ADS500B - Final Group Project

Bikram Gill & Paul Jeong

Introduction

For the group project, we have chosen the housesales.csv dataset.

We have been given a set of house sales records; containing independent variables which describe the characteristics of the house, along with its sale price.

Our goal during this project is to answer the following question: **Can we use the independent variables provided in this dataset to accurately predict house sale prices?**

In order to accomplish this, we will provide an overview of the data file (Section 1), clean and prepare our data (Section 2), thoroughly review our feature set (Section 3) and then create an apply a supervised machine learning (regression model) to predict house sale prices (Section 4).

We will provide a brief summary of our findings at the conclusion of this document.

Please note - several of our key visualizations exist only in the Appendix. They will be referred to throughout this document and serve as the primary means of evaluating our dataset.

Presentation

Our final presentation video has been uploaded to Youtube. Please see link below.

https://www.youtube.com/watch?v=0W8jWpZWoAQ

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import pathlib
import os
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
pd.options.mode.chained_assignment = None


filepath = "G:/My Drive/Masters of Applied Data Sciences/Spring 2021/ADS 500B - Data Science Programming/Week 7 - Final Project/Data/house_sales.csv"
housesales_df = pd.read_csv(filepath, sep=",", header=0)
housesales_df.head()
```

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	7129300520	20141013T000000	221900.0	3.0	1.00	1180.0	5650.0	1.0	0	0	...	7	1180	0	1955	0	98178	47.5112	-122.257	1340	5650
1	6414100192	20141209T000000	538000.0	3.0	2.25	2570.0	7242.0	2.0	0	0	...	7	2170	400	1951	1991	98125	47.7210	-122.319	1690	7639
2	5631500400	20150225T000000	180000.0	2.0	1.00	770.0	10000.0	1.0	0	0	...	6	770	0	1933	0	98028	47.7379	-122.233	2720	8062
3	2487200875	20141209T000000	604000.0	4.0	3.00	1960.0	5000.0	1.0	0	0	...	7	1050	910	1965	0	98136	47.5208	-122.393	1360	5000
4	1954400510	20150218T000000	510000.0	3.0	2.00	1680.0	8080.0	1.0	0	0	...	8	1680	0	1987	0	98074	47.6168	-122.045	1800	7503

5 rows × 21 columns

1. Data Importing and Pre-processing

All actions in this section are conducted for the purpose of ensuring any data fed into our algorithm is useful and meaningful.

1.1 provides a simple summary of a data and data files.

1.2-1.4 focus on cleaning the data, removing outliers, standarizing data where required and making informed decisions to remove redundant data.

Comments have been provided throughout this exercise to describe any actions or analysis.

1.1 Import dataset and describe characteristics such as dimensions, data types, file types, and import methods used

Import Methods

Data was imported at the very beginning of this jupyter notebook, along with all necessary libraries. Pandas read_csv command was used to read the house_sales.csv file into the housesales_df dataframe.

Dimensions

We can see below there are 21,613 rows and and 21 columns in the original data set.

In [2]:
```
print(housesales_df.shape)
```

```
(21613, 21)
```

Data Types

All data types listed below. The majority of these columns are numeric in nature.

In [3]:
```
print(housesales_df.dtypes)
```

```
id                 int64
date              object
price            float64
bedrooms         float64
bathrooms        float64
sqft_living      float64
sqft_lot         float64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

File Type

The command below locates the filepath, and splits into the name and the extension (filetype). This has been printed below to show .csv.

In [4]:
```
name, filetype = os.path.splitext(filepath)
print(filetype)
```

```
.csv
```

1.2 Clean, wrangle, and handle missing data

In section 1.2, we will handle missing values. Section 1.3 will focus on removing outliers, standardizing and normalizing data.

The following command shows a count of missing values in each column of the data set.

In [5]:
```
print(housesales_df.isnull().sum())
```

```
id                 0
date               0
price              0
bedrooms        1134
bathrooms       1068
sqft_living     1110
sqft_lot        1044
floors             0
waterfront         0
view               0
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

As we can see, there are missing values in the following columns:

- bedrooms
- bathrooms
- sqft_living
- sqft_lot

We also know from 1.1 that these are all numeric data types.

The most common approach to handling missing numeric values is to use the mean, median or mode of the column as a substitute (Shah, 2020).

The mean works well for normally distibuted data, whilst the median and mode are better for skewed datasets (Shah, 2020).

In order to find the appropriate method, please see Appendix 2 which shows the boxplots of our columns.

From the analysis in the appendix, we can observe the following for the four columns - most values seem to exist within a certain range, but all exhibit outliers towards higher numbers, thus the mean does not seem to be a good option as a replacement.

In Appendix 1, the mode is presenting more than one option for the columns, and hence also does not provide great insight into a suitable replacement value.

Hence, the median will be chosen as the best replacement value - it provides the best balance of entering realistic numbers and avoiding outliers or skewed results.

See code below which replaces missing values in each column with the median.

In [6]:
```python
#Define variables to fill missing values.
##Median of each column calculated. Will be used to fill missing values.
bedrooms_median = housesales_df['bedrooms'].median().round(decimals=2)
bathrooms_median = housesales_df['bathrooms'].median().round(decimals=2)
sqft_living_median = housesales_df['sqft_living'].median().round(decimals=2)
sqft_lot_median = housesales_df['sqft_lot'].median().round(decimals=2)


#Update Nan's in each column of housesales_df dataset with Median variables.
housesales_df['bedrooms'] = housesales_df['bedrooms'].fillna(bedrooms_median)
housesales_df['bathrooms'] = housesales_df['bathrooms'].fillna(bathrooms_median)
housesales_df['sqft_living'] = housesales_df['sqft_living'].fillna(sqft_living_median)
housesales_df['sqft_lot'] = housesales_df['sqft_lot'].fillna(sqft_lot_median)

#Print new count of missing values to ensure they have been filled.
print(housesales_df.isnull().sum())
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

1.3 Transform data appropriately using techniques such as aggregation, normalization, and feature construction

In order to normalize our dataset for further use, the concepts of exploratory data analysis (Patil, 2018) and data aggregation (What is Data Aggregation?, 2019) will be used.

In essence, we will try to understand the high-level view of our data, through aggregation techniques such as summary statistics, which will then facilitate further normalization and feature construction.

Feature construction is the exercise of creating new data/columns/features from an existing dataset that will facilitate new insights for machine learning algorithms (Omarsawan, n.d.). The primary features being constructed during this step is to break apart the 'date' column into subcomponents (see below).

The describe function below prints summary statistics for the entire dataset. As most columns in housesales_df use numeric values, these statistics will be useful in observing outliers or abnormal data.

To complement the summary statistics below, we will once again refer to our boxplots in Appendix 2.

From this initial analysis, we an see the following trends:

- Normalization: Columns price, sqft_above and sqft_basement appear to have a dense clustering of outliers. These will be removed.
- Normalization: Column view is supposed to be binary (0 or 1) according to the ReadMe file. Yet the values in this column exist in a domain of 0-4 according to its boxplot in the appendix. This will be transformed.
- Feature Construction: Our hypothesis is that the time of a sale (particularly year and month) is a key component in its selling price. Hence, the date will be decomposed to create two new features, a 'sale_year' column and a 'sale_month' column.
- Comment: Columns zipcode, lat and long are numeric in value, but categorical in nature. These will not be changed at this stage.
- Comment: Any column not specifically mentioned was either deemed to exist in a suitable domain/range, or will be removed in 1.4 as redundant data.

In order to normalize price, sqft_above, and sqft_basement (remove outliers), we will use the summary statistics quartile information and interquartile range (IQR). Outliers will be defined as anything 1.5 x IQR below the first quartile and 1.5 x IQR above the third quartile.

See code below which removes outliers for these columns.

In [7]:
```python
#Retreive summary statistics to remove outliers; quartile values.
pd.set_option('display.float_format', lambda x: '%.2f' % x)
housesales_summary = housesales_df.describe()
housesales_df.describe()
```

Out[7]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
count	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00	21613.00
mean	4580301520.86	540088.14	3.35	2.12	2072.80	14814.65	1.49	0.01	0.23	3.41	7.66	1788.39	291.51	1971.01	84.40	98077.94	47.56	-122.21	1986.55	12768.46
std	2876565571.31	367127.20	0.91	0.75	891.94	40504.19	0.54	0.09	0.77	0.65	1.18	828.09	442.58	29.37	401.68	53.51	0.14	0.14	685.39	27304.18
min	1000102.00	75000.00	0.00	0.00	290.00	520.00	1.00	0.00	0.00	1.00	1.00	290.00	0.00	1900.00	0.00	98001.00	47.16	-122.52	399.00	651.00

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
25%	2123049194.00	321950.00	3.00	1.75	1450.00	5140.00	1.00	0.00	0.00	3.00	7.00	1190.00	0.00	1951.00	0.00	98033.00	47.47	-122.33	1490.00	5100.00
50%	3904930410.00	450000.00	3.00	2.25	1920.00	7620.00	1.50	0.00	0.00	3.00	7.00	1560.00	0.00	1975.00	0.00	98065.00	47.57	-122.23	1840.00	7620.00
75%	7308900445.00	645000.00	4.00	2.50	2510.00	10404.00	2.00	0.00	0.00	4.00	8.00	2210.00	560.00	1997.00	0.00	98118.00	47.68	-122.12	2360.00	10083.00
max	9900000190.00	7700000.00	33.00	8.00	12050.00	1651359.00	3.50	1.00	4.00	5.00	13.00	9410.00	4820.00	2015.00	2015.00	98199.00	47.78	-121.31	6210.00	871200.00

In [8]:
```
#Variables set to identify the price, sqft_above and sqft_basement low and high range.
#Anything outside of these ranges will be removed as an outlier.

##Define quartiles 1 and 3 for each column.
price_quartile_1 = housesales_summary.loc["25%","price"]
price_quartile_3 = housesales_summary.loc["75%","price"]

sqft_above_quartile_1 = housesales_summary.loc["25%","sqft_above"]
sqft_above_quartile_3 = housesales_summary.loc["75%","sqft_above"]

sqft_basement_quartile_1 = housesales_summary.loc["25%","sqft_basement"]
sqft_basement_quartile_3 = housesales_summary.loc["75%","sqft_basement"]


#define interquartile range (IQR) for each column.
price_IQR = (price_quartile_3 - price_quartile_1)
sqft_above_IQR = (sqft_above_quartile_3 - sqft_above_quartile_1)
sqft_basement_IQR = (sqft_basement_quartile_3 - sqft_basement_quartile_1)


#Create new subset housesales_normalized_df. All outliers will be removed from housesales_df.
##Remove price outliers from original housesales_df.
housesales_normalized_df = housesales_df.loc[(housesales_df['price'] <= (price_quartile_3 + (1.5*price_IQR))) & (housesales_df['price'] >= (price_quartile_1 - (1.5*price_IQR)))]
##Remove sqft_above outliers from new housesales_normalized_df.
housesales_normalized_df = housesales_normalized_df.loc[(housesales_normalized_df['sqft_above'] <= (sqft_above_quartile_3 + (1.5*sqft_above_IQR))) & (housesales_normalized_df['sqft_above'] >= (sqft_above_qu
##Remove sqft_basement outliers from housesales_normalized_df.
housesales_normalized_df = housesales_normalized_df.loc[(housesales_normalized_df['sqft_basement'] <= (sqft_basement_quartile_3 + (1.5*sqft_basement_IQR))) & (housesales_normalized_df['sqft_basement'] >= (
```

The 'view' column will be normalized. We can see from the data description text file that it is supposed to be binary (0 = house has no views, 1 = house has views). Our hypothesis is that view information will have an impact on house sale price, and hence this will be normalized.

The assumption being made is that 0 values are accurate, and any value >=1 should actually be 1 (house has a view). Code below will correct any values >1 to 1.

In [9]:
```
housesales_normalized_df['view'] = housesales_normalized_df['view'].apply(lambda x: 1 if x >= 1 else 0)
```

Feature Construction: 'date' column will be broken down into new 'year' and 'month' columns as discussed earlier.

First, the 'date' column is converted to datetime, and then new columns created by extracting dateparts.

In [10]:
```
#Convert 'date' column to datetime data type.
##Datatypes printed to show conversion was successful.
housesales_normalized_df.loc[:,"date"] = pd.to_datetime(housesales_normalized_df["date"])
print("date column datatype:", housesales_normalized_df.loc[:,'date'].dtypes)

housesales_normalized_df['sale_year'] = pd.DatetimeIndex(housesales_normalized_df['date']).year
housesales_normalized_df['sale_month'] = pd.DatetimeIndex(housesales_normalized_df['date']).month
housesales_normalized_df.head()
```

```
date column datatype: datetime64[ns]
```

Out[10]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15	sale_year	sale_month
0	7129300520	2014-10-13	221900.00	3.00	1.00	1180.00	5650.00	1.00	0	0	...	0	1955	0	98178	47.51	-122.26	1340	5650	2014	10
1	6414100192	2014-12-09	538000.00	3.00	2.25	2570.00	7242.00	2.00	0	0	...	400	1951	1991	98125	47.72	-122.32	1690	7639	2014	12
2	5631500400	2015-02-25	180000.00	2.00	1.00	770.00	10000.00	1.00	0	0	...	0	1933	0	98028	47.74	-122.23	2720	8062	2015	2
3	2487200875	2014-12-09	604000.00	4.00	3.00	1960.00	5000.00	1.00	0	0	...	910	1965	0	98136	47.52	-122.39	1360	5000	2014	12
4	1954400510	2015-02-18	510000.00	3.00	2.00	1680.00	8080.00	1.00	0	0	...	0	1987	0	98074	47.62	-122.05	1800	7503	2015	2

5 rows × 23 columns

1.4 Reduce redundant data and perform need based discretization

The following columns are considered redundant and will be dropped (reasons provided).

- id: this column is useful in a relational database setting, but will not provide any useful information for a machine learning algorithm. Hence will be dropped.
- date: because new features 'sale_year' and 'sale_month' were constructed, this column is now redundant.
- grade: the data description does not provide much information on the difference between 'condition' and 'grade.' As such, condition will be retained whilst grade is removed. The hypothesese is that these features are relevant, but the values in the condition column exist within a more normalized distribution.
- zipcode: location will be an important factor, but as we have latitude and longitude co-ordinates (which are more specific and likely to work better in a machine learning algorithm), zipcode is dropped.
- sqft_living15: this column is not listed in the data description text file and it is unclear what it represents. It will hence be dropped.
- sqft_lot15: same reason as sqft_living15.

See code below which only retains columns not deemed redundant.

In [11]:
```
#Remove redundant data columns
housesales_normalized_df = housesales_normalized_df.drop(['id','date','grade','zipcode','sqft_living15','sqft_lot15'], axis=1)
housesales_normalized_df.head()
```

Out[11]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	lat	long	sale_year	sale_month
0	221900.00	3.00	1.00	1180.00	5650.00	1.00	0	0	3	1180	0	1955	0	47.51	-122.26	2014	10
1	538000.00	3.00	2.25	2570.00	7242.00	2.00	0	0	3	2170	400	1951	1991	47.72	-122.32	2014	12
2	180000.00	2.00	1.00	770.00	10000.00	1.00	0	0	3	770	0	1933	0	47.74	-122.23	2015	2
3	604000.00	4.00	3.00	1960.00	5000.00	1.00	0	0	5	1050	910	1965	0	47.52	-122.39	2014	12
4	510000.00	3.00	2.00	1680.00	8080.00	1.00	0	0	3	1680	0	1987	0	47.62	-122.05	2015	2

One form of need-based discretization of data that can assist machine learning algorithms is converting continuous data points into meaningul bins or intervals (Gupta, 2019).

The following columns are good canidates for discretization, as they provide valuable information but would be easier to interpret in smaller domain of values.

- sqft_basement:

The hypothesis here is that the information on whether a house has a basement is more important to the sale price than the exact sqft of the basement itself, particularly as we already have information on the sqft of the lot and living room. Based on the boxplot of sqft_basement in Appendix 2 and removal of outliers in section 1.3, sqft_basement will be converted into the following intervals - 0=no basement, 1=basement between 0-500, 2= basement between 501-100, 3 = basement > 1000.

In [12]:
```
#Convert sqft_basement into bins mentioned above.
housesales_normalized_df.loc[(housesales_normalized_df['sqft_basement'] >0) & (housesales_normalized_df['sqft_basement'] <= 500),'sqft_basement']=1
housesales_normalized_df.loc[(housesales_normalized_df['sqft_basement'] >500) & (housesales_normalized_df['sqft_basement'] <= 1000),'sqft_basement']=2
housesales_normalized_df.loc[housesales_normalized_df['sqft_basement'] >1000,'sqft_basement']=3

housesales_normalized_df.head()
```

Out[12]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	lat	long	sale_year	sale_month
0	221900.00	3.00	1.00	1180.00	5650.00	1.00	0	0	3	1180	0	1955	0	47.51	-122.26	2014	10
1	538000.00	3.00	2.25	2570.00	7242.00	2.00	0	0	3	2170	1	1951	1991	47.72	-122.32	2014	12

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	lat	long	sale_year	sale_month
2	180000.00	2.00	1.00	770.00	10000.00	1.00	0	0	3	770	0	1933	0	47.74	-122.23	2015	2
3	604000.00	4.00	3.00	1960.00	5000.00	1.00	0	0	5	1050	2	1965	0	47.52	-122.39	2014	12
4	510000.00	3.00	2.00	1680.00	8080.00	1.00	0	0	3	1680	0	1987	0	47.62	-122.05	2015	2

Our dataset has now been cleaned substantially. However, before creating our machine learning algorithm to attempt to use the independent variables in the data set to create a house sale price prediciton algorithm; there will be a final review of the data based on the knowledge gathered in section 3.

2. Data Importing and Pre-processing

2.1 Identify categorical, ordinal, and numerical variables within data

Categorical variables are descriptive in nature, and values within such a variable column are usually one of a finite list of choices (values are often repeated). Categorical variables can exist in numerical form - known as nominal variables (Shah, 2020)

Ordinal variables imply some form of ranking amongst the data, or signify a different class level. E.g. military ranks may be numeric, and distinguish one soldier from another. (Shah, 2020)

Finally, numerical variables exist within an infinite set of numbers, but can be categorized further (interval, continuous, ratio etc.).

- Price: numerical (interval)
- Bedrooms: numerical (interval)
- Bathrooms: numerical (interval)
- sqft_living: numerical (interval)
- sqft_lot: numerical (interval)
- floors: numerical (interval)
- waterfront: categorical (nominal)
- view: categorical (nominal)
- condition: ordinal
- grade: ordinal
- sqft_above: numerical (interval)
- sqft_basement: numerical (interval)
- yr_built: numerical (interval)
- yr_renovated: numerical (interval)
- lat: numerical (continuous)
- long: numerical (continuous)
- sale_year: numerical (integer)
- sale_month: numerical (integer)

2.2 Provide measures of centrality and distribution with visualizations

From section 1, measures of centrality/dispersion were created with **Appendix 1 - Mean, Median, Mode** and **Appendix 2 - Box Plots**.

To aid these, **Appendix 3 - Histograms** has been created to aid with visual analysis of the distributions of the data in question.

From the histograms in Appendix 3, we see that most values in our columns are heavily distributed in a small range, however from Appendix 2 (Boxplots), we also see that most columns show heavy outliers towards higher values.

This makes sense in the context of our dataset, most houses will have a certain minimum area, size, bedrooms etc., but outliers will exist where homes are unusually large.

2.3 Diagnose for correlations between variables and determine independent and dependent variables

A correlation matrix of the data set has been provided below to establish the relationship between the independent variables and house sale price. This will help with section 2.4 to discover patterns of interest and final feature selection for the machine learning model.

This has also been converted to a visualization for easier interpretation, through the correlation matrix visualisation.

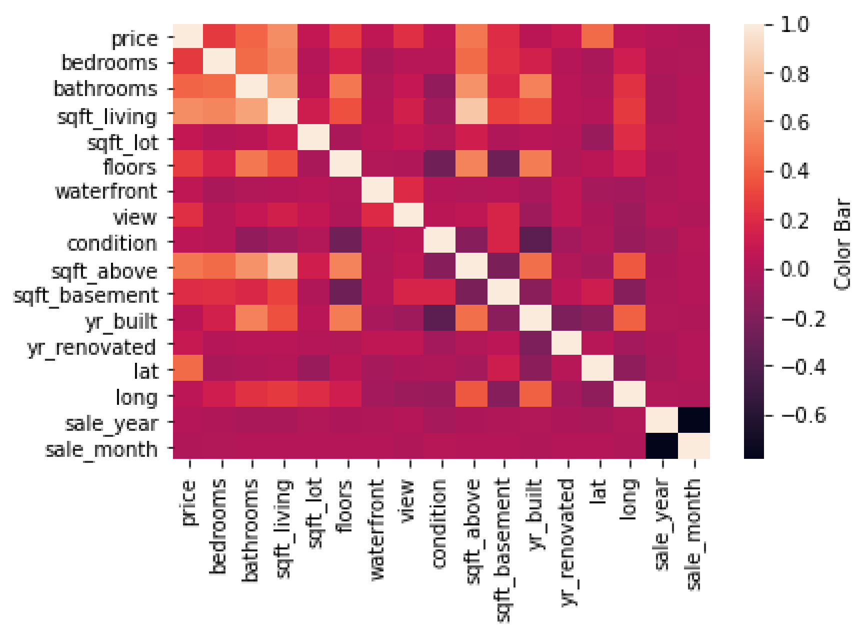
In our dataset, price is the dependent variable whilst everything else is independent.

In [13]:
```
#Correlation matrix in table format.
housesales_normalized_df.corr().head(1)
```

Out[13]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	lat	long	sale_year	sale_month
price	1.00	0.27	0.42	0.58	0.07	0.27	0.06	0.22	0.05	0.50	0.21	0.04	0.09	0.44	0.04	0.01	-0.02

In [14]:
```
#Correlation matrix in visual format.
corrMatrix1=housesales_normalized_df.corr()
sns.heatmap(corrMatrix1, annot=False, cbar_kws={'label': 'Color Bar'})
plt.show()
Ebrahim, M. (2019, March 26). Seaborn heatmap tutorial (Python Data Visualization). Like Geeks. https://likegeeks.com/seaborn-heatmap-tutorial/
```



2.4 Perform exploratory analysis in combination with visualization techniques to discover patterns and features of interest

Brownlee (2020) provides insight on best practices for feature selection for regression problems. We have decided to go with the correlation approach.

Essentially, what we are looking for are features that have a high correlation (positive or negative) to house sale price.

From our correlation analysis above, there are not too many features that exhibit a high correlation to price, and many have quite a low correlation. Based on the values observed above, *good* correlation for the purpose of our house sale price prediction will be anything <-0.4 or >0.4.

The columns that fit this criteria are **bathrooms, sqft_living, sqft_above and lat.**

Note that many of the columns we cleaned or normalized did not make the cut - this is simply part of the process. We converted these data points into their best form to see if they would be helpful for our problem, and now can confirm they are not.

The code below subsets our dataframe to just the remaining columns.

In [15]:
```
housesales_regression_df = housesales_normalized_df[['bathrooms', 'sqft_living', 'sqft_above', 'lat', 'price']]
housesales_regression_df.head()
```

Out[15]:

	bathrooms	sqft_living	sqft_above	lat	price
0	1.00	1180.00	1180	47.51	221900.00
1	2.25	2570.00	2170	47.72	538000.00
2	1.00	770.00	770	47.74	180000.00
3	3.00	1960.00	1050	47.52	604000.00
4	2.00	1680.00	1680	47.62	510000.00

3. Data Analytics

3.1 Determine the need for a supervised or unsupervised learning method and identify dependent and independent variables

The main difference between supervised and unsupervised learning algorithms comes from the presence (or lack) of labelled data and results.

Supervised algorithms are given labelled datasets where the algorithm can use known 'answers' to validate its predictions or outputs (Salian, 2018).

This is applicable to our use case. We will use the actual house sale prices as the answer key to train our model. Regression analysis will be used to find relationships between our final chosen variables and the sale price on a majority subset of the data, which in turn will allow us to predict values and compare against a smaller unseen subset.

Our final chosen variable list is as below:

- Independent (Predictor) Variables: bathrooms, sqft_living, sqft_above and lat
- Dependent (Outcome) Variables: price

3.2 Train, test, and provide accuracy and evaluation metrics for model results

Prepare the model

Below we are preparing our prediction model.

Price is assigned to the y variable (outcome) and the remaining features assigned to x (predictors). We will scale the x variables to ensure they exist within a certain range and that features with higher values do not inadvertently weight the regression equation.

In [16]:
```python
# Initial variables created (x and y)
y = housesales_regression_df['price']
x = housesales_regression_df[['bathrooms', 'sqft_living', 'sqft_above', 'lat']]

#x variables are scaled. Chosen method if MinMaxScaler which converts each variable to a value between 0 and 1.
x_scaled = MinMaxScaler().fit_transform(x)
x_scaled = sm.add_constant(x_scaled)

# Variables split into training and test sets (80/20).
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2)
```

Fit the model and create regression equation

The model has been fitted and OLS Regression Results provided below. These results also list the constant and co-efficients for the scaled features, which act as the final regression equation.

Because the features have been scaled, the co-efficients are much higher than if the raw data was used.

The OLS Regression Results will be one of our key evaluation metrics, which will be explore after the predictions are created using the regression equation below.

In [17]:
```python
lr_model = sm.OLS(y_train,x_train).fit()

print(lr_model.summary())
print(lr_model.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.533
Model:                            OLS   Adj. R-squared:                  0.533
Method:                 Least Squares   F-statistic:                     4536.
Date:                Sun, 18 Apr 2021   Prob (F-statistic):               0.00
Time:                        11:17:13   Log-Likelihood:            -2.1092e+05
No. Observations:               15916   AIC:                         4.218e+05
Df Residuals:                   15911   BIC:                         4.219e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -8.04e+04   4700.531    -17.105      0.000   -8.96e+04   -7.12e+04
x1         9.916e+04   1.26e+04      7.883      0.000    7.45e+04    1.24e+05
x2         5.937e+05   1.46e+04     40.628      0.000    5.65e+05    6.22e+05
x3         1.298e+05   1.02e+04     12.780      0.000     1.1e+05     1.5e+05
x4         3.975e+05   4840.391     82.115      0.000    3.88e+05    4.07e+05
==============================================================================
Omnibus:                     1757.670   Durbin-Watson:                   2.025
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2793.683
Skew:                           0.796   Prob(JB):                         0.00
Kurtosis:                       4.295   Cond. No.                         21.5
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
const   -80400.96
x1       99157.71
x2      593669.36
x3      129816.23
x4      397469.75
dtype: float64
```

Create the predictions

The data below takes the regression equation from the training set and applies to the test set. We will evaluate the results further below.

In [18]:
```python
predictions = pd.DataFrame(lr_model.predict(x_test))
predictions.columns = ['Predicted_HouseSalesPrice']
print(predictions)
```

```
      Predicted_HouseSalesPrice
0                     513431.11
1                     352288.98
2                     508319.77
3                     183611.58
4                     597517.45
...                         ...
3975                  424390.43
3976                  561263.84
3977                  408961.58
3978                  390582.25
3979                  386816.15

[3980 rows x 1 columns]
```

Evaluation metrics and comments

The focus of this evaluation will be on two main components; the OLS Regression Results from above and graphical representations below.

- **R-Squared value is 0.53 and std errors are quite high (Yadav, 2019)**

This shows that while the regression equation has established a meaningful relationship between our predictor and outcome variables, there is a sizeable distance between the regressin equation line and most of the actual values (i.e., the model is not terrible, but also not highly accurate).

- **Prob (F-Statistic) is 0.00 (Yadav, 2019)**

This shows that we can reject the null hypothesis, or in other words, the regression did produce a meaningful outcome.

- **Other statistics (Interpreting Regression Output, n.d.)**

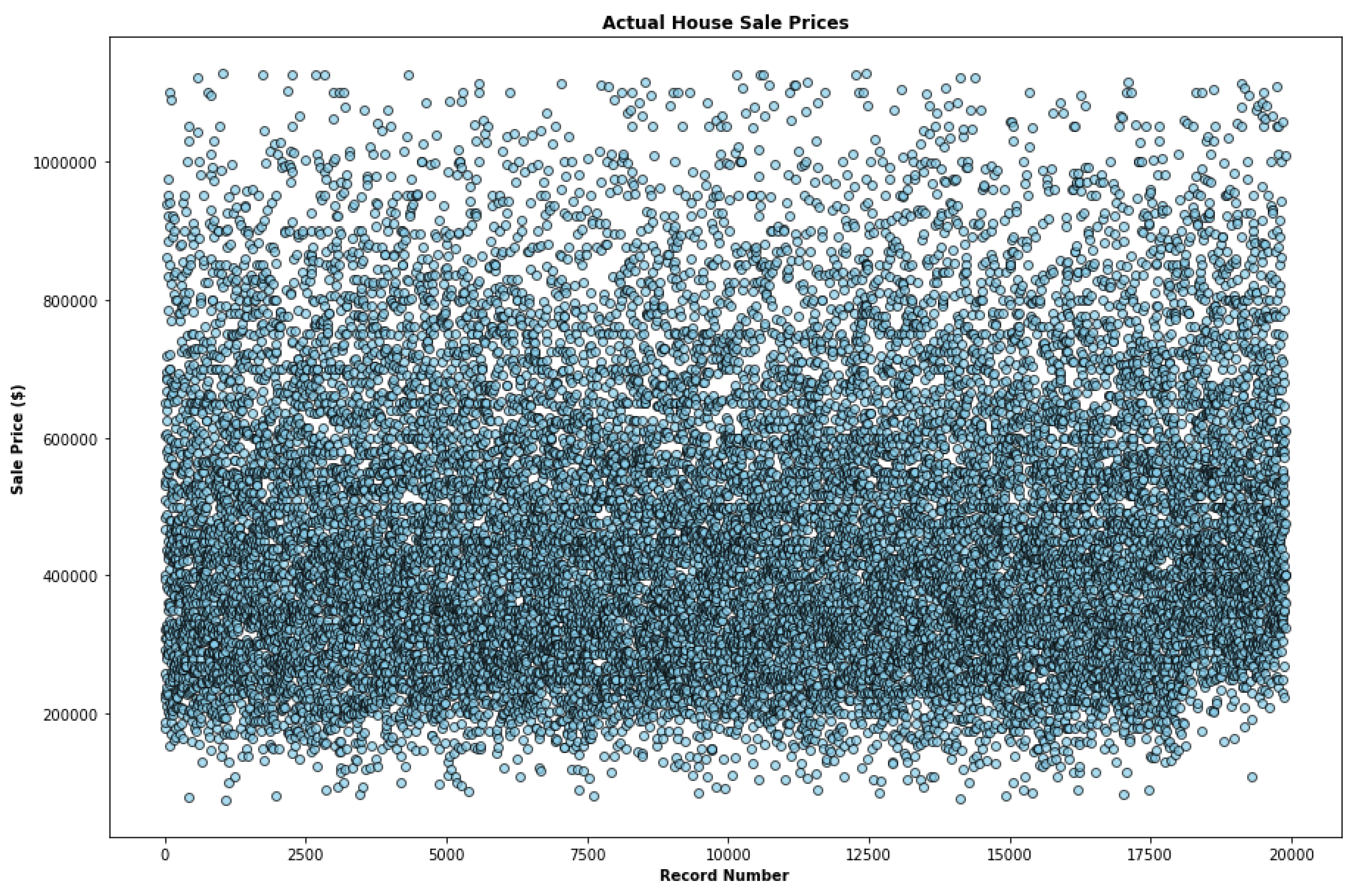
The t value backs up the findings of r-squared and std errors. t values are reasonably high in this model, meaning there is a higher standard error (deviation from regression line of actual values).

- **Graphical interpretation**

Finally, see graphic below. We have plotted the actual house sales prices in a scatter plot. We see such a vast array of values that it would be very difficult to establish a regression line that fits this data set in a meaningul way.

Our overall finding from this exercise is that there may indeed be some set of variables that could be used to predict house sales price with high degree of confidence, but we were not able to find them in this specific data set. However, our predictive model shows that there is some relationship between these variables and house sale price; and would warrant further investigation/iteration (outside the scope of this assignment).

In [19]:
```
#Plot all actual house sale prices against regression equation from above.
##Actuals
plt.figure(figsize=(15,10))
housesales_regression_df = housesales_regression_df.reset_index()
housesales_regression_df['RecordNo'] = housesales_regression_df.index

plt.scatter(housesales_regression_df['RecordNo'],
            housesales_regression_df['price'],
            color="skyblue",
            alpha=0.7,
            ec="black")
plt.ticklabel_format(style='plain')
plt.title("Actual House Sale Prices",fontweight='bold')
plt.xlabel("Record Number",fontweight='bold')
plt.ylabel("Sale Price ($)",fontweight='bold')
plt.show()
```



References

Brownlee, J. (2020). *How to perform feature selection for regression data.* Machine Learning Mastery. https://machinelearningmastery.com/feature-selection-for-regression-data/#:~:text=Feature%20selection%20is%20the%20process,target%20for%20regression%20predictive%20modeling.

Ebrahim, M. (2019, March 26). *Seaborn heatmap tutorial (Python Data Visualization)*. Like Geeks. https://likegeeks.com/seaborn-heatmap-tutorial/

Gupta, R. (2019). *An introduction to discretization in data science.* towards data science. https://towardsdatascience.com/an-introduction-to-discretization-in-data-science-55ef8c9775a2.

Interpreting Regression Output. (n.d.). *Princeton University Library.* Retrieved April 10, 2021 from https://dss.princeton.edu/online_help/analysis/interpreting_regression.htm.

Omarsawan. (n.d.). *Feature-construction-and-Categorical-features-tutorial.* GitHub. https://github.com/Omarsawan/Feature-construction-and-Categorical-features-tutorial/blob/master/feature_construction_and_categorical_features.ipynb.

Patil, P. (2018). *What is exploratory data analysis?* towards data science. https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15.

Salian, I. (2018). *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* NVIDIA. https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/#:~:text=In%20a%20supervised%20learning%20model,and%20patterns%20on%20its%20own

Shah, C. (2020). *A Hands-on Introduction to Data Science.* Cambridge University Press, p. 70.

What is Data Aggregation? Examples of Data Aggregation by Industry (2019, October 22). Import.io. Retrieved April 01, 2021 from https://www.import.io/post/what-is-data-aggregation-industry-examples/.

Yadav, J. (2019). *Statistics: How Should I interpret results of OLS?* https://jyotiyadav99111.medium.com/statistics-how-should-i-interpret-results-of-ols-3bde1ebeec01.

Appendix

Appendix 1 - Mean, Median, Mode - All Columns

Mean, Median and Mode statistics have been created to help with exploratory analysis throughout our assignment.

In [20]:
```
mean = housesales_df['price'].mean().round(decimals=2)
median = housesales_df['price'].median().round(decimals=2)
stddev = housesales_df['price'].std().round(decimals=2)
mode = housesales_df['price'].mode(dropna=True).round(decimals=2)

print("price Mean = " + repr(mean))
print("price Median = " + repr(median))
print("price Std Dev = " + repr(stddev))
print("price Mode = " + repr(mode))
```

```
price Mean = 540088.14
price Median = 450000.0
price Std Dev = 367127.2
price Mode = 0   350000.00
1   450000.00
dtype: float64
```

In [21]:
```
mean = housesales_df['bedrooms'].mean().round(decimals=2)
median = housesales_df['bedrooms'].median().round(decimals=2)
stddev = housesales_df['bedrooms'].std().round(decimals=2)
mode = housesales_df['bedrooms'].mode(dropna=True).round(decimals=2)

print("bedrooms Mean = " + repr(mean))
print("bedrooms Median = " + repr(median))
print("bedrooms Std Dev = " + repr(stddev))
print("bedrooms Mode = " + repr(mode))
```

```
bedrooms Mean = 3.35
bedrooms Median = 3.0
bedrooms Std Dev = 0.91
bedrooms Mode = 0   3.00
dtype: float64
```

In [22]:
```
mean = housesales_df['bathrooms'].mean().round(decimals=2)
median = housesales_df['bathrooms'].median().round(decimals=2)
stddev = housesales_df['bathrooms'].std().round(decimals=2)
mode = housesales_df['bathrooms'].mode(dropna=True).round(decimals=2)

print("bathrooms Mean = " + repr(mean))
print("bathrooms Median = " + repr(median))
print("bathrooms Std Dev = " + repr(stddev))
print("bathrooms Mode = " + repr(mode))
```

```
bathrooms Mean = 2.12
bathrooms Median = 2.25
bathrooms Std Dev = 0.75
bathrooms Mode = 0    2.50
dtype: float64
```

In [23]:

```
mean = housesales_df['sqft_living'].mean().round(decimals=2)
median = housesales_df['sqft_living'].median().round(decimals=2)
stddev = housesales_df['sqft_living'].std().round(decimals=2)
mode = housesales_df['sqft_living'].mode(dropna=True).round(decimals=2)

print("sqft_living Mean = " + repr(mean))
print("sqft_living Median = " + repr(median))
print("sqft_living Std Dev = " + repr(stddev))
print("sqft_living Mode = " + repr(mode))
```

```
sqft_living Mean = 2072.8
sqft_living Median = 1920.0
sqft_living Std Dev = 891.94
sqft_living Mode = 0    1920.00
dtype: float64
```

In [24]:

```
mean = housesales_df['sqft_lot'].mean().round(decimals=2)
median = housesales_df['sqft_lot'].median().round(decimals=2)
stddev = housesales_df['sqft_lot'].std().round(decimals=2)
mode = housesales_df['sqft_lot'].mode(dropna=True).round(decimals=2)

print("sqft_lot Mean = " + repr(mean))
print("sqft_lot Median = " + repr(median))
print("sqft_lot Std Dev = " + repr(stddev))
print("sqft_lot Mode = " + repr(mode))
```

```
sqft_lot Mean = 14814.65
sqft_lot Median = 7620.0
sqft_lot Std Dev = 40504.19
sqft_lot Mode = 0    7620.00
dtype: float64
```

In [25]:

```
mean = housesales_df['floors'].mean().round(decimals=2)
median = housesales_df['floors'].median().round(decimals=2)
stddev = housesales_df['floors'].std().round(decimals=2)
mode = housesales_df['floors'].mode(dropna=True).round(decimals=2)

print("floors Mean = " + repr(mean))
print("floors Median = " + repr(median))
print("floors Std Dev = " + repr(stddev))
print("floors Mode = " + repr(mode))
```

```
floors Mean = 1.49
floors Median = 1.5
floors Std Dev = 0.54
floors Mode = 0    1.00
dtype: float64
```

In [26]:

```
mean = housesales_df['waterfront'].mean().round(decimals=2)
median = housesales_df['waterfront'].median().round(decimals=2)
stddev = housesales_df['waterfront'].std().round(decimals=2)
mode = housesales_df['waterfront'].mode(dropna=True).round(decimals=2)

print("waterfront Mean = " + repr(mean))
print("waterfront Median = " + repr(median))
print("waterfront Std Dev = " + repr(stddev))
print("waterfront Mode = " + repr(mode))
```

```
waterfront Mean = 0.01
waterfront Median = 0.0
waterfront Std Dev = 0.09
waterfront Mode = 0    0
dtype: int64
```

In [27]:

```
mean = housesales_df['view'].mean().round(decimals=2)
median = housesales_df['view'].median().round(decimals=2)
stddev = housesales_df['view'].std().round(decimals=2)
mode = housesales_df['view'].mode(dropna=True).round(decimals=2)

print("view Mean = " + repr(mean))
print("view Median = " + repr(median))
print("view Std Dev = " + repr(stddev))
print("view Mode = " + repr(mode))
```

```
view Mean = 0.23
view Median = 0.0
view Std Dev = 0.77
view Mode = 0    0
dtype: int64
```

In [28]:

```
mean = housesales_df['condition'].mean().round(decimals=2)
median = housesales_df['condition'].median().round(decimals=2)
stddev = housesales_df['condition'].std().round(decimals=2)
mode = housesales_df['condition'].mode(dropna=True).round(decimals=2)

print("condition Mean = " + repr(mean))
print("condition Median = " + repr(median))
print("condition Std Dev = " + repr(stddev))
print("condition Mode = " + repr(mode))
```

```
condition Mean = 3.41
condition Median = 3.0
condition Std Dev = 0.65
condition Mode = 0    3
dtype: int64
```

In [29]:

```
mean = housesales_df['grade'].mean().round(decimals=2)
median = housesales_df['grade'].median().round(decimals=2)
stddev = housesales_df['grade'].std().round(decimals=2)
mode = housesales_df['grade'].mode(dropna=True).round(decimals=2)

print("grade Mean = " + repr(mean))
print("grade Median = " + repr(median))
print("grade Std Dev = " + repr(stddev))
print("grade Mode = " + repr(mode))
```

```
grade Mean = 7.66
grade Median = 7.0
grade Std Dev = 1.18
grade Mode = 0    7
dtype: int64
```

In [30]:

```
mean = housesales_df['sqft_above'].mean().round(decimals=2)
median = housesales_df['sqft_above'].median().round(decimals=2)
stddev = housesales_df['sqft_above'].std().round(decimals=2)
mode = housesales_df['sqft_above'].mode(dropna=True).round(decimals=2)

print("sqft_above Mean = " + repr(mean))
print("sqft_above Median = " + repr(median))
print("sqft_above Std Dev = " + repr(stddev))
print("sqft_above Mode = " + repr(mode))
```

```
sqft_above Mean = 1788.39
sqft_above Median = 1560.0
sqft_above Std Dev = 828.09
sqft_above Mode = 0    1300
dtype: int64
```

In [31]:

```
mean = housesales_df['sqft_basement'].mean().round(decimals=2)
median = housesales_df['sqft_basement'].median().round(decimals=2)
stddev = housesales_df['sqft_basement'].std().round(decimals=2)
mode = housesales_df['sqft_basement'].mode(dropna=True).round(decimals=2)
```

```
print("sqft_basement Mean = " + repr(mean))
print("sqft_basement Median = " + repr(median))
print("sqft_basement Std Dev = " + repr(stddev))
print("sqft_basement Mode = " + repr(mode))
```

```
sqft_basement Mean = 291.51
sqft_basement Median = 0.0
sqft_basement Std Dev = 442.58
sqft_basement Mode = 0    0
dtype: int64
```

In [32]:

```
mean = housesales_df['yr_built'].mean().round(decimals=2)
median = housesales_df['yr_built'].median().round(decimals=2)
stddev = housesales_df['yr_built'].std().round(decimals=2)
mode = housesales_df['yr_built'].mode(dropna=True).round(decimals=2)

print("yr_built Mean = " + repr(mean))
print("yr_built Median = " + repr(median))
print("yr_built Std Dev = " + repr(stddev))
print("yr_built Mode = " + repr(mode))
```

```
yr_built Mean = 1971.01
yr_built Median = 1975.0
yr_built Std Dev = 29.37
yr_built Mode = 0    2014
dtype: int64
```

In [33]:

```
mean = housesales_df['yr_renovated'].mean().round(decimals=2)
median = housesales_df['yr_renovated'].median().round(decimals=2)
stddev = housesales_df['yr_renovated'].std().round(decimals=2)
mode = housesales_df['yr_renovated'].mode(dropna=True).round(decimals=2)

print("yr_renovated Mean = " + repr(mean))
print("yr_renovated Median = " + repr(median))
print("yr_renovated Std Dev = " + repr(stddev))
print("yr_renovated Mode = " + repr(mode))
```

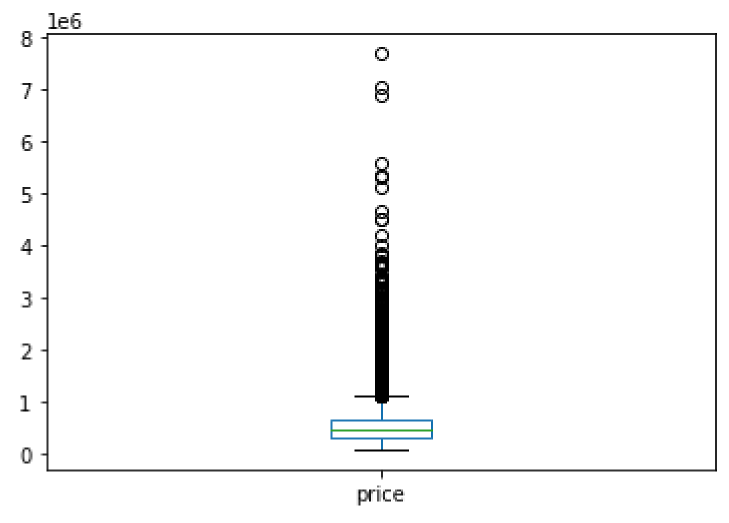
```
yr_renovated Mean = 84.4
yr_renovated Median = 0.0
yr_renovated Std Dev = 401.68
yr_renovated Mode = 0    0
dtype: int64
```

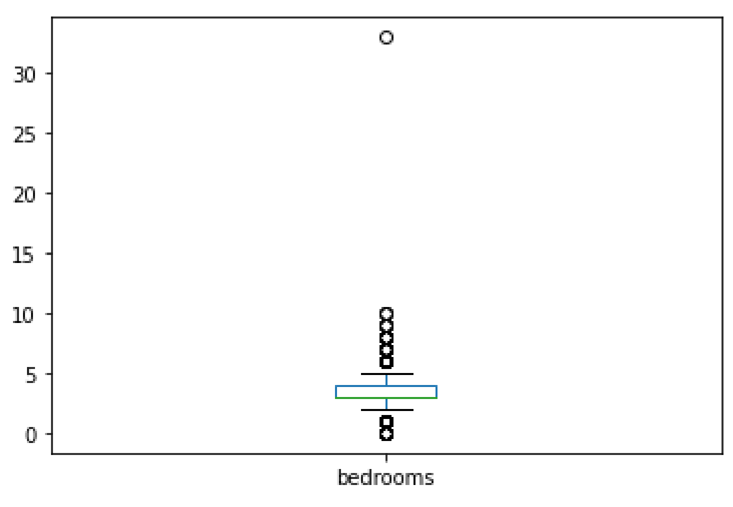
Appendix 2: Boxplots

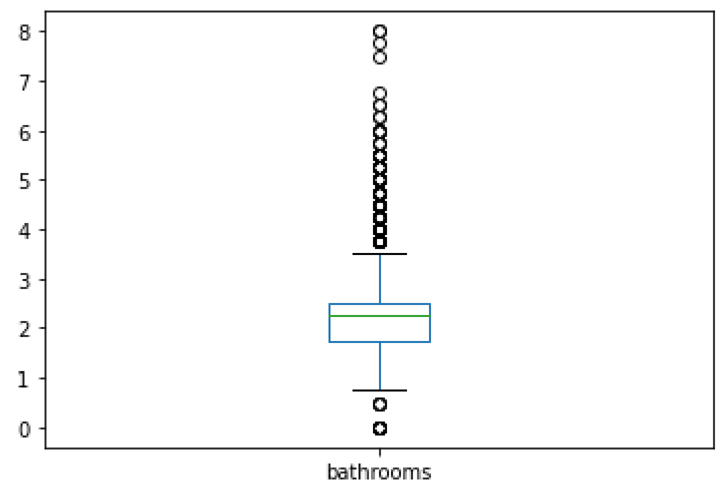
The following visual aids were used throughout our assignment to identify columns with uneven distributions and heavy outliers.

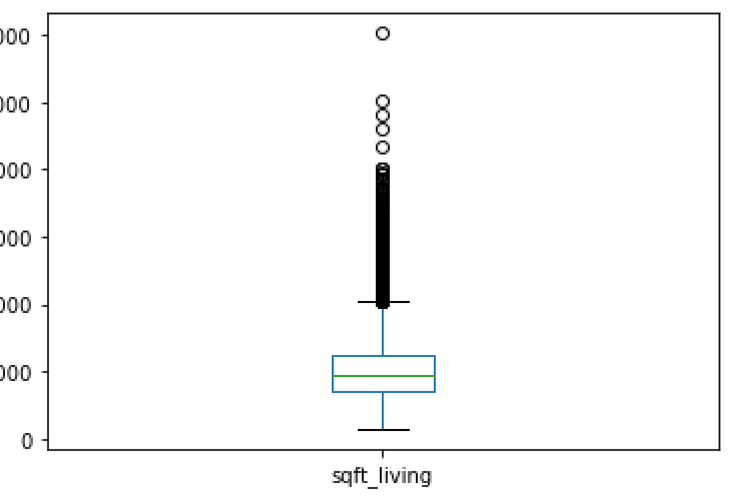
In [34]:

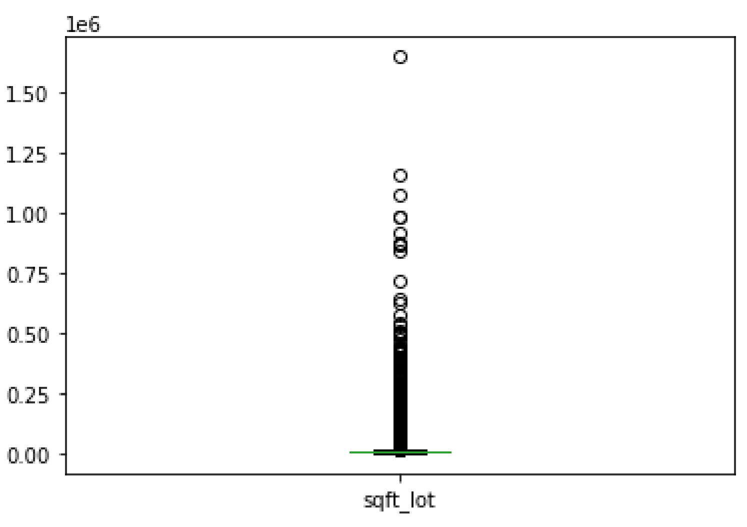
```
housesales_df.boxplot(column='price', grid=False)
plt.show()

housesales_df.boxplot(column='bedrooms', grid=False)
plt.show()

housesales_df.boxplot(column='bathrooms', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_living', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_lot', grid=False)
plt.show()

housesales_df.boxplot(column='floors', grid=False)
plt.show()

housesales_df.boxplot(column='waterfront', grid=False)
plt.show()

housesales_df.boxplot(column='view', grid=False)
plt.show()

housesales_df.boxplot(column='condition', grid=False)
plt.show()

housesales_df.boxplot(column='grade', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_above', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_basement', grid=False)
plt.show()

housesales_df.boxplot(column='yr_built', grid=False)
plt.show()

housesales_df.boxplot(column='yr_renovated', grid=False)
plt.show()

housesales_df.boxplot(column='zipcode', grid=False)
plt.show()

housesales_df.boxplot(column='lat', grid=False)
plt.show()

housesales_df.boxplot(column='long', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_living15', grid=False)
plt.show()

housesales_df.boxplot(column='sqft_lot15', grid=False)
plt.show()
```

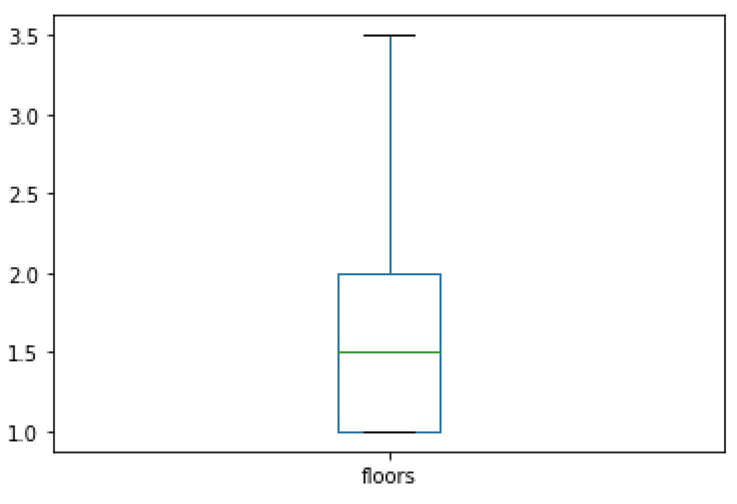


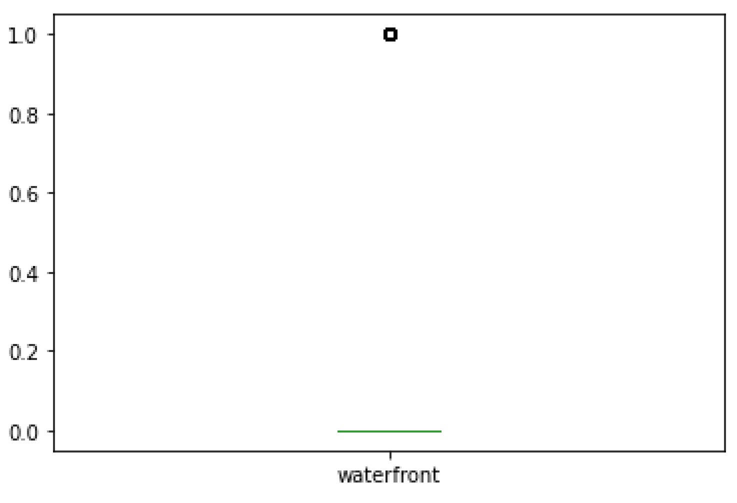


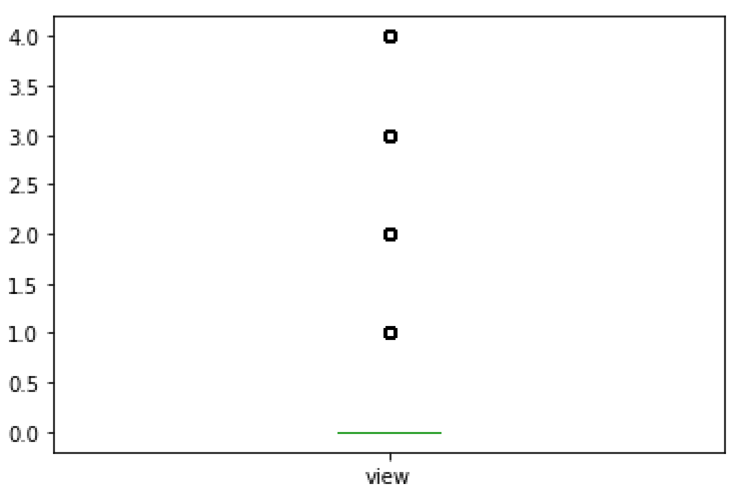

bathrooms

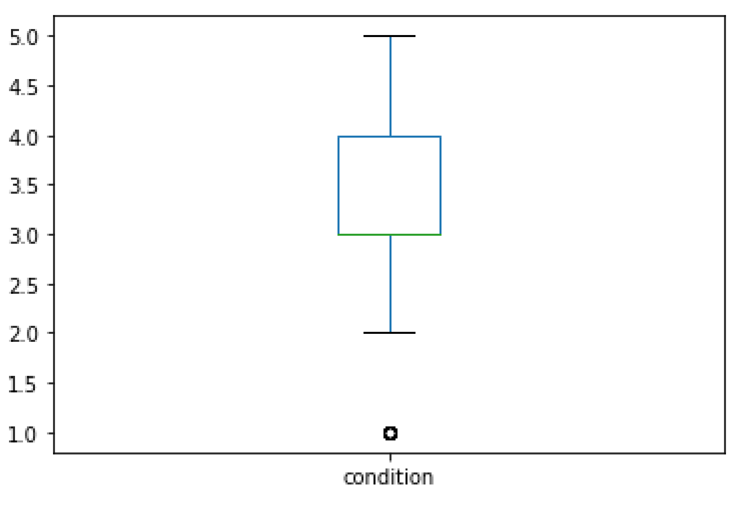

sqft_living

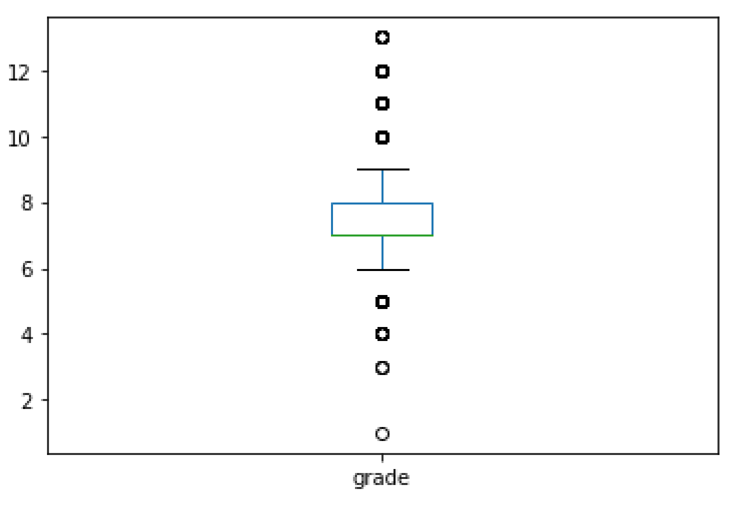

1e6
sqft_lot

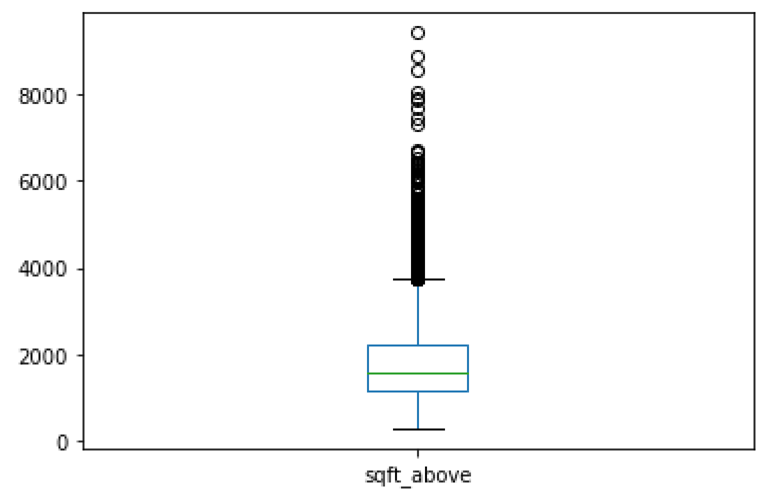

floors

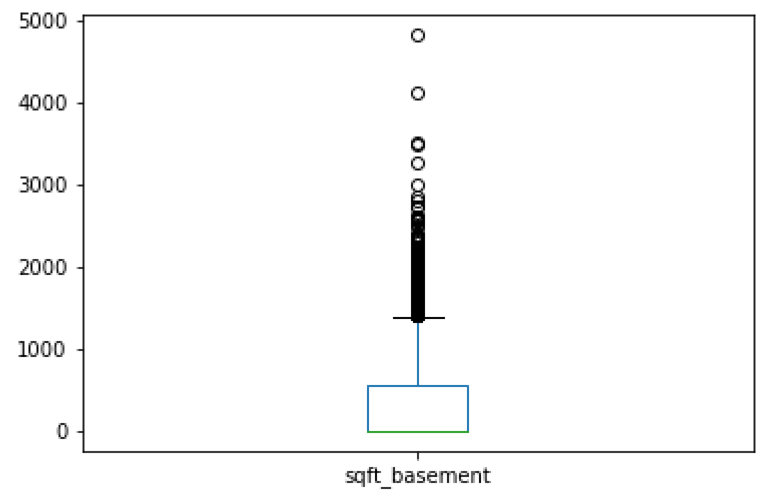

waterfront

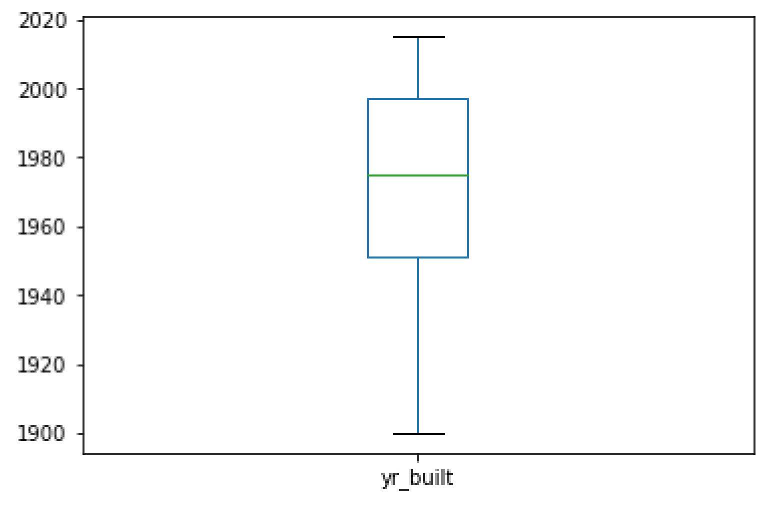

view

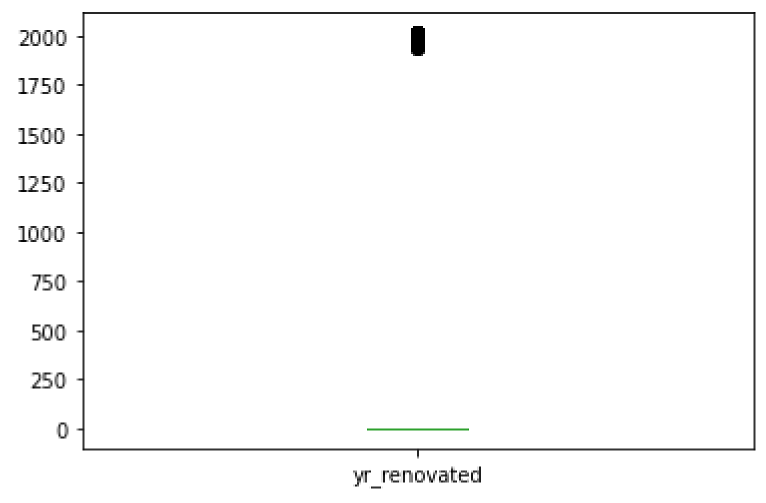

condition

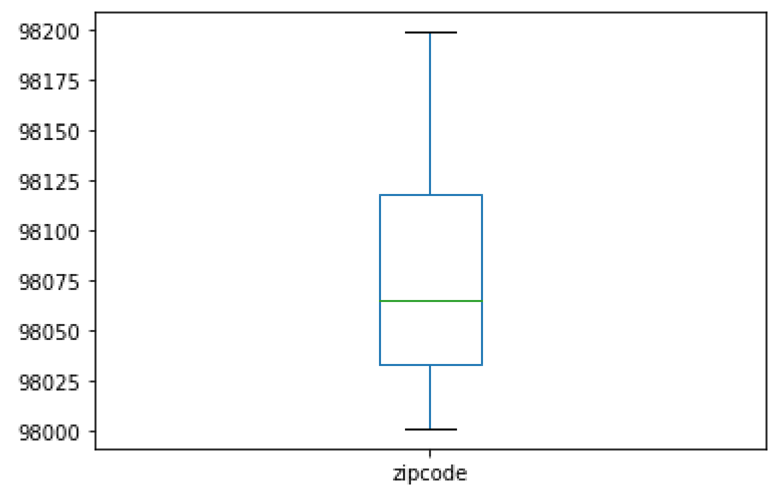

grade

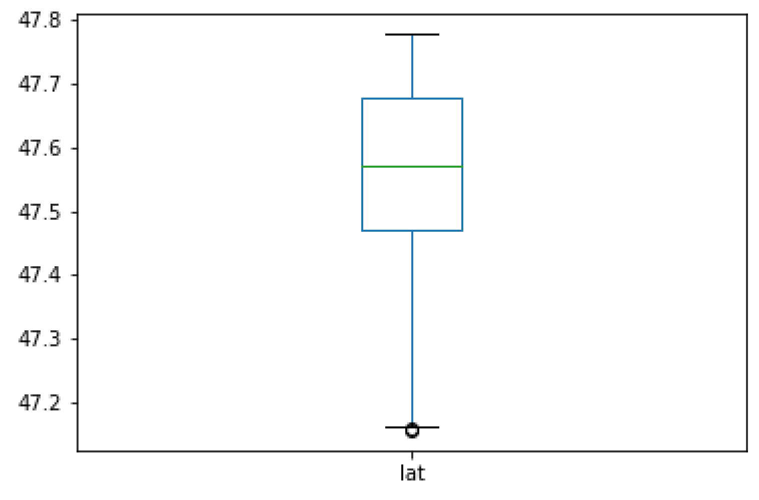

0
2000
4000
6000
8000
sqft_above

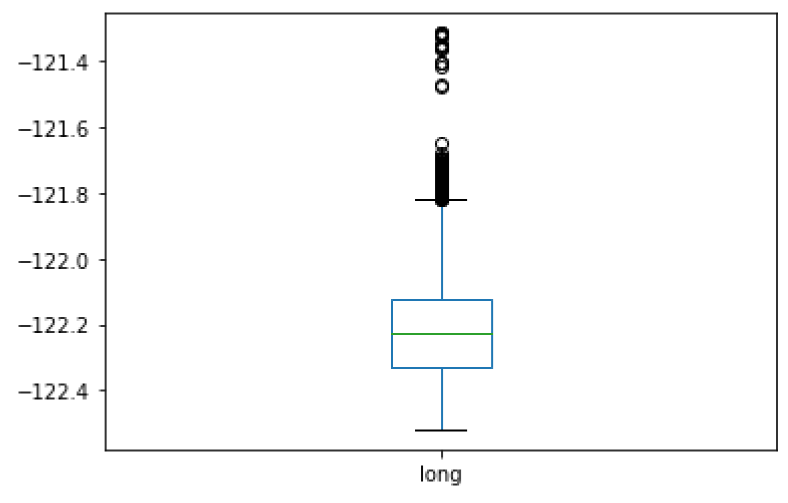

0
1000
2000
3000
4000
5000
sqft_basement

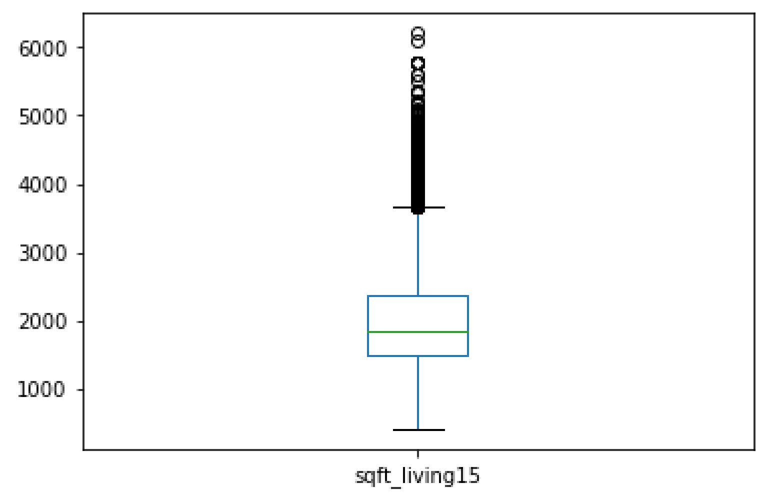

1900
1920
1940
1960
1980
2000
2020
yr_built

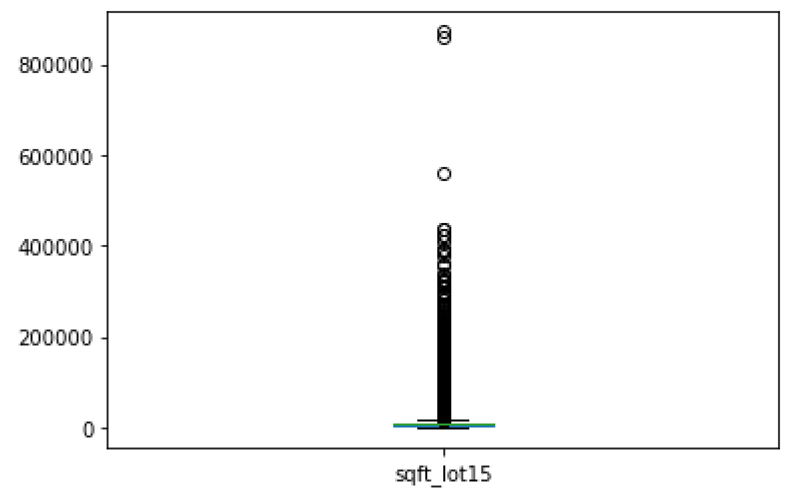

0
250
500
750
1000
1250
1500
1750
2000
yr_renovated


98000
98025
98050
98075
98100
98125
98150
98175
98200
zipcode


47.2
47.3
47.4
47.5
47.6
47.7
47.8
lat


−122.4
−122.2
−122.0
−121.8
−121.6
−121.4
long


1000
2000
3000
4000
5000
6000
sqft_living15

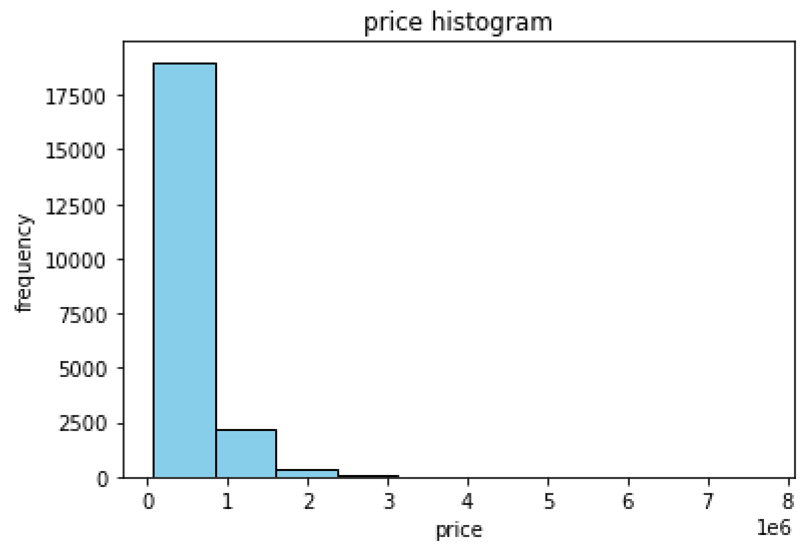


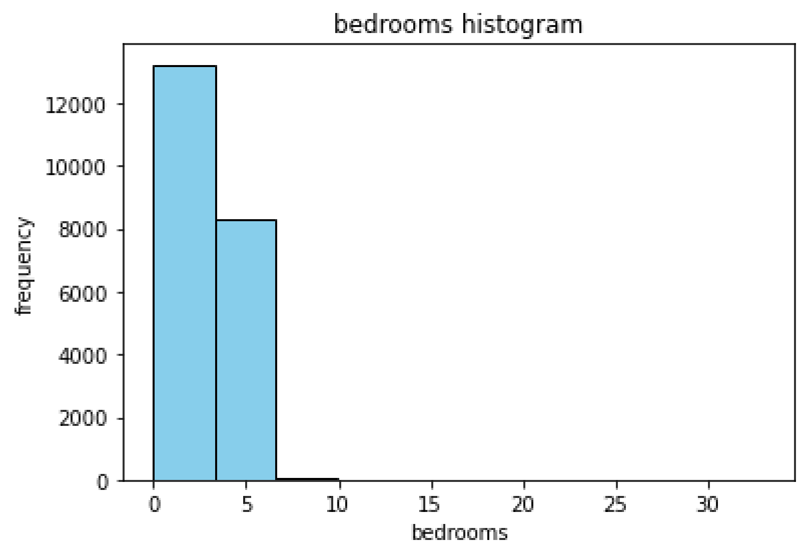
Appendix 2: Histograms

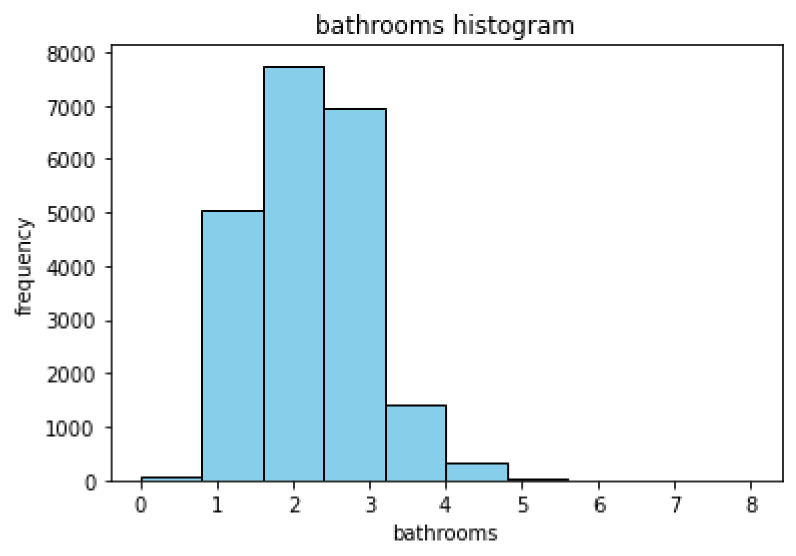
To further aid our analysis of distributions, in tandem with boxplots above.

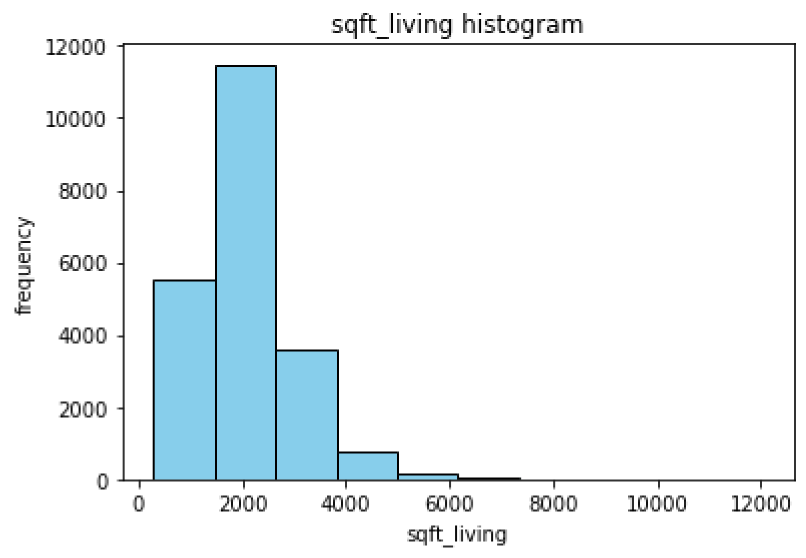
Certain columns were omitted as the distributions didn't provide meaningful information, or the columns were removed later in the assignment as they were redundant.

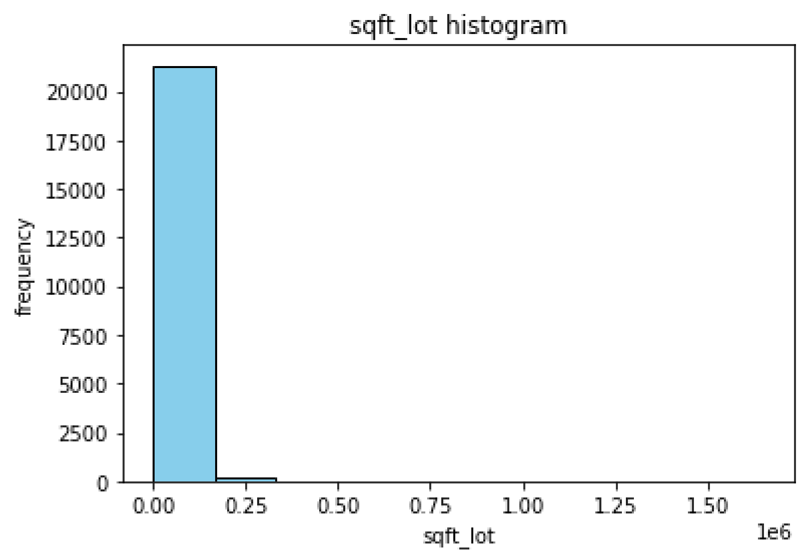
In [35]:
```
plt.hist(housesales_df['price'],bins=10,color="skyblue", ec="black")
plt.title("price histogram")
plt.xlabel("price")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['bedrooms'],bins=10,color="skyblue", ec="black")
plt.title("bedrooms histogram")
plt.xlabel("bedrooms")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['bathrooms'],bins=10,color="skyblue", ec="black")
plt.title("bathrooms histogram")
plt.xlabel("bathrooms")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['sqft_living'],bins=10,color="skyblue", ec="black")
plt.title("sqft_living histogram")
plt.xlabel("sqft_living")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['sqft_lot'],bins=10,color="skyblue", ec="black")
plt.title("sqft_lot histogram")
plt.xlabel("sqft_lot")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['floors'],bins=10,color="skyblue", ec="black")
plt.title("floors histogram")
plt.xlabel("floors")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['waterfront'],bins=10,color="skyblue", ec="black")
plt.title("waterfront histogram")
plt.xlabel("waterfront")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['view'],bins=10,color="skyblue", ec="black")
plt.title("view histogram")
plt.xlabel("view")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['condition'],bins=10,color="skyblue", ec="black")
plt.title("condition histogram")
plt.xlabel("condition")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['grade'],bins=10,color="skyblue", ec="black")
plt.title("grade histogram")
plt.xlabel("grade")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['sqft_above'],bins=10,color="skyblue", ec="black")
plt.title("sqft_above histogram")
plt.xlabel("sqft_above")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['sqft_basement'],bins=10,color="skyblue", ec="black")
plt.title("sqft_basement histogram")
plt.xlabel("sqft_basement")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['yr_built'],bins=10,color="skyblue", ec="black")
plt.title("yr_built histogram")
plt.xlabel("yr_built")
plt.ylabel("frequency")
plt.show()

plt.hist(housesales_df['yr_renovated'],bins=10,color="skyblue", ec="black")
plt.title("yr_renovated histogram")
plt.xlabel("yr_renovated")
plt.ylabel("frequency")
plt.show()
```

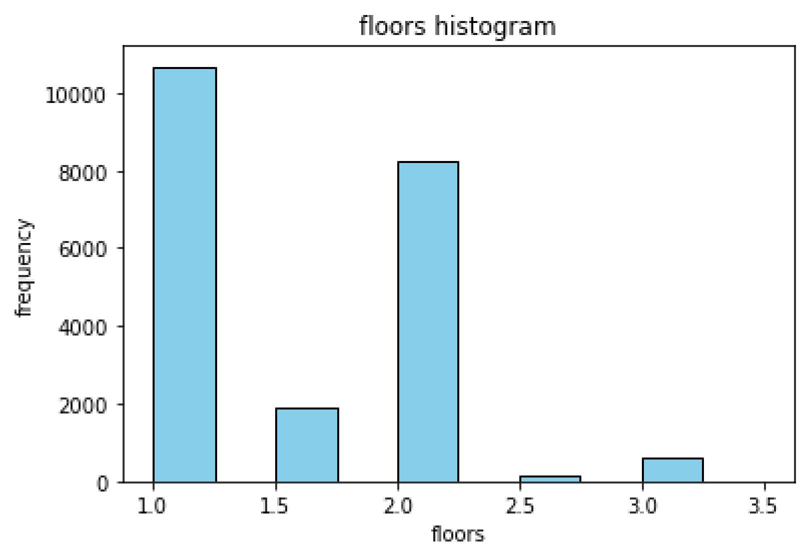


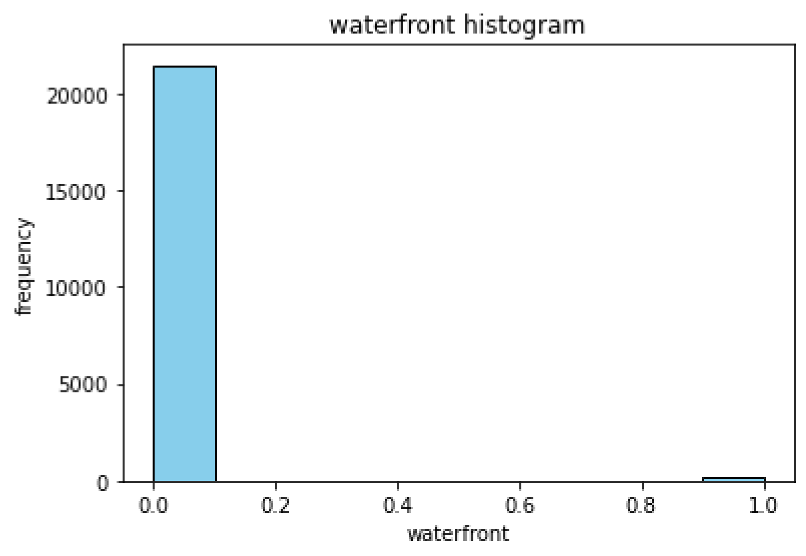

bedrooms histogram
frequency
bedrooms

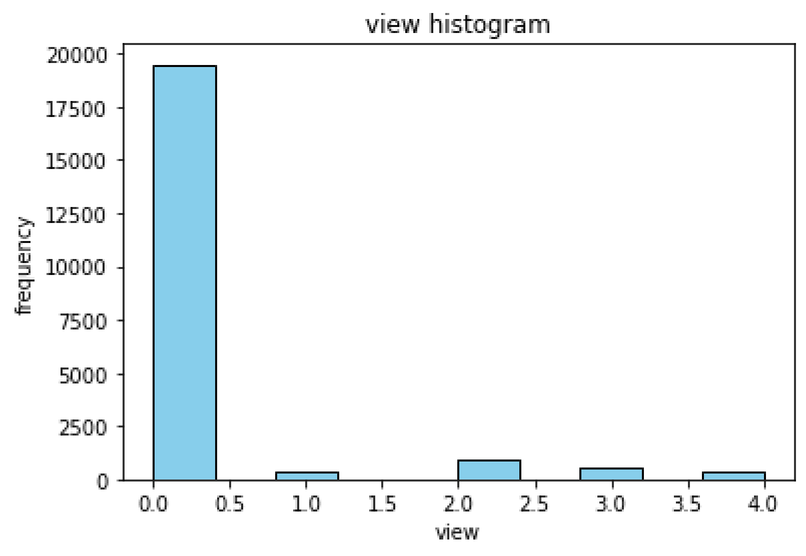

bathrooms histogram
frequency
bathrooms

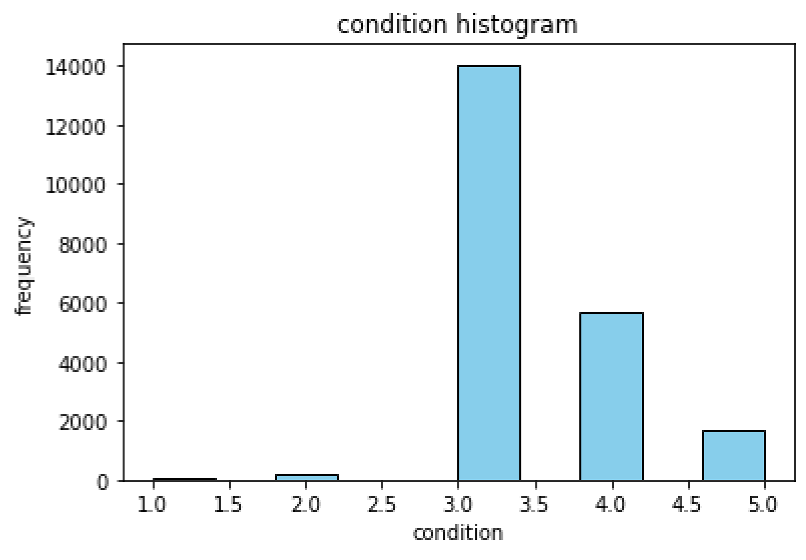

sqft_living histogram
frequency
sqft_living

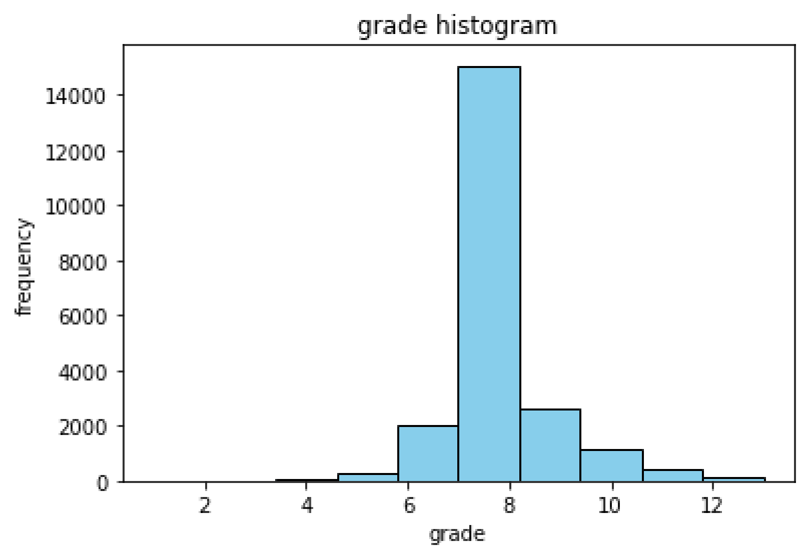

sqft_lot histogram
frequency
sqft_lot
1e6

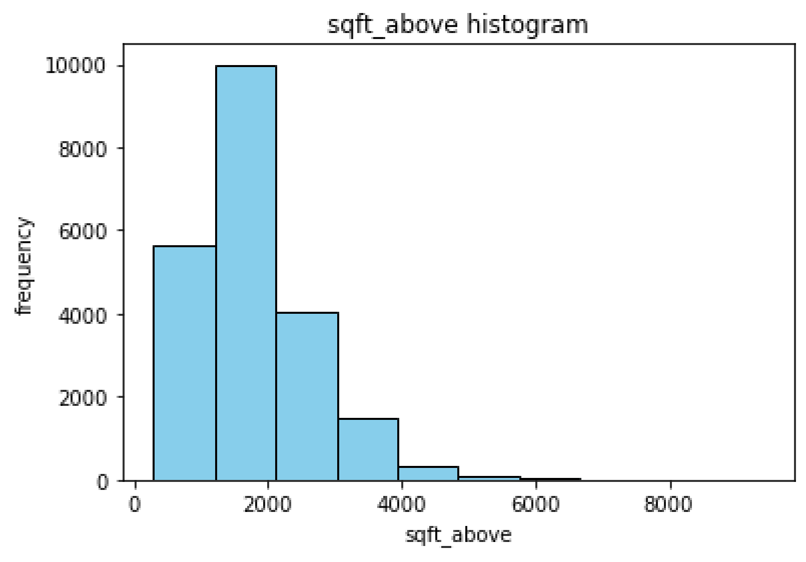

floors histogram
frequency
floors

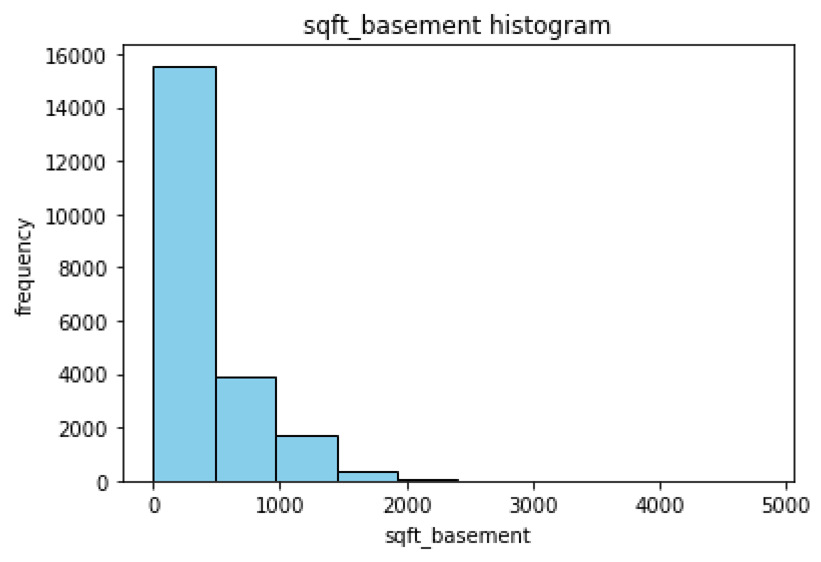

waterfront histogram
frequency
waterfront

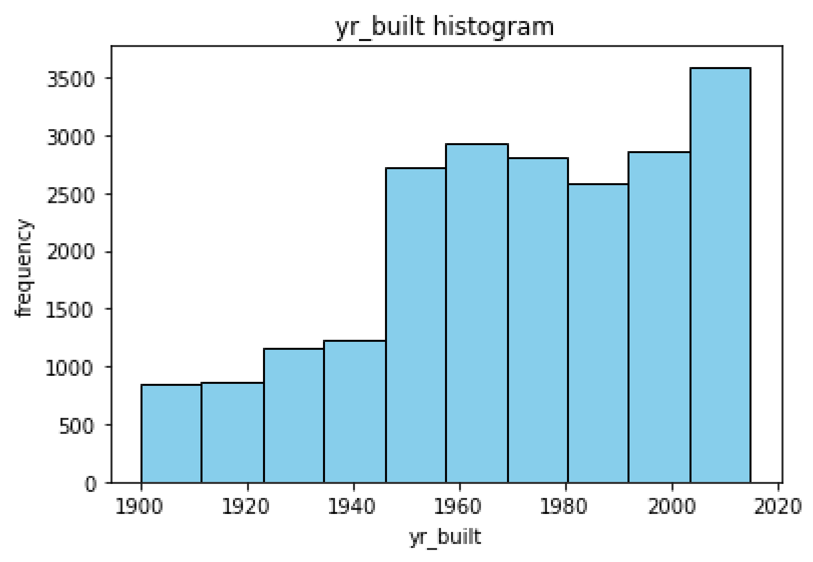

view histogram
frequency
view

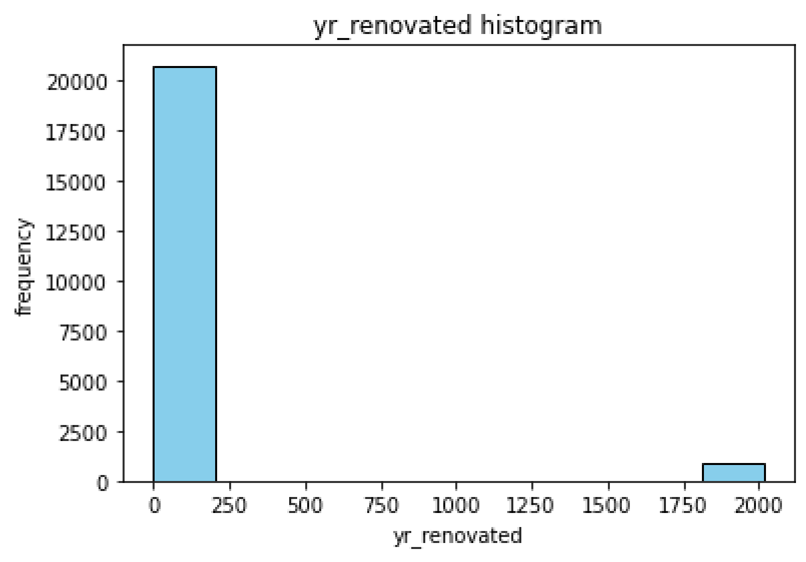

condition histogram
frequency
condition


grade histogram
frequency
grade


sqft_above histogram
frequency
sqft_above


sqft_basement histogram
frequency
sqft_basement


yr_built histogram
frequency
yr_built


yr_renovated histogram
frequency
yr_renovated