```
import pandas as pd
import numpy as np
import missingno as msn
import math
import seaborn as sns
from scipy import stats

import statsmodels.api as sm
```

EDA

The first step is to make exploratory analysis to understand the data frame and if it is fit to be used in the project it is required.

The data frame

This datafrae was provided as an excel file. First step was to convert it to CSV format. The dataframe is going to be read in order to proceed with the exploratory analysis.

```
raw_data = pd.read_csv('/content/drive/MyDrive/Enhance IT Data Science Course/Week 1/A
raw_data
```

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

Columns

Column quantity

The dataframe has 16 different columns.

```
len(raw_data.columns)
```

```
16
```

Column names

The following are the column names in the dataframe.

```
raw_data.columns
```

```
Index(['MLS', 'sold_price', 'zipcode', 'longitude', 'latitude', 'lot_acres',
       'taxes', 'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',
       'kitchen_features', 'fireplaces', 'floor_covering', 'HOA'],
      dtype='object')
```

Column Types

Each column is a diferent variable and these are their types. There are some variables who are objects. It won't be possible to handle the properly so something important to consider is to change them to its respective type.

```
raw_data.dtypes
```

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

Missing data

First glance

At first glance, there is missing data. The instances with the more missing are the variables with 14 missing values. Theres is a problem though. There are some columns in the dataframe with type 'object'. And in the graphic below they are considered as a not missing value. Also, in some cases, some not missing values could be managed diferently to the variable's context. Further explication will be provided later in this document.

The dataaset has missing values.

```
raw_data.isnull().values.any()
```

```
True
```

Missing values per column.

```
raw_data.isnull().sum()
```

```
MLS                0
sold_price         0
zipcode            0
longitude          0
latitude           0
lot_acres         10
taxes              0
```

```
year_built          0
bedrooms            0
bathrooms           0
sqrt_ft             0
garage              0
kitchen_features    0
fireplaces         25
floor_covering      0
HOA                 0
dtype: int64
```

Total missing values

```
raw_data.isnull().sum().sum()
```

```
35
```

```
msn.matrix(raw_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dea5cc5d0>
```

Column Values

Next step is exploring the columns one by one.

MLS - Int

```
raw_data['MLS'].sort_values().unique()
```

```
array([ 3042851,  3044500,  3044867, ..., 21925429, 21926082, 21928565])
```

sold_price - Float

```
raw_data['sold_price'].sort_values().unique()
```

```
array([ 169000.,  300000.,  375000., ..., 4200000., 4500000., 5300000.])
```

```
len(raw_data['sold_price'].sort_values().unique())
```

```
1092
```

```
min(raw_data['sold_price'])
```

```
169000.0
```

```
max(raw_data['sold_price'])
```

```
5300000.0
```

Zipcode - Int

```
raw_data['zipcode'].sort_values().unique()
```

```
array([85118, 85192, 85541, 85601, 85602, 85603, 85605, 85609, 85610,
       85611, 85614, 85615, 85619, 85621, 85622, 85623, 85624, 85625,
       85629, 85630, 85637, 85638, 85640, 85641, 85643, 85645, 85646,
       85648, 85658, 85701, 85704, 85705, 85710, 85711, 85712, 85713,
       85715, 85716, 85718, 85719, 85730, 85737, 85739, 85742, 85743,
```

```
           85745, 85747, 85748, 85749, 85750, 85755, 85901, 85929, 85935,
           86024, 86323])
```

```
len(raw_data['zipcode'].sort_values().unique())
```

```
56
```

Longitude - Float

```
raw_data['longitude'].sort_values().unique()
```

```
array([-112.520168, -111.430863, -111.33586 , ..., -109.826222,
       -109.685284, -109.454637])
```

```
len(raw_data['longitude'].sort_values().unique())
```

```
4762
```

Latitude - Float

```
raw_data['latitude'].sort_values().unique()
```

```
array([31.356362, 31.361562, 31.375394, ..., 34.314889, 34.596971,
       34.927884])
```

```
len(raw_data['latitude'].sort_values().unique())
```

```
4821
```

lot_acres

```
raw_data['lot_acres'].sort_values().unique()
```

```
           3.99000e+00, 4.00000e+00, 4.01000e+00, 4.03000e+00, 4.04000e+00,
           4.07000e+00, 4.08000e+00, 4.09000e+00, 4.11000e+00, 4.12000e+00,
           4.13000e+00, 4.14000e+00, 4.15000e+00, 4.16000e+00, 4.17000e+00,
           4.20000e+00, 4.21000e+00, 4.22000e+00, 4.23000e+00, 4.24000e+00,
           4.26000e+00, 4.27000e+00, 4.28000e+00, 4.29000e+00, 4.30000e+00,
           4.31000e+00, 4.32000e+00, 4.33000e+00, 4.34000e+00, 4.37000e+00,
           4.38000e+00, 4.39000e+00, 4.40000e+00, 4.41000e+00, 4.43000e+00,
           4.45000e+00, 4.46000e+00, 4.48000e+00, 4.49000e+00, 4.50000e+00,
           4.52000e+00, 4.53000e+00, 4.55000e+00, 4.56000e+00, 4.57000e+00,
           4.58000e+00, 4.60000e+00, 4.61000e+00, 4.62000e+00, 4.63000e+00,
           4.64000e+00, 4.65000e+00, 4.66000e+00, 4.68000e+00, 4.69000e+00,
           4.72000e+00, 4.73000e+00, 4.75000e+00, 4.77000e+00, 4.78000e+00,
```

```
       4.80000e+00, 4.83000e+00, 4.85000e+00, 4.86000e+00, 4.94000e+00,
       4.96000e+00, 4.98000e+00, 5.00000e+00, 5.01000e+00, 5.06000e+00,
       5.08000e+00, 5.10000e+00, 5.13000e+00, 5.15000e+00, 5.17000e+00,
       5.19000e+00, 5.20000e+00, 5.21000e+00, 5.25000e+00, 5.26000e+00,
       5.27000e+00, 5.28000e+00, 5.34000e+00, 5.40000e+00, 5.41000e+00,
       5.51000e+00, 5.55000e+00, 5.58000e+00, 5.64000e+00, 5.67000e+00,
       5.75000e+00, 5.76000e+00, 5.77000e+00, 5.85000e+00, 5.88000e+00,
       5.90000e+00, 5.97000e+00, 5.98000e+00, 5.99000e+00, 6.00000e+00,
       6.01000e+00, 6.03000e+00, 6.07000e+00, 6.11000e+00, 6.12000e+00,
       6.21000e+00, 6.31000e+00, 6.39000e+00, 6.50000e+00, 6.52000e+00,
       6.60000e+00, 6.62000e+00, 6.63000e+00, 6.64000e+00, 6.70000e+00,
       6.73000e+00, 6.93000e+00, 6.97000e+00, 7.00000e+00, 7.09000e+00,
       7.10000e+00, 7.11000e+00, 7.12000e+00, 7.22000e+00, 7.27000e+00,
       7.31000e+00, 7.36000e+00, 7.38000e+00, 7.49000e+00, 7.65000e+00,
       7.73000e+00, 7.74000e+00, 7.75000e+00, 7.76000e+00, 7.79000e+00,
       7.80000e+00, 7.83000e+00, 7.87000e+00, 7.93000e+00, 7.96000e+00,
       7.97000e+00, 7.99000e+00, 8.00000e+00, 8.03000e+00, 8.10000e+00,
       8.11000e+00, 8.15000e+00, 8.27000e+00, 8.29000e+00, 8.37000e+00,
       8.41000e+00, 8.44000e+00, 8.46000e+00, 8.77000e+00, 8.80000e+00,
       8.81000e+00, 9.10000e+00, 9.18000e+00, 9.20000e+00, 9.30000e+00,
       9.37000e+00, 9.54000e+00, 9.55000e+00, 9.76000e+00, 9.81000e+00,
       9.84000e+00, 1.00000e+01, 1.01000e+01, 1.03100e+01, 1.04000e+01,
       1.05600e+01, 1.10000e+01, 1.14400e+01, 1.14900e+01, 1.15700e+01,
       1.20000e+01, 1.20300e+01, 1.20600e+01, 1.30000e+01, 1.31600e+01,
       1.32000e+01, 1.33600e+01, 1.34900e+01, 1.35000e+01, 1.36200e+01,
       1.45200e+01, 1.49000e+01, 1.50000e+01, 1.50500e+01, 1.63000e+01,
       1.63300e+01, 1.69900e+01, 1.77800e+01, 1.81300e+01, 1.81700e+01,
       1.85700e+01, 1.88900e+01, 1.90000e+01, 1.91100e+01, 1.99100e+01,
       2.00000e+01, 2.13900e+01, 2.15600e+01, 2.20600e+01, 2.47200e+01,
       2.49200e+01, 2.71000e+01, 2.96000e+01, 3.00000e+01, 3.09000e+01,
       3.18900e+01, 3.20000e+01, 3.24500e+01, 3.44500e+01, 3.51000e+01,
       3.60000e+01, 3.60100e+01, 3.60200e+01, 3.62000e+01, 3.63000e+01,
       3.65000e+01, 3.66100e+01, 3.66300e+01, 3.68700e+01, 3.72200e+01,
       3.74200e+01, 3.75700e+01, 3.83500e+01, 3.88000e+01, 3.89800e+01,
       3.90900e+01, 3.91000e+01, 3.96600e+01, 4.00000e+01, 4.04100e+01,
       4.04900e+01, 4.08700e+01, 4.10400e+01, 4.13000e+01, 4.15000e+01,
       4.19000e+01, 4.64100e+01, 4.75200e+01, 5.00000e+01, 5.20000e+01,
       5.70500e+01, 5.78400e+01, 5.80000e+01, 5.93000e+01, 6.05200e+01,
       6.05700e+01, 6.87000e+01, 7.20000e+01, 7.26200e+01, 7.33300e+01,
       7.34200e+01, 7.60300e+01, 7.66700e+01, 7.69200e+01, 7.72000e+01,
       7.91800e+01, 8.12000e+01, 9.17000e+01, 9.30000e+01, 9.30600e+01,
       9.40700e+01, 9.46800e+01, 1.03000e+02, 1.04800e+02, 1.17020e+02,
       1.19790e+02, 1.31000e+02, 1.47180e+02, 1.64300e+02, 1.72760e+02,
       2.20000e+02, 2.73030e+02, 2.77000e+02, 4.44930e+02, 4.71000e+02,
       5.55600e+02, 6.36670e+02, 1.04818e+03, 1.70700e+03, 2.15400e+03,
             nan])
```

```
len(raw_data['lot_acres'].sort_values().unique())
```

```
646
```

```
raw_data['lot_acres'].isna().sum()
```

```
10
```

Taxes - Float

```
raw_data['taxes'].sort_values().unique()
```

```
array([0.0000000e+00, 1.0000000e+00, 2.0000000e+00, ..., 3.2442220e+04,
       6.6805900e+05, 1.2215075e+07])
```

```
len(raw_data['taxes'].sort_values().unique())
```

```
4719
```

Year built - Int

```
raw_data['year_built'].sort_values().unique()
```

```
array([   0, 1893, 1900, 1901, 1902, 1905, 1907, 1910, 1911, 1913, 1914,
       1917, 1918, 1919, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928,
       1929, 1930, 1931, 1932, 1934, 1935, 1936, 1937, 1938, 1939, 1940,
       1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951,
       1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962,
       1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973,
       1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984,
       1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995,
       1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006,
       2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
       2018, 2019])
```

```
len(raw_data['year_built'].sort_values().unique())
```

```
112
```

Bedrooms - Int

```
raw_data['bedrooms'].sort_values().unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 13, 18, 19, 36])
```

```
len(raw_data['year_built'].sort_values().unique())
```

```
112
```

Bathrooms - Object

```
raw_data['bathrooms'].sort_values().unique()
```

```
array(['1', '10', '11', '14', '15', '18', '2', '2.5', '3', '3.5', '35',
       '36', '4', '4.5', '5', '6', '7', '8', '9', 'None'], dtype=object)
```

```
len(raw_data['bathrooms'].sort_values().unique())
```

```
20
```

Squared feet - Object

```
raw_data['sqrt_ft'].sort_values().unique()
```

```
array(['10258', '10318', '10417', ..., '9630', '9858', 'None'],
      dtype=object)
```

```
len(raw_data['sqrt_ft'].sort_values().unique())
```

```
2362
```

Garage - Object

```
raw_data['garage'].sort_values().unique()
```

```
array(['0', '1', '10', '11', '12', '13', '15', '2', '2.5', '20', '22',
       '3', '3.5', '30', '4', '4.5', '5', '6', '7', '8', '9', 'None'],
      dtype=object)
```

```
len(raw_data['garage'].sort_values().unique())
```

```
22
```

Kitchen features - Object

```
kf = raw_data['kitchen_features'].sort_values().unique()
for i in kf:
  print(i)
```

```
Dishwasher, Refrigerator, Microwave: microwave, Oven: oven/ stove
Dishwasher, Refrigerator, Oven
Double Sink, Electric Range, Garbage Disposal, Island, Pantry: Closet, Refrig
Double Sink, Electric Range, Island, Pantry: Walk-In, Refrigerator, Wet Bar,
Double Sink, Freezer, Gas Range, Island, Pantry: Walk-In, Refrigerator, Appli
```

```
Double Sink, Garbage Disposal, Gas Range, Island
Double Sink, Garbage Disposal, Gas Range, Island, Pantry: Butler, Refrigerato
Double Sink, Garbage Disposal, Gas Range, Lazy Susan, Pantry: Closet, Refrige
Double Sink, Garbage Disposal, Gas Range, Refrigerator
Double Sink, Garbage Disposal, Island, Pantry: Closet, Refrigerator, Microwav
Double Sink, Garbage Disposal, Island, Pantry: Walk-In, Prep Sink, Appliance
Double Sink, Gas Range, Island, Appliance Color: Stainless, Countertops: Gran
Double Sink, Gas Range, Island, Pantry: Cabinet, Appliance Color: Black, Coun
Electric Range
Electric Range, Garbage Disposal, Island, Refrigerator, Appliance Color: Stai
Electric Range, Island, Countertops: Quartz
Electric Range, Refrigerator, Appliance Color: Stainless
Freezer, Garbage Disposal, Gas Range, Island, Refrigerator, Appliance Color:
Freezer, Refrigerator, Appliance Color: Stainless, Countertops: wood and gran
Garbage Disposal
Garbage Disposal, Gas Range, Island, Lazy Susan, Pantry: Walk-In, Refrigerato
Garbage Disposal, Gas Range, Island, Lazy Susan, Pantry: Walk-In, Refrigerato
Garbage Disposal, Gas Range, Island, Pantry: Cabinet, Prep Sink, Countertops:
Garbage Disposal, Gas Range, Island, Pantry: Closet, Prep Sink, Appliance Col
Garbage Disposal, Gas Range, Island, Pantry: Walk-In, Appliance Color: Stainl
Garbage Disposal, Gas Range, Refrigerator, Oven: -
Garbage Disposal, Island, Refrigerator, Countertops: SS, Oven: .
Garbage Disposal, Microwave, Oven
Garbage Disposal, Oven
Garbage Disposal, Pantry: Closet, Appliance Color: Stainless, Countertops: gr
Garbage Disposal, Pantry: Walk-In, Refrigerator, Countertops: Granite
Garbage Disposal, Refrigerator
Garbage Disposal, Refrigerator, Microwave
Garbage Disposal, Refrigerator, Microwave, Oven
Garbage Disposal, Refrigerator, Oven
Gas Range, Appliance Color: Black, Countertops: Granite

Gas Range, Island
Gas Range, Island, Pantry: Butler, Refrigerator, Appliance Color: Stainless,
Gas Range, Pantry: Butler, Refrigerator, Countertops: Granite
Gas Range, Refrigerator
Island
Island, Countertops: granite, Missing: appliances
Island, Pantry: Walk-In, Refrigerator, Countertops: granite
Microwave, Oven
Missing: All Appliances
Missing: kitchen
None
Oven
Pantry: Closet
Pantry: Walk-In, Appliance Color: Stainless, Countertops: quartz, Oven: stain
Prep Sink
Refrigerator
Refrigerator, Appliance Color: Black, Countertops: Granite
Refrigerator, Microwave, Oven
Refrigerator, Oven
Wet Bar
```

```
len(raw_data['kitchen_features'].sort_values().unique())
```

```
1872
```

Fireplaces - Int

```
raw_data['fireplaces'].sort_values().unique()
```

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., nan])
```

```
len(raw_data['fireplaces'].sort_values().unique())
```

```
11
```

```
raw_data['fireplaces'].isna().sum()
```

```
25
```

Floor covering - Object

```
raw_data['floor_covering'].sort_values().unique()
```

```
       'Ceramic Tile, Other: porcelain wood tile',
       'Ceramic Tile, Other: vinyl planks', 'Ceramic Tile, Vinyl',
       'Ceramic Tile, Vinyl, Other: Cement tiles/Bamboo',
       'Ceramic Tile, Vinyl, Wood', 'Ceramic Tile, Wood',
       'Ceramic Tile, Wood, Other',
       'Ceramic Tile, Wood, Other: Saltillo on Patio',
       'Ceramic Tile, Wood, Other: Travertine Accents', 'Concrete',
       'Concrete, Laminate', 'Concrete, Laminate, Natural Stone',
       'Concrete, Laminate, Natural Stone, Wood',
       'Concrete, Mexican Tile', 'Concrete, Mexican Tile, Other',
       'Concrete, Mexican Tile, Wood',
       'Concrete, Mexican Tile, Wood, Other: concrete tile',
       'Concrete, Natural Stone', 'Concrete, Natural Stone, Wood',
       'Concrete, Natural Stone, Wood, Other', 'Concrete, Other',
       'Concrete, Other: Cork', 'Concrete, Other: Polished Concrete',
       'Concrete, Other: Real polishd aggrgt',
       'Concrete, Other: Saltillo', 'Concrete, Other: Stained concrete',
       'Concrete, Other: porclain tile', 'Concrete, Vinyl, Wood',
       'Concrete, Wood', 'Concrete, Wood, Other',
       'Concrete, Wood, Other: Marble',
       'Concrete, Wood, Other: Mesquite wood floors',
       'Concrete, Wood, Other: Porcelain tile',
       'Concrete, Wood, Other: Tile bathrooms',
       'Concrete, Wood, Other: flagstone',
       'Laminate, Mexican Tile, Natural Stone',
       'Laminate, Mexican Tile, Natural Stone, Wood',
       'Laminate, Natural Stone', 'Laminate, Other: Porcelain tile 24x24',
       'Laminate, Vinyl', 'Laminate, Wood, Other: Porcelain Tile',
```

```
       'Mexican Tile', 'Mexican Tile, Natural Stone',
       'Mexican Tile, Natural Stone, Other',
       'Mexican Tile, Natural Stone, Wood',
       'Mexican Tile, Natural Stone, Wood, Other: studio laminate',
       'Mexican Tile, Other', 'Mexican Tile, Other: CONCRETE TILE',
       'Mexican Tile, Other: Porcelain', 'Mexican Tile, Other: Saltillo',
       'Mexican Tile, Other: San Marcos Mex Tile',
       'Mexican Tile, Other: saltillo', 'Mexican Tile, Wood',
       'Mexican Tile, Wood, Other', 'Mexican Tile, Wood, Other: Brick',
       'Mexican Tile, Wood, Other: scored concrete', 'Natural Stone',
       'Natural Stone, Other', 'Natural Stone, Other: Limestone',
       'Natural Stone, Other: Porcelain-wood',
       'Natural Stone, Other: Rock', 'Natural Stone, Other: Travertine',
       'Natural Stone, Other: Travertine & Slate',
       'Natural Stone, Other: Wood Laminate', 'Natural Stone, Wood',
       'Natural Stone, Wood, Other', 'Natural Stone, Wood, Other: Cork',
       'Natural Stone, Wood, Other: Marble',
       'Natural Stone, Wood, Other: Organic Wool Carpet', 'None', 'Other',
       'Other: 100% Porcelain Tile', 'Other: Brick', 'Other: Flagstone',
       'Other: Italian Tile', 'Other: Italian tile',
       'Other: Luxury Vinyl', 'Other: None', 'Other: Polish concrete',
       'Other: Polished Brick', 'Other: Porcelain',
       'Other: Porcelain Tile', 'Other: Porcelain tile',
       'Other: Porcelyn', 'Other: Quartzite', 'Other: Recycled Porcelain',
       'Other: Saltillo tile', 'Other: TBD', 'Other: Tile',
       'Other: Tile-Other', 'Other: Travertine', 'Other: travertine',
       'Vinyl, Wood', 'Wood', 'Wood, Other', 'Wood, Other: Lime Stone',
       'Wood, Other: Porcelain tile', 'Wood, Other: Travertine',
       'Wood, Other: Travertine/Marble', 'Wood, Other: porcelain tile'],
      dtype=object)
```

```
len(raw_data['floor_covering'].sort_values().unique())
```

```
311
```

HOA - Object

```
raw_data['HOA'].sort_values().unique()
```

```
array(['0', '1', '1,000', '1,010', '1,100', '1,200', '1,270', '1,290',
       '1,600', '1,717', '1,769', '10', '10.83', '100', '101', '102',
       '103', '104', '105', '106', '107', '108', '109', '11', '11.08',
       '110', '111', '112', '112.38', '113', '114', '115', '116', '117',
       '118', '119', '119.66', '12', '120', '121', '122', '123', '123.44',
       '124', '125', '126', '127', '128', '129', '13', '130', '131',
       '132', '132.66', '133', '134', '134.5', '135', '136', '137', '138',
       '139', '14', '14.58', '140', '141', '141.66', '141.67', '142',
       '143', '144', '145', '145.83', '146', '147', '148', '149',
       '149.04', '149.5', '15', '15.41', '15.45', '150', '151', '152',
       '153', '154', '155', '156', '157', '157.33', '158', '159', '16',
       '16.66', '16.67', '160', '161', '162', '164', '165', '166',
       '166.66', '167', '168', '168.92', '169', '17', '170', '171', '172',
```

```
       '173', '174', '175', '176', '177', '177.34', '178', '179', '18',
       '18.75', '180', '183', '184', '185', '186', '187', '188', '188.33',
       '189', '19', '19,480', '190', '191', '192', '193', '193.5', '194',
       '194.51', '195', '198', '199', '2', '2,000', '2.08', '20',
       '20,000', '20.83', '200', '202', '202.75', '203', '205', '208',
       '209', '21', '210', '211', '212', '212.38', '212.88', '213',
       '213.88', '214', '215', '216', '219', '22', '220', '221', '225',
       '225.21', '226', '23', '232', '233', '233.33', '234', '238', '24',
       '240', '241', '242', '243', '247', '249', '25', '250', '252',
       '253', '257', '258', '258.08', '259', '26', '263', '267', '269',
       '270', '273', '275', '275.08', '28', '283', '285', '29', '290',
       '294', '295', '299', '3', '30', '300', '303', '31', '311', '317',
       '32', '320', '322', '323', '324', '325', '328', '33', '33.33',
       '330', '332.66', '332.67', '333', '337', '34', '34.17', '342',
       '343', '35', '35.83', '350', '357', '36', '36.02', '36.66', '368',
       '37', '37.5', '38', '38.55', '39', '39.59', '390', '4', '4.16',
       '4.17', '40', '40.55', '40.77', '40.78', '41', '41.08', '41.61',
       '41.66', '42', '42.38', '421', '422', '425', '43', '43.01',
       '43.71', '43.75', '437', '44', '45', '45.9', '46', '46.95', '47',
       '48', '487.88', '49', '49.43', '5', '5,900', '5.25', '5.4', '5.41',
       '5.5', '50', '50.42', '500', '506', '51', '516', '52', '53',
       '53.34', '54', '54.16', '55', '55.58', '550', '56', '57', '57.33',
       '57.5', '58', '58.33', '58.36', '59', '6', '6.25', '60', '60.5',
       '609', '61', '62', '62.5', '63', '63.33', '63.98', '64', '65',
       '66', '66.66', '66.67', '67', '68', '68.66', '69', '69.16', '7',
       '70', '700', '71', '72', '73', '73.33', '73.72', '74', '75', '750',
       '76', '765', '77', '78', '78.65', '79', '8', '8,333', '8.33',
       '8.34', '80', '81', '82', '83', '83.33', '83.34', '84', '84.75',
       '85', '86', '87', '87.66', '88', '88.33', '89', '9', '90', '91',
       '92', '925', '93', '94', '95', '96', '97', '97.66', '98', '99',
       '99.66', 'None'], dtype=object)
```

```
len(raw_data['HOA'].sort_values().unique())
```

```
381
```

Data Cleaning

For data cleaning it is necesary to perform the following tasks:

- Changing object columns to string, int or float depending the case.
- Delete rows and/or Predict missing values

Object type columns

The columns who has object type are the following:

- bathrooms
- sqrt_ft
- garage

- kitchen_features
- floor_covering
- HOA

Changing object columns to int or float

First step is to change all 'None' values to nan values.

```
#Change all 'none' to nan
raw_data = raw_data.fillna(value=np.nan)
```

```
raw_data.isnull().sum()
```

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

```
raw_data.isnull().sum().sum()
```

```
35
```

No changes has been made after changing 'None' values to nan values, so the 'None' values in the object columns are a different data type. ie: string

Some things to consider

- In the 'floor_covering' column, there are two strings to consider as nan: 'None' and 'Other: None'. In this project both will be considered as nan due to both means the same.
- In the 'kitchen_features' column there are 'None' values and actual string values who have the 'none' status in microwave, so only the first one will be consider as nan since in the second case 'none' is considered part of the kitchen information.

- Object columns who have string and int values have any extraordinary situation and will be considered as their respective values after replacing 'None' for nan.

Replacing values column by column

```
raw_data = raw_data.replace('None', np.nan)
raw_data['floor_covering'] = raw_data['floor_covering'].replace('Other: None', np.nan
```

```
raw_data.isnull().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              56
garage                7
kitchen_features     33
fireplaces           25
floor_covering        2
HOA                 562
dtype: int64
```

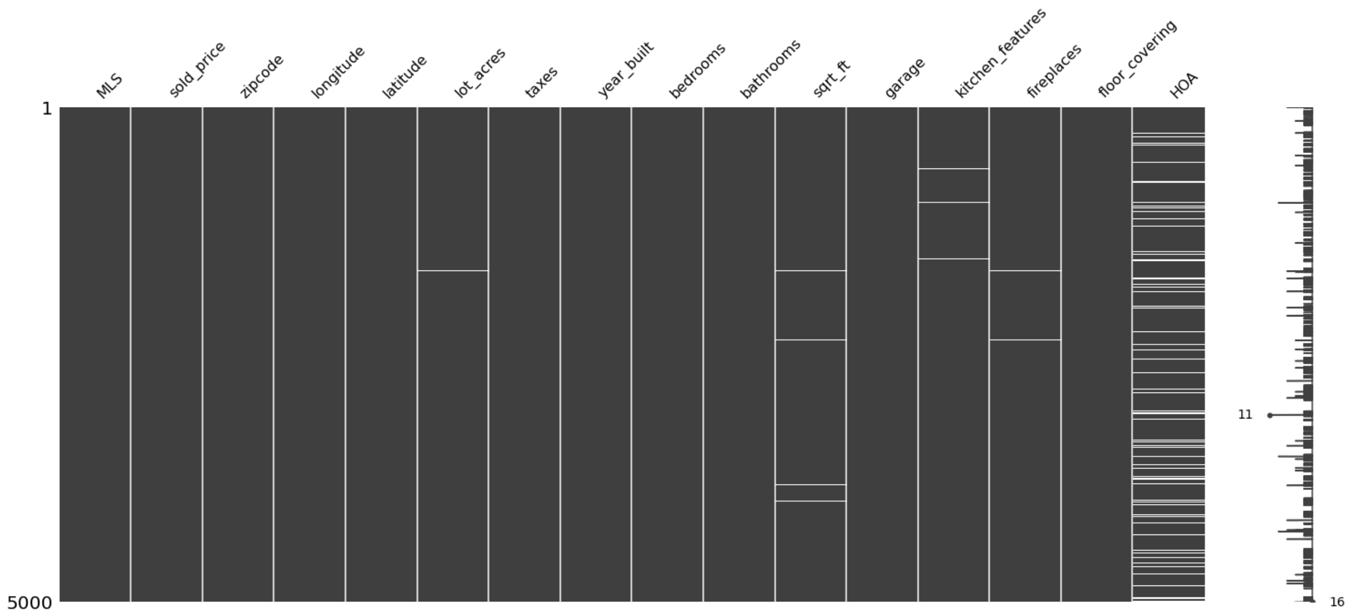
```
raw_data.isnull().sum().sum()
```

```
701
```

Now we went from 35 nan values to 701 nan values.

```
msn.matrix(raw_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dea461750>
```



Changing object types to float or int.

For changing the object column types first it is important to understand which values are floats, so the remaining columns who are not sring type values can be converted to integer.

```
# First we change all values as numeric, then we check if any element is an instance o
# float and the we check if there are float values in each element.
raw_data.apply(pd.to_numeric, errors="ignore").applymap(lambda x: isinstance(x, float)
```

```
MLS                 False
sold_price           True
zipcode             False
longitude            True
latitude             True
lot_acres            True
taxes                True
year_built          False
bedrooms            False
bathrooms            True
sqrt_ft              True
garage               True
kitchen_features    False
fireplaces           True
floor_covering      False
HOA                 False
dtype: bool
```

The list avobe classifies all float type columns as true and the rest as false. Now the column types can be classified as follows:

- MLS - Int
- sold_price - Float
- zipcode - Int
- longitude - Float
- latitude - Float
- lot_acres - Float
- taxes - Float
- year_built - Int
- bedrooms - Int
- bathrooms - Float
- sqrt_ft - Float
- garage - Float
- kitchen_features - String
- fireplaces - Float
- floor_covering - String
- HOA - Int

Things to consider

- Even though HOA column is considered not a float, some float elements and int elements with commas. Further cleanning will be needed.

```
# Change of the object columns to int or float (Excep for 'HOA' column)
raw_data['bathrooms'] = raw_data['bathrooms'].astype('float')
raw_data['sqrt_ft']  = raw_data['sqrt_ft'].astype('float')
raw_data['garage'] = raw_data['garage'].astype('float')
raw_data['kitchen_features'] = raw_data['kitchen_features'].astype('string')
raw_data['floor_covering'] = raw_data['floor_covering'].astype('string')
```

```
# Replaces commas in the strings with ''.
raw_data['HOA'] = raw_data['HOA'].replace(',','', regex=True)
# Converts 'HOA' column into float.
raw_data['HOA'] = raw_data['HOA'].astype('float')
```

```
raw_data.dtypes
```

```
MLS                      int64
sold_price             float64
zipcode                  int64
```

```
longitude            float64
latitude             float64
lot_acres            float64
taxes                float64
year_built             int64
bedrooms               int64
bathrooms            float64
sqrt_ft              float64
garage               float64
kitchen_features      string
fireplaces           float64
floor_covering        string
HOA                  float64
dtype: object
```

Now all columns have its respective types.

Deletion and prediction of missing values

There are two important thing to remember. The dataset has 701 nan values. 562 of them are from 'HOA' column, making it the column with the most missing values. The other 139 are spread within 'lot_acres', 'sqrt_ft', 'garage', 'kitchen_features', 'fireplaces' and 'floor_covering' columns.

```
raw_data.isnull().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              56
garage                7
kitchen_features     33
fireplaces           25
floor_covering        2
HOA                 562
dtype: int64
```

```
raw_data.isnull().sum().sum()
```

```
701
```

This means that besides 'HOA' column the datasest has up to 139 instances who, at least, have 1 missing value.

In this project the rows with missing values in any column except for 'HOA' will be deleted.

Then, since the dataset has more than 10% of HOA column missing data, a prediction model will be

```
len(raw_data)
```

```
5000
```

HOA column missing values prediction

First it is necesary to know the correlation between the independent variable (HOA) and the dependent variables (the rest).

```
corr_matrix = raw_data.corr()
corr_matrix['HOA'].sort_values(ascending = False)
```

```
HOA           1.000000
sold_price    0.171170
latitude      0.030892
year_built    0.015036
fireplaces    0.006481
bathrooms     0.005243
taxes         0.004560
sqrt_ft       0.002485
lot_acres    -0.008533
MLS          -0.018158
longitude    -0.021703
zipcode      -0.024722
garage       -0.039678
bedrooms     -0.067988
Name: HOA, dtype: float64
```

Correlation is poor. The only variable who will be used in the prediction will be 'sold_price'.

Prediction

For the prediction a linear regression with the least squares methos will be used. It is not recommended to expect good results.

```
# define x and y variables.
train_data = raw_data.drop(raw_data[raw_data.HOA.isnull()].index)
x = train_data['sold_price']
y = train_data['HOA']
```

```
#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
model = sm.OLS(y, x).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    HOA   R-squared:                       0.029
Model:                            OLS   Adj. R-squared:                  0.029
Method:                 Least Squares   F-statistic:                     133.9
Date:                Tue, 16 Aug 2022   Prob (F-statistic):           1.57e-30
Time:                        03:47:04   Log-Likelihood:                -34213.
No. Observations:                4438   AIC:                         6.843e+04
Df Residuals:                    4436   BIC:                         6.844e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -127.4913     21.505     -5.928      0.000    -169.653     -85.330
sold_price     0.0003   2.56e-05     11.571      0.000       0.000       0.000
==============================================================================
Omnibus:                    11871.064   Durbin-Watson:                   2.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        233310353.826
Skew:                          32.102   Prob(JB):                         0.00
Kurtosis:                    1124.420   Cond. No.                     2.23e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
[2] The condition number is large, 2.23e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117: FutureWa
  x = pd.concat(x[::order], 1)
```

Prediction results

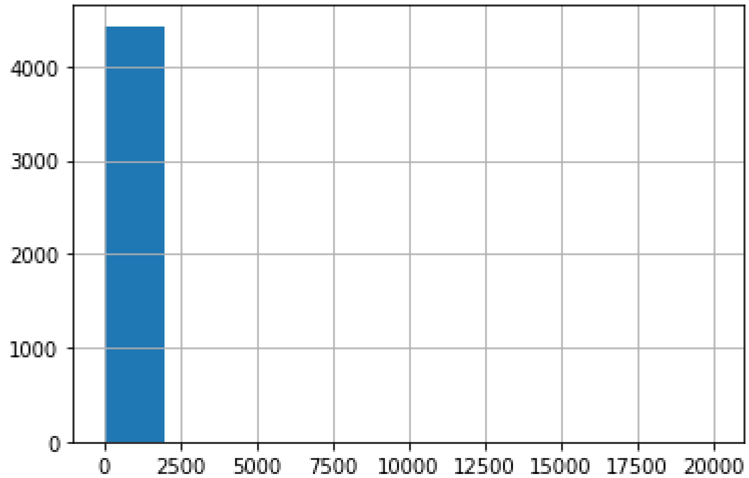
Since the R-suquared of our model is 0.029 It is not recommended to substitute missing values in ´HOA´ column with the predictions this model can generate. The results ar so poor, it will be practically the same to input random data. In conclusion, it will be necesary to look for another substitution method.

Data substitution

First it is necesary to look at the distribution of the ´HOA´ variable.

```
raw_data['HOA'].hist()
```

    <matplotlib.axes._subplots.AxesSubplot at 0x7f8dea3ec110>



Since 'HOA' feature has outliers, the best thig will be to replace the nan values with the median.

```
np.median(train_data['HOA'])
```

    56.0

The median is 56. Next step is to replace nan thevalues with this number.

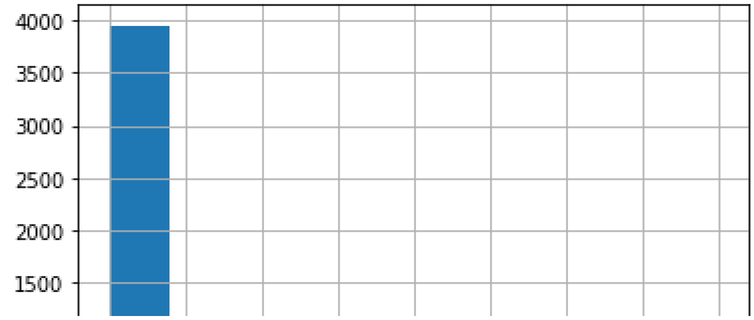
```
# Assign the median to the nan values
new_hoa = pd.Series(raw_data['HOA'])
median = np.median(train_data['HOA'])
for j, i in enumerate(new_hoa):
  if pd.isna(i):
    new_hoa[j] = median


# Replace the old HOA with the predicted HOA
raw_data = raw_data.drop(labels = 'HOA', axis = 1)
raw_data = raw_data.assign(HOA = new_hoa)
```

Now the 'HOA' feature is complete

```
raw_data['HOA'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dea4dd410>
```



The distribution was changed but this is maybe the best thng to do in this particular case.

Deletion of rows with missing values.

A deletion of the missing values for the rest of the variables will proceed.

```
# Deletion of rows with missing values
raw_data = raw_data.drop(raw_data[raw_data.lot_acres.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.bathrooms.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.sqrt_ft.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.garage.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.kitchen_features.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.fireplaces.isnull()].index)
raw_data = raw_data.drop(raw_data[raw_data.floor_covering.isnull()].index)
```

All the instances with missing values were deleted except for the missing values in the 'HOA' column. 93 instances were deleted in total.

```
raw_data.isnull().sum()
```

```
MLS                 0
sold_price          0
zipcode             0
longitude           0
latitude            0
lot_acres           0
taxes               0
year_built          0
bedrooms            0
bathrooms           0
sqrt_ft             0
garage              0
kitchen_features    0
fireplaces          0
floor_covering      0
HOA                 0
dtype: int64
```

Our data is complete, no more missing values!

Data Exploration Part Two

Now that the data is complete we can do some more exploration.

Data correlation

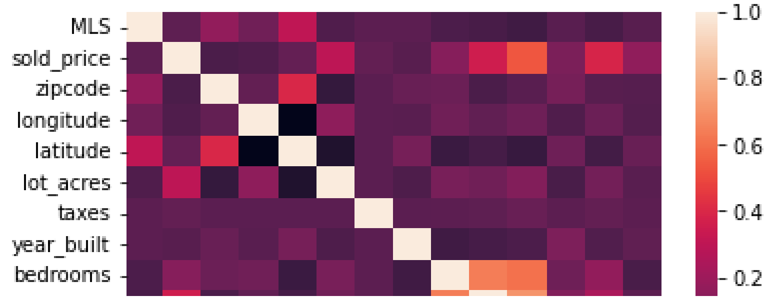
The correlation between variables ar as follows:

```
raw_data.corr()
```

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes
MLS	1.000000	0.006897	0.165119	0.066491	0.305317	-0.037203	0.002355
sold_price	0.006897	1.000000	-0.054891	-0.039780	0.027504	0.300523	0.023462
zipcode	0.165119	-0.054891	1.000000	0.024815	0.399569	-0.128703	-0.002074
longitude	0.066491	-0.039780	0.024815	1.000000	-0.311329	0.157587	-0.001182
latitude	0.305317	0.027504	0.399569	-0.311329	1.000000	-0.200858	0.000037
lot_acres	-0.037203	0.300523	-0.128703	0.157587	-0.200858	1.000000	-0.000726
taxes	0.002355	0.023462	-0.002074	-0.001182	0.000037	-0.000726	1.000000
year_built	0.004706	-0.013218	0.041256	-0.008810	0.087406	-0.044280	0.000060
bedrooms	-0.045562	0.130678	0.052018	0.065387	-0.108524	0.092989	0.005198
bathrooms	-0.064809	0.354633	-0.051133	0.020144	-0.075302	0.065832	0.009049
sqrt_ft	-0.090568	0.537213	-0.005091	0.062130	-0.116949	0.120350	0.038007
garage	-0.006397	0.100783	0.083879	-0.038961	0.062169	-0.059013	0.005581
fireplaces	-0.062865	0.385343	-0.010321	0.049870	-0.077375	0.072893	0.022757
HOA	-0.010035	0.163641	-0.018561	-0.019412	0.037044	-0.008999	0.004655

```
sns.heatmap(raw_data.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8dea328710>
```

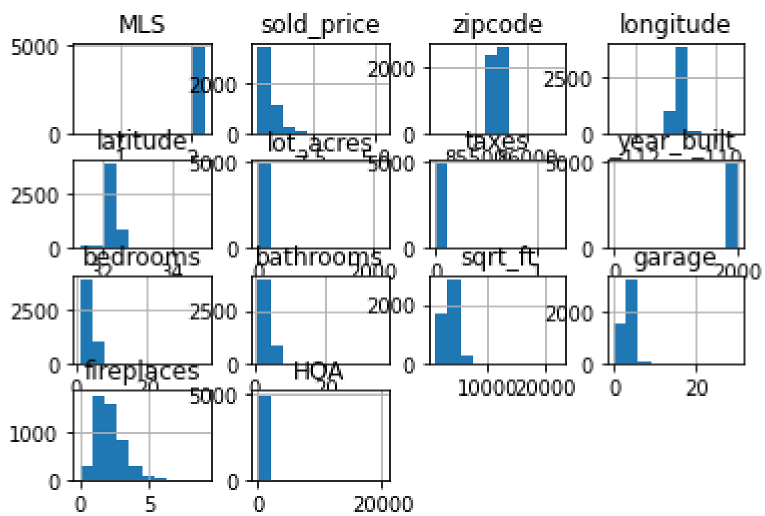


Data distribution

The following is the distribution of each numeric column. As seen, there are some outliers who are messing with the data.

```
raw_data.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea225f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea1d89d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea203b10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea152610>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea186c10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea14c250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea1028d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea0b9e10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea0b9e50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea07b590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9ff80d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9fb06d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9f65cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9f2b310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9ee1910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9e98f10>]],
      dtype=object)
```



It would be necesary to drop the rows with outliers.

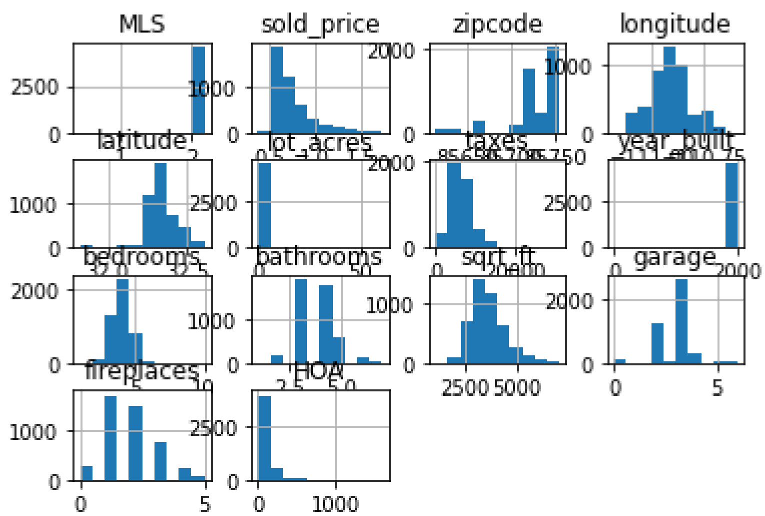
```
clean_data = raw_data.drop(labels = ['MLS', 'zipcode', 'year_built', 'bedrooms', 'kitc
z = np.abs(stats.zscore(clean_data))
raw_data = raw_data[(z<3).all(axis=1)]
raw_data.shape
```

```
(4570, 16)
```

As it can be seen, now the dataset has 4570 cleaned instances. As it can be seen bellow the distribution has no more outliers.

```
raw_data.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9cc6610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea3a7f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea1a40d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8dea034d50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9e248d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9bd4590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9afba50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9aa9b10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9ac0050>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9a78710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de99f1250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de99a8810>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8de995fe10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9922450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de98dba50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8de9908b90>]],
      dtype=object)
```



Our data is ready!

Conclussions

- The original dataset had 5000 instances.
- The data types where: 4 ints, 10 floats an 2 strings.
- In the data exploration phase, there were some challenges dealing with missing data like: Finding object type columns, dealing with strings, etc.
- Data deletion and data prediction were needed in this project to clean the dataframe.
- 'HOA' feature has the most missing values. More than 10% were missing.
- Linear regression model for predicting the 'HOA' feature values has a poor performance so, it was necesary to replace nan values with the median due to outliers.
- Some outliers have to be deleted.
- After the cleaning process, the dataset has 4570 instances.
- 430 instances had to be deleted.
- Correlations between variables tend to be poor.
- Columns and their type once cleaned:
- MLS - Int
- sold_price - Float
- zipcode - Int
- longitude - Float
- latitude - Float
- lot_acres - Float
- taxes - Float
- year_built - Int
- bedrooms - Int
- bathrooms - Float
- sqrt_ft - Float
- garage - Float
- kitchen_features - String
- fireplaces - Float
- floor_covering - String

- HOA - Float

✓ 0 s se ejecutó 23:51