PROJECT SAUDDY
DATA MINING AND INSIGHT DISCOVERY 2

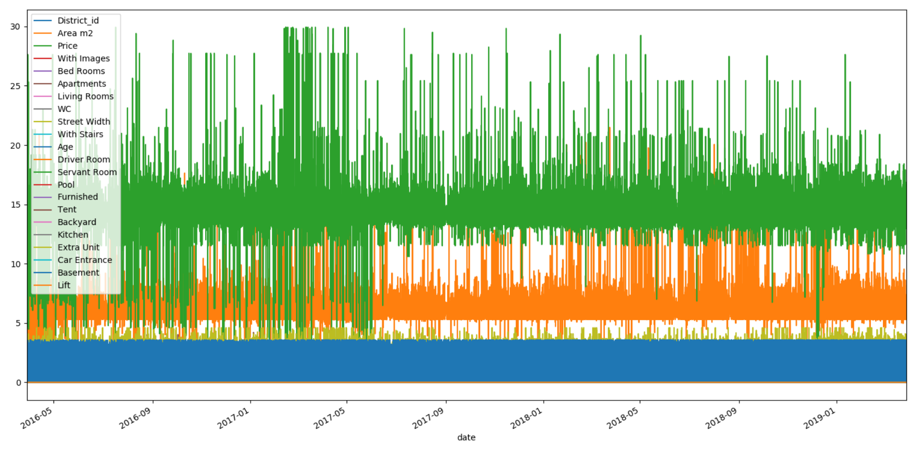
INDONESIAN HOUSING PRICES 2

By :
E. KENNETH

For :
SAUDDY

TITLE
MACHINE LEARNER/DATA SCIENTIST

April 12, 2019



Contents

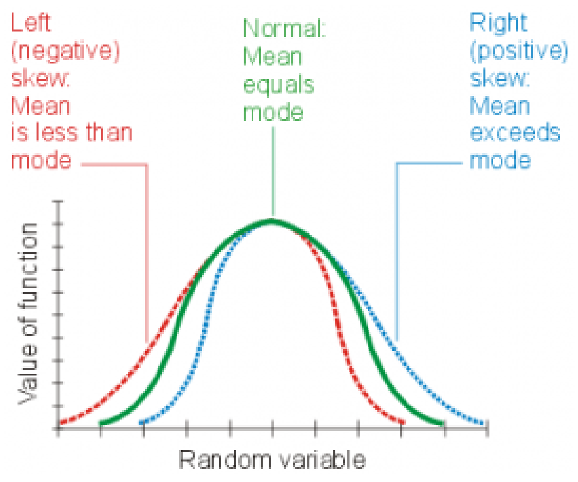
List of Figures

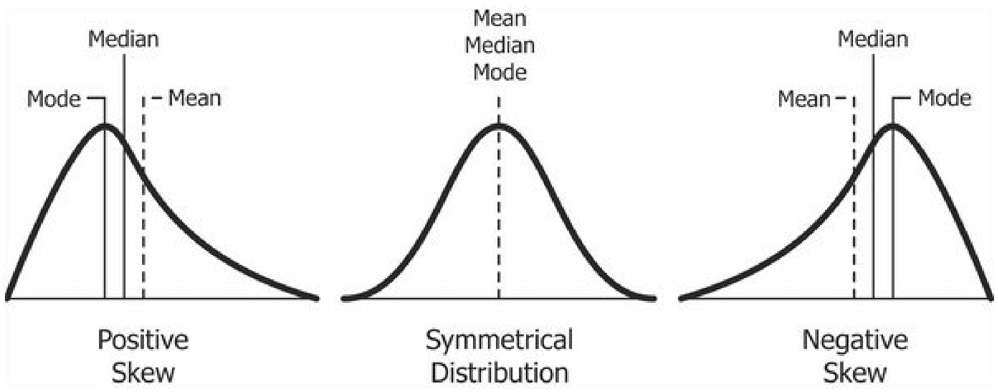
List of Tables

1 Introduction

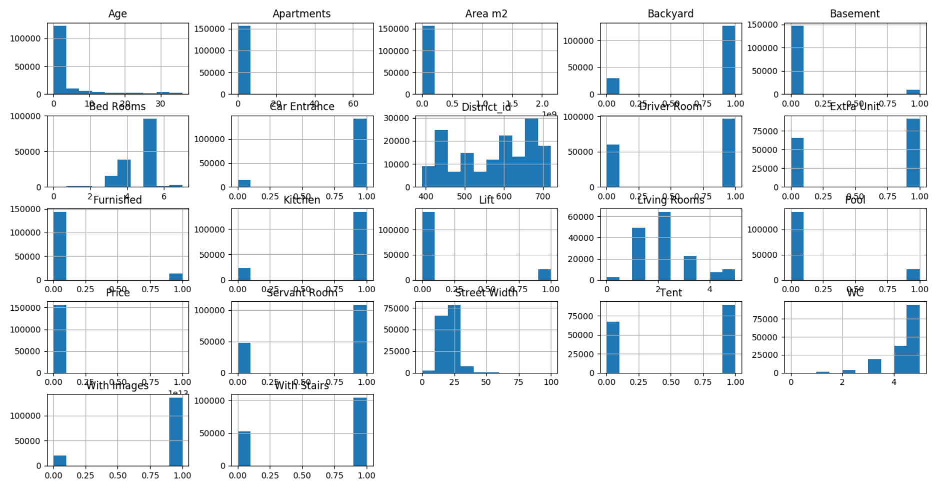
For this project i have taken the standard machine learning approach to solving every analysis problem. I began by taking a statistical measure of the data i am working, then followed by preprocessing of the data in an understandable form. I will begin by expplaining the two basic terms to introduce us to the type of data we are working on.

- Skew: Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. As explained in the graph below.


Left (negative) skew: Mean is less than mode
Normal: Mean equals mode
Right (positive) skew: Mean exceeds mode
Value of function
Random variable



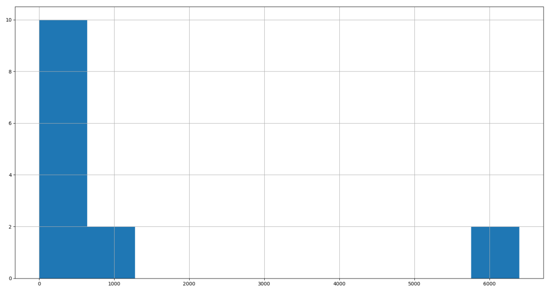
In our case, the data is positive skewed as the mean and median exceeds the mode price of the houses. In essence, the average price of houses exceeds the highest price of any particular house. This can be seen in the histogram below.



Area, Price, Apartments, Living Rooms, Street Width, Age, Pool, Furnished, Basement and Lift all have positive skew, meaning their average and median values is more than the highest occuring value. so it would be better to remove this mode values to balance the dataset.

The solution to balancing the data is removing outliers and that we would consider at later stage of the project

- kurtosis: the sharpness of the peak of a frequency-distribution curve. Just as explained above for prices we would also observe that there are area where prices are very high and areas where it is very low.



Kurtosis plot

High kurtosis in a data set is an indicator that data has heavy tails or outliers. If there is a high kurtosis, then, we need to investigate why do we have so many outliers. It indicates a lot of things, maybe wrong data entry or other things. Investigate in next section!

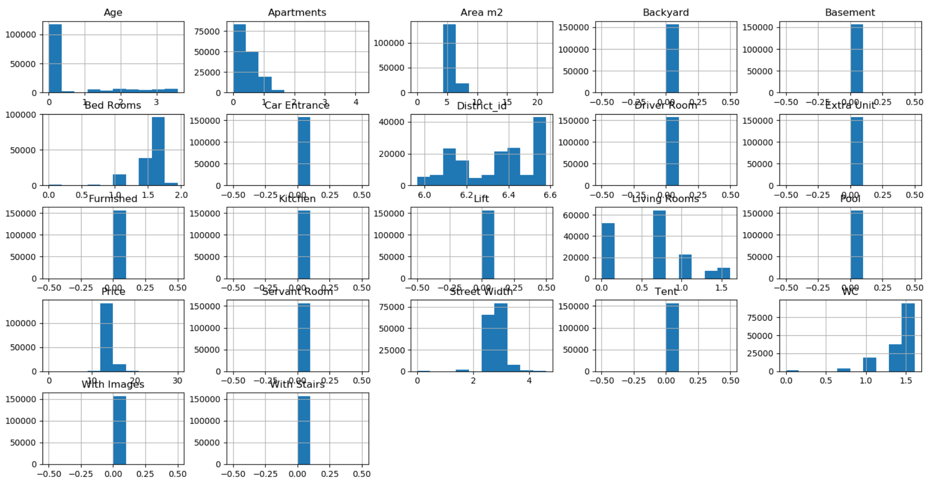
2 Exploratory analysis

This section deals with both preprocessing of the data and exploration to understand the trends and possible existence of outliers. We will also see a little about this outliers and how to handle them.
We begin by writing a function to return either a standardized data, scaled log data or normalized data as the case may be.

you can see the notebook here: Notebook

```
def standardize_houseprize(df, standardize = None,
                           logg = None, normalize = None):
  df = df.copy(deep = True)
  #drop all objects
  #and leaving all float64 and int64 datatypes
  for ii in df.columns:
    if df[ii].dtype == object:
      df = df.drop(ii, axis = 1)

  '''
  #standardize values
        x - mean of x
  z = --------------------
          sd of x

  #log values

  z = log(x)

  #normalize values

          x - min(x)
  z = --------------------
          max(x) - min(x)
  '''

  #standard deviation
  def stdev(df):
    return np.std(df, axis = 0)
  #mean deviation
  def mean_dev(df):
    return df - np.mean(df, axis = 0)
  #log of data
  def logg_dat(df):
    return np.log(df)
  #standardized values for columns
  if standardize:
    for ii, ij in enumerate(df.columns):
      print(ii, ij)
      df['{}'.format(ij)] = mean_dev(df.loc[:, '{}'.format(ij)])/stdev(df.loc[:, '
  elif logg:
    df = logg_dat(df)
    df = df.replace([np.inf, -np.inf, np.nan], 0)
  elif normalize:
    for ii, ij in enumerate(df.columns):
      df['{}'.format(ij)] = (df.loc[:, '{}'.format(ij)] - min(df.loc[:, '{}'.forma
      (max(df.loc[:, '{}'.format(ij)]) - min(df.loc[:, '{}'.format(ij)]))
  else:
    pass

  return df

df = standardize_houseprize(hosue_df)
df_standard = standardize_houseprize(hosue_df, standardize = True)
log_data = standardize_houseprize(hosue_df, logg=True)
df_normal = standardize_houseprize(hosue_df, normalize = True)
```

The Essence of standardizing the data is to remove the unbalance effect in the dataset. And to ensure it is scalled within a certain range. the formula for this is found inside the function above. In most cases i prefer the log_data since it is well scaled.



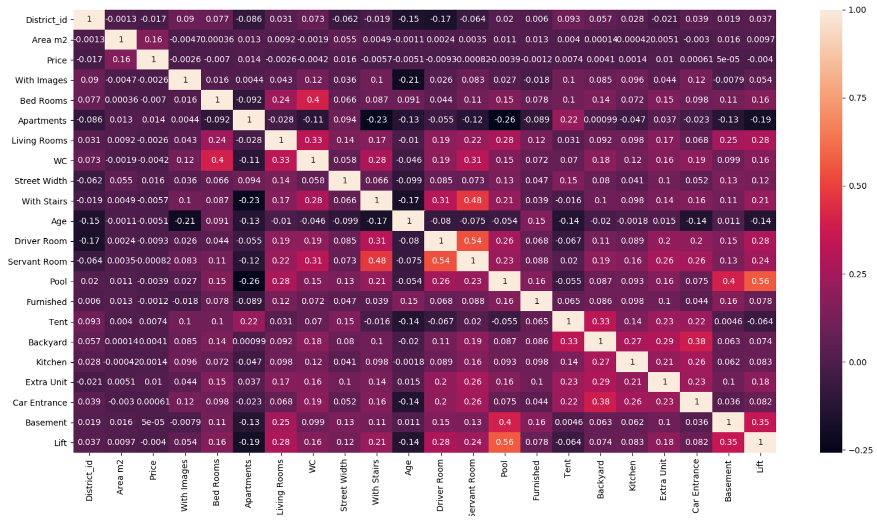
Plot of log data

The following code shows the distribution of the pairplot as they are correlated to eachother.

```
sns.pairplot(log_data)
```

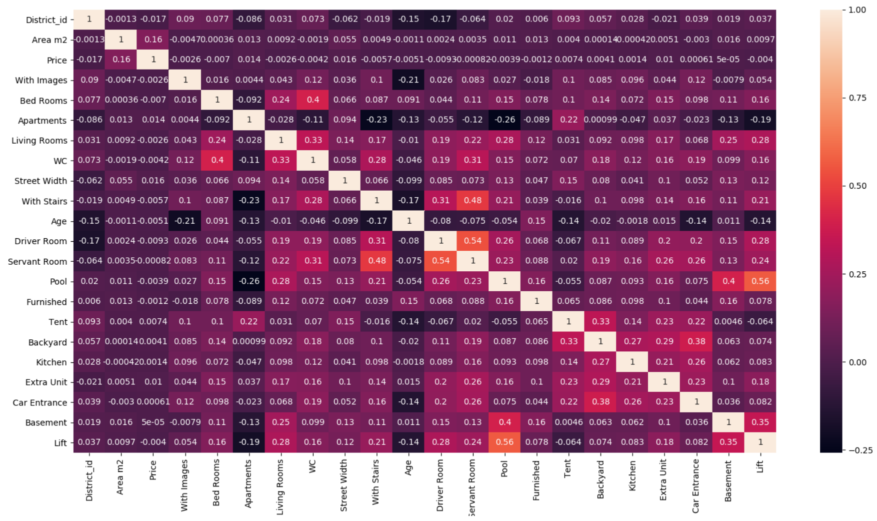
Feature Correlation

The dataset is an all numerical dataset hence, just a few processing is required. Understanding the correlation of the dataset is explained using a heatmap(a plot of correlation coefficient between all the features vector in the dataset).



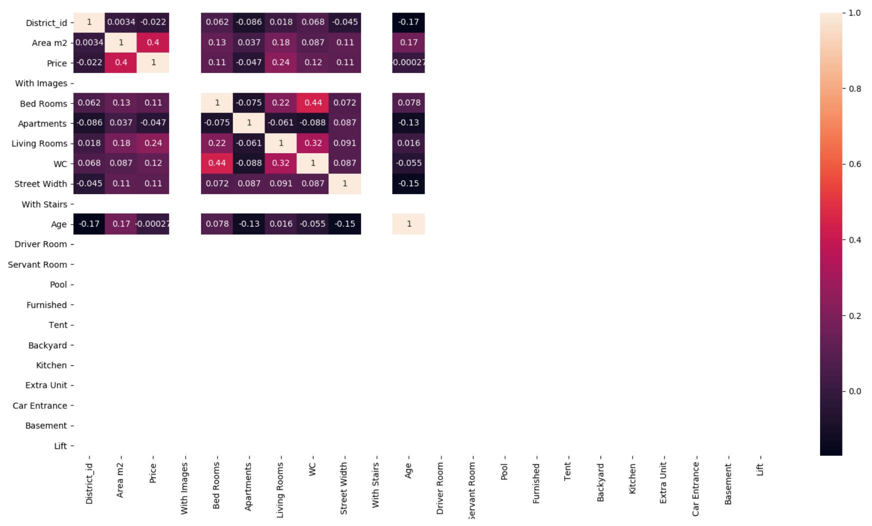
Correlation heat map of raw unscalled dataset.

We will then company this with both standardized dataset and the log scalled dataset just to check if there are meaning insights worth extracting.



Correlation heat map of Standardized dataset.

The Standardized dataset correlation heat is very similar if not the same with the unstandardized dataset. So the same inference or deductions would be made for this plot.



Correlation heat map of log dataset.

However, this is not the case here as log correlation heat map. This is because the dataset contains quite a huge number of zeros and ones. From mathematics we knw the log od zero is underfined but my function returns zero anyways. and the log of 1 is also zero. Hence, features having zeros and ones will result zero in the correlation plot. This in turn means, we not be using the log dataset when developing our model or even in feature selection. We would rather use the standardized dataset for this purpose.

Here are the deduction from the Correlation heatmap.

1. Houses with Pool most likely have a Basement and a Lift
2. Very high correlation between Price and houses with stairs..The higher the numbers of stairs in a house, the higher the prices of such houses.

3. Houses with Stairs most correlated with Pool and Lift. meaning houses with Stairs most like would have a pool and a lift.

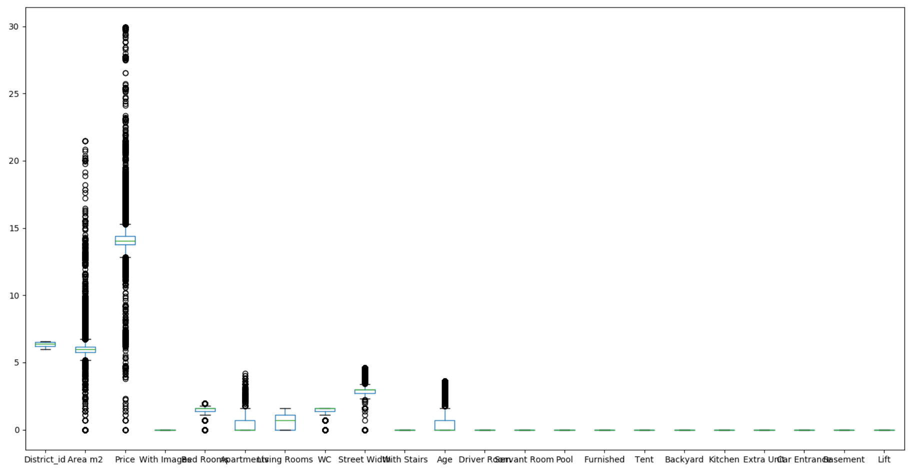
4. High apartments numbers are correlated with areas with large street widths

5. Houses with Extra units have high prices as they are highly correlated.

6. House prices have very low correlation with area of a house. As well as with basement.

Every other correlation is just normal. For example, their is a high correlation between bedrooms and WC. We know this should be true for every house. If that is true, then it is also true that that this is a dataset of mostly residential houses.

Since we know the features with high correlation, we know what is responsible for high prices and co. What we would like to do next is check for the presence of outliers.
We introduce a **boxplot** to see how much of these outliers are present in the data.



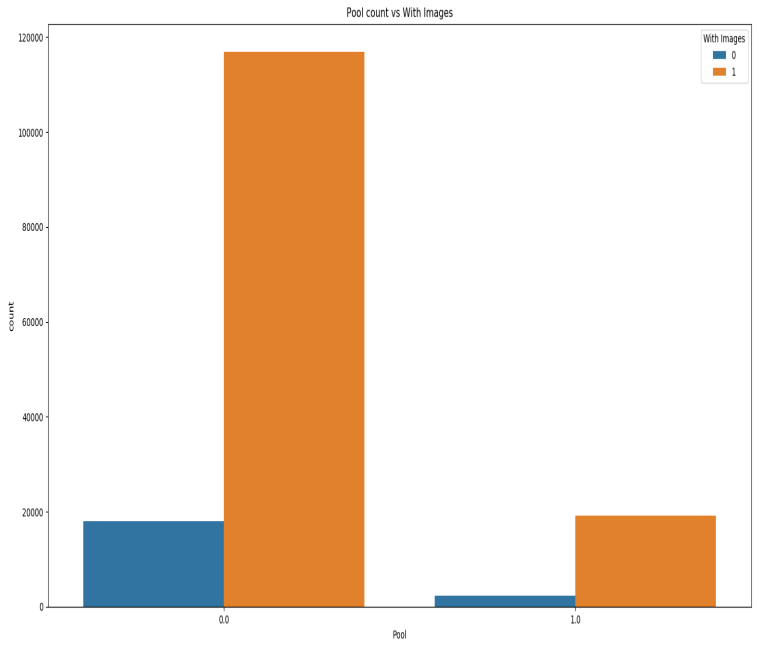
Boxplot of log data

Unlike the first dataset, this one happens to have a lot more outliers. Mostly in Prices and Area and a few couple of others.

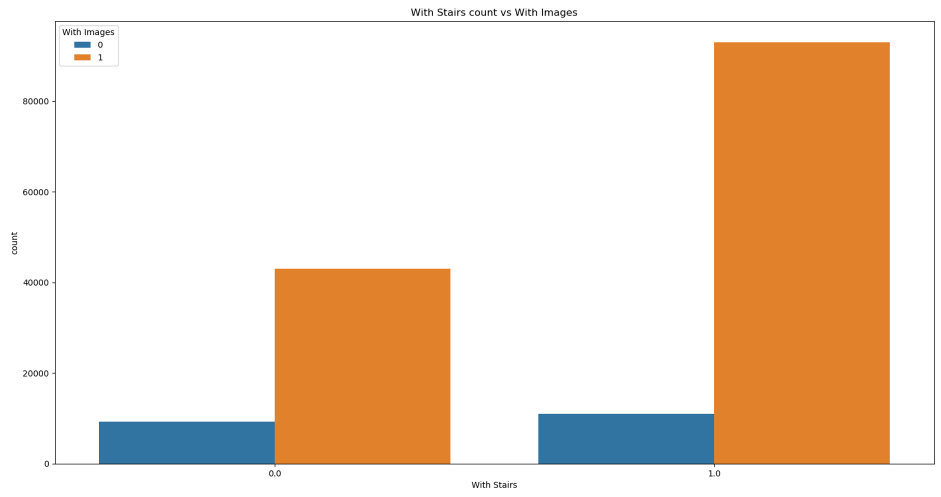
Further exploration before removing outliers

This section is meant to extract what we think may be underlying trends or facts in the dataset

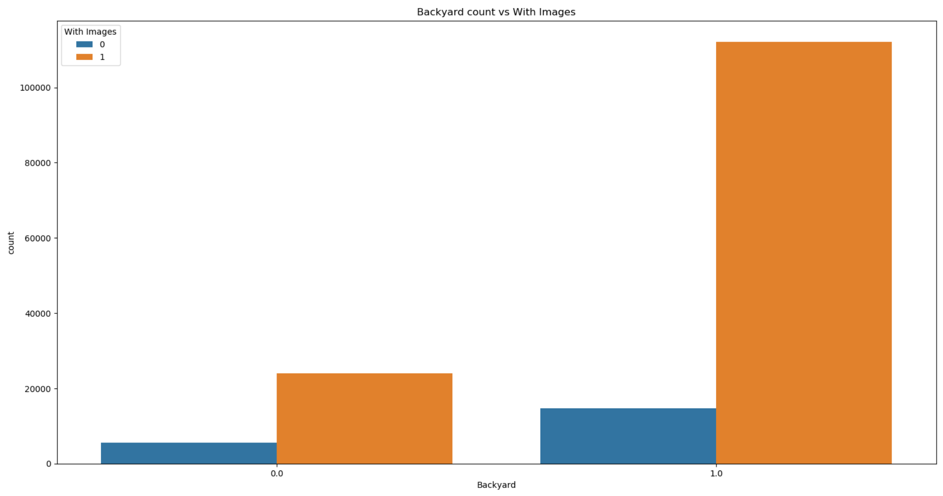
1. **Grouping and visualization**



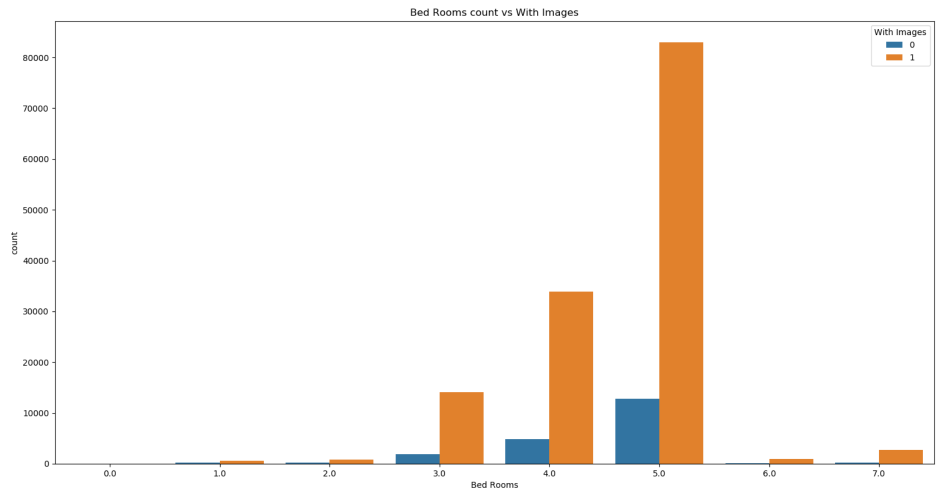
- Images are available for houses without pool than those with pool)



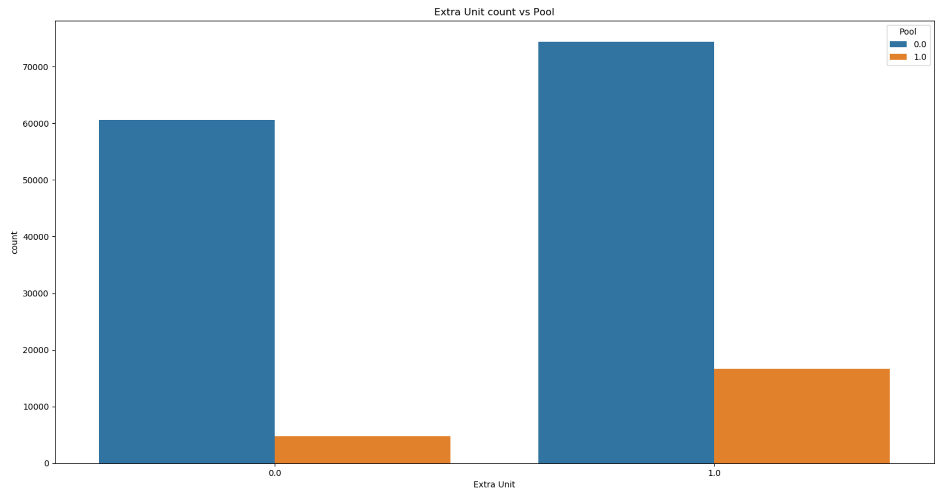
- Images are available for houses with stairs than those without



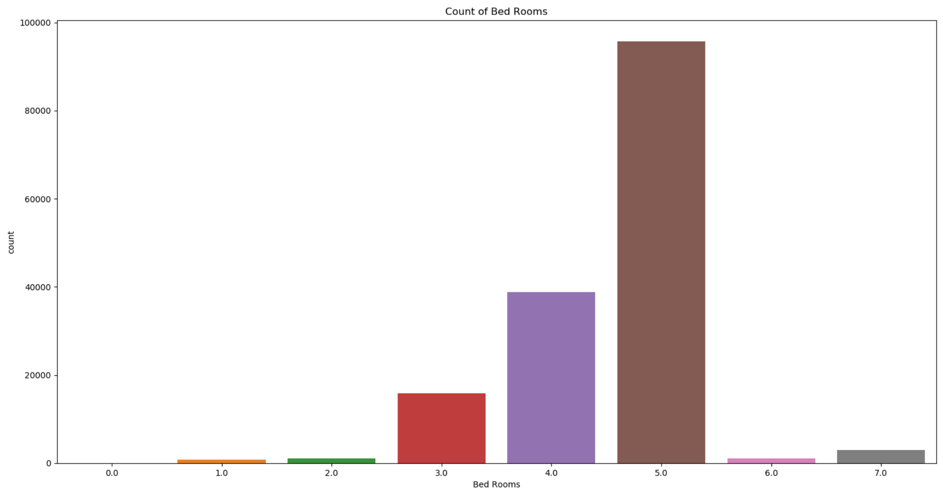
- Images are available for houses with backyeards than those without



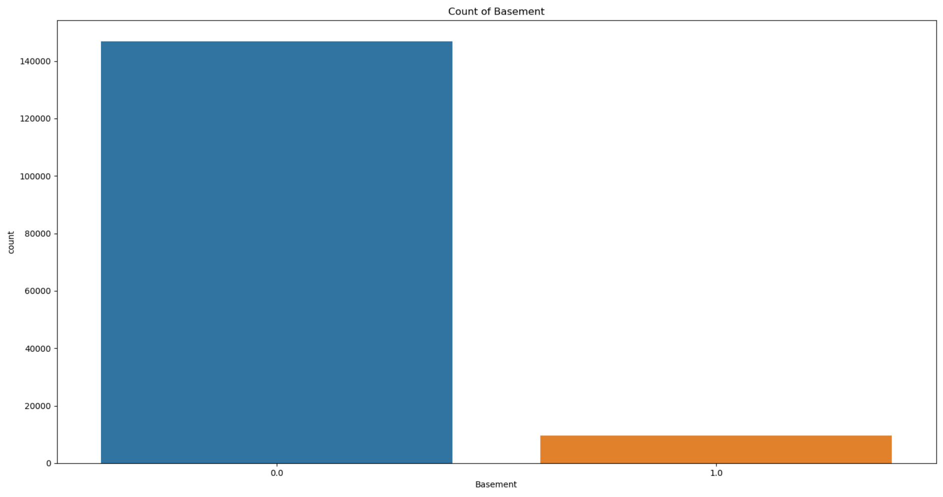
- Images are available for houses with 5,4 and 3bedrooms that than others



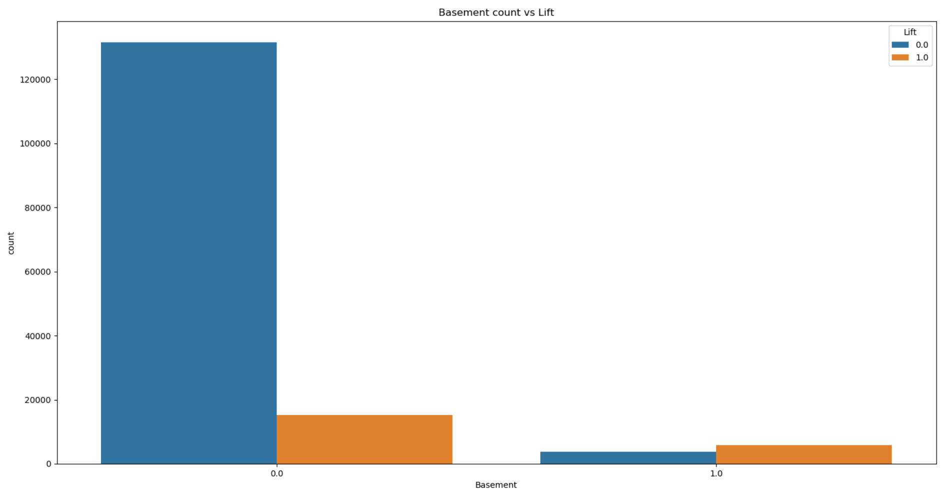
- Pool is available for houses with extra unit as well as for houses without extra unit. This means the residential houses in this dataset are all in very wealthy environ.



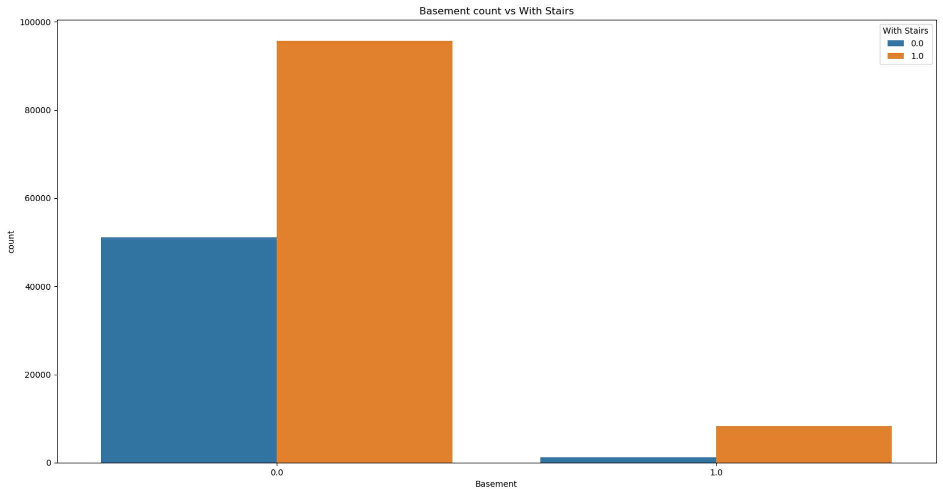
- Almost half of the houses in this dataset have 5bedrooms.



- Just a few of the houses have basement. Around 1/8th of the dataset have basement, the rest don't have

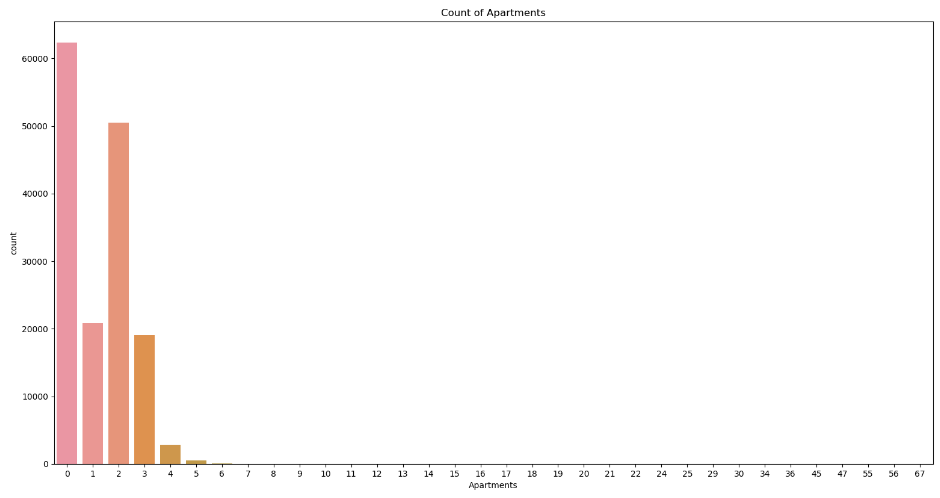


- More than half of the houses that have basement have lift. The rest would most probably have stairs as shown below.

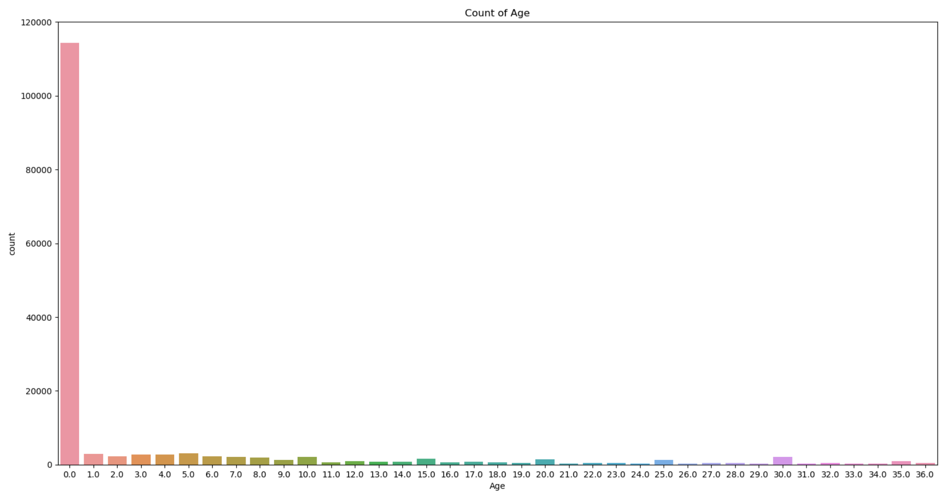


- Most houses with basement have stairs.

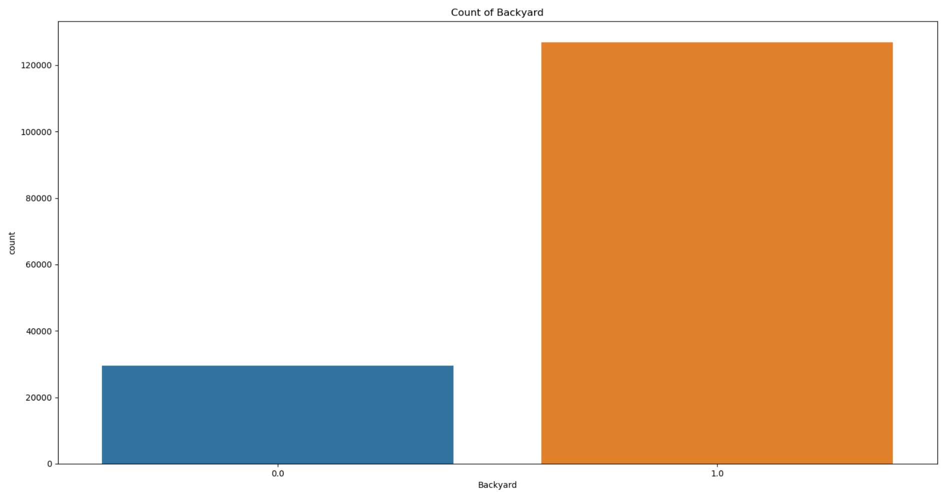
Count of some of the features



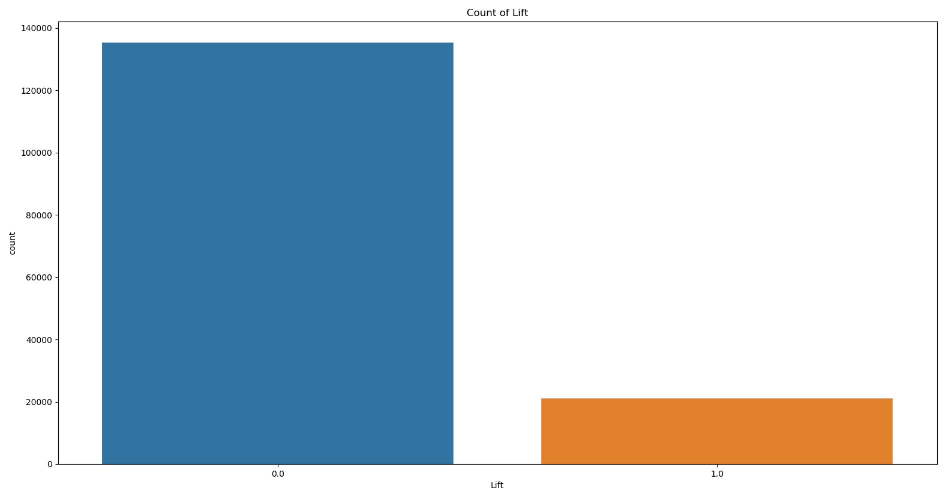
- 0 and 2 apartment houses are common in this housing dataset.



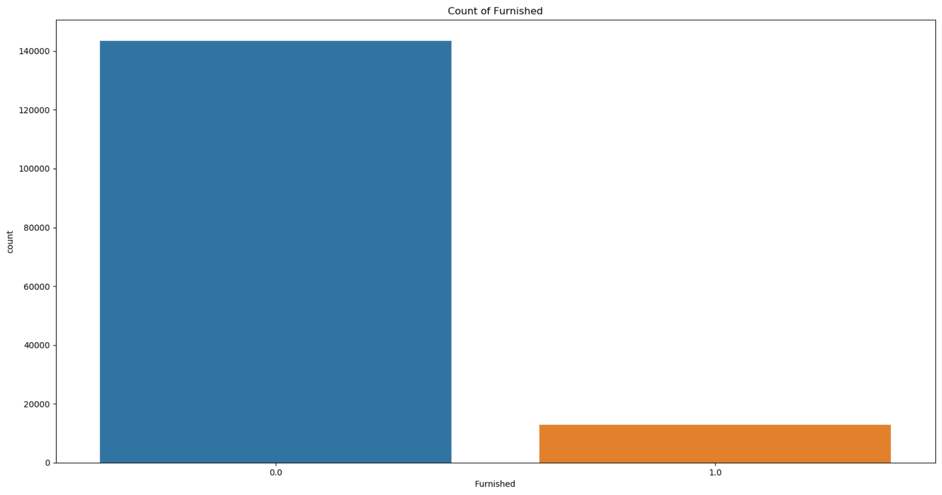
- Most of the houses are relatively new. 0years



- Most of the houses have backyard. Only 25% of them do not



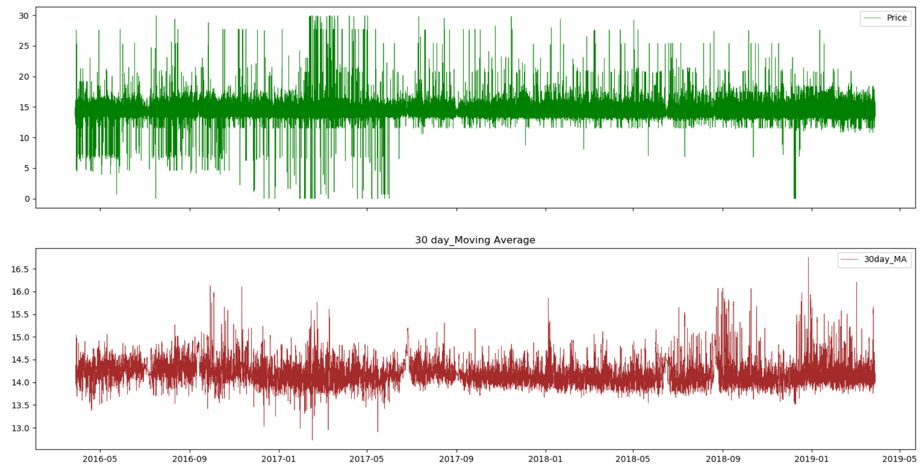
- Just like we saw earlier, most houses dont have lift. especially ones without basement.



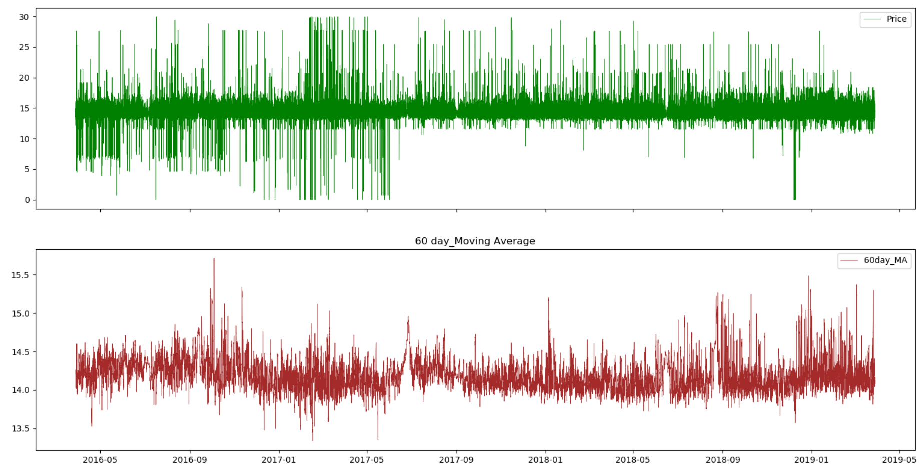
- Most of the houses here are not furnished

MOVING AVERAGES

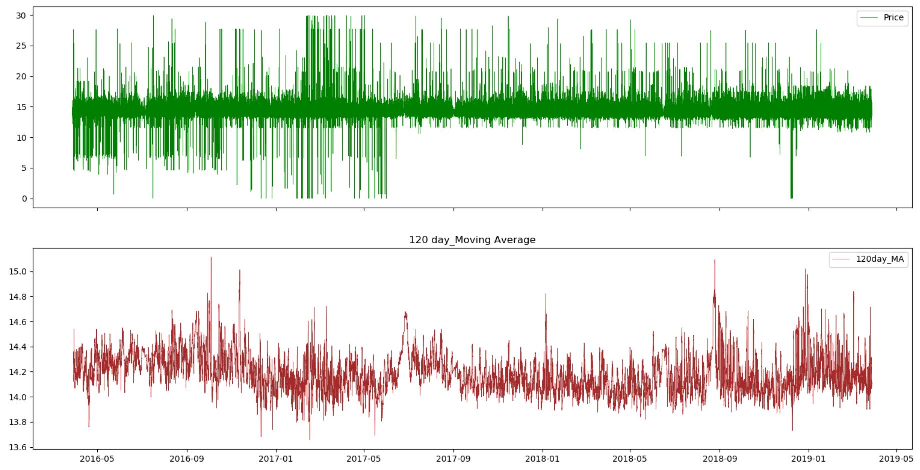
Here we would like to see how the fluctuation is price over the past 3years. Checking the trend and perhaps this would help both developers and house investors.



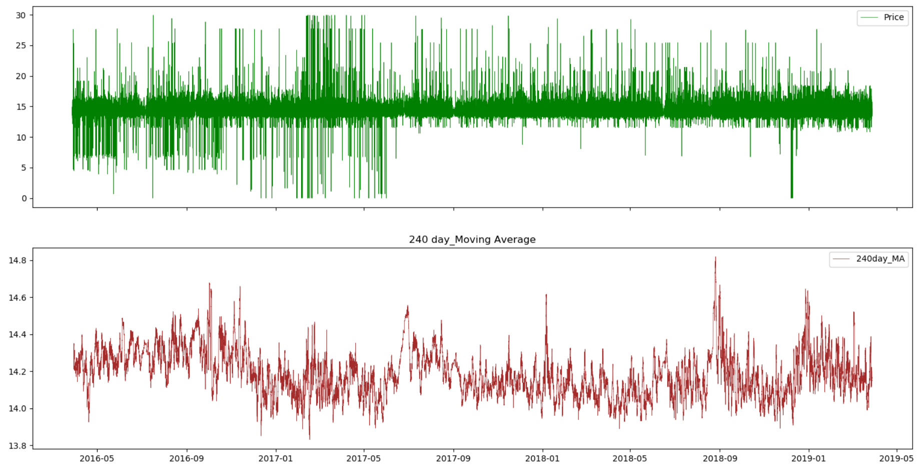
- Moving average over 30days



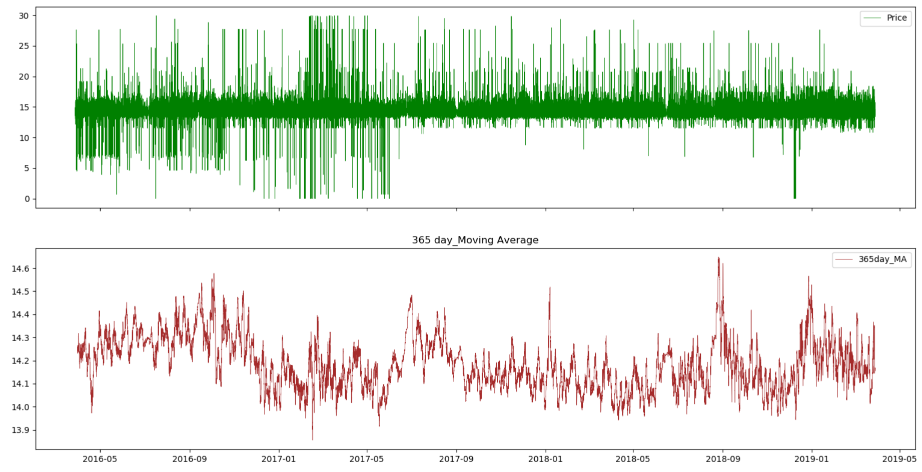
- Moving average over 60days



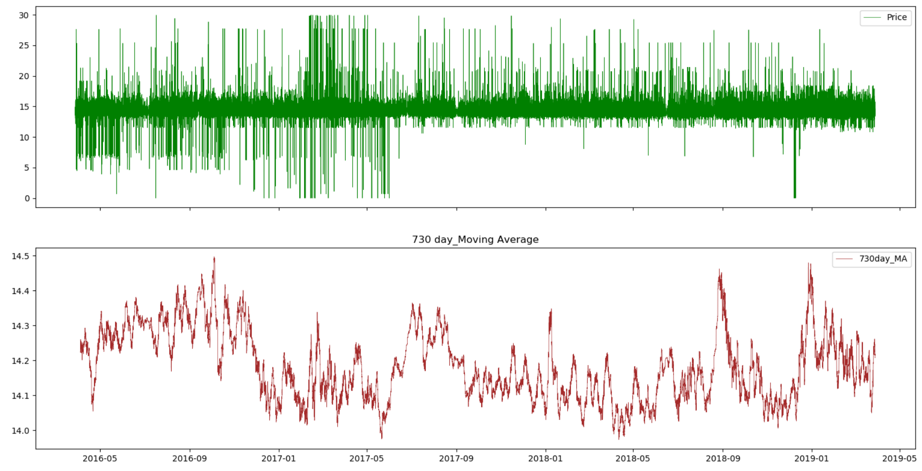
- Moving average over 120days



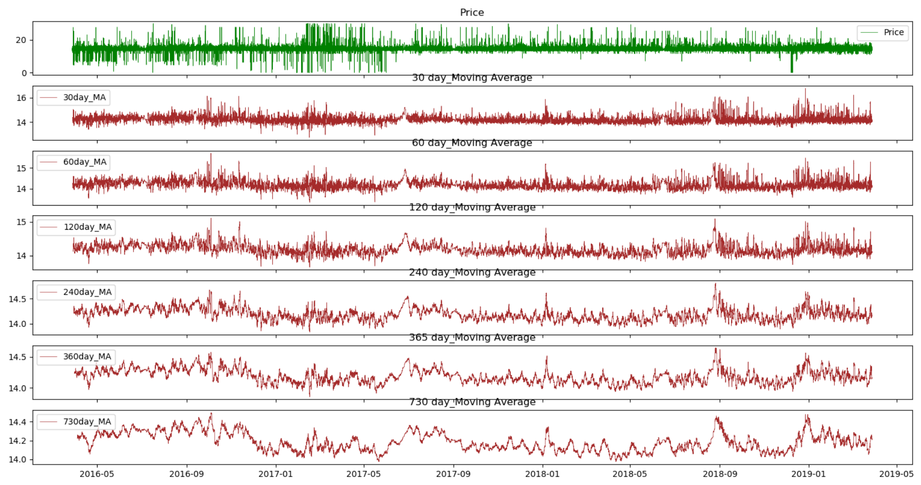
- Moving average over 240days



- Moving average over 365days



- Moving average over 730days



- Summing them all together

OBSERVATION 1: House prices were down on the 22nd of April 2016 but started moving up to reach a high on 5th of October same year.

OBSERVATION 2: 2017 started with Low prices. prices continued ranging with a reasonable high until. Prices however hit doubles highs on 5th and 15th of July same year. house prices also started declining on 25th of August same year.

OBSERVATION 3: 2018 started with Low prices but followed shortly by increased demands leading to high prices on 8th of Jan. After which prices began ranging. prices thereafter hit another high just before the end of August. precisely on the 29th of August 2018.

OBSERVATION 4: 2019 was an exception as house prices started high. Highly price was sustained and kept ranging into February of 2019.

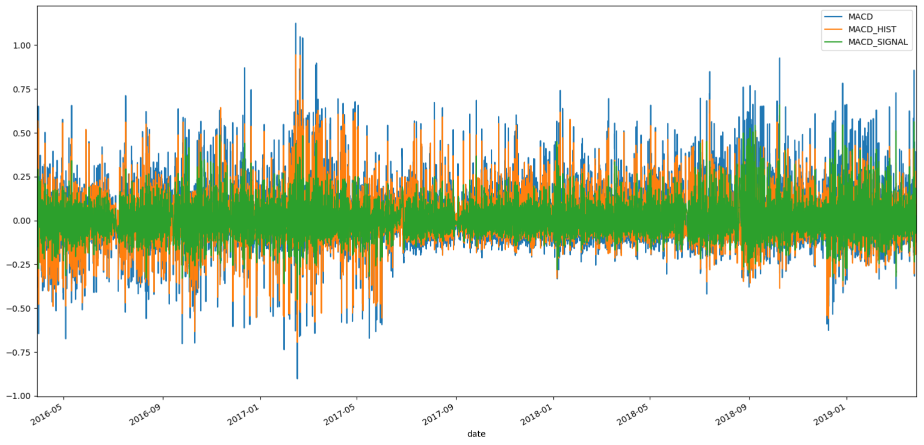
MOVING AVERAGES CONVERGENCE DIVERGENCE(MACD)

This technical formular is a signalling that is capable of telling investors when to buy or sell a house. It is as well capable of showing us the trend of prices at any given time.

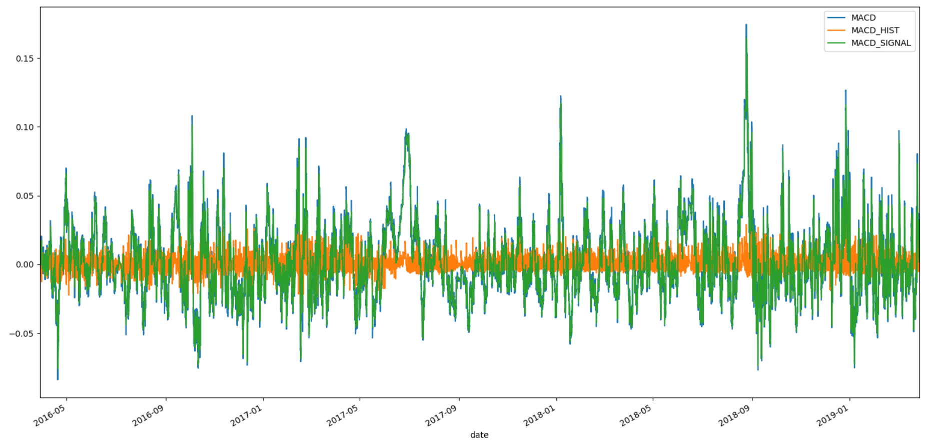
How to compute MACD

```
def ema(df, n):
  '''
  :params
    :df: feature, can be price, area or any numerical value
    :n: duration we want to check price
  '''
  return df.ewm(n).mean()

def MACD(price, n_fast, n_slow, signal):
    '''
    :Arguments:
      :n_fast: <integer> representing fast exponential
               moving average

      :n_slow: <integer> representing slow exponential
               moving average

      :signal: Signal line

    :Return:
      MACD: fast, slow and signal.
    '''

    n_fast = n_fast
    n_slow = n_slow
    signal = signal
```

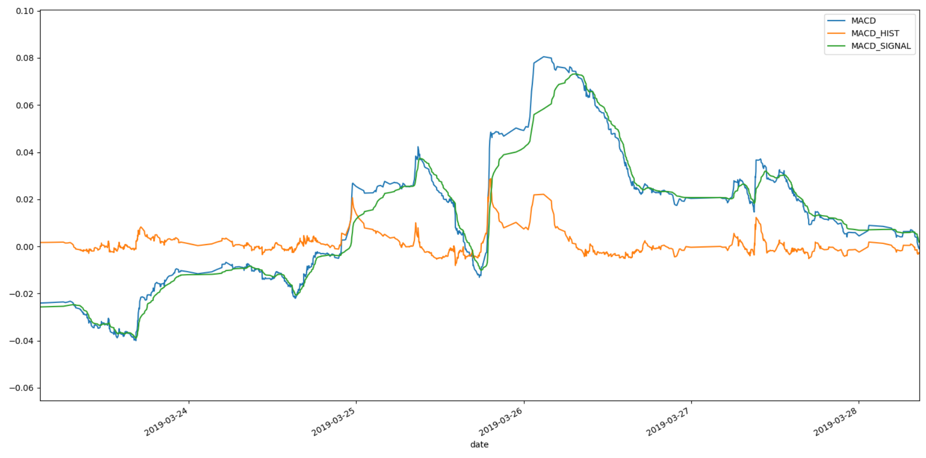
```
#defin MACD
macd = ema(price, n_fast) - ema(price, n_slow)
#MACD signal
macd_signal = ema(macd, signal)
#MACD histo
macd_histo_ = macd - macd_signal
return pd.DataFrame({'MACD': macd, 'MACD_HIST': macd_histo_,
                     'MACD_SIGNAL': macd_signal})
```



- Plot of MACD using default settings with a period of 12days



- Plot of MACD using defined setting of over 365days

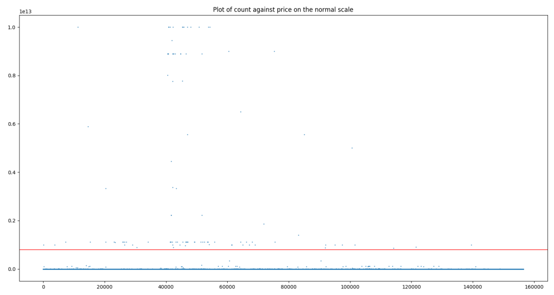


- Current house trend shows market price is going down and it may be the best time for an real estate investor to buy houses and later resale when prices are moving up.

3 Categorical data and Outliers

Luckily for us most of the entry in this dataset are all numerical. Even though they are not entirely numerical, some are just interpretation of TRUE or FALSE encoded with 0 and 1. Hence, to work with this data, we would have to standardize this dataset.

First let us understand how the data is scattered using a scatter plot.



- The red line is where we would like to cut out as outlier.

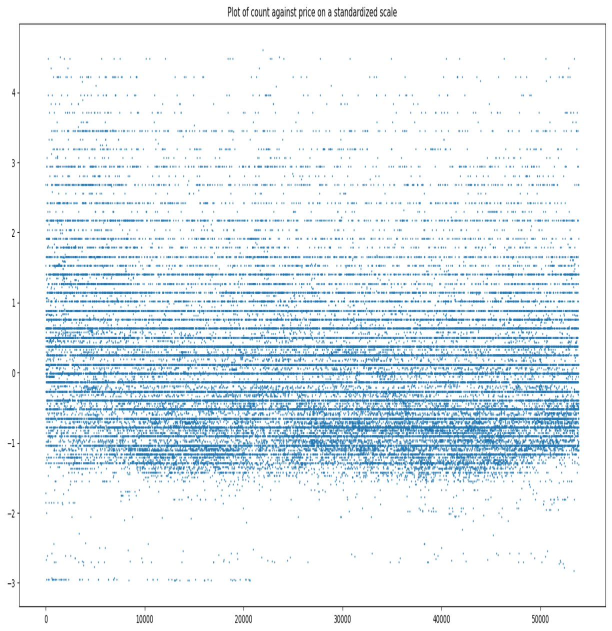
Since we already wrote a function to handle this, all we need do is just call the function and jobs done. To do this we employ the use of the function **remove_outliers**. Where we particularly made use of percentile to remove the outliers. The following coed helps us achieve this result.

```
lower_quart = .25 #using the 25th percentile for the lower quartile
upper_quart = .75 #and the 75th percentile for the upper percentile
multiplier = 1.5
quart_1 = df.quantile(lower_quartile)
quart_2 = df.quantile(upper_quartile)
diff_quart = quart_2 - quart_1
```

```
df = df[~((df < (quart_1 - 1.5 * diff_quart)) |(df > (quart_2 + 1.5
                    diff_quart))).any(axis=1)]
```

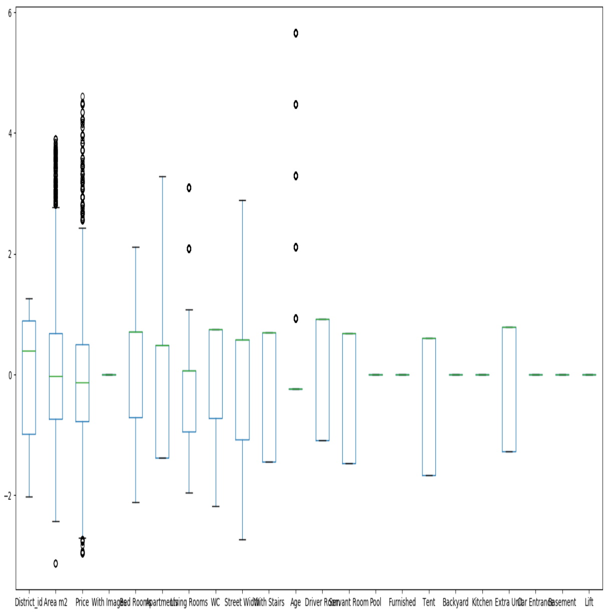
* Note that df is the house price dataset

This would have a huge impact on our dataset by cutting out more than half the data.



- Scatter plot after roving outliers. This operation removes the outliers and more than 50% of the dataset.

Now lets see how the data look using the boxplot.



- Box plot of dataset after removing outliers. You wil find the long tail in the previous box plot is no longer visible.

We have succeeded in removing most of the largely deviated data. Remember the removed data is responsible for the inbalance in our dataset by causing a positive skewness.
Removing this also affect our model as we would find out soon.

4 feature Importance and Extraction

In this section we require the use of a powerful boosting algorithm to understand which features affect housing price the most.
The findings are interesting as it shows us the features more likely to drive price fluctuations over the past 3years.

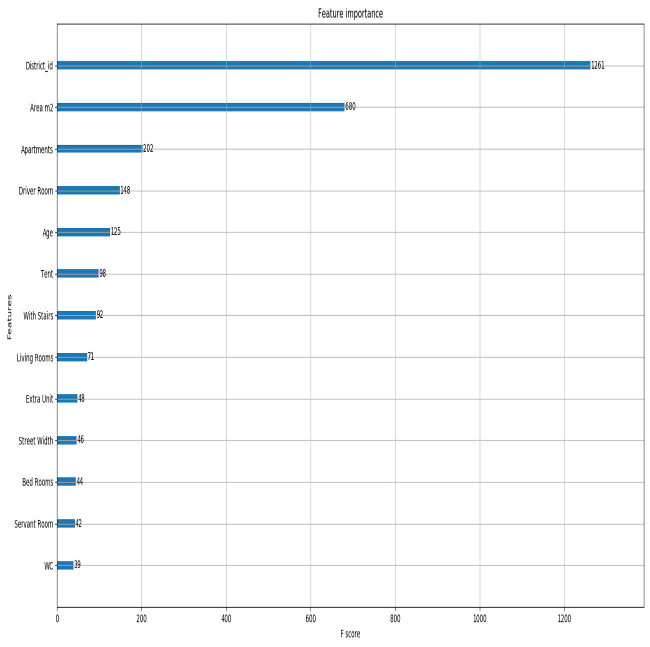
Solidifying this result requires a machine learning approach called **Gridsearch** to find the best estimation parameter for our model. This parameter would be used finally to predict the final outcome of the features.

The code required for this stage is shown below.

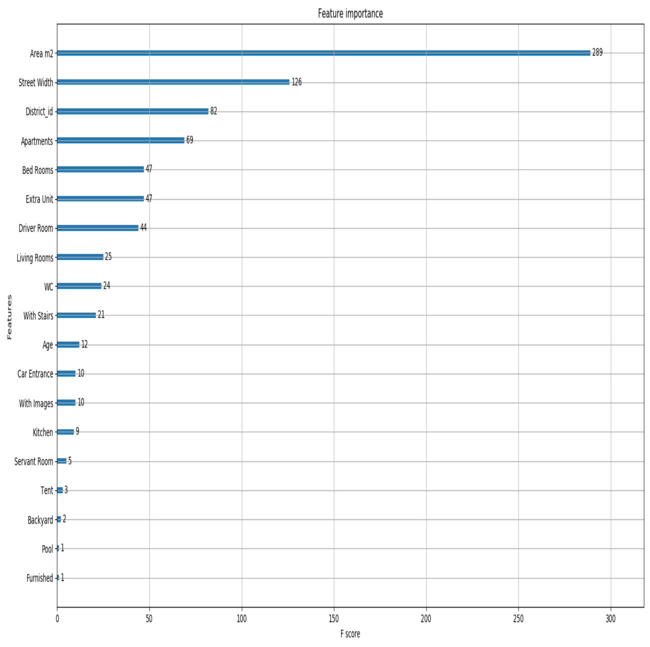
```
#plot feature importance
def plot_features(model):
  figsize = [20, 16]
  fig, ax = plt.subplots(1, 1, figsize = figsize)
  return plot_importance(model)


def categorical_handler(df, standardize = None, remove_objects = Tru
                    logg = None, normalize = None, \
                    lower_quartile = None, upper_quartile = None, \
                    multiplier = None):

  df_dum = df.copy(deep = True)
  df_num = df.copy(deep = True)
  #seperate numerical variables
  for ii in df_num.columns:
    if df_num[ii].dtypes == object:
      df_num = df_num.drop(ii, axis = 1)
  #seperate categories
  for ii in df_dum.columns:
    if df_dum[ii].dtypes != object:
      df_dum = df_dum.drop(ii, axis = 1)
  #remove outliers
  quart_1 = df_num.quantile(lower_quart)
```

```
quart_2 = df_num.quantile(upper_quart)
diff_quart = abs(quart_1 - quart_2)
df_num = df_num[~((df_num < (quart_1 - multiplier * diff_quart))\
          | (df_num > (quart_2 + multiplier * diff_quart))).any(ax
#convert categorical variables to numerical var
df_dum = pd.get_dummies(df_dum, dtype = float)
#merge
df = pd.merge(df_num.reset_index(drop = True),\
           df_dum.reset_index(drop = True), \
           left_index=True, right_index=True)

df.set_index(df_num.index, inplace = True)
#create additional time features

#standard deviation
def stdev(df):
  return np.std(df, axis = 0)
#mean deviation
def mean_dev(df):
  return df - np.mean(df, axis = 0)
#log of data
def logg_dat(df):
  return np.log(df)

#standardized values for columns
if standardize:
  for ii, ij in enumerate(df.columns):
    print(ii, ij)
    df['{}'.format(ij)] = mean_dev(df.loc[:, '{}'.format(ij)])\
                          /stdev(df.loc[:, '{}'.format(ij)])
    df = df.replace([np.inf, -np.inf, np.nan], 0)
elif logg:
  df = logg_dat(df)
  df = df.replace([np.inf, -np.inf, np.nan], 0)
elif normalize:
  for ii, ij in enumerate(df.columns):
    df['{}'.format(ij)] = (df.loc[:, '{}'.format(ij)] -
    min(df.loc[:, '{}'.format(ij)]))/\
    (max(df.loc[:, '{}'.format(ij)]) - min(df.loc[:, '{}'.format(i
    df = df.replace([np.inf, -np.inf, np.nan], 0)
else:
  pass


return df
```

1. Feature importance on full datatset (dataset containing outliers.)



- Feature importance on unstandadized data



- Feature importance on standardized dataset

- The feature importance plot clearly shows the features affecting price fluctuation. The most influencing factors Area, street width, District, Extra units and driver rooms. In essence, the first seven features can be the best influencing features.

References

A Appendix

No provided at the moment. See doc folder if any.