In [1]:

```
import pandas as pd
```

In [2]:

```
import numpy as np
```

In [3]:

```
df=pd.read_csv("C:\\Users\\cozze\\OneDrive\\Desktop\\nyc_rolling_sales.csv")
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84548 entries, 0 to 84547
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   Unnamed: 0                      84548 non-null  int64 
 1   BOROUGH                         84548 non-null  int64 
 2   NEIGHBORHOOD                    84548 non-null  object
 3   BUILDING CLASS CATEGORY         84548 non-null  object
 4   TAX CLASS AT PRESENT            84548 non-null  object
 5   BLOCK                           84548 non-null  int64 
 6   LOT                             84548 non-null  int64 
 7   EASE-MENT                       84548 non-null  object
 8   BUILDING CLASS AT PRESENT       84548 non-null  object
 9   ADDRESS                         84548 non-null  object
 10  APARTMENT NUMBER                84548 non-null  object
 11  ZIP CODE                        84548 non-null  int64 
 12  RESIDENTIAL UNITS               84548 non-null  int64 
 13  COMMERCIAL UNITS                84548 non-null  int64 
 14  TOTAL UNITS                     84548 non-null  int64 
 15  LAND SQUARE FEET                84548 non-null  object
 16  GROSS SQUARE FEET               84548 non-null  object
 17  YEAR BUILT                      84548 non-null  int64 
 18  TAX CLASS AT TIME OF SALE       84548 non-null  int64 
 19  BUILDING CLASS AT TIME OF SALE  84548 non-null  object
 20  SALE PRICE                      84548 non-null  object
 21  SALE DATE                       84548 non-null  object
dtypes: int64(10), object(12)
memory usage: 14.2+ MB
```

In [5]:

```
#Check for duplicated data
```

In [6]:
```
df.duplicated()
```

Out[6]:
```
0        False
1        False
2        False
3        False
4        False
         ...  
84543    False
84544    False
84545    False
84546    False
84547    False
Length: 84548, dtype: bool
```

In [7]:
```
print(df.duplicated().value_counts())
```

```
False    84548
dtype: int64
```

In [8]:
```
df=df.drop_duplicates()
```

In [9]:
```
print(df.duplicated().value_counts())
```

```
False    84548
dtype: int64
```

In [10]:
```
#Check for missing data
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
Unnamed: 0                        0
BOROUGH                           0
NEIGHBORHOOD                      0
BUILDING CLASS CATEGORY           0
TAX CLASS AT PRESENT              0
BLOCK                             0
LOT                               0
EASE-MENT                         0
BUILDING CLASS AT PRESENT         0
ADDRESS                           0
APARTMENT NUMBER                  0
ZIP CODE                          0
RESIDENTIAL UNITS                 0
COMMERCIAL UNITS                  0
TOTAL UNITS                       0
LAND SQUARE FEET                  0
GROSS SQUARE FEET                 0
YEAR BUILT                        0
TAX CLASS AT TIME OF SALE         0
BUILDING CLASS AT TIME OF SALE    0
SALE PRICE                        0
SALE DATE                         0
dtype: int64
```

In [12]:

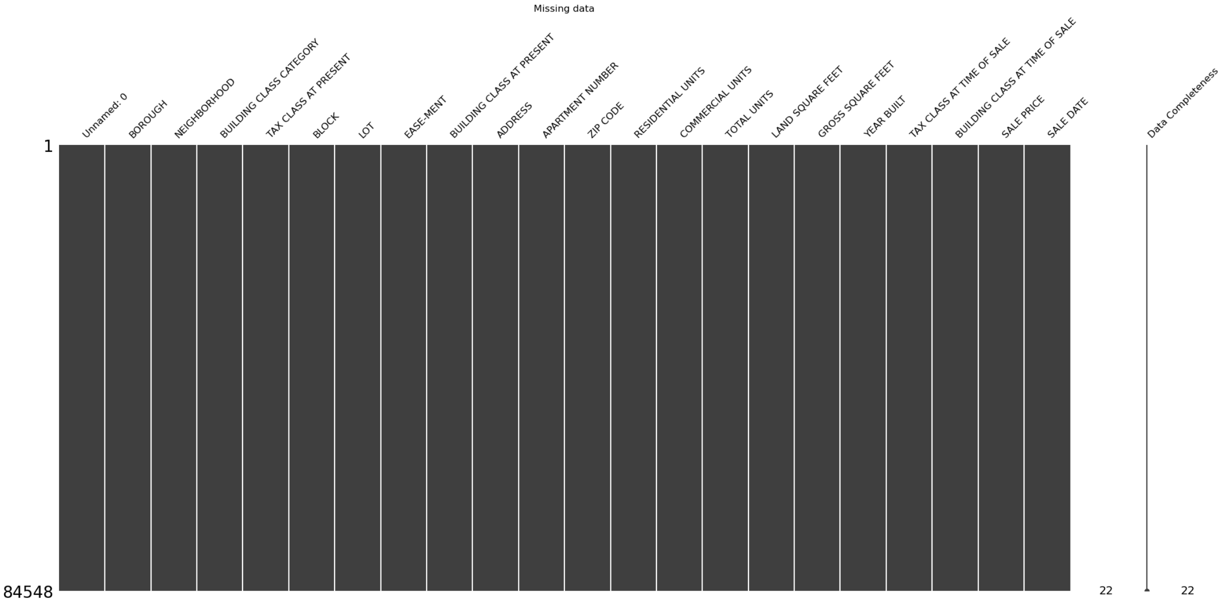
```
import missingno as msno
```

In [13]:

```
import matplotlib.pyplot as plt
```

In [14]:

```
msno.matrix(df, fontsize=12, labels=True)
plt.title('Missing data')
plt.show()
```



In [15]:

```
#Convert variables to a proper format and drop hidden null values among quant
```

In [16]:

```
df['LAND SQUARE FEET'] = pd.to_numeric(df['LAND SQUARE FEET'],
                                       errors='coerce')
```

In [17]:

```
df = df.dropna(subset=['LAND SQUARE FEET'])
```

In [18]:

```
df['GROSS SQUARE FEET'] = pd.to_numeric(df['GROSS SQUARE FEET'],
                                        errors='coerce')
```

In [19]:

```
df = df.dropna(subset=['GROSS SQUARE FEET'])
```

In [20]:

```
df['SALE PRICE'] = pd.to_numeric(df['SALE PRICE'], errors='coerce')
```

In [21]:

```
df = df.dropna(subset=['SALE PRICE'])
```

In [ ]:

```

```

In [22]:
```
df.head()
```

Out[22]:

	Unnamed: 0	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDI CLA PRESE
0	4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6		
3	7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21		
4	8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55		
6	10	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	406	32		
9	13	1	ALPHABET CITY	08 RENTALS - ELEVATOR APARTMENTS	2	387	153		

5 rows × 22 columns

In [ ]:
```

```

In [ ]:
```

```

In [23]:
```
####################################################################
```

In [24]:
```
#Fix variables so that they're in integer format
```

In [25]:
```
df.BOROUGH.unique()
```

Out[25]:
```
array([1, 2, 3, 4, 5], dtype=int64)
```

In [26]:

```
df['RESIDENTIAL UNITS'].unique()
```

Out[26]:

```
array([   5,   10,    6,    8,   24,    3,    4,    0,    1,    2,   22,
          9,   15,   30,   35,   11,   28,    7,   18,   12,   16,   20,
         21,   19,   48,  529,  476,  317,   14,   42,  107,   31,   36,
         34,   33,   74,   29,   23,   25,  286,  256,  771,  257,   38,
        311,   41,   51,   76,   45,   72,   56,   68,   37,   50,   26,
         17,   61,   60,  894,   67,  184,   78,  181,   13,  102,  121,
         52,   27,   70,  369,   77,   40,  233,   91,   32,  109,  387,
        153,  193,   62,  146,   94,   46,   44,   84,   75,   65,   95,
         49,   63,   64,  100,   54,   43,  716,  680,   47,  179,   71,
         39,   53,   55,  283,   66,  114,   59,   89,   73,   79,   83,
        498,   81,  162,  127,  222,   99,  130,   90,   58,  159,  462,
        142,  948,  129,  889,  271,  150,  120,   57,  117,  106,   85,
        103,  118,  164,  139,  165,  122,  113,  134,  278,  135,  324,
        180,   96,  144,  152,   88,  136,  291, 1844,  198,  148,  446,
        335,  143,  128], dtype=int64)
```

In [27]:

```
df['RESIDENTIAL UNITS_numeric']=df['RESIDENTIAL UNITS']
```

In [28]:

```
dict_res={'RESIDENTIAL UNITS_numeric':{'5.':'5','10.':'10','6.':'6',
                                       '8.':'8',
                                       '1.':'1','3.':'3',
                                       '4.':'4','0.':'0','2.':'2',
                                       '9.':'9','15.':'15','11.':'11',
                                       '7.':'7',
                                       '18.':'18','12.':'12',
                                       '16.':'16','20.':'20',
                                      '19.':'19','14.':'14','17.':'17',
                                       '13.':'13'}}
```

In [29]:

```
df.replace(dict_res, inplace=True)
```

In [30]:

```
df['COMMERCIAL UNITS'].unique()
```

Out[30]:

```
array([   0,    1,    2,    3,   13,    5,    4,   19,   10,   14,    8,
          6,   35,   55,   17,   12,   15,    9,   23,   52,  318,   11,
        254,    7,   26,   59,   62,   42,   32,   20,   22,   28,  147,
        184,   25,  172,  436,   16, 2261,   51,   18,   21,  126],
      dtype=int64)
```

In [31]:

```
df['COMMERICAL UNITS_numeric']=df['COMMERCIAL UNITS']
```

In [32]:
```
dict_comm={'COMMERCIAL UNITS_numeric':{'0.':'0','1.':'1','2.':'2',
                                       '3.':'3',
                                       '13.':'13','5.':'5',
                                       '4.':'4','19.':'19',
                                       '10.':'10',
                                       '14.':'14','8.':'8','6.':'6',
                                       '17.':'17','12.':'12',
                                       '15.':'15','9.':'9','23.':'23',
                                      '11.':'11','7.':'7','26.':'26',
                                       '20.':'20','22.':'22',
                                       '28.':'28','25.':'25',
                                       '16.':'16',
                                     '18.':'18','21.':'21'}}
```

In [33]:
```
df.replace(dict_comm, inplace=True)
```

In [34]:
```
df = df.dropna(subset=['GROSS SQUARE FEET'])
```

In [35]:
```
df['GROSS SQUARE FEET'] = df['GROSS SQUARE FEET'].replace(' - ', np.nan)  #
```

In [36]:
```
mean_value = df['GROSS SQUARE FEET'].mean()
```

In [37]:
```
df['LAND SQUARE FEET'].unique()
```

Out[37]:
```
array([  1633.,   2272.,   2369., ...,  11088., 208033.,  10796.])
```

In [38]:
```
df['LAND SQUARE FEET'] = df['LAND SQUARE FEET'].astype(int)
```

In [39]:
```
df['GROSS SQUARE FEET'].unique()
```

Out[39]:
```
array([ 6440.,  6794.,  4615., ...,   977.,  2683., 64117.])
```

In [40]:
```
df['GROSS SQUARE FEET'] = df['GROSS SQUARE FEET'].astype(int)
```

In [41]:
```
#Ordinal Encoding
```

In [42]:
```
df['TAX CLASS AT PRESENT'].unique()
```

Out[42]:
```
array(['2A', '2B', '2', '4', '1', '2C', '1A', '1B', '3', ' ', '1C'],
      dtype=object)
```

In [43]:
```
df['TAX_numeric']=df['TAX CLASS AT PRESENT']
```

In [44]:
```
dict_tax={'TAX_numeric':{' ':0, '1':1, '1A':2,
                         '1B':3,'1C':4,'2':5,'2A':6,
                      '2B':7,'2C':8,'3':9,'4':10}}
```

In [45]:
```
df.replace(dict_tax, inplace=True)
```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [46]:

```python
#Check for outliers
```

In [47]:

```python
import seaborn
```

In [48]:

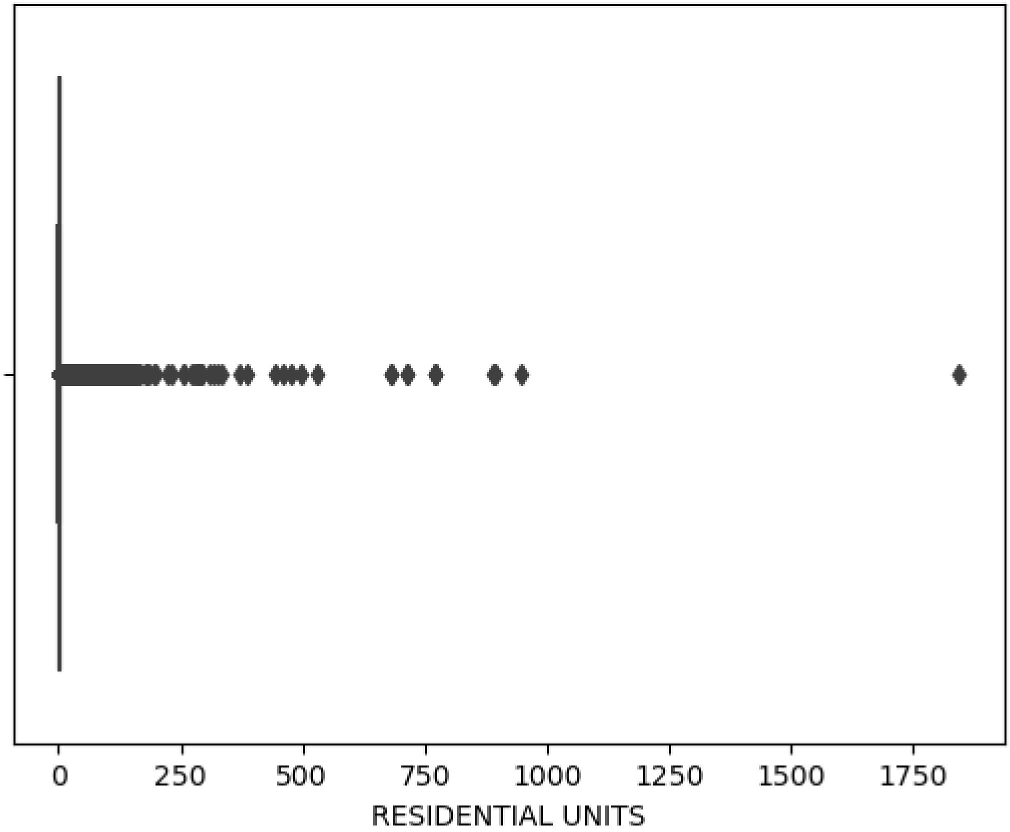
```python
from pandas import DataFrame
```

In [49]:

```python
import scipy.stats as stats
```

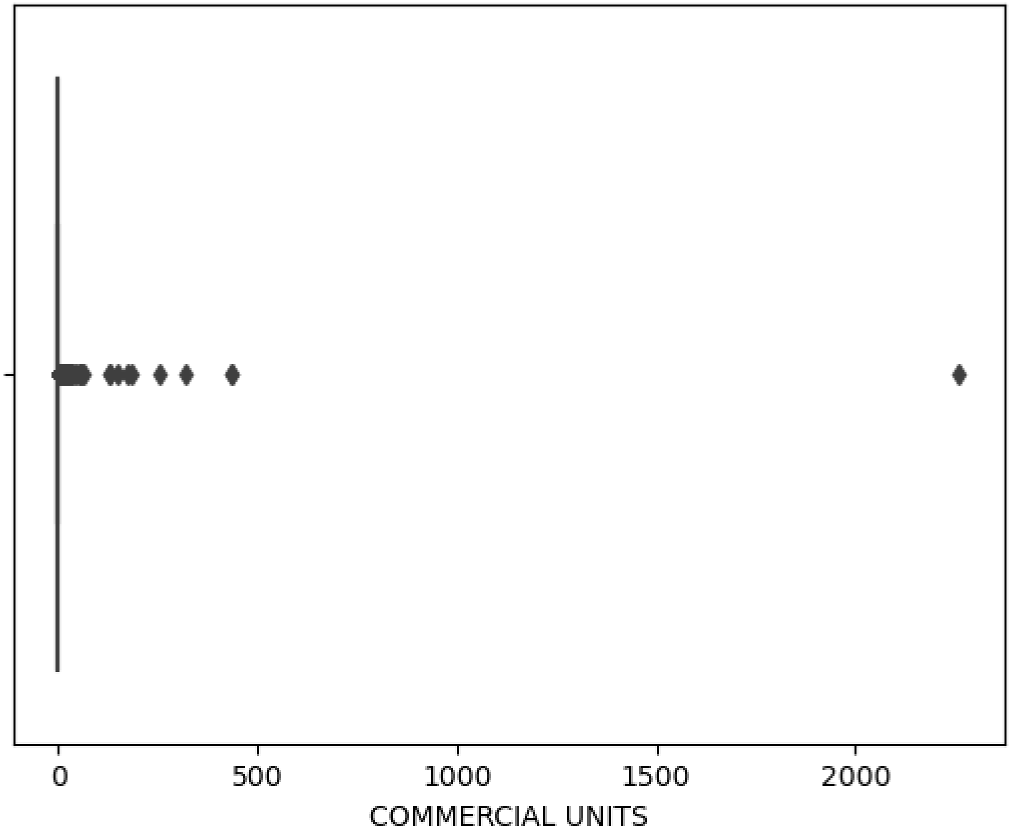
In [50]:

```python
boxplot1=seaborn.boxplot(x='RESIDENTIAL UNITS', data=df)
```



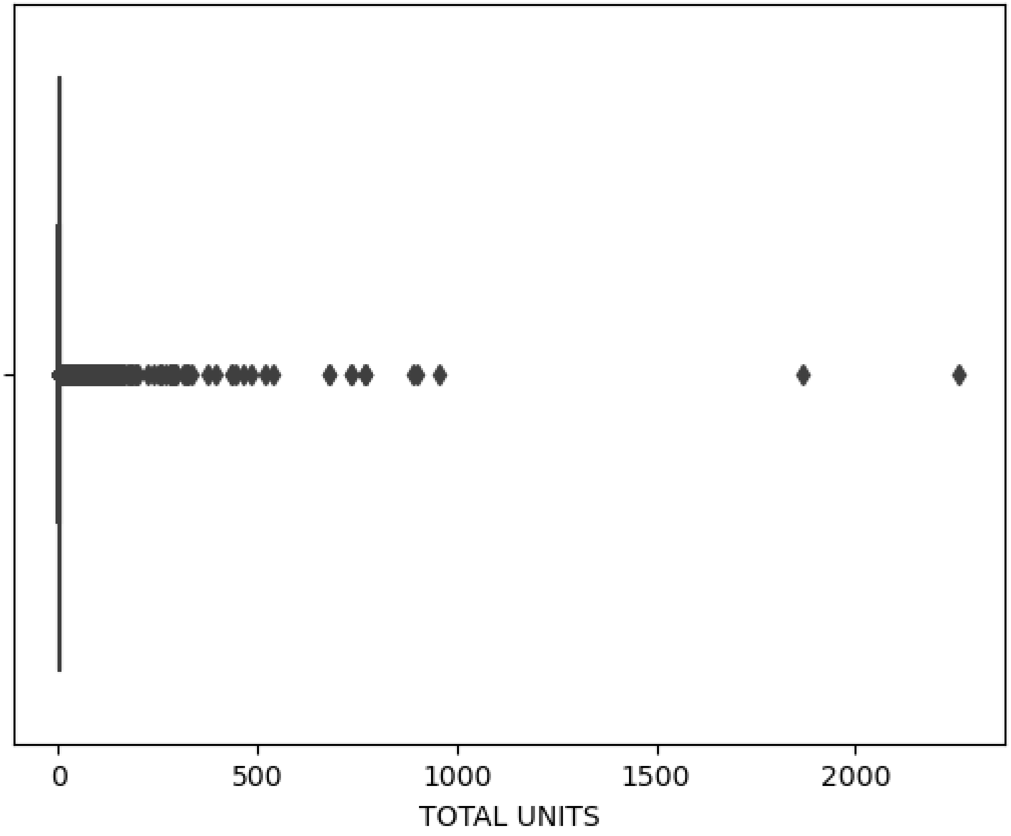
In [51]:

```
boxplot2=seaborn.boxplot(x='COMMERCIAL UNITS', data=df)
```



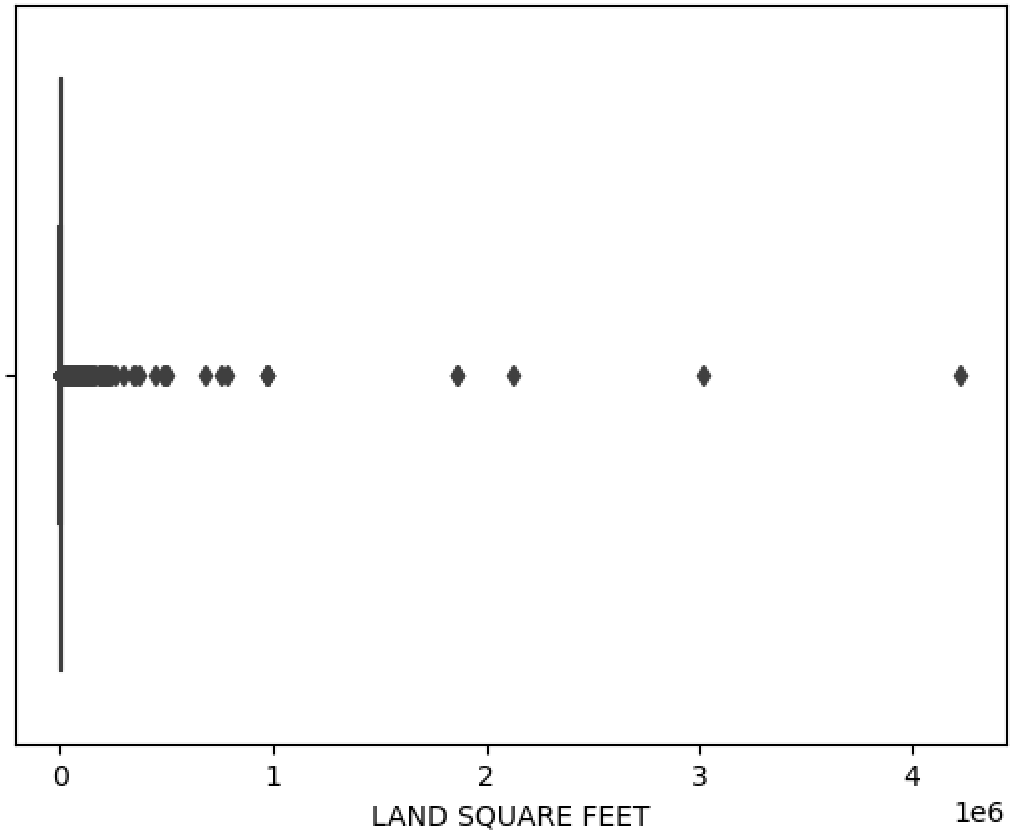
In [52]:

```
boxplot3=seaborn.boxplot(x='TOTAL UNITS', data=df)
```



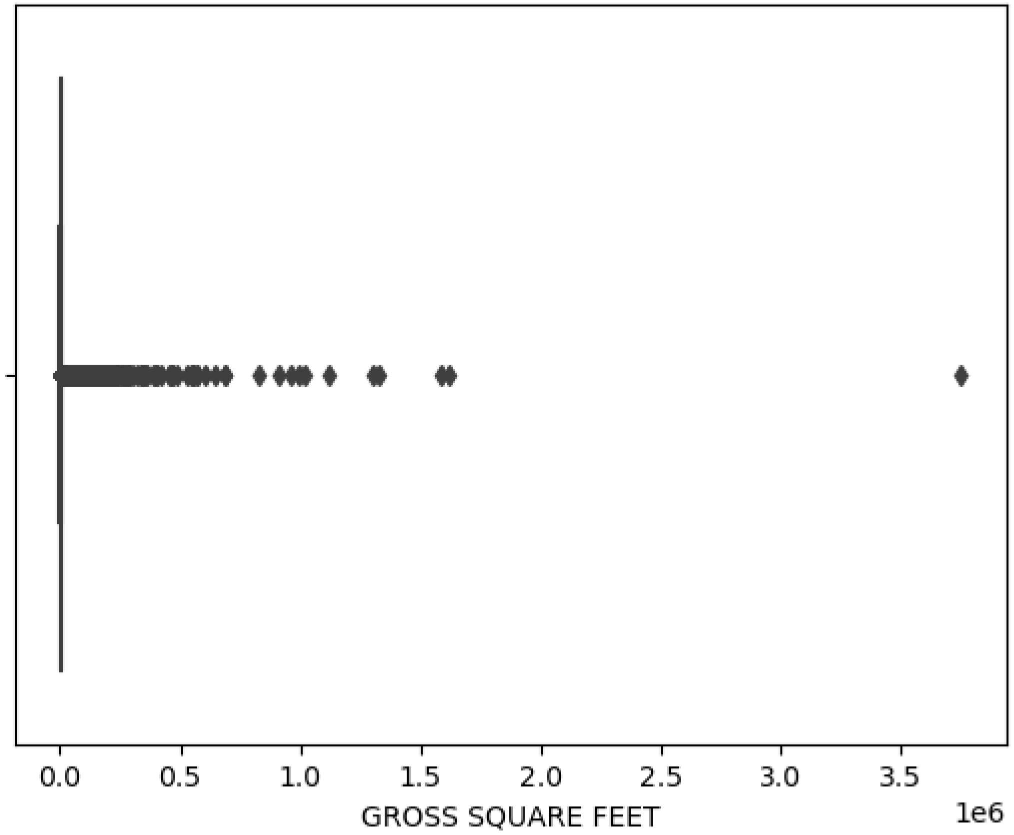
In [53]:

```
boxplot4=seaborn.boxplot(x='LAND SQUARE FEET',data=df)
```



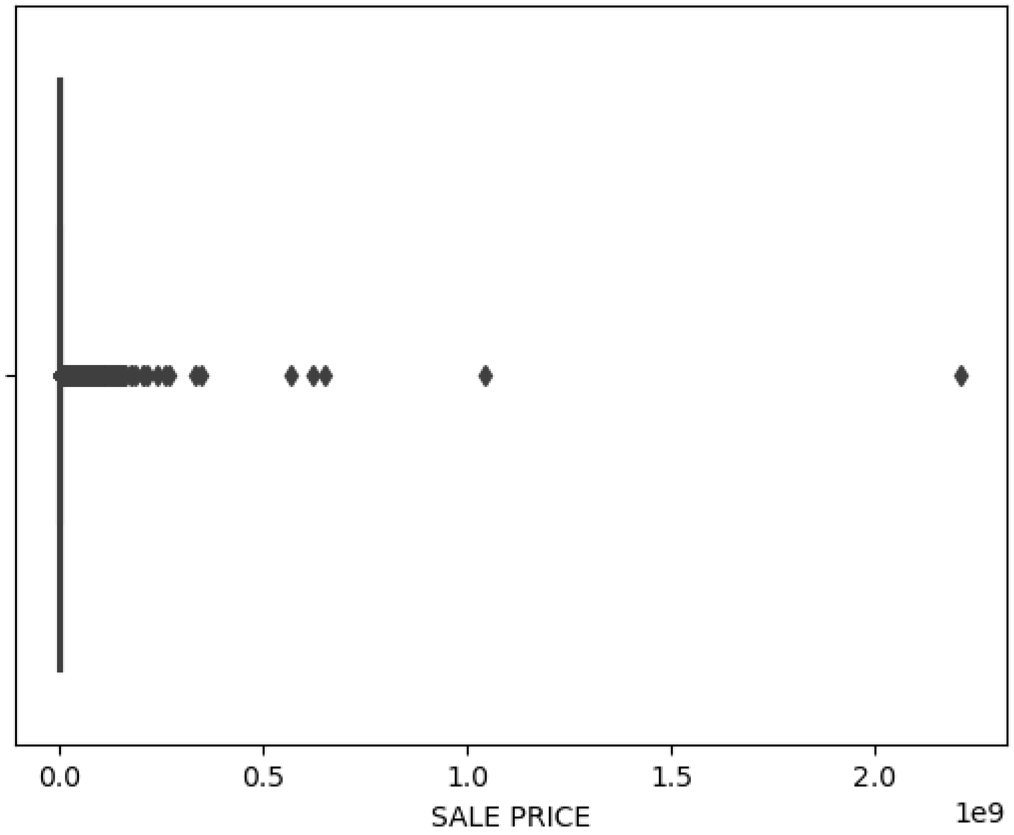
In [54]:

```
boxplot5=seaborn.boxplot(x='GROSS SQUARE FEET',data=df)
```



In [55]:

```
boxplot6=seaborn.boxplot(x='SALE PRICE', data=df)
```



In [ ]:

```

```

In [56]:

```
#Fix outliers
```

In [57]:

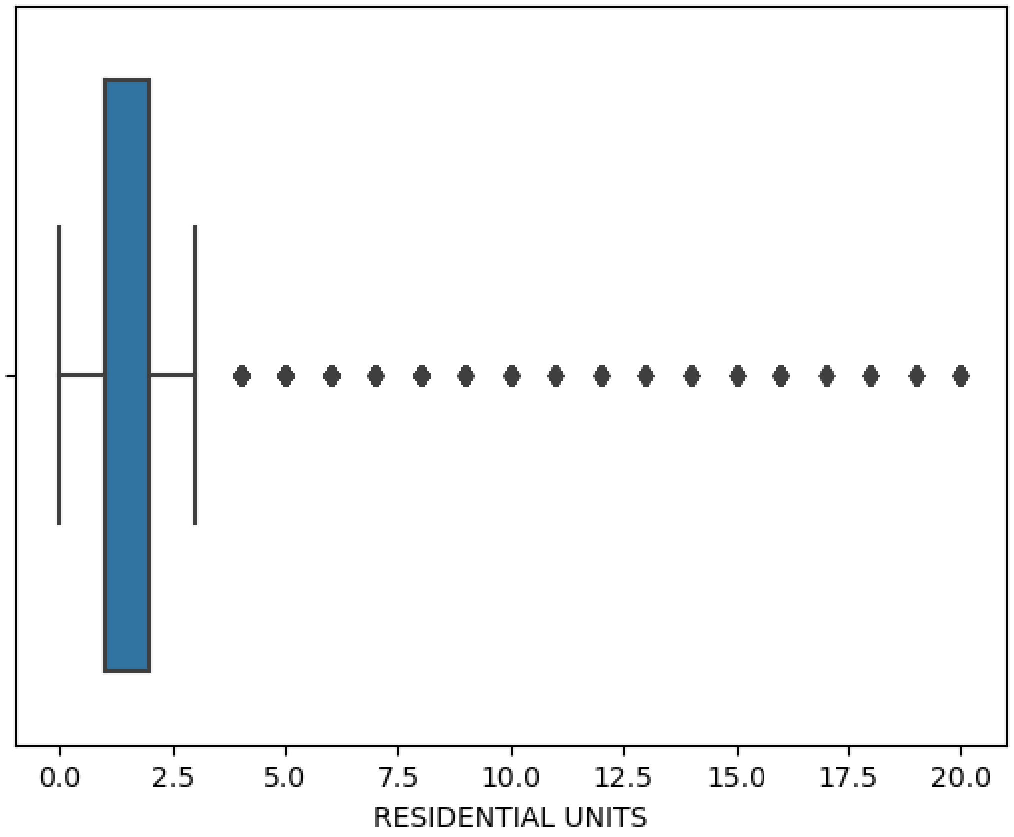
```
median=np.median(df['RESIDENTIAL UNITS'])
```

In [58]:

```
df['RESIDENTIAL UNITS']=np.where(df['RESIDENTIAL UNITS']>20,
                                 median, df['RESIDENTIAL UNITS'])
```

In [59]:

```
boxplot_n1=seaborn.boxplot(x='RESIDENTIAL UNITS',data=df)
```



In [60]:

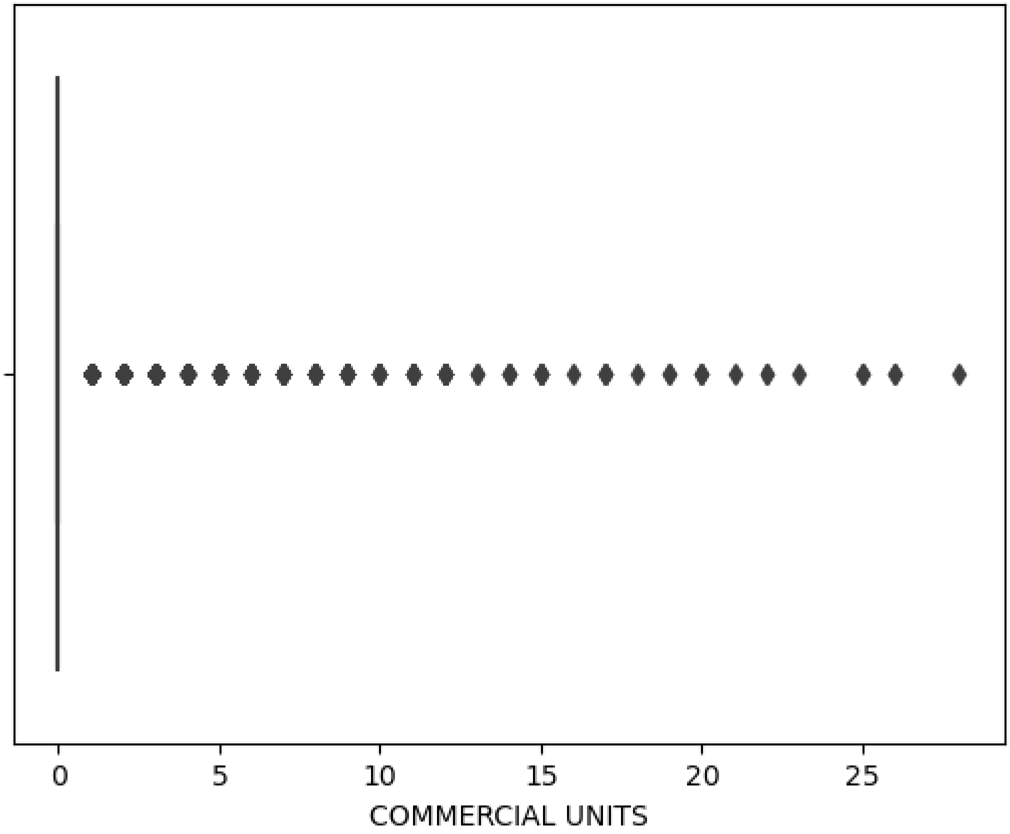
```
median2=np.median(df['COMMERCIAL UNITS'])
```

In [61]:

```
df['COMMERCIAL UNITS']=np.where(df['COMMERCIAL UNITS']>30,
                                 median2, df['COMMERCIAL UNITS'])
```

In [62]:

```
boxplot_n2=seaborn.boxplot(x='COMMERCIAL UNITS', data=df)
```



In [63]:

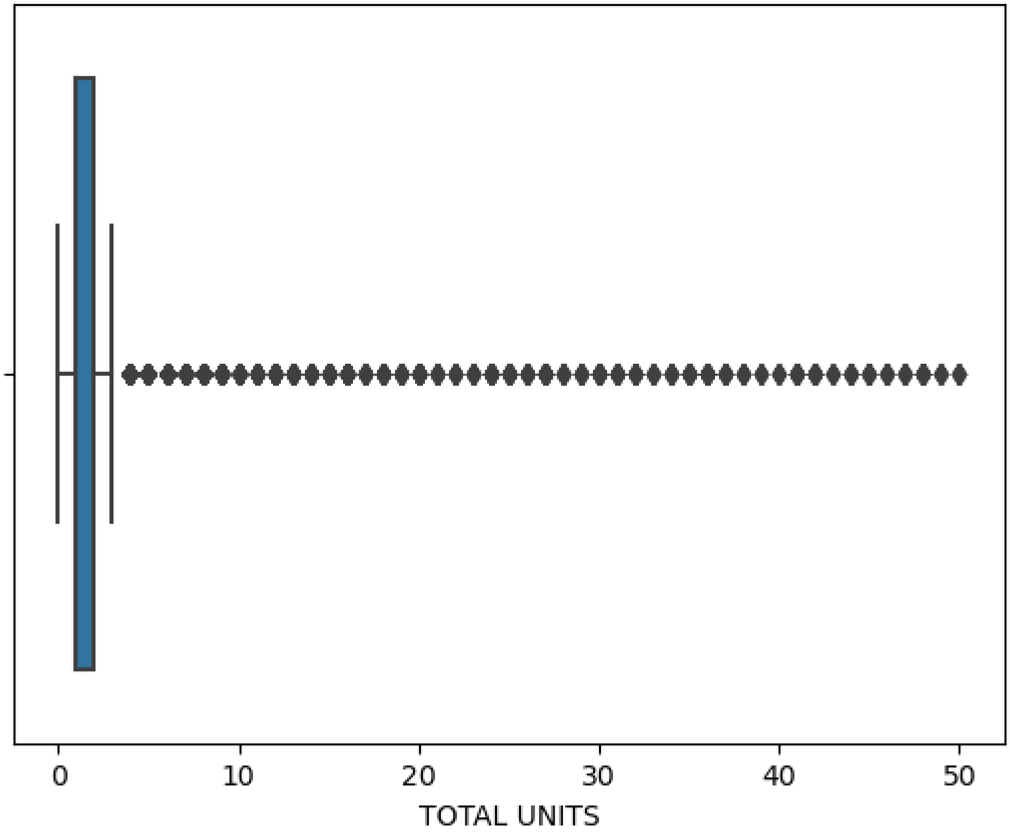
```
median3=np.median(df['TOTAL UNITS'])
```

In [64]:

```
df['TOTAL UNITS']=np.where(df['TOTAL UNITS']>50,
                           median3, df['TOTAL UNITS'])
```

In [65]:

```
boxplot_n3=seaborn.boxplot(x='TOTAL UNITS', data=df)
```



In [66]:

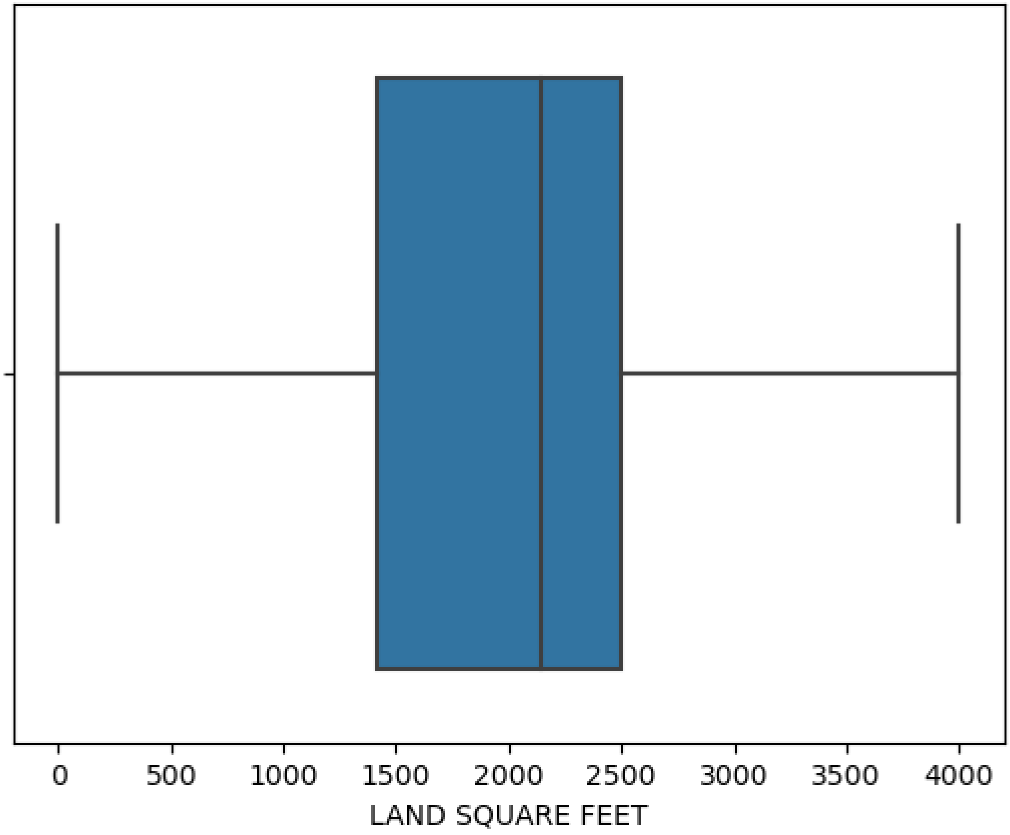
```
median4=np.median(df['LAND SQUARE FEET'])
```

In [67]:

```
df['LAND SQUARE FEET']=np.where(df['LAND SQUARE FEET']>4000,
                                   median4, df['LAND SQUARE FEET'])
```

In [68]:

```
boxplot_n4=seaborn.boxplot(x='LAND SQUARE FEET', data=df)
```



In [69]:

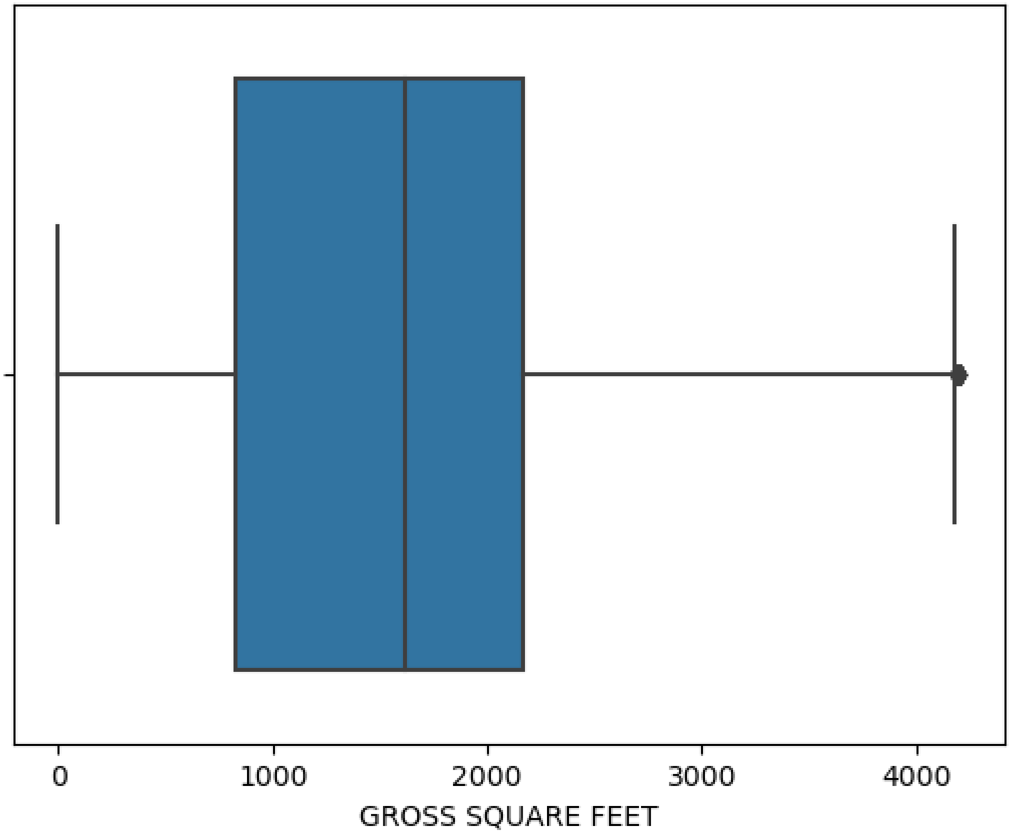
```
median5=np.median(df['GROSS SQUARE FEET'])
```

In [70]:

```
df['GROSS SQUARE FEET']=np.where(df['GROSS SQUARE FEET']>4200,
                            median5, df['GROSS SQUARE FEET'])
```

In [71]:

```python
boxplot_n5=seaborn.boxplot(x='GROSS SQUARE FEET', data=df)
```



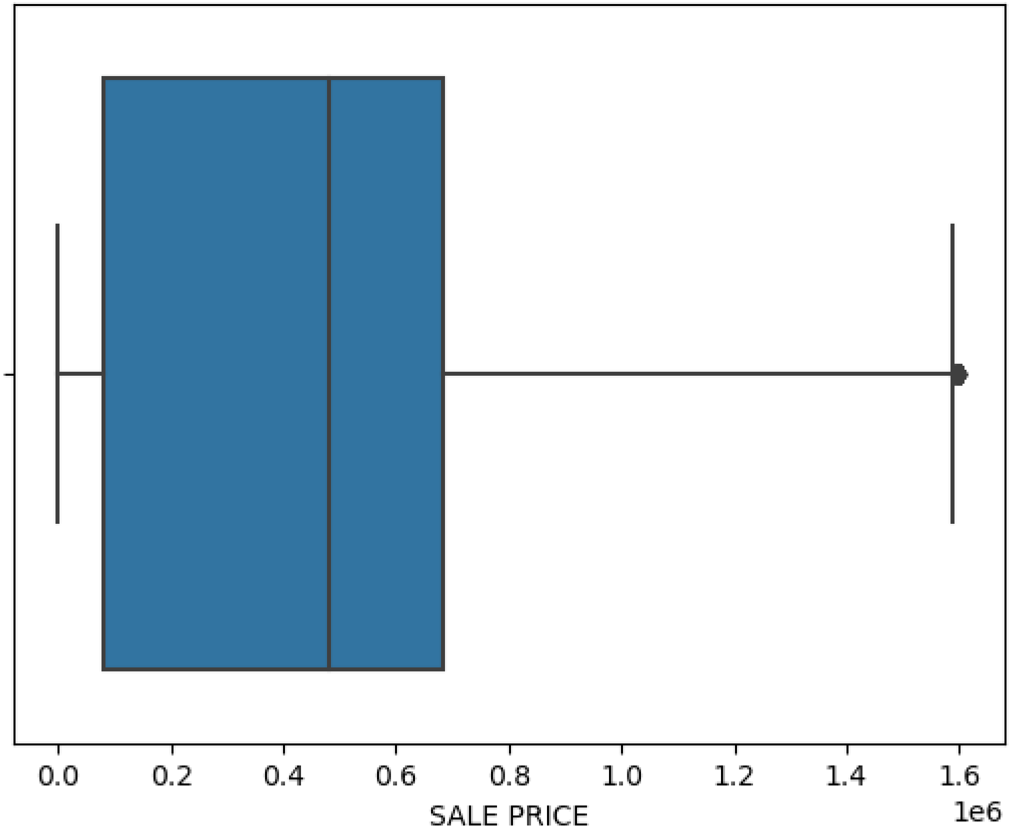
In [72]:

```python
median6=np.median(df['SALE PRICE'])
```

In [73]:

```python
df['SALE PRICE']=np.where(df['SALE PRICE']>1600000,
                            median6, df['SALE PRICE'])
```

```
In [74]: boxplot_n6=seaborn.boxplot(x='SALE PRICE', data=df)
```



```
In [ ]: 
```

```
In [75]: #Export cleaned dataset
```

```
In [76]: df.to_csv('NYC_clean.csv')
```

```
In [ ]: 
```

```
In [ ]: 
```

```
In [ ]: 
```

```
In [ ]: 
```

```
In [ ]: 
```

```
In [77]: ##RQ- What characteristics of the units influence the price of the unit?
```

```
In [78]: import statsmodels.formula.api as smf
```

```
In [79]: from sklearn import linear_model
```

In [80]:

```
import statsmodels.api as sm
```

In [81]:

```
X=df[['BOROUGH','RESIDENTIAL UNITS','COMMERCIAL UNITS',
      'LAND SQUARE FEET','GROSS SQUARE FEET','TAX_numeric']]
```

In [82]:

```
y=df['SALE PRICE']
```

In [83]:

```
model=smf.ols('y~X', data=df)
```

In [84]:

```
res=model.fit()
```

In [85]:

```
print(res.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.052
Model:                            OLS   Adj. R-squared:                  0.052
Method:                 Least Squares   F-statistic:                     441.0
Date:                Wed, 19 Jul 2023   Prob (F-statistic):               0.00
Time:                        00:11:02   Log-Likelihood:            -6.8728e+05
No. Observations:               48244   AIC:                         1.375e+06
Df Residuals:                   48237   BIC:                         1.375e+06
Df Model:                           6                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1.811e+05   8371.894     21.638      0.000    1.65e+05    1.98e+05
X[0]        8.795e+04   2054.530     42.806      0.000    8.39e+04     9.2e+04
X[1]        7059.6424   1056.201      6.684      0.000    4989.474    9129.810
X[2]        2530.8511   2364.956      1.070      0.285   -2104.494    7166.196
X[3]         -25.0549      2.065    -12.131      0.000     -29.103     -21.007
X[4]          32.9606      2.196     15.011      0.000      28.657      37.264
X[5]       -6450.9245    749.658     -8.605      0.000   -7920.265   -4981.584
==============================================================================
Omnibus:                     3194.669   Durbin-Watson:                   1.421
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3871.056
Skew:                           0.691   Prob(JB):                         0.00
Kurtosis:                       3.125   Cond. No.                     1.40e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.4e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [86]:

```
res.summary()
```

Out[86]:

```
OLS Regression Results

    Dep. Variable:                y          R-squared:        0.052
           Model:               OLS      Adj. R-squared:        0.052
          Method:     Least Squares         F-statistic:        441.0
            Date:  Wed, 19 Jul 2023  Prob (F-statistic):         0.00
            Time:          00:11:02      Log-Likelihood:  -6.8728e+05
No. Observations:             48244                 AIC:    1.375e+06
    Df Residuals:             48237                 BIC:    1.375e+06
        Df Model:                 6
 Covariance Type:         nonrobust

                  coef    std err        t  P>|t|      [0.025     0.975]
Intercept    1.811e+05   8371.894   21.638  0.000    1.65e+05   1.98e+05
     X[0]    8.795e+04   2054.530   42.806  0.000    8.39e+04    9.2e+04
     X[1]    7059.6424   1056.201    6.684  0.000    4989.474   9129.810
     X[2]    2530.8511   2364.956    1.070  0.285   -2104.494   7166.196
     X[3]     -25.0549      2.065  -12.131  0.000     -29.103    -21.007
     X[4]      32.9606      2.196   15.011  0.000      28.657     37.264
     X[5]   -6450.9245    749.658   -8.605  0.000   -7920.265  -4981.584

      Omnibus:  3194.669      Durbin-Watson:     1.421
Prob(Omnibus):     0.000   Jarque-Bera (JB):  3871.056
         Skew:     0.691           Prob(JB):      0.00
     Kurtosis:     3.125          Cond. No.  1.40e+04


Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.4e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [87]:

```
#Calculate MSE
```

In [88]:

```
from sklearn.linear_model import LinearRegression
```

In [89]:

```
from sklearn.metrics import mean_squared_error
```

```
In [90]: lin_regress=LinearRegression()
```

```
In [91]: lin_regress.fit(X,y)
```

```
Out[91]: LinearRegression()
```

```
In [92]: y_pred=lin_regress.predict(X)
```

```
In [93]: y_pred
```

```
Out[93]: array([278167.82104213, 291005.00862662, 266787.03549798, ...,
                577637.87564476, 573865.51527501, 584389.70537782])
```

```
In [94]: mse=mean_squared_error(y,y_pred)
```

```
In [95]: X_=df[['BOROUGH','RESIDENTIAL UNITS','COMMERCIAL UNITS',
                'TAX_numeric','LAND SQUARE FEET','GROSS SQUARE FEET']]
```

```
In [96]: print(mse)
```

```
138470011655.5684
```