```
#imports
import pandas as pd
import numpy as np
import math as mat
from google.colab import drive
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import tensorflow as tf
import statsmodels.api as sm

```

```
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Houses/raw_house_data.csv')

```

```
data['garage'].replace('None', np.nan, inplace=True)
data['bathrooms'].replace('None', np.nan, inplace=True)
data['sqrt_ft'].replace('None', np.nan, inplace=True)
data['HOA'].replace('None', np.nan, inplace=True)
data['floor_covering'].replace('None', np.nan, inplace=True)
data['sold_price'].replace('None', np.nan, inplace=True)
data.head(10)
```

Saved successfully!

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built
0	21530491	5300000.0	85637	-110.378200	31.356362	2154.00	5272.00	1941
1	21529082	4200000.0	85646	-111.045371	31.594213	1707.00	10422.36	1997
2	3054672	4200000.0	85646	-111.040707	31.594844	1707.00	10482.00	1997
3	21919321	4500000.0	85646	-111.035925	31.645878	636.67	8418.58	1930
4	21306357	3411450.0	85750	-110.813768	32.285162	3.21	15393.00	1995

```
data["sqrt_ft"] = data.sqrt_ft.astype(float)
data["bathrooms"] = data.bathrooms.astype(float)
data["year_built"] = data.year_built.astype(int)
data["garage"] = data.garage.astype(float)
data['HOA'] = data['HOA'].replace(',','', regex=True)
data["HOA"] = data.HOA.astype(float)
```

```
data.dtypes
```

```
MLS                 int64
sold_price        float64
zipcode             int64
longitude         float64
latitude          float64
year_built          int64
bedrooms            int64
bathrooms         float64
sqrt_ft           float64
garage            float64
kitchen_features   object
fireplaces        float64
floor_covering     object
HOA               float64
dtype: object
```

Saved successfully!

```
data['HOA'] = data['HOA'].fillna(data['HOA'].median())
data['sqrt_ft'] = data['sqrt_ft'].fillna(data['sqrt_ft'].median())
data.isna().sum()
```

```
MLS                 0
sold_price          0
```

```
zipcode             0
longitude           0
latitude            0
lot_acres          10
taxes               0
year_built          0
bedrooms            0
bathrooms           6
sqrt_ft             0
garage              7
kitchen_features    0
fireplaces         25
floor_covering      1
HOA                 0
dtype: int64
```

```
df = data
```

```
df = df.dropna()
```

```
df['year_built'].min()
```

```
0
```

```
df= df[df['year_built'] != 0]
```

```
df['year_built'].min()
2
```

```
1893
```

Saved successfully!

```
2019
```

```
1
```

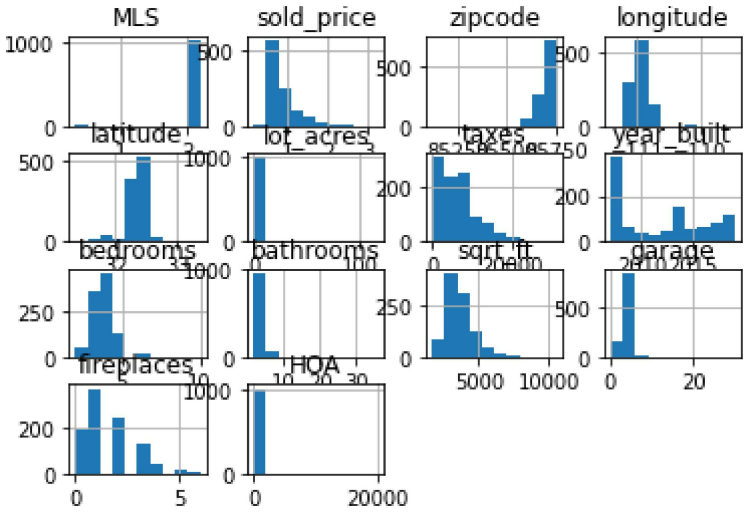
We create 4 categories base on the marketing for houses

```
dfA = df[df['year_built'] >= 2007]
dfA.shape
```

```
(1021, 16)
```

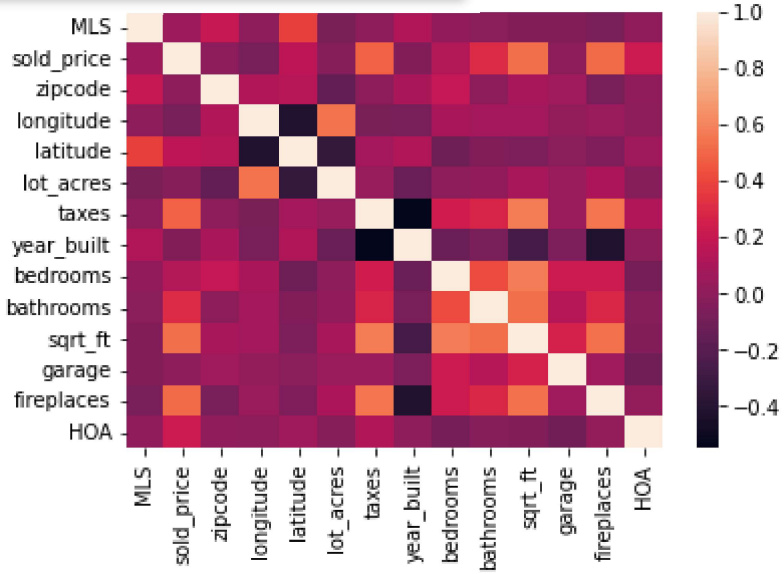
```
dfA.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8c0720d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8c085f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8fce2950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8fc94b90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8fa53510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8ccadc10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bf6b190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bf216d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bf21710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bf56e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bed1950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8be88f50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8be4c590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8be02b90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bdc51d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5d8bd7d7d0>]],
      dtype=object)
```



Saved successfully!

```
Subplot at 0x7f5d8bb93e10>
```



```
1 test = dfA.loc(axis=1)['sold_price','taxes','sqrt_ft']
```

```
1 test.dtypes
```

```
sold_price    float64
taxes         float64
sqrt_ft       float64
dtype: object
```

```
1
```

```
1
```

```
1 # Shuffle your dataset
2 shuffle_df = test.sample(frac=1)
3
4 # Define a size for your train set
5 train_size = int(0.7 * len(test))
6
7 # Split your dataset
8 train_datar = shuffle_df[:train_size]
9 test_datar = shuffle_df[train_size:]
```

```
1 train_datar = train_datar.to_numpy()
2 test_datar = test_datar.to_numpy()
```

```
1 x,y = train_datar[:,1:],train_datar[:,0]
2 xt, yt = test_datar[:,1:], test_datar[:,0]
```

Saved successfully!

```
(714, 2)
```

```
1 model = sm.OLS(y,x)
2 results = model.fit()
```

```
1 print(results.summary())
```

```
                                 OLS Regression Results
================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.886
Model:                            OLS   Adj. R-squared (uncentered):              0.886
Method:                 Least Squares   F-statistic:                              2776.
Date:                Tue, 30 Aug 2022   Prob (F-statistic):                        0.00
Time:                        03:54:36   Log-Likelihood:                         -10073.
No. Observations:                 714   AIC:                                  2.015e+04
```

```
Df Residuals:                      712   BIC:                             2.016e+04
Df Model:                            2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            18.9553      2.890      6.558      0.000      13.280      24.630
x2           200.4252      6.126     32.717      0.000     188.398     212.452
==============================================================================
Omnibus:                      203.669   Durbin-Watson:                   1.979
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              883.916
Skew:                           1.247   Prob(JB):                    1.15e-192
Kurtosis:                       7.847   Cond. No.                         4.81
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a cor
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
```

```
1 Pricepersqrtf = df['sold_price']/df['sqrt_ft']
2 Pricepersqrtf
```

```
0       504.761905
1       575.342466
3       498.946668
4       533.372420
5       475.007308
           ...
4994    254.036087
4995    137.461816
4996    237.273512
4997    268.058691
4998    147.690655
```

Saved successfully!

```
1 taxesperm = df['taxes']/12
2 taxesperm
```

```
0        439.333333
1        868.530000
3        701.548333
4       1282.750000
5       2316.903333
           ...
4994     367.833333
4995     168.083333
4996     401.834167
4997      83.333333
4998     485.244167
Name: taxes, Length: 4961, dtype: float64
```

taxes per month, number of rooms, +- score and recomendations

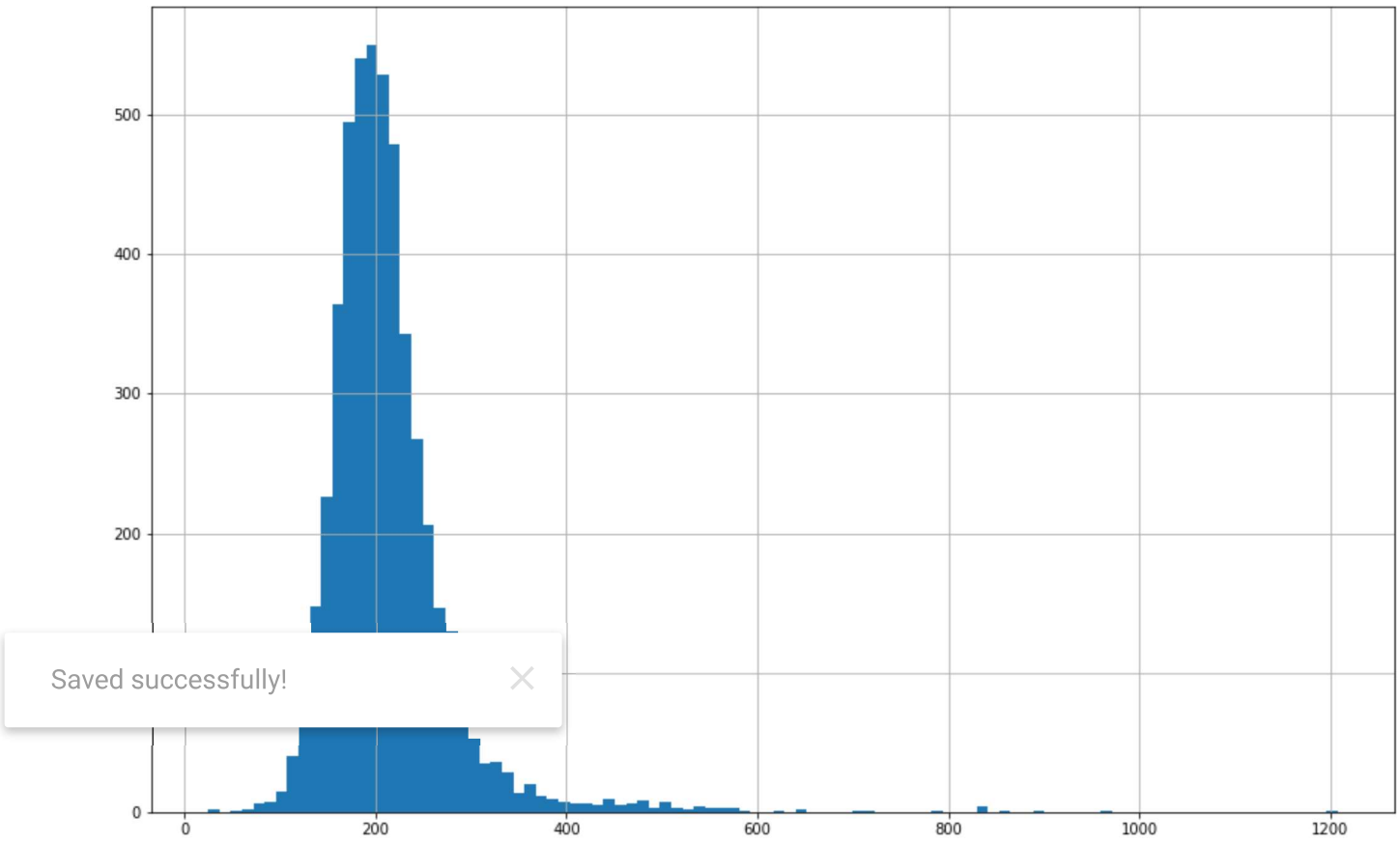
```
1 df['Pricesqrt'] = Pricepersqrtf
```

```
1 df['Pricesqrt'].nunique()
```

```
4816
```

```
1 fig = plt.figure(figsize = (15,10))
2 ax = fig.gca()
3
4 df['Pricesqrt'].hist(ax=ax,bins=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5d8b9c1510>
```




```
1 def transform(x):
2   if x >= 0 and x <=150:
3     return 1
4   if x >=151 and x <=250:
```

```
    return 2
  if x >= 251 and x <=350:
    return 3
  if x >= 351 and x <= 450:
    return 4
  if x >= 451:
    return 5
```

```
def transform(x):
  if x >= 0 and x <=150:
    return 1
  if x >=151 and x <=250:
    return 2
  if x >= 251 and x <=350:
    return 3
  if x >= 351:
    return 4
```

```
df["Pricesqrt"] = df['Pricesqrt'].map(transform)
```

```
class KNNClassifier():
  def fit(self, X, y):
    self.X = X
    self.y =y.astype(int)
  def predict(self,X, k, epsilon=1e-4):
    N = len(X)
    y_hat = np.zeros(N)
```

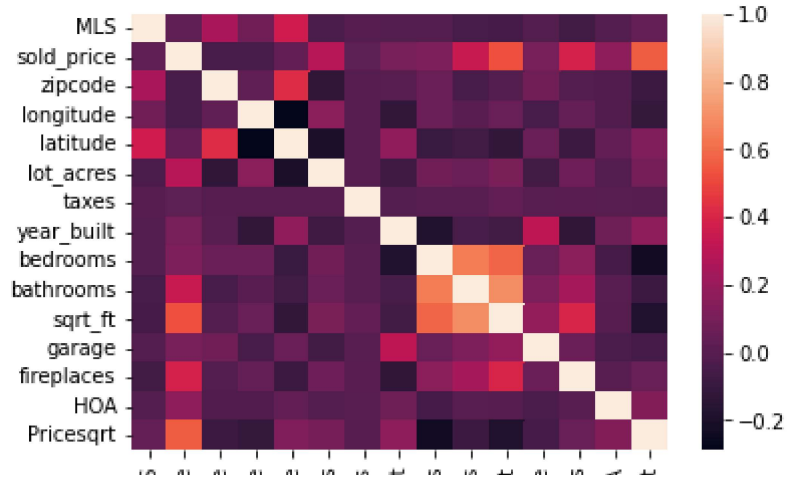
Saved successfully!

```
      dist2 = np.sum((self.X-X[i])**2,axis=1)
      idxt = np.argsort(dist2)[:k]
      gamma_k = 1/(np.sqrt(dist2[idxt]+epsilon))
      y_hat[i] = np.bincount(self.y[idxt], weights =gamma_k).argmax()
    return y_hat
```

```
def accuracy(y, y_hat):
  return np.mean(y==y_hat)
```

```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5d8ba1f110>
```




```
datclass = df.loc[:,['longitude','latitude','Pricesqrt','bathrooms','bedrooms']]
datclass.isna().sum()

```

```
longitude     0
latitude      0
Pricesqrt    36
bathrooms     0
bedrooms      0
dtype: int64
```

```
datclass = datclass.dropna()
```

```
datclass.isna().sum()
```

```
longitude    0
latitude     0
```

Saved successfully!

```
dtype: int64
```

```
from math import radians, cos, sin, asin, sqrt

def single_pt_haversine(lat, lng, degrees=True):
    """
    'Single-point' Haversine: Calculates the great circle distance
    between a point on Earth and the (0, 0) lat-long coordinate
    """
    r = 6371 # Earth's radius (km). Have r = 3956 if you want miles

    # Convert decimal degrees to radians
    if degrees:
        lat, lng = map(radians, [lat, lng])

    # 'Single-point' Haversine formula
```

```
    a = sin(lat/2)**2 + cos(lat) * sin(lng/2)**2
    d = 2 * r * asin(sqrt(a))

    return d
```

```
datclass['coor'] = datclass.apply(lambda x: single_pt_haversine(lat = x['latitude'], lng
```

```
datclass['coor'].head()
```

```
0    11931.100535
1    11988.050359
3    11986.037832
4    11951.394495
5    11959.113631
Name: coor, dtype: float64
```

```
datclass = datclass.loc[:,['coor','bedrooms','bathrooms','Pricesqrt']]
```

```
def absolute_maximum_scale(series):
    return series / series.abs().max()

```

```
datclass['coor'] = (datclass['coor']-datclass['coor'].min())/(datclass['coor'].max()-datc
```

```
# Shuffle your dataset
shuffle_df = datclass.sample(frac=1)

# Define a size for your train set
train_size = int(0.9 * len(datclass))
```

Saved successfully!

```
size]
test_dataknn = shuffle_df[train_size:]
```

```
train_dataknn = train_dataknn.to_numpy()
test_dataknn = test_dataknn.to_numpy()
```

```
train_dataknn

```

```
array([[0.55906289, 5.        , 4.        , 2.        ],
       [0.68101563, 4.        , 3.        , 2.        ],
       [0.59895059, 3.        , 3.        , 2.        ],
       ...,
       [0.60613837, 3.        , 3.        , 3.        ],
       [0.60217234, 5.        , 3.        , 2.        ],
       [0.56994026, 3.        , 3.        , 2.        ]])
```

```
1 x = train_dataknn[:,:3]
2 y =  train_dataknn[:,-1]
```

```
1 xtest= test_dataknn[:,:3]
2 ytest = test_dataknn[:,-1]
```

```
1 modelknn = KNNClassifier()
```

```
1 modelknn.fit(x,y)
```

```
1 y_hat = modelknn.predict(xtest,20)
2
```

```
1   accuracy(ytest,y_hat)
```

```
0.7647058823529411
```

```
1 unique, counts = np.unique(y_hat, return_counts=True)
2
3 result = np.column_stack((unique, counts))
4 print (result.round(2))
```

```
[[  1.   7.]
 [  2. 467.]
 [  3.  13.]
 [  4.   6.]]
```

Saved successfully!

For this section, it was estimated the possible price base on quantity of bedroom and the area where they are found

```
1 gendat = datclass.to_numpy()
2 xknn = gendat[:,:3]
3 xknn
```

```
array([[ 0.4898821 , 13.        , 10.        ],
       [ 0.7504651 ,  2.        ,  2.        ],
       [ 0.74125647,  7.        ,  5.        ],
       ...,
       [ 0.59845064,  4.        ,  3.        ],
       [ 0.4889346 ,  3.        ,  2.        ],
       [ 0.68342353,  4.        ,  4.        ]])
```

```
1 y_ultknn = modelknn.predict(xknn,15)
```

```
1 add = datclass
2 add['ClassPred'] = y_ultknn
```

```
1 add.head()
```

	coor	bedrooms	bathrooms	Pricesqrt	ClassPred
0	0.489882	13	10.0	4.0	4.0
1	0.750465	2	2.0	4.0	4.0
3	0.741256	7	5.0	4.0	4.0
4	0.582740	4	6.0	4.0	2.0
5	0.618060	3	4.0	4.0	2.0

```
1 ndf = df
2
```

```
1 ndf = ndf.join(add, lsuffix='l', rsuffix='la')
```

```
1   ndf.head()
```

Saved successfully!

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built

```
#class 1
cate1 = ndf[ndf['ClassPred']==1]
cate1.head()
```

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_bui
30	21906412	2350000.0	85750	-110.858327	32.331597	1.55	5931.00	20
47	21605559	2197500.0	85750	-110.860852	32.330665	1.14	7292.00	20
60	21706549	1995000.0	85750	-110.860305	32.323825	1.15	3292.00	20
89	21625846	1750000.0	85749	-110.761590	32.302163	3.71	13947.44	20
115	21823642	1695000.0	85718	-110.917749	32.300739	1.11	19110.45	20

5 rows × 22 columns

Saved successfully!

```
#class 2
cate2 = ndf[ndf['ClassPred']==2]
cate2.shape
```

```
(4573, 22)
```

```
#class 3
cate3 = ndf[ndf['ClassPred']==3]
cate3.shape
```

```
(208, 22)
```

```
#class 4
cate4 = ndf[ndf['ClassPred']==4]
cate4.shape
```

```
(57, 22)
```

```
#class 5
cate5 = ndf[ndf['ClassPred']==5]
cate5.shape
```

```
(0, 22)
```

Linear regressions

```
# Shuffle your dataset
shuffle_df = cate1.sample(frac=1)

# Define a size for your train set
train_size = int(0.9 * len(cate1))

# Split your dataset
train_datar = shuffle_df[:train_size]
test_datar = shuffle_df[train_size:]
```

```
train_datar = train_datar.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
test_datar = test_datar.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
```

```
train_datar
```

Saved successfully!

```
1 train_datar = train_datar.to_numpy()
2 test_datar = test_datar.to_numpy()
```

```
3029     650000.0           5    4279.0           6.0
```

```
1 x,y = train_datar[:,1:],train_datar[:,0]
2
3 xt, yt = test_datar[:,1:], test_datar[:,0]
```

```
4029     565000.0           5    4279.0           5.0
```

```
1 model = sm.OLS(y,x)
2 results = model.fit()
3 print(results.summary())
```

```
                                 OLS Regression Results
==============================================================================================
Dep. Variable:                          y   R-squared (uncentered):                   0.737
Model:                                OLS   Adj. R-squared (uncentered):              0.727
Method:                     Least Squares   F-statistic:                              70.22
Date:                    Tue, 30 Aug 2022   Prob (F-statistic):                    1.00e-21
Time:                            05:02:09   Log-Likelihood:                         -1130.0
No. Observations:                      78   AIC:                                      2266.
Df Residuals:                          75   BIC:                                      2273.
Df Model:                               3
Covariance Type:                nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1         -4.194e+04   3.34e+04     -1.257      0.213   -1.08e+05    2.45e+04
x2           204.1738     28.086      7.270      0.000     148.224     260.124
x3         -3.887e+04   4.32e+04     -0.899      0.372   -1.25e+05    4.73e+04
==============================================================================
Omnibus:                       12.954   Durbin-Watson:                   1.843
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               44.615
Skew:                          -0.119   Prob(JB):                     2.05e-10
Kurtosis:                       6.698   Cond. No.                     6.64e+03
==============================================================================
```

Saved successfully!

```
Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a cor
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
[3] The condition number is large, 6.64e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
1   yest·=·results.predict(xt)
2   yest
```

```
array([ 469605.54963094,  982081.77107924,  441267.3920658 ,
        600114.60333543,  461889.97190702, 1452625.21694796,
        763163.41141561,  533145.59909837,  706975.67192527])
```

```
# Shuffle your dataset
shuffle_df2 = cate2.sample(frac=1)

# Define a size for your train set
train_size2 = int(0.9 * len(cate2))

# Split your dataset
train_datar2 = shuffle_df2[:train_size2]
test_datar2 = shuffle_df2[train_size2:]
```

```
train_datar2 = train_datar2.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
test_datar2 = test_datar2.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
```

```

train_datar2 = train_datar2.to_numpy()
test_datar2 = test_datar2.to_numpy()

```

```
x2,y2 = train_datar2[:,1:],train_datar2[:,0]
xt2, yt2 = test_datar2[:, 1:], test_datar2[:,0]
```

```
x2.shape
```

```
(4115, 3)
```

```
xt2.shape
```

```
(458, 3)
```

```
model2 = sm.OLS(y2,x2)
```

Saved successfully!

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.924
Model:                            OLS   Adj. R-squared (uncentered):              0.924
Method:                 Least Squares   F-statistic:                          1.668e+04
Date:                Tue, 30 Aug 2022   Prob (F-statistic):                        0.00
Time:                        05:05:10   Log-Likelihood:                         -56507.
No. Observations:                4115   AIC:                                  1.130e+05
Df Residuals:                    4112   BIC:                                  1.130e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1         -2.414e+04   3923.251     -6.153      0.000   -3.18e+04   -1.64e+04
x2           186.2101      4.653     40.017      0.000     177.087     195.333
x3          4.247e+04   4097.591     10.363      0.000    3.44e+04    5.05e+04
```

```
==============================================================================
Omnibus:                     2465.810   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            77123.534
Skew:                           2.318   Prob(JB):                         0.00
Kurtosis:                      23.696   Cond. No.                     5.14e+03
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a cor
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
[3] The condition number is large, 5.14e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
1 yest = results2.predict(xt2)
```

```
1 # Shuffle your dataset
2 shuffle_df3 = cate3.sample(frac=1)
3
4 # Define a size for your train set
5 train_size3 = int(0.9 * len(cate3))
6
7 # Split your dataset
8 train_datar3 = shuffle_df3[:train_size3]
9 test_datar3 = shuffle_df3[train_size3:]
```

```
1 train_datar3 = train_datar3.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
2 test_datar3 = test_datar3.loc(axis=1)['sold_price','bedroomsl','sqrt_ft','bathroomsl']
3 train_datar3 = train_datar3.to_numpy()
4 test_datar3 = test_datar3.to_numpy()
5 x3,y3 = train_datar3[:,1:],train_datar3[:,0]
                            t_datar3[:,0]
```

Saved successfully!

```
1 model3 = sm.OLS(y3,x3)
2 results3 = model3.fit()
3 print(results3.summary())
```

```
                                 OLS Regression Results
==============================================================================================
Dep. Variable:                        y   R-squared (uncentered):                   0.949
Model:                              OLS   Adj. R-squared (uncentered):              0.948
Method:                   Least Squares   F-statistic:                              1138.
Date:                  Tue, 30 Aug 2022   Prob (F-statistic):                   1.67e-118
Time:                          04:53:33   Log-Likelihood:                         -2575.3
No. Observations:                   187   AIC:                                      5157.
Df Residuals:                       184   BIC:                                      5166.
Df Model:                             3
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
x1         -6750.7863   9454.243     -0.714      0.476   -2.54e+04    1.19e+04
x2         -6750.7863   9454.243     -0.714      0.476   -2.54e+04    1.19e+04
x3           234.8345     17.425     13.477      0.000     200.456     269.213
x4          1.782e+04   1.18e+04      1.506      0.134   -5520.674    4.12e+04
x5          1.782e+04   1.18e+04      1.506      0.134   -5520.674    4.12e+04
==============================================================================
Omnibus:                        8.385   Durbin-Watson:                   2.210
Prob(Omnibus):                  0.015   Jarque-Bera (JB):               12.764
Skew:                           0.230   Prob(JB):                      0.00169
Kurtosis:                       4.194   Cond. No.                      2.37e+19
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a cor
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
[3] The smallest eigenvalue is 4.84e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
1 yest3 = results3.predict(xt3)
```

Live part

```
1   from geopy import Nominatim
```

```
1   def coord(x):
2
3     geolocator = Nominatim(user_agent='TFClan')
4     location = geolocator.geocode(x)
                                    cation.latitude,location.longitude)
```

Saved successfully!

```
1   coord("1202 E Irvington rd Tucson AZ")
```

```
11966.853401080774
```

```
1   xas = [11966.853401080774]
2   y_hatest = modelknn.predict(xas,15)
3   y_hatest
```

```
array([1.])
```

```
1   xas = [6,4000,5]
2
```

+ Code + Text

```
1   yesti = results.predict(xas)
2   vesti
```

```
array([370700.40747885])
```

✓ 0s completed at 1:07 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Saved successfully!