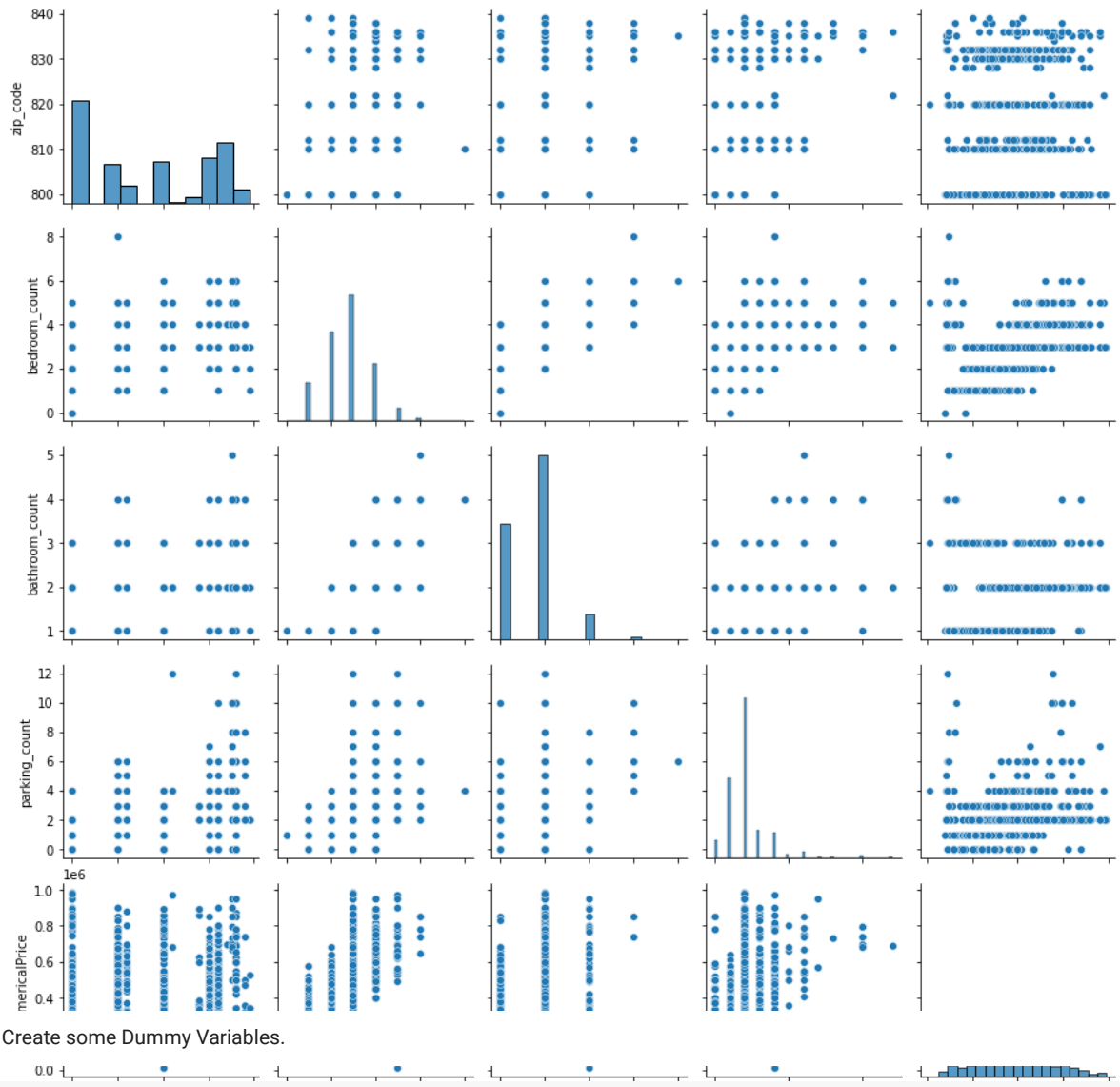
Let's now Build a New DataFrame, containing all the Columns that we will use for our Machine Learning.

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

Let's Check for some Correlations

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

```
<seaborn.axisgrid.PairGrid at 0x7fadc8544d90>
```



Let's Create some Dummy Variables.

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

```
newdf.head()
```

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

Now that we Have Cleaned and Prepared our Data, let's move on with Modeling.

The Steps for Building Our Linear Regression Model Are:

- Create the X Features Variable and Y the Predicted Variable
- Split the DataSet into a TrainSet and a TestSet
- Fit the Model
- Use the Model to make Predictions
- Calculate Errors

- Evaluate the Model

```
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"]
```

```
xTrain, xTest, yTrain, yTest = TTS(x, y, test_size = 0.3)
```

```
LR_ = LR()
```

```
LR_.fit(xTrain, yTrain)
```

```
LinearRegression()
```

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

```
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
```

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

```
print(mae)
print(mse)
print(rmse)
print(r2)
```

```
109072.1537758192
25133039454.05762
158534.03247901576
-1.1835998141164925
```

As we've Guessed Earlier, by Looking at the Errors, we can conclude that the Linear Regression is not the best Model to fit our Data. Since we have Non-Linear Data, but we Still need to Address a Regression Problem, we will Try out 2 more Models:

- K Nearest Neighbors Regressor
- Decision Tree Regressor

The Steps for Building Our Models are pretty much the same with some Variations:

- Create the X Features Variable and Y the Predicted Variable
- Split the DataSet into a TrainSet and a TestSet
- Search for the Best Parameters with the GridSearchCV Method
- Fit the Model
- Use the Model to make Predictions
- Calculate Errors
- Cross Validate the Model
- Evaluate the Model

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

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

```
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)
```

```
'''
#Evaluate KNN Neighbors with Elbow Method

Scores = []

for k in range(1, 11):
    K = KNR(algorithm = "auto", n_neighbors = k, p = 1)
    K.fit(xTrain, yTrain)
    Preds = K.predict(xTest)
    Scores.append(MSE(yTest, Preds, squared = False))

print(Scores)

plt.plot(Scores)
'''
```

```
'\n#Evaluate KNN Neighbors with Elbow Method\n\nScores = []\n\nfor k in range(1, 11):\n    K = KNR(algorithm = "auto", n_neighbors = k, p = 1)\n    K.fit(xTrain, yTrain)\n    Preds = K.predict(xTest)\n    Scores.append(MSE(yTest, Preds, squared = False))\n\nprint(Scores)\n\nplt.plot(Scores)\n'
```

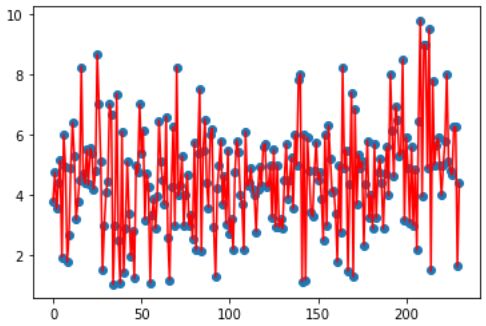
```
Parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNR()

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

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

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

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

```
[<matplotlib.lines.Line2D at 0x7fadb951df50>]
```




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

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

```
Errors
```

	KNR
MAE	0.002485
MSE	0.000086
RMSE	0.009249
R2	0.999972

```
from sklearn.model_selection import cross_val_score as CrossVal
```