```
from google.colab import drive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
"""
Links that helped me further understand the data.
hoa
  https://corporatefinanceinstitute.com/resources/knowledge/other/homeowners-association-hoa/
mls
  https://www.investopedia.com/terms/m/multiple-listing-service-mls.asp#toc-what-is-an-mls-nu
"""
```

```
df = pd.read_csv('/content/raw_house_data.csv')
print(df.shape)
df.head(3)
```

```
(5000, 16)
```

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built
0	21530491	5300000.0	85637	-110.378200	31.356362	2154.0	5272.00	1941
1	21529082	4200000.0	85646	-111.045371	31.594213	1707.0	10422.36	1997
2	3054672	4200000.0	85646	-111.040707	31.594844	1707.0	10482.00	1997

```
# Remove duplicated rows if they exist
df.drop_duplicates()
print(df.shape)
```

```
(5000, 16)
```

```
df.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')
```

```
# Removing logitude and lattitude because we have zip code
```

```
df1 = df.drop(['longitude', 'latitude'], axis=1)
print(df1.shape)
```

```
(5000, 14)
```

```
df1.columns
```

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

```
# Convert all 'None' values to nan
cols = ['MLS', 'sold_price', 'zipcode', 'lot_acres', 'taxes', 'year_built',
       'bedrooms', 'bathrooms', 'sqrt_ft', 'garage', 'kitchen_features',
       'fireplaces', 'floor_covering', 'HOA']
for i in cols:
  df1[i].replace('None', np.nan, inplace=True)
df1.head(3)
```

	MLS	sold_price	zipcode	lot_acres	taxes	year_built	bedrooms	bathrooms
0	21530491	5300000.0	85637	2154.0	5272.00	1941	13	10
1	21529082	4200000.0	85646	1707.0	10422.36	1997	2	2
2	3054672	4200000.0	85646	1707.0	10482.00	1997	2	3

```
df1.isnull().sum()
```

```
MLS                 0
sold_price          0
zipcode             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      1
HOA               562
dtype: int64
```

ASSUMPTION: I am assuming nan values for ['HOA', 'fireplaces', 'garage'] denotes that the property doesnt have the corresponding item. I am therefore converting those nan values to 0 to represent my assumption.

```
# With that assumption, I will change those nan values to 0
cols_nan_to_0 = ['HOA', 'fireplaces', 'garage']
for i in cols_nan_to_0:
  df1[i].replace(np.nan, 0, inplace=True)
df1.isnull().sum()
```

```
MLS                 0
sold_price          0
zipcode             0
lot_acres          10
taxes               0
year_built          0
bedrooms            0
bathrooms           6
sqrt_ft            56
garage              0
kitchen_features   33
fireplaces          0
floor_covering      1
HOA                 0
dtype: int64
```

```
(df1.isnull().sum().sum())/5000
```

```
0.0212
```

Only 2% of the data contains missing values at this point. Being that it is less than 5% of the total data, I think it is safe to remove them.

```
df2 = df1.dropna()
df2.isnull().sum()
```

```
MLS                0
sold_price         0
zipcode            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
```

```
df2.shape
```

```
(4912, 14)
```

```
df2.dtypes
```

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

```
# Converting the object types to a numerical type
df2.bathrooms = df2.bathrooms.astype(float)
df2.sqrt_ft = df2.sqrt_ft.astype(float)
df2.garage = df2.garage.astype(float).astype(int)
df2.fireplaces = df2.fireplaces.astype(int)
```

```
df2.dtypes
```

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

```
df2.HOA = df2.HOA.replace(',','', regex=True)
```

```
df2.HOA = df2.HOA.astype(float)
df2.dtypes
```

Now that the dtypes have been dealth with I will create another dataframe to start cleaning another aspect of the data.

```
df3 = df2.copy()
df3.shape
```

```
(4912, 14)
```

```
kitchen_items = set([])
for i in df3['kitchen_features'].unique():
  for item in i.split(','):
    kitchen_items.add(item.strip(' '))
kitchen_items = sorted(kitchen_items)
for i in kitchen_items:
  print(i)
```

```
Oven: double - electric
Oven: double SS
Oven: double convection
Oven: double oven
Oven: double oven built-in
Oven: double ovens
Oven: double range
Oven: double wall
Oven: elec/gas/convection
Oven: electric
Oven: electric built in
Oven: electric wall
Oven: electric/dual
Oven: electric/gas stove
Oven: freestanding
Oven: gas
Oven: in range
Oven: included
Oven: multiple wall ovens
Oven: mutiple
Oven: oven
Oven: oven / range
Oven: oven/ stove
Oven: part of range & elec
Oven: professional 48''
Oven: slide in
Oven: ss
Oven: stainless
Oven: stainless built-in
Oven: stainless high-end
Oven: stove/oven
Oven: total of 2
Oven: two
Oven: under cabinet
```

```
Oven: under cabinet
Oven: viking
Oven: vintage
Oven: wall
Oven: wall (electric)
Oven: wall mounted
Oven: wall oven
Oven: wall with convection
Oven: white
Oven: wolf
Oven: x 2
Oven: xtra-electric
Oven: yes
Pantry: Butler
Pantry: Cabinet
Pantry: Closet
Pantry: Walk-In
Prep Sink
Quartzite
Refrigerator
Reverse Osmosis
Stainless
Tile
Warming Drawer
Water Purifier
```

After seeing how dirty the 'kitchen_features' data is I decided to remove it until further notice. I think it can be processed and added to the dataset using a one-hot-encoding style in the future, but until I develope a method for doing so I believe it is better to leave it out of the dataset.

```
df3 = df3.drop(['kitchen_features'], axis=1)
print(df3.shape)
df3.head(3)
```

```
(4912, 13)
```

	MLS	sold_price	zipcode	lot_acres	taxes	year_built	bedrooms	bathrooms
0	21530491	5300000.0	85637	2154.00	5272.00	1941	13	10.0
1	21529082	4200000.0	85646	1707.00	10422.36	1997	2	2.0

```
floor_items = set([])
for i in df3['floor_covering'].unique():
  for item in i.split(','):
    floor_items.add(item.strip(' '))
floor_items = sorted(floor_items)
for i in floor_items:
  print(i)
```

Carpet
Ceramic Tile
Concrete
Granite
Indoor/Outdoor
Laminate
Mexican Tile
Natural Stone
Other
Other: 100% Porcelain Tile
Other: Bamboo
Other: Brazilian Pergo
Other: Brick
Other: Brick Floor
Other: Brick Pavers
Other: Brick inlayed
Other: CONCRETE TILE
Other: Canterra Stone
Other: Carpet bedrooms only
Other: Cement tiles/Bamboo
Other: Concrete tile
Other: Cork
Other: Custom Saltillo
Other: Dyed Concrete
Other: Egytian sandstone
Other: Eng wood
Other: Engineered Wood
Other: Flagstone
Other: Hardwood
Other: High End Laminate
Other: Itailian Porclaine
Other: Italian Tile
Other: Italian tile
Other: Lime Stone
Other: Limestone
Other: Lux Vinyl
Other: Luxury Vinyl
Other: Marble
Other: Marble-Master Bath
Other: Master Bedroom/ Tile
Other: Mesquite wood floors
Other: Multiple Types
Other: NEW Plank Tile
Other: NEW WOOD PLANK TILE
Other: None
Other: Organic Wool Carpet
Other: PLANK TILE
Other: Parquet
Other: Pergo
Other: Polish concrete
Other: Polished Brick
Other: Polished Concrete
Other: Porcelain
Other: Porcelain Plank Tile
Other: Porcelain Tile
Other: Porcelain Wood Tile

```
Other: Porcelain tile
```

I am removing floor_coverings for the same reason I removed kitchen_items

```
df3 = df3.drop(['floor_covering'], axis=1)
print(df3.shape)
df3.head(3)
```

```
(4912, 12)
```

	MLS	sold_price	zipcode	lot_acres	taxes	year_built	bedrooms	bathrooms
0	21530491	5300000.0	85637	2154.00	5272.00	1941	13	10.0
1	21529082	4200000.0	85646	1707.00	10422.36	1997	2	2.0
3	21919321	4500000.0	85646	636.67	8418.58	1930	7	5.0

I believe the important information in the HOA column is whether a property does or doesn't have an attached HOA, so I one-hot-encoded the HOA column to represent that. I think this is a better representation because some area's will naturally have a higher population, thus a larger HOA, and that could skew the results.

```
df3.HOA = df3.HOA.apply(lambda x: 1 if x > 0.0 else 0)
df3.HOA.unique()
```

```
array([0, 1])
```

I am creating two columns ['price_per_sqft', 'price_per_acre'] because I believe they will be good price estimators. Before I can do that I will check & remove rows where either of those columns are 0 as long as I don't end up removing more than 5% of the original data.

```
df4 = df3.copy()
print(df4[df4['lot_acres'].apply(lambda x: x == 0.0)].shape)
print(df4[df4['sqrt_ft'].apply(lambda x: x == 0.0)].shape)
```

```
(35, 12)
(0, 12)
```

```
df5 = df4[~df4['lot_acres'].apply(lambda x: x == 0.0)]
df5 = df5[~df5['sqrt_ft'].apply(lambda x: x == 0.0)]
```

Since it was only 35 rows I know I can remove them without going over my 5% limit. In total, I removed a little less than 3% of the data.

```
df5['price_per_sqft'] = round(df5['sold_price']/df5['sqrt_ft'], 2)
df5['price_per_acre'] = round(df5['sold_price']/df5['lot_acres'], 2)
df5.head()
```

	MLS	sold_price	zipcode	lot_acres	taxes	year_built	bedrooms	bathrooms
0	21530491	5300000.0	85637	2154.00	5272.00	1941	13	10.0
1	21529082	4200000.0	85646	1707.00	10422.36	1997	2	2.0
3	21919321	4500000.0	85646	636.67	8418.58	1930	7	5.0
4	21306357	3411450.0	85750	3.21	15393.00	1995	4	6.0
5	21528016	3250000.0	85718	1.67	27802.84	1999	3	4.0