

### Australian Housing Prices prediction

This dataset can be used to predict housing prices in Australia. This dataset can be used to find relationships between housing prices and location. This dataset can be used to find relationships between housing prices and features such as size, number of bedrooms, and number of bathrooms

Hint: RealEstateAU\_1000\_Samples.csv file

### Instructions:

1. Use Lifecycle of Data Science
2. Use necessary data Preprocess techniques
3. Use various Regression and Classification techniques for comparison
4. Use metrics for regression and classification when needed.
5. Use various Pipeline/Hyperparameter tuning techniques for improving performance

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

```
%matplotlib inline
```

```
df = pd.read_csv("/content/RealEstateAU_1000_Samples.csv")
df
```

	index	TID	breadcrumb	category_name	property_type	building_size	land_size	preferred_si
0	0	1350988	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	Na
1	1	1350989	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m <sup>2</sup>	NaN	171r
2	2	1350990	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	NaN	Na

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')
```

df.drop(["index", "TID", "breadcrumb", "open\_date", "phone", "RunDate"], axis = 1, inplace = True)

df.head()

	category_name	property_type	building_size	land_size	preferred_size	listing_agency	price	location_
0	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Professionals - DARWIN CITY	\$435,000	135
1	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m <sup>2</sup>	NaN	171m <sup>2</sup>	Nick Mousellis Real Estate - Eview Group Member	Offers Over \$320,000	139
2	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	NaN	NaN	Habitat Real Estate - THE GARDENS	\$310,000	139
3	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Ray White - NIGHTCLIFF	\$259,000	135
4	Real Estate & Property for sale in DARWIN CITY...	Unit	201m <sup>2</sup>	NaN	201m <sup>2</sup>	Carol Need Real Estate - Fannie Bay	\$439,000	139

5 rows x 21 columns

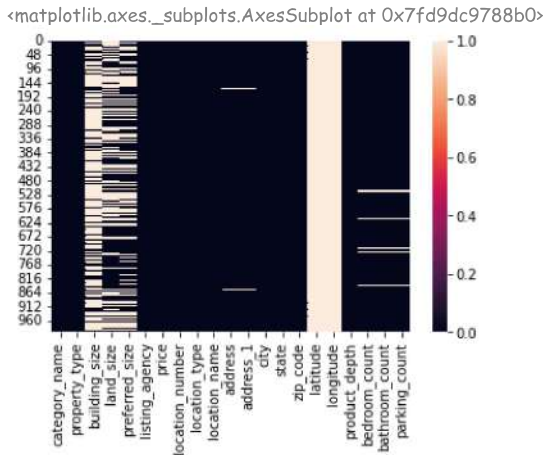


null = df.isnull().sum()  
null

```
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
```

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

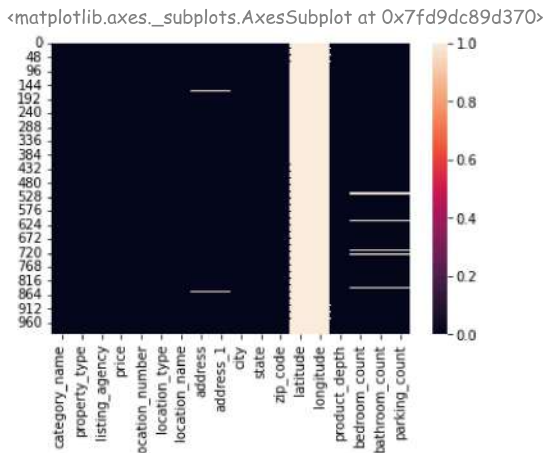


The "Building Size", "Land Size" and "Preferred Size" are sadly columns to drop, since there are too many missing values.

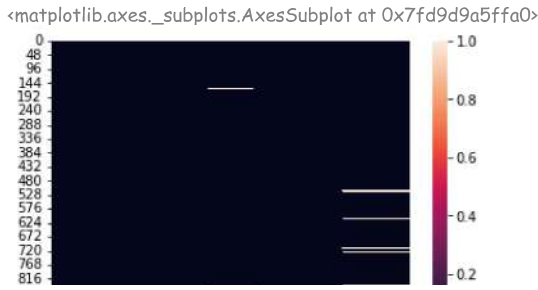
Alternatively, we could create a new DataFrame to be analyzed with Only the Rows that are not Null for the 3 Columns mentioned, but that would reduce the size of our samples, which is already small.

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

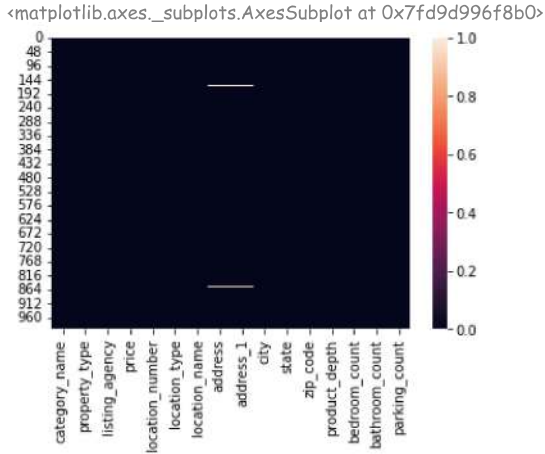
```
sns.heatmap(df.isnull())
```



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



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



```
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
```

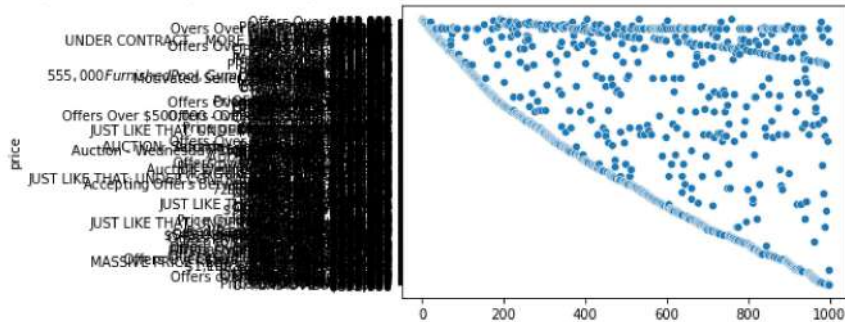
```
df["city"].value_counts()
```

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"])
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd9d988b610>



```
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))

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

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

960	Offers over \$419,000	4.190000e+05
961	Offers over \$380,000	3.800000e+05
962	\$430,000	4.300000e+05
963	\$244,000	2.440000e+05
964	\$599,000	5.990000e+05
965	Offers Over \$480,000	4.800000e+05
966	\$560,000	5.600000e+05
967	\$315,000	3.150000e+05
968	\$585,000 & \$595,000	5.850006e+11
969	\$1,550,000	1.550000e+06
970	Offers over \$450,000	4.500000e+05
971	Under Offer	NaN
972	Offers Over \$630,000	6.300000e+05
973	Price Guide Mid \$300,000	3.000000e+05
974	\$685,000	6.850000e+05
975	UNDER CONTRACT	NaN
976	\$469,000	4.690000e+05
977	Negotiable	NaN
978	\$439,000	4.390000e+05
979	\$299,000 Negotiable	2.990000e+05
980	\$380,000	3.800000e+05
981	Contact Agent	NaN
982	\$339,000	3.390000e+05
983	\$275,000	2.750000e+05
984	\$425,000	4.250000e+05
985	Offers over \$370,000	3.700000e+05
986	OFFERS OVER \$415,000	4.150000e+05
987	\$550,000	5.500000e+05
988	PRICE GUIDE \$1,070,000	1.070000e+06
989	Offers over \$740,000	7.400000e+05
990	UNDER CONTRACT	NaN
991	\$270,000	2.700000e+05
992	\$390,000	3.900000e+05
993	\$495,000	4.950000e+05
994	UNDER CONTRACT	NaN
995	2 Residence	2.000000e+00
996	\$601,000	6.010000e+05
997	\$655,000	6.550000e+05
998	\$675,000	6.750000e+05
999	\$399,000	3.990000e+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

<ipython-input-21-e1de7eabf4fa>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

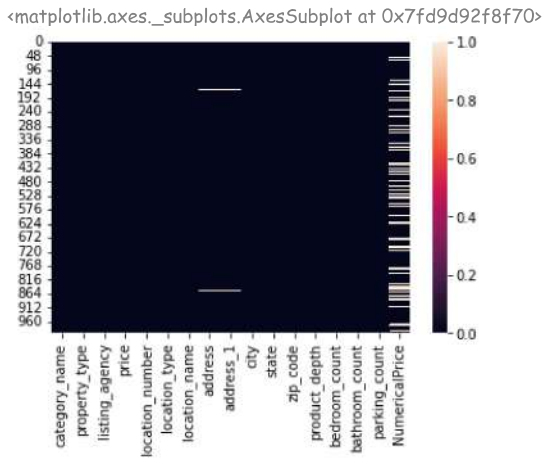
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy.
df["NumericalPrice"][c] = x

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



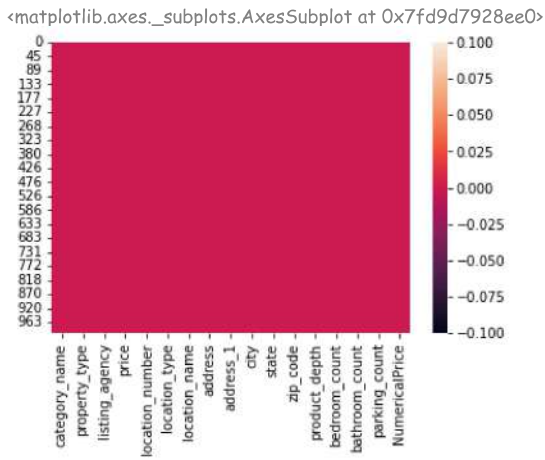
960	Offers over \$419,000	419000.0
961	Offers over \$380,000	380000.0
962	\$430,000	430000.0
963	\$244,000	244000.0
964	\$599,000	599000.0
965	Offers Over \$480,000	480000.0
966	\$560,000	560000.0
967	\$315,000	315000.0
968	\$585,000 & \$595,000	585000.0
969	\$1,550,000	155000.0
970	Offers over \$450,000	450000.0
971	Under Offer	NaN
972	Offers Over \$630,000	630000.0
973	Price Guide Mid \$300,000	300000.0
974	\$685,000	685000.0
975	UNDER CONTRACT	NaN
976	\$469,000	469000.0
977	Negotiable	NaN
978	\$439,000	439000.0
979	\$299,000 Negotiable	299000.0
980	\$380,000	380000.0
981	Contact Agent	NaN
982	\$339,000	339000.0
983	\$275,000	275000.0
984	\$425,000	425000.0
985	Offers over \$370,000	370000.0
986	OFFERS OVER \$415,000	415000.0
987	\$550,000	550000.0
988	PRICE GUIDE \$1,070,000	107000.0
989	Offers over \$740,000	740000.0
990	UNDER CONTRACT	NaN
991	\$270,000	270000.0
992	\$390,000	390000.0
993	\$495,000	495000.0
994	UNDER CONTRACT	NaN
995	2 Residence	NaN
996	\$601,000	601000.0
997	\$655,000	655000.0
998	\$675,000	675000.0
999	\$399,000	399000.0

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



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

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



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

```
966 560000.0
967 315000.0
968 585000.0
969 155000.0
970 450000.0
972 630000.0
973 300000.0
974 685000.0
976 469000.0
978 439000.0
979 299000.0
980 380000.0
982 339000.0
983 275000.0
984 425000.0
985 370000.0
986 415000.0
987 550000.0
988 107000.0
989 740000.0
991 270000.0
992 390000.0
993 495000.0
996 601000.0
997 655000.0
998 675000.0
999 399000.0
Name: NumericalPrice, dtype: float64
```

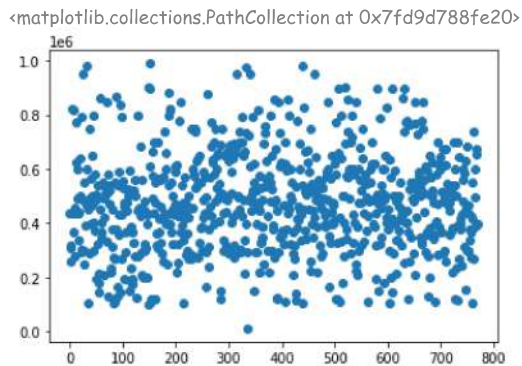
```
df["NumericalPrice"].astype("float")
```

```

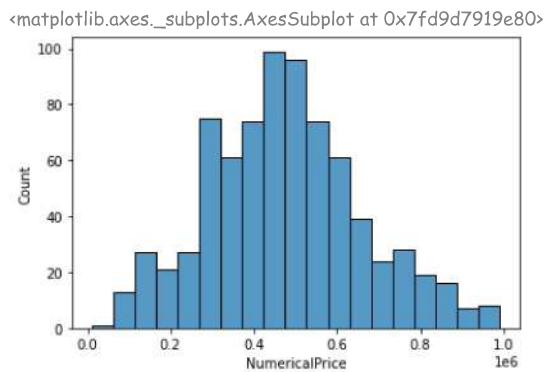
996 601000.0
997 655000.0
998 675000.0
999 399000.0
Name: NumericalPrice, dtype: float64

```

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



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



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

```
df.columns
```

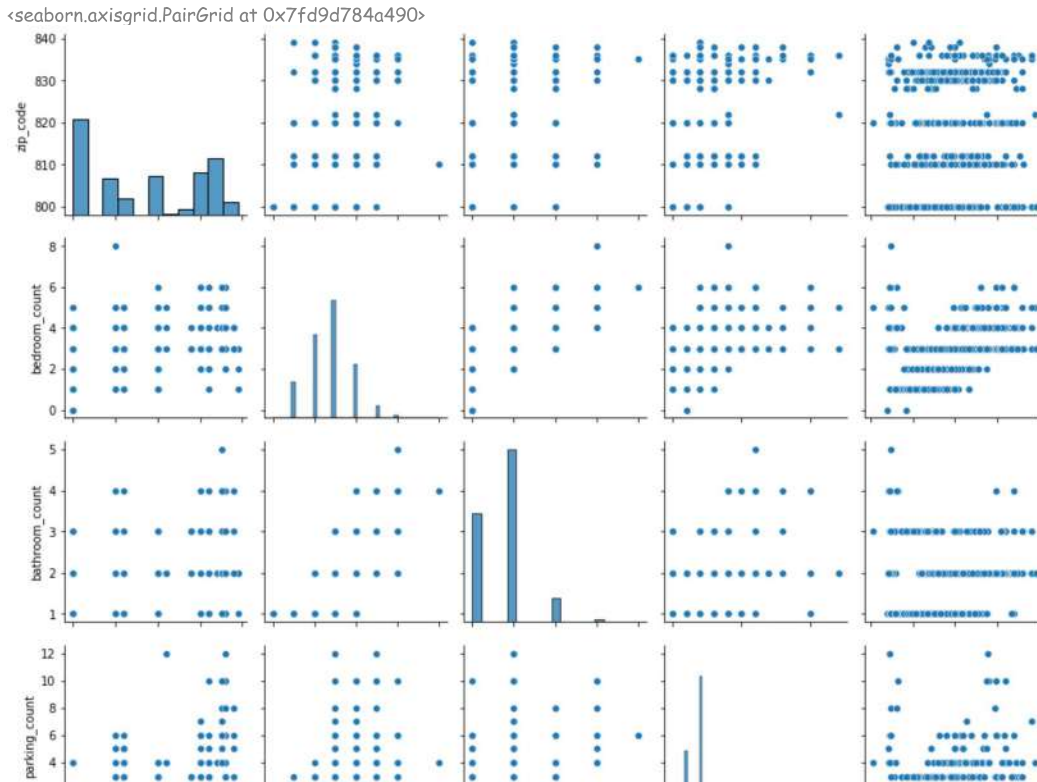
```

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')

```

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

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



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

```
newdf.head()
```

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

```
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)

110480.09710388833
25126686160.28905
158513.99357876595
-1.9465748442855029

```

As we've Gussed 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:

- 1) K Nearest Neighbors Regressor
- 2) Decision Tree Regressor

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

- 1) Create the X Features Variable and Y the Predicted Variable
- 2) Split the DataSet into a TrainSet and a TestSet
- 3) Search for the Best Parameters with the GridSearchCV Method
- 4) Fit the Model
- 5) Use the Model to make Predictions
- 6) Calculate Errors
- 7) Cross Validate the Model
- 8) 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)

```

```
'\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, yTr
```

```

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